An Investigation into Steganalysis Techniques in Media File Forensics and the Use of Machine Learning in Identifying Affected Files

Jessica Sammon

School of Design and Informatics

Abertay University

Bachelor of Science with Honours in Ethical Hacking

Table of Contents

Table of Figures

Table of Tables

Acknowledgements

I would like to thank my family for their unwavering support during my time at university, and for always pushing me to do my best.

I would also like to thank my Honours project supervisor, Salma ElSayed, for her support and guidance over the course of this project.

Abstract

With the rise of digital media and the availability of file editing software, the ability for secret messages to be hidden inside other files has become more accessible for general computer users through the use of steganography. This can be useful for the exchange of confidential messages but can be exploited to disguise and share illicit information. Steganalysis can be used to detect the presence of such steganographic messages in data across a variety of media.

The aim of this project was to prototype a system that implemented current steganalysis techniques and used machine learning to classify target image and video files as clean or stego files, in order to examine the effectiveness of the implemented steganalysis and machine learning algorithms.

A Python program was created that used support vector machines (SVMs) and logistic regression (LR) alongside image and video steganalysis techniques in order to examine the performance of the chosen algorithms. Image and video data from a variety of online datasets was used to create a dataset for training the implemented machine learning algorithms, and then testing the final steganalysis system.

The results of this project show that the process of steganalysis can be very effective at detecting the presence of steganographic messages in digital image and video files. They also show that the use of machine learning can be a great asset to the field of steganography but that the effectiveness of machine learning classifiers varies greatly depending on the size of the dataset used for training and also on the specific machine learning algorithm implemented.

Abbreviations, Symbols and Notation

ANOVA: Analysis of Variance

DCT: Discrete Cosine Transformation

IQM: Image Quality Measure

LR: Logistic Regression

SVM: Support Vector Machine

1. Introduction

1.1. Background

The invention and popularity of modern computing over the last few decades have allowed digital storage to become much more affordable and accessible for general users, which in turn has allowed significantly more people to store and share data electronically. This shift towards digital media has also introduced far easier methods of altering digital information in order to edit, remove or hide messages inside digital data. In an environment where almost anyone is able to easily change digital files in such a manner, the ability to determine the authenticity and security of said files becomes all the more important; especially in a legal or judicial context.

In cases where digital data does have to be examined as evidence in the course of criminal investigations, the data is often inspected by analysts who specialise in digital forensics (Warkentin, Bekkering & Schmidt, 2008) to search for signs of potentially suspicious data or activity. One of the indicators of suspicious activity often explored in digital forensics investigations is the presence of files affected by steganography.

Steganography is the process of hiding one message inside another message, a technique that has been used for covert communications since as early as 480 B.C. (Judge, 2001). The method of detecting and potentially deciphering these steganographic messages is called steganalysis, the specifics of which vary depending on the type of message used as a cover for hiding data.

As outlined in Figure 1, the process of steganography involves a secret message being embedded into a cover file (a cover image in this instance) using a secret key, which then creates a file containing a steganographic message, called a stego file. The secret message can then be extracted from this file by someone who has the appropriate secret key. Steganalysis is performed on the stego file by an interloper who does not have the secret key needed for steganographic extraction, in order to determine if there has been a steganographic message embedded in the file and, if possible, what that message may be.

Although the specifics of steganalysis differ depending on the cover media, the main premise of a secret message being embedded inside a stego file remains the same.

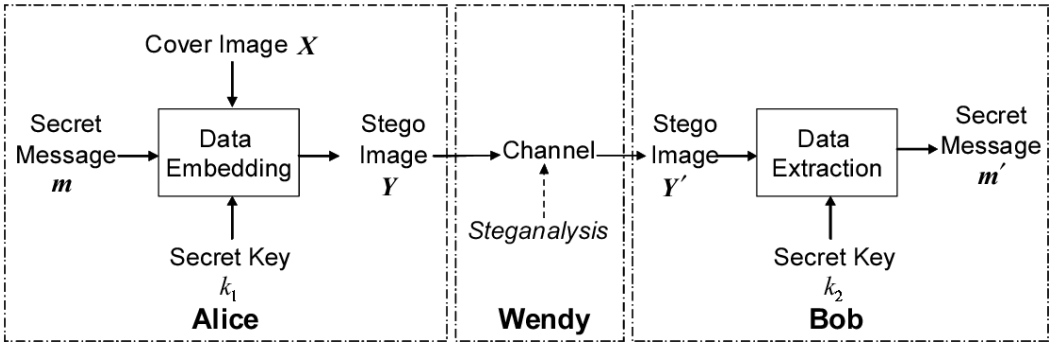

Figure 1: "The model of steganography and steganalysis" (Li et al*., 2011, p. 143)*

In recent research, steganalysis has been conducted with the help of machine learning; a type of artificial intelligence that uses statistics to classify data into pre-determined categories (Jordan & Mitchell, 2015). There are four main types of machine learning – supervised, unsupervised, semi-supervised, and reinforcement learning – which each handle data and form predictions in their own different ways. Supervised machine learning algorithms are used to make predictions on data based on specific statistics gathered from training data wherein the predicted values are already known. Unsupervised algorithms use training data where the predicted values are not already known, and they can also analyse the data to determine patterns in the data statistics that could be used for future predictions. Semi-supervised learning uses a combination of techniques from both supervised and unsupervised learning, while reinforcement learning employs a rewards-based approach to the training process.

1.2. Scope

The purpose of this project is to provide an investigation into steganalysis and steganalysis techniques currently in use, and to examine the viability of machine learning in aiding the detection of steganographic messages within image and video files.

1.2.1. Research Question

The purpose of this investigation is to answer the following research question:

How can steganalysis and machine learning techniques be used to identify the existence of steganographic messages in image or video files?

1.2.2. Aim

The main aim of this research is to develop a prototype system to analyse the operation and effectiveness of various steganalysis techniques against image and video files.

1.2.3. Objectives

The main objectives of this project are to:

- Investigate and review existing steganalysis tools and algorithms for image and video files
- Investigate how machine learning is currently used within steganalysis
- Select, with reasoning, 3-6 steganalysis algorithms to investigate further
- Select, with reasoning, 2 machine learning algorithms to investigate further
- Prototype a suitably featured program/tool to demonstrate implemented steganalysis and machine learning algorithms
- Obtain performance data of the program/tool and the effectiveness of implemented algorithms
- Analyse and evaluate the data
- Make recommendations and suggest future work

1.3. Overview

The remainder of this dissertation focuses on the steps that were taken to address the research question and meet the project aims and objectives throughout the course of the project. Section 2 provides a thorough review on current literature in the field of digital media steganalysis and the use of machine learning in steganalysis. Section 3 details the

methodology followed in creating and testing a prototype steganalysis system for use in examining the effectiveness of several steganalysis algorithms, with the results of the testing carried out outlined in Section 4. Section 5 then offers a discussion of the results and how well the overall project not only answers the research question but also how it meets the project aims and objectives, as well as an overview of the challenges faced during the project. Finally, the project is concluded, and potential areas of future work are highlighted in Section 6.

2. Literature Review

As digital steganography has become both more popular and more utilised, the field of steganalysis focusing on digital media has also become more crucial in attempting to identify and decipher current steganographic techniques. This section provides an analysis of the current techniques used and proposed in digital steganalysis, the recent trend of using machine learning for steganalysis, and the role of steganalysis in the field of digital forensics.

2.1. Current Digital Steganalysis Techniques

In the current digital landscape, there are an extremely wide variety of methods of steganography, which vary depending not only on the type of media (e.g. image, audio, video), but also on the specific file type that the cover media is stored as (e.g. mp4, jpeg, png). As a result, there are also a myriad of steganalysis techniques that have been developed in order to counter the influx of steganographic methods.

In general, steganalysis techniques can be divided into two main categories (Meghanathan & Nayak, 2010): specific steganalysis, which focuses on detecting a singular known steganographic technique only; and blind, or universal, steganalysis, where the potential steganographic algorithm used is unknown. For the purpose of this research, universal steganalysis algorithms focusing on image and video cover files is the primary focus.

2.1.1. Image Steganalysis

Universal steganalysis techniques focusing on image files tend to utilise one of three main methods; statistical analysis, wavelet-based analysis, or feature-based analysis.

One method proposed by Liu *et al.* (2009) used block-based discrete cosine transform (DCT) to decompose an image before reorganising the DCT coefficients. These reorganised DCT coefficients were then utilised as part of the proposed steganalysis algorithm with successful results. A similar method was proposed by Hashemipour and Rahmati (2012) which calculated the DCT coefficients and the local entropy measures to create “information

content criterion" (Hashemipour & Rahmati, 2012, p. 95) for use as steganalysis features. Although both of these techniques prove effective at identifying files with hidden steganographic messages, the use of DCT coefficients is only really efficient with JPEG image files and would not be particularly viable to use with other image formats. Avcibas, Memon and Sankur (2003) also proposed a statistical method of steganalysis, which uses image quality measures (IQMs), specifically analysis of variance (ANOVA) testing, in order to determine if there exist significant statistical differences between steganographic and non-steganographic filtered images. A further method proposed by Shi *et al.* (2005) uses the statistical moments of the characteristic functions of an image, as well as a prediction-error image generated by subtracting the predicted pixel grayscale values from each respective pixel in the image, as the key statistical features for steganalysis.

Wavelet decomposition has also been used to generate statistics for use in steganalysis. Lyu and Farid (2003) first proposed wavelet-based decomposition of an image to obtain higher order statistics in 2003. Yang *et al*. (2009) propose a similar technique which implements wavelet packet decomposition against the image and also introduces the use of an empirical transition matrix. Yu (2018) also outlines a steganalysis technique which, similar to Lyu and Farid's (2003) work, uses wavelet packet decomposition to obtain higher order statistics to be used in the steganalysis process; although they highlight that the methodology could be better implemented as their testing did not give a success rate any higher than 90% (Yu, 2018, p. 90). Another methodology defined by Luo *et al.* (2010) uses wavelet packet transform to define and extract characteristics to be used as features for steganalysis. A method of steganalysis using wavelet coefficient correlation is detailed by Zong, Liu and Luo (2012) although this research again focuses solely on JPEG files. The wavelet absolute moments of image files have also been proposed for use in image steganalysis (Goljan, Fridrich & Holotyak, 2006).

In feature-based steganalysis, certain elements of the image are selected, extracted, and used as 'features' for steganalysis classification. Many of the methodologies described above are also integrated with feature-based steganalysis as methods of feature extraction: Lyu and Farid (2003) for example, use the statistics generated from wavelet decomposition as a 24-dimensional feature vector; and Zong, Liu and Luo (2012) extract a variety of

statistical data from their images' wavelet coefficients for use as a 126-dimensional feature space. Further feature extraction methodologies, such as the non-sampled contourlet transform proposed by Xie, Lai and Zheng (2010) and the method of spectrum-based feature extraction from Deepa *et al.* (2012), are outlined by Babu, Rangu and Manogna (2017). There are a vast number of different methods that can be used for feature selection and extraction, since any number and combination of features could be used for steganalysis. This, however, brings up the issue of model complexity (Schaathun, 2012a); where the more features are introduced, the more complex the model becomes, and so the more likely the resulting classification system will result in incorrect classifications. Thus, there is a trade-off between the number of features used and the acceptable levels of errors for the feature model.

2.1.2. Video Steganalysis

In image files, a steganographic message can only be embedded utilising the pixels of the image, however video files introduce a whole host of new places where information can be hidden without discernible visual changes to the file (Deng *et al*., 2013). Thus, it is not particularly effective to attempt to perform steganalysis of video files using the same algorithms and methodologies that would be used against image files.

One technique of steganalysis more effective against video files involves considering inter-frame features. Xu, Dong and Tan (2012) propose using inter-frame features as well as intra-frame features, through a method of inter-frame feature extraction which considers the difference in pixels between each frame. A similar scheme from Kancherla and Mukkamala (2012) also examines information obtained by examining the differences in between frames, although their research focuses on the pattern noise between each frame instead of the differences in pixels between frames. The ability to use the motion vectors of a video has also resulted in many steganalysis algorithms that examine these areas for identifying features (Deng *et al*., 2013; Xu *et al.*, 2013).

2.2. Machine Learning in Steganalysis

Several of the methodologies outlined above implement machine learning for the classification of data as clean or stego files. Lyu and Farid (2003), for example, use support vector machines (SVMs) as a machine learning classifier, and compare and contrast the differences between linear separable SVMs, linear non-separable SVMs, and non-linear SVMs and their appropriateness within the context of their research.

An SVM was also used by Kancherla and Mukkamala (2012) and Xia *et al.* (2016) above; as well as by Sadat, Faez and Saffai Pour (2018) for video steganalysis, and also by Schaathun (2012a) for image steganalysis. SVMs are one of the more popular methods of machine learning that is used with steganalysis, as much of the available research uses SVMs as the machine learning classifier of choice.

Many other machine learning algorithms can be implemented to work with steganalysis, however. Linear regression was implemented by Gomis *et al*. (2018) to classify the abilities of an image steganalysis algorithm that used a 486-dimensional feature vector against JPEG images. A logistic regression algorithm was utilised by Lubenko and Ker (2011) with the 686-dimensional SPAM feature vector (Pevny, Bas & Fridrich, 2010) as a comparison between logistic regression and SVMs for use in steganalysis. Convolutional neural networks (CNNs) have also been used alongside image steganalysis (Zhang *et al*., 2018; Xu, Wu & Shi, 2016). Ensemble classifiers were implemented as random forests by Kodovsky, Fridrich and Holub (2012) and found to be a viable alternative to SVMs. Hence, although SVMs are the most common method of machine learning implemented for steganalysis, there are many other methods that are just as appropriate.

2.3. Steganalysis in Forensics

The widespread usage of digital media for data storage and transfer means it can be extremely easy to implement steganography digitally and share media that contains steganographic messages. As digital media is also fundamentally linked to modern life, with one in five adults spending over 40 hours a week on the internet (Ofcom, 2018), there is significant potential for general users to be able to learn how to implement steganography without a high amount of effort. Thus, when it comes to digital forensics investigations, there may be a large amount of data that requires thorough analysis for any suspicious alterations, including the insertion of steganographic messages.

Warkentin, Bekkering and Schmidt (2008) highlight the importance of using steganalysis in digital forensics investigations to identify illicit materials, due to the increasing ease in accessing steganography software and the vast amounts of data generally held by digital users; noting that, in an ideal world, every digital forensics investigation should implement steganalysis, but it is instead usually only conducted in investigations directly involving digital media files.

2.4. Summary

Steganalysis can be employed against digital files using a large range of techniques. Image steganalysis can be conducted through a number of different statistical methods that look at either the image itself, or the wavelets of the decomposed image, or even through the use of image transformations. Video steganalysis often focuses on the inter-frame video features, or towards motion-vector based signatures.

The use of feature extraction techniques for steganalysis that utilises machine learning algorithms is also a popular area of research, with new feature extraction algorithms and methods of utilising machine learning with steganalysis being proposed as recently as 2018. A lot of the research that has been conducted into methods of steganalysis that implement machine learning utilise SVMs as the machine learning classifier of choice, however there has also been a great deal of research into steganalysis algorithms which are integrated with

different machine learning classifiers, such as linear regression, logistic regression, neural networks, and ensemble classifiers.

Steganalysis also plays an important role in a digital forensics environment; as more and more general computer users are interacting with a high amount of digital data, the more possible it is for non-technical users to gain access to steganography software. Although steganography tools can be used for legitimate purposes, they can also be exploited for use in the hiding and sharing of illicit information. As such, steganalysis can prove vital in criminal investigations involving digital media, as a crucial method of detecting hidden data.

3. Methodology

The overall methodology was divided into four main phases: design, development, testing, and evaluation. A waterfall methodology was implemented for this process, although an iterative approach was adopted during the design and development stages to allow for potential changes to the planned design in case unexpected issues occurred when implementing the code.

3.1. Design

In order to best facilitate the development of the project, several key aspects of its design needed to be considered before the development phase could begin, including the decision of which programming language to be used and also the way in which data should be handled by the system.

3.1.1. Programming Language

Several points needed to be considered for choosing the programming language that was to be used for developing the steganalysis system; most importantly, the suitability of the language to the project at hand, and the availability of existing steganalysis implementations in the language of choice.

There are two main publicly available instances of image steganalysis algorithms: Fridrich, Holub and Denemark's (2018) MATLAB implementations, and Schaathun's (2012b) PySteg Python package. Fridrich, Holub and Denemark (2018) provide MATLAB instances of over twenty different feature extraction algorithms for image steganalysis, as well as the accompanying academic papers for each algorithm, while the PySteg Python package provides a Python implementation of several feature extraction methods for image steganalysis, each described further in Schaathun (2012a). The MATLAB algorithms extracted a significantly higher number of features compared to the algorithms available in PySteg: the smallest feature set available via the MATLAB implementations extracted 212 features.

Although the aforementioned image steganalysis algorithms are easily accessible, there exist very few publicly available implementations of feature extraction algorithms for video steganalysis. Zhang (2019) provides three such algorithms (Wang, Zhao & Wang, 2014; Zhang, Cao & Zhao, 2017) as pre-compiled executables of C/C++ scripts. However, they can only be applied to video files that are in the form of raw H.264 bitstreams, which is a severe restriction on the breadth of video steganalysis that could be performed.

Ultimately, Python was chosen as the programming language for the project, due to the availability and encompassing features of the PySteg package, as well as the author's familiarity with the language. Zhang's (2019) pre-compiled video feature extraction algorithms could also be run from Python through the use of subprocesses, and so it would be easier for both types of steganalysis to be performed from the same script.

Although PySteg was written in Python 2.7, Python 2.7 is reaching end of life and will not be supported after 2020 (Team Anaconda, 2019). It is, however, possible to call Python 2.7 code from a Python 3 project using subprocesses, and so Python 3 (specifically Python 3.6, the most recent version of Python 3) was chosen as the main programming language. Developing in Python 3 also meant that it would be easier to alter the code if PySteg was updated from Python 2.7 to Python 3 at any point during the project lifecycle.

Because subprocesses were required for running PySteg, the decision was also made to restrict the development, testing and evaluation of the prototype system to Unix systems, specifically Ubuntu 18.04 (Canonical Ltd., 2019) for the purpose of this project.

3.1.2. Feature Extraction Methods

The PySteg package offers several feature extraction methods as part of its 'features' module; the 'farid', 'wam', 'ac', and 'bsm' methods of feature extraction were chosen as the main methods of feature extraction to focus on for this project. Many of the algorithms included in PySteg focus on grayscale images, however they can still be used with colour images if the features are extracted for each colour channel in the image (Schaathun, 2012a). The 'farid' features, based on the wavelet-based algorithm proposed by (Lyu &

Farid, 2003) identify 36 features per colour channel. The 'wam' features are based on Goljan, Fridrich and Holotyak's (2006) WAM features, and extract 27 features per colour channel. The 'ac' features focus on the 10 autocorrelation features proposed by Yadollahpour and Naimi (2009), and the 'bsm' features target the 12 binary similarity measures proposed by Avcibas *et al.* (2005). These algorithms were chosen over the other available methods as they could be applied directly to an image itself, without requiring any prior segmentation or editing to have occurred.

Table 1: Overview of feature extraction methods from PySteg package planned for use in image steganalysis

Algorithm	Features per colour channel	Total features (over 3 colour channels)
farid	36	108
wam	27	81
ac	10	30
bsm	12	36

The three feature extraction methods provided by Zhang (2019) each target a different number of features. The 'AoSO' method, based on Wang, Zhao and Wang's (2014) method of motion vector steganalysis, extracts a mere 18 features, while the 'NPELO' method, based on Zhang, Cao and Zhao's (2017) method of steganalysis based on near-perfect estimation extracts 36. The third method 'IDFB' extracts 768 features, but is based on research that has not yet been published and so has very little documentation. Thus, the NPELO algorithm was chosen for use against video files.

3.1.3. Machine Learning Classifiers

One of the key objectives of the project was to implement and examine the use of two distinct machine learning algorithms with steganalysis. As discussed in Section 2, SVMs are a popular choice of machine learning classifier in steganalysis research and so SVMs were chosen as one of the two machine learning algorithms to implement and evaluate.

Other machine learning algorithms previously used in digital media steganalysis included linear regression, logistic regression, neural networks, and ensemble classifiers. The main issues that were considered in order to decide which algorithm to implement alongside SVMs included the suitability of the algorithm in handling the input data and making classification predictions, and the availability of resources to implement the algorithm in Python.

Further research (Li, 2017) suggested that linear regression would have potentially been more suited to the analysis of purely numerical data. Neural networks, on the other hand, would have been suited to the classification of clean and steganographic data, but would have required a very large dataset of training materials (Raschka, 2016), which would have taken a very long time to process and so would not have been suitable for this project. Although ensemble classifiers implemented as random forests would have been an appropriate algorithm for predicting the classifications of input data in steganalysis, they are often slow to use for predictions once trained (Donges, 2018). A wholly different method of machine learning could have been implemented, but the intention of this project was to investigate current techniques. Logistic regression would be well suited to data classification (Li, 2017), and would not have not required as much training data as neural networks. Hence, logistic regression (LR) was chosen as the second machine learning algorithm to be examined for this project.

3.1.4. Data Flow Diagrams

The processes of training the machine learning classifiers and performing steganalysis were each planned out in pseudocode and then modelled in data flow diagrams as detailed in Figures Figure 2 and Figure 3, so as to begin development with a clear outline of the steps required. This also meant that the expected input and output of each section of the system were well-defined in advance.

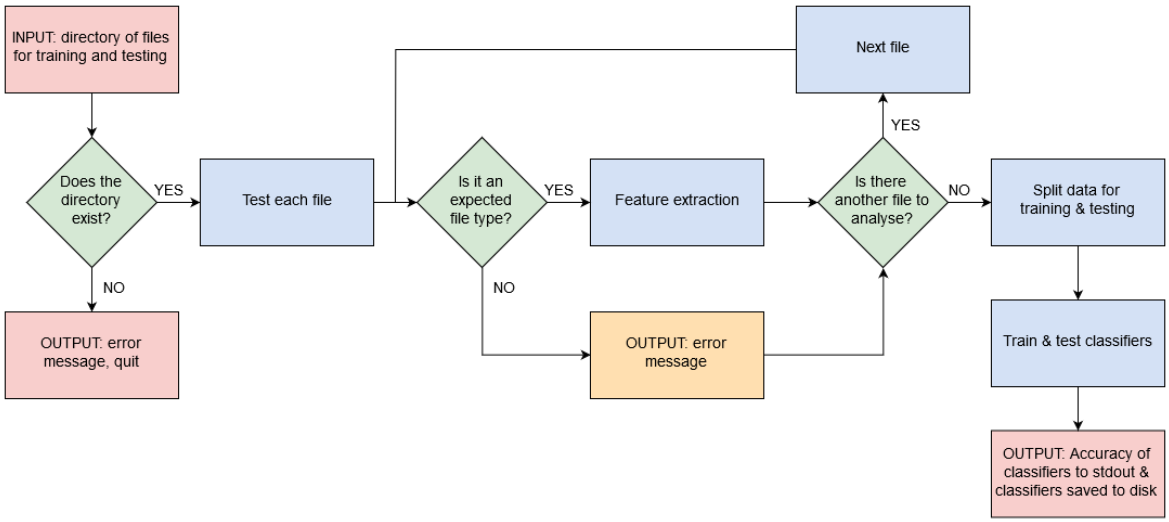

Figure 2: Data flow diagram for the process of training the machine learning classifiers

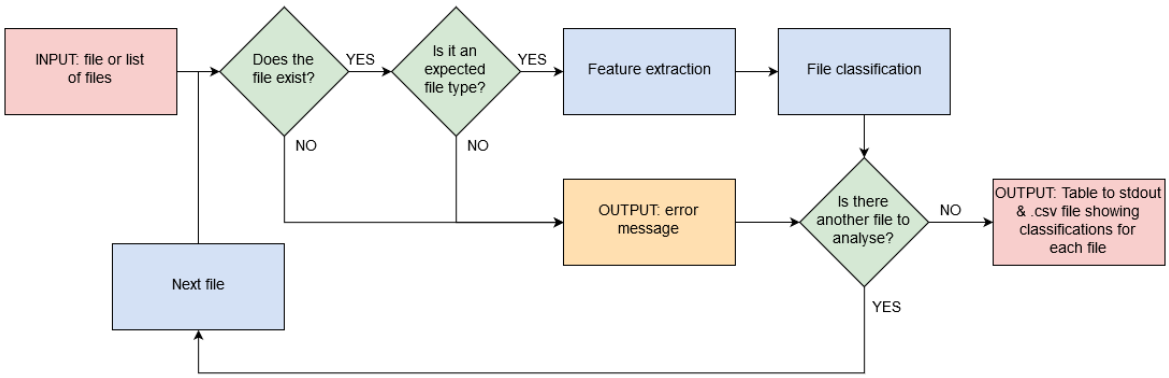

Figure 3: Data flow diagram for the steganalysis process

3.1.5. Preparing Data Sets for Training and Testing

The final design consideration that had to be addressed before development could begin focused on the image and video data required for training the machine learning algorithms and testing the prototype system. Given the large amount of data required in training machine learning algorithms, it was more prudent to make use of existing data sets for training purposes rather than build new data sets entirely from scratch.

The StegoAppDB (Newman *et al.*, 2019) database of clean and stego JPG and PNG files, specifically created for use in image steganalysis research, contained stego PNG files which

were suitable for use, however none of the JPG files could be effectively utilised due to their relatively large file sizes which had significant effects on training time. There were many other data sets of images available that could be modified for use in steganalysis through the use of steganography programs; as such, randomly selected files from the Mendeley (Al-Jarrah, 2018) data set of BMP images, ImageNet's (Deng *et al*., 2009) database of JPG images, and the STL-10 (Coates, Lee & Ng, 2011) PNG images were utilised alongside the appropriate stego PNG images to create a mixed dataset of JPG, PNG and BMP images. Half of the files selected from the ImageNet (Deng *et al*., 2009) and Mendeley (Al-Jarrah, 2018) datasets were embedded with steganographic messages using *steghide* (Hetzl, 2003) to ensure a mix of clean and stego images. The final composition of the utilised training data is described in Table 2.

Table 2: Breakdown of the image data used for training and testing the machine learning classifiers

Data set	**File type**	**Clean files**	**Stego files created with steghide**	**Stego files**	**Total clean files**	**Total stego files**
StegoAppDB	PNG	-	-	100	0	100
ImageNet	JPG	100	100	-	100	100
Mendeley	BMP	100	100	-	100	100
STL-10	PNG	100	-	-	100	0

There are not any datasets that exist explicitly for video steganalysis, although there are many video datasets that have been created for other purposes, such as object detection or behaviour recognition, which could be adapted for use in video steganalysis through video steganography software. The BEHAVE (Blundenson & Fisher, 2010) dataset of AVI videos for labelling human interaction and the VIRAT (Oh *et al.*, 2011) dataset of MPG videos for action recognition were selected as appropriate datasets here as their file types matched those which could be used by the video steganography software *OpenPuff* (EmbeddedSw, 2018). Video files from both datasets were segmented into videos of approximately one minute in length using *ffmpeg* (ffmpeg, 2019), in order to both maximise the size of the data set for training and minimise the length of time required to perform feature extraction on each file.

Half of the segmented videos were then altered to include steganographic messages via OpenPuff.

Table 3: Breakdown of the video data used for training and testing the machine learning classifiers

Data set	**Original file type**	**Number of original files**	**Number of files used**	**Number of approx. 1-minute segments created**	**Number of segments used as clean files**	**Number of segments used as stego files**
BEHAVE	AVI	8	4	8	4	4
VIRAT	MPG	24	20	80	36	44

The segmented clean and stego video files were then converted into H264 files using ffmpeg, as that was the only file format supported by the video feature extraction algorithm that was to be implemented. OpenPuff did not support the H264 file type, and no datasets including H264 files could be found, and so this was the implemented method of creating the video steganalysis dataset.

3.2. Development

Two main Python scripts were created as the basis for the prototype steganalysis system: *train-classifiers.py* and *steganalyse.py*. The machine learning classifiers were created and trained using the *train-classifiers.py* script, while *steganalyse.py* was the main program that actually performed steganalysis on given input files, using the trained classifiers created by *train-classifiers.py* to make predictions on the state of each file.

3.2.1. Initial Setup

As discussed in Section 3.1.1., Python 3 was chosen as the main programming language for the project. PyCharm (JetBrains, 2019) was used as an IDE throughout development, as it

creates a specific Python virtual environment for each project which allowed for easier set-up and requirements handling across the project.

A private GitHub repository was also set up for use as an online back-up of the project and as a method of version control. The repository was updated after every major change in the development, so that changes could be rolled back as required if a significant issue in the implementation arose.

3.2.2. Input Handling

The first steps in the development process were to ensure that data could be properly passed into the system, and that input files could be found in the filesystem and then verified as a supported file type by the system.

The chosen method of handling this was to have input data be a string denoting the path to the target file in the filesystem. An option to enter data as a text file containing the path locations of target files on each new line was also included, in order to make it easier to analyse large numbers of target files.

The '*Fleep*' (Paliienko, 2018) Python module was used to check the file type of input image files via their file signature. Only valid raster image files were considered to be valid image files for the system, RAW images were not. Video file type verification was done in a similar manner but without the use of Fleep, as the video file format required by the NPELO feature extraction algorithm was not in the known list of file signatures contained within the module.

3.2.3. Feature Extraction

Feature extraction occurred in the same way in both *train-classifiers.py* and *steganalyse.py*, with features being stored as a dict object associated with each input file after extraction in both scripts. In the *train-classifiers.py* script, the extracted features of the input data were then saved to .csv files. Due to the major differences in their feature extraction algorithms,

the feature extractions of the image files and video files were handled separately, and image features were saved to one .csv while the features extracted from the video files were saved to another.

3.2.3.1. Image Files

As the PySteg package was written in Python 2.7 and the developed system was written in Python 3, Python subprocesses had to be implemented in order to properly run the PySteg modules. A separate Python 2.7 script (*p2-img-feature-extraction.py*) was created as a handler for the PySteg functions, which was called each time image features needed to be extracted. As can be seen in Figure 4, the name of the target features was passed to the *p2-img-feature-extraction.py* script to indicate which feature extraction method to use.

```
# create subprocess to handle features - pysteg is a python 2 package/collection and so needs to be run in p2
p2_process = subprocess.Popen(['./p2-img-feature-extraction.py', 'farid', file.file_name], stdout=subprocess.PIPE)
stdout, _ = p2_process.communicate()
```

Figure 4: Subprocess called to run p2-img-feature-extraction.py for Farid feature extraction

Once the image had been passed to the target feature extraction function in the Python 2.7 script, it was segmented into its three colour channels, as shown by way of the created function for Farid feature extraction in Figure 5.

```
# FUNCTION: GET FARID FEATURES
def get_farid_features(file_path):

    # get image file from file path
    img = cv2.imread(file_path)

    # copy file x3 for each colour channel
    img_red = img.copy()
    img_green = img.copy()
    img_blue = img.copy()

    # isolate colour channels
    img_red[:, :, 1] = 0
    img_red[:, :, 2] = 0
    img_green[:, :, 0] = 0
    img_green[:, :, 2] = 0
    img_blue[:, :, 0] = 0
    img_blue[:, :, 1] = 0

    # get features for each colour channel
    colour_channels = [img_red, img_green, img_blue]
    for channel in colour_channels:
        print features.farid36(channel)
```

Figure 5: get_farid_features function from p2-img-feature-extraction.py

The PySteg module was then used to extract the target features from each colour channel in the image. The output of this function call was then sent directly back to the original Python 3 program (*train-classifiers.py* or *steganalyse.py*). The resulting features were then parsed into a dict structure and saved as part of the features dict associated with the corresponding file's File object, to be processed for use with machine learning later.

The same process was repeated for the AC, BSM and WAM feature extraction methods, however these resulted in ValueError exceptions (as shown in Figures Figure 6, Figure 7, and Figure 8, respectively) due to the way that the image array was being handled by the modules.

```
Traceback (most recent call last):
  File "./p2-img-feature-extraction.py", line 135, in <module>
    get_ac_features(file_path)
  File "./p2-img-feature-extraction.py", line 38, in get_ac_features
    print features.ac(channel)
  File "/home/jessica/PycharmProjects/Diss-3/pysteg/features/ac.py", line 46, in ac
    if I == None:
ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()
```

Figure 6: Exception thrown when the AC features were attempted to be extracted from an input image

```
Traceback (most recent call last):
  File "./p2-img-feature-extraction.py", line 139, in <module>
    get_bsm_features(file_path)
  File "./p2-img-feature-extraction.py", line 63, in get_bsm_features
    print features.bsm12(channel)
  File "/home/jessica/PycharmProjects/Diss-3/pysteg/features/bsm.py", line 218, in bsm12
    if I == None:
ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()
```

Figure 7: Exception thrown when the BSM features were attempted to be extracted from an input image

```
Traceback (most recent call last):
  File "./p2-img-feature-extraction.py", line 142, in <module>
    get_wam_features(file_path)
  File "./p2-img-feature-extraction.py", line 88, in get_wam_features
    print features.wamNxN(channel)
  File "/home/jessica/PycharmProjects/Diss-3/pysteg/features/wam.py", line 39, in wamNxN
    if I == None: return wamNxNnames(N,nmom)
ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()
```

Figure 8: Exception thrown when the WAM features were attempted to be extracted from an input image

After inspecting the source code for PySteg, the problem appeared to be that the modules were written to comply with an outdated version of *NumPy* (Oliphant, T. E. *et al.,* 2019), which has since updated how it handles arrays and longer supports the syntax used by these modules in PySteg. A clean Python 2.7 build was created with an earlier version of NumPy which did support the array syntax, but this caused more dependency issues and also did not work. The PySteg package was distributed under a GNU Public Licence v3 which, during the development phase of this project, the author believed to prohibit any modifications to the source code material. Hence, after many failed attempts to get the PySteg methods of AC, BSM and WAM feature extractions working without making any modifications to the PySteg source code, the decision was made to focus solely on the Farid features, which could be extracted without issue.

3.2.3.2. Video Files

Because the feature extraction algorithms provided by (Zhang, 2019) were pre-compiled executables, it was originally assumed that they could be run as stand-alone programs. However, it was quickly discovered that the executable was a Windows executable and could not be run on Unix systems. In order to circumvent this restriction, *Wine* (WineHQ, 2019) was installed and a subprocess was used to call the extractor executable from the main Python code, as shown by the section of code in Figure 9.

```
# FUNCTION: GET NPELO FEATURES
def get_npelo_features(file):

    # set up files for bash cmds
    input_file = file.file_name
    output_file = 'temp-features.csv'
    extractor = 'NPELO_extractor/extractor.exe'
    bash_cmd = 'wine {} -s -t 12 -i {} -o {}'.format(extractor, input_file, output_file)

    if os.path.exists(output_file):
        os.remove(output_file)
    print('... Calling subprocess ')
    video_extraction_process = subprocess.Popen([bash_cmd], stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)
    output, error = video_extraction_process.communicate()
    decoded_output = output.decode('utf-8')

    print('... Handling frames')
```

Figure 9: Section of get_npelo_features function used in steganalysis.py and train-classifiers.py showing subprocess command

The NPELO features were extracted after every 12 frames in the video file (Wang, Cao & Zhao, 2017) and saved to a .csv file. These features were then read by the Python program and all extracted features were saved to the features dict associated with the corresponding video File object, to be processed for use with machine learning later.

3.2.4. Training the Machine Learning Classifiers

The SVM and LR classifiers were trained with the image and video datasets created during the design phase (Section 3.2), using the .csv files of input file features created during feature extraction. Due to the high amount of data required for training the classifiers, the input was changed to accept a directory path location containing the training dataset instead of individual files.

The svm and linear_model modules of the *sklearn* (Pedregosa *et al.*, 2011) Python package were used for creating the SVM and LR classifiers, respectively, as shown in Figure 10 and Figure 11. The input data was split into training and testing data, with the training data used to train the algorithm and the testing data then being used to determine the accuracy of the trained classifier by making predictions on the classification of known data. For logistic regression, the input data was standardised before it was used for training the classifier in order to help ensure reliable training could take place (scikit-learn developers, 2018).

The trained classifiers were then saved to disk as .joblib files to be loaded in to *steganalyse.py* as required for model persistence, so that the classifiers would not need to be retrained each time the system is run. The *train-classifiers.py* could be run at any point, however, if new classifiers were required.

```
def create_svm_classifier(file_type):
    # check file type + load csv
    if file_type == 'image':
        csv_file = '{}/img-features.csv'.format(input_dir)
        joblib_file = 'img-svm.joblib'
    else:
        csv_file = '{}/vid-features.csv'.format(input_dir)
        joblib_file = 'vid-svm.joblib'
    # create + train svm
    print('=== Handling SVM for {} files ... ==='.format(file_type))
    print('[*] Reading {} ... '.format(csv_file))
    training_data = pandas.read_csv(csv_file)
    print('[*] Getting x and y ... ')
    x = training_data.drop(['file_name', 'class'], axis=1)
    y = training_data['class']
    print('[*] Splitting data ... ')
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=split)
    print('[*] Setting classifier ... ')
    classifier = svm.SVC(kernel='linear')
    print('[*] Training classifier (note: this may take a while) ... ')
    classifier.fit(x_train, y_train)
    print('[*] Getting predictions ... ')
    y_pred = classifier.predict(x_test)
    print('[*] Finding accuracy ... ')
    print('... Accuracy: {}'.format(metrics.accuracy_score(y_test, y_pred)))
    print('[*] Saving classifier as {} ... \n'.format(joblib_file))
    joblib.dump(classifier, joblib_file)
```

Figure 10: create_svm_classifier function created to handle the training and testing of SVM classifiers

```
def create_lr_classifier(file_type):
    # check file type + load csv
    if file_type == 'image':
        csv_file = '{}/img-features.csv'.format(input_dir)
        joblib_file = 'img-lr.joblib'
    else:
        csv_file = '{}/vid-features.csv'.format(input_dir)
        joblib_file = 'vid-lr.joblib'
    # create and train lr classifier
    print('=== Handling Logistic Regression for {} files ... ==='.format(file_type))
    print('[*] Reading {} ... '.format(csv_file))
    training_data = pandas.read_csv(csv_file)
    print('[*] Getting x and y ... ')
    x = training_data.drop(['file_name', 'class'], axis=1)
    y = training_data['class']
    print('[*] Scaling x ... ')
    scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
    x = scaler.fit_transform(x)
    print('[*] Splitting data ... ')
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=split)
    print('[*] Setting classifier ... ')
    classifier = linear_model.LogisticRegression()
    print('[*] Training classifier (note: this may take a while) ... ')
    classifier.fit(x_train, y_train)
    print('[*] Finding accuracy ... ')
    print('Accuracy: {}'.format(classifier.score(x_test, y_test)))
    print('[*] Saving classifier as {} ... \n'.format(joblib_file))
    joblib.dump(classifier, joblib_file)
```

Figure 11: create_lr_classifier function created to handle the training and testing of LR classifiers

3.2.5. Classifying New Data

Once feature extraction and machine learning classification had been completed, the *steganalyse.py* program was finalised to handle the classification of new input data. Files underwent feature extraction as in Section 3.3.3. in order to obtain the necessary file features for classification. The .joblib classifier files were loaded in and unpacked, and were then handled as SVM and LR classifiers by the system. Once the classifiers were successfully loaded, they were used for making predictions on the steganographic content of the target input files based on the files extracted features.

3.3. Testing

The testing phase of the methodology examined the system's ability to run as expected with both expected and unexpected input, and also examined the effect of changing the data size for training the classifiers on the accuracy of the trained classifiers.

3.3.1. Classification Accuracy

In order to verify the accuracy of the classifiers, the classifiers were re-trained twice more with the same data. The nature of the 'train_test_split()' function (Pedregosa *et al.*, 2011), used to split the input data into training and testing data, as shown in Figure 12, meant that the pre-processed files were chosen at random for each category; thus, different files from the input data would be used for training and testing every time the classifier was trained.

```
print('[*] Splitting data ... ')
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

Figure 12: train_test_split() function from the model_selection module of the sklearn Python package (Pedregosa et al.*, 2011)*

As can be seen by the 'test_size=0.2' in Figure 12, the data was split into training and testing data with 20% of the data used for testing and the other 80% of the data used for training. The classifiers were also retrained twice again with different divisions of training and testing data, to examine any affect that changing the training size may have on the accuracy of the classifiers; they were first retrained with an 85% training data and 15% testing data split, and again with a 75% training data and 25% testing data split.

3.3.2. Input Handling

A mixture of tested and untested data was used to test the performance and accuracy of the *steganalysis.py* program. Both expected and unexpected file types were used, and non-existent files were included in the input list, in order to test and refine the input handling performed by the system.

3.4. Evaluation

Files were randomly selected from the datasets discussed in Section 3.2.5. to create a mix of files that had and had not been previously used for training purposes. An equal number of clean and stego files were used for each file format, as outlined in Table 4.

Table 4: Breakdown of files used for testing and evaluating the system

File type	Number of clean files	Number of stego files	Total number of files
JPG	20	20	40
PNG	20	20	40
BMP	20	20	40
H264	8	8	12
All files	68	68	132

These files were then used as new input data for *steganalyse.py*, in order to evaluate the performance of the steganalysis system. The implemented classifiers were trained using a split of 80% training and 20% testing data (Table 5).

Table 5: Accuracy of the classifiers used for system evaluation

Focus of classifier	Accuracy of SVM classifier	Accuracy of LR classifier
Images	0.641667	0.575
Videos	0.546599	0.574501

3.5. Summary

The tests described above were conducted in order to examine the accuracy, and thus the effectiveness and appropriateness, of the implemented algorithms against the supported media file types. Secure input handling targeting the system's ability to withstand malicious or deliberately disruptive input was not tested, as the goal of the project was to create a prototype system and so fully secure input handling was not the focus of the project. Despite this, efforts were made to ensure the system could adequately handle unexpected input.

4. Results

This section presents an overview of the results from the testing and evaluation conducted in Sections 3.3 and 3.4.

4.1. Effect of Training Size on Classifier Accuracy

The results of testing described in Section 3.4 are detailed in full in Appendix B, with the accuracy of the classifiers given to 6 decimal places. The average accuracy of the trained classifiers by ratio of training and testing data across the three iterations of testing is shown in Figure 13. The highest level of accuracy across the majority of the classifiers was obtained from an 80% to 20% split of training and testing data, with the performance of the 75% to 25% and 85% to 15% training to testing data giving similar but lower results. The exception to this was the LR image classifier, which had significantly less accurate results from a 75% to 25% training to testing split of input data than from any other division of data.

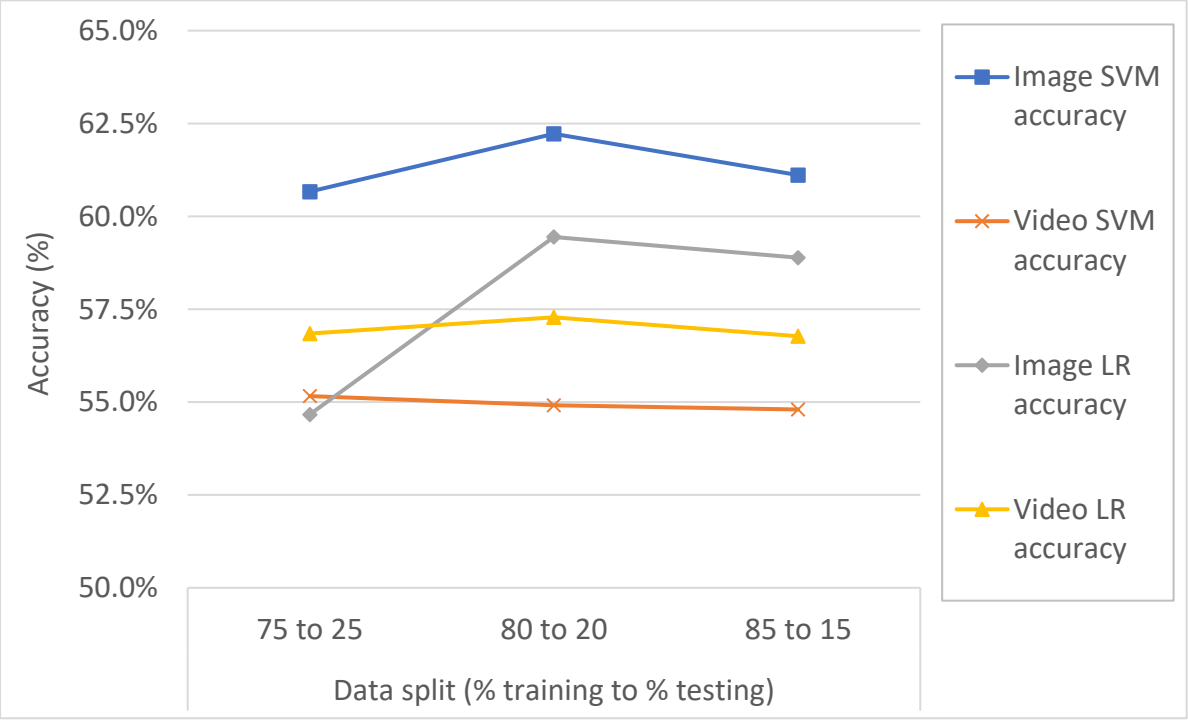

Figure 13: Line graph showing the average accuracy of trained classifiers by ratio of training and testing data used

4.2. Accuracy of Steganalysis System

The steganalysis predictions made by the system during evaluation (Section 3.5) were compared with the known classifications for each given input file, in order to determine the accuracy of the system against new data. The number of correct predictions by the SVM and LR classifiers is detailed in Table 6.

Table 6: Results of steganalysis against new input data

File type	Number of files	Correct SVM predictions	SVM accuracy	Correct LR predictions	LR accuracy
All files	132	82	62.12%	60	45.45%
JPG	40	16	40.00%	17	42.50%
PNG	40	32	80.00%	19	47.50%
BMP	40	28	70.00%	19	47.50%
H264	12	6	50.00%	5	41.67%

As can be seen from the table above, the overall accuracy of both of the trained classifiers against new data was roughly the same as their accuracy against the testing data used during training. The accuracy of the LR classifiers were relatively consistent across file types, however, the accuracy of the SVM classifiers did vary greatly between each file type. For example, the SVM image classifier had the highest level of accuracy against PNG files, and the lowest level, half of the PNG accuracy, against JPG files.

4.3. Summary

In general, the accuracies of the trained classifiers did not vary greatly when different sizes of training data were used; there was no more than a 5% difference in the accuracy of each classifier when faced with the different divisions of training data. The accuracy of the final steganalyse.py system ranged from 40% to 80%, depending on the type of classifier used and the file type of the targeted file. The implemented SVM classifiers also performed slightly better than their LR counterparts across almost all tested file types.

5. Discussion

The testing and evaluation results outlined in Section 4 offered a great deal of insight into the effectiveness of the algorithms implemented in the prototype steganalysis system. The effectiveness of these algorithms was crucial to defining the success and suitability of the methods used, and could also be used to offer potential insight into sections of the project where conclusive results could not be obtained.

5.1. Discussion of Results

From the results of the accuracy and performance testing as described in Section 4, it can be seen that the implemented steganalysis algorithms and classifiers were all over 40% accurate, with the general accuracy of all predictions averaging at around 58% (Table 10, Appendix B). The system was most accurate at predicting the classifications of PNG images, with the accuracy of the SVM image classifier reaching 80% against these files, and was least accurate at identifying steganography in videos and JPG images, with SVM and LR classifiers both performing with approximately 40%-50% accuracy against these files. Based on existing research into image steganalysis as discussed in Section 2, it was unexpected that the accuracy of both the SVM and LR classifiers would be so low against JPG files. There are, however, a number of reasons why the accuracy could have been so low.

Since the accuracy of the SVM and LR classifiers were roughly the same for JPG images, it is possible that the level of successful stego detection in JPG files was so low is because the extracted features based on Lyu and Farid's (2003) wavelet characteristics were not the most suitable features for JPG steganalysis. However, Lyu and Farid's (2003) research yielded generally higher levels of accuracy against JPG images while also using an SVM classifier, and so it is unlikely that the classifiers were the issue.

Similarly, the low levels of accuracy in detecting video steganography may have been because of issues with the NPELO method of feature extraction, or because of the small sample size of video files used to train the machine learning classifiers. The original research that proposed this feature extraction algorithm (Zhang, Cao & Zhao, 2017) obtained accuracies no lower than 70%, but used sample sizes of either 50, 200 or 800 files – all

significantly more than the 40 files used to train the machine learning classifiers implemented in this prototype system. However, the low level of accuracy in identifying stego video files may have instead been because the videos were not in H264 format originally; steganography was performed on AVI and MPG files which were then converted into H264 files. The effects of converting these files into the H264 format may have had a significant impact on the ability to detect the messages in the stego files.

A well-trained, accurate classifier is clearly instrumental to the steganalysis prediction process conducted by machine learning. Such classifiers require a significant amount of training materials in order to make accurate predictions, and the training set should be large enough to be split into training and testing data in order to verify the accuracy of the trained classifier before it is implemented against new data. Although the effect of different divisions of training and testing data was examined, the implemented data was still drawn from the dataset outlined in Table 4. It is possible that a significantly larger or smaller dataset would cause significantly different results.

As well as this, the results from Section 4 also show that the LR classifier performed less accurately than the SVM classifier at correctly identifying the presence of steganography across all tested file formats. This suggests that the LR classifiers were not as effective at distinguishing between clean and stego input files. This may have been because the classifiers were not trained with enough data, but it may instead suggest that LR is not as effective for use with steganalysis as SVMs.

The accuracy of the steganalysis system varied between 40% and 80%, which could have been due to a variety of reasons. For example, a relatively small number of features were used for the decision-making process, and it is possible that a more accurate system could have been produced if a larger number of features was used.

5.2. Addressing the Research Question, Aims, and Objectives

The proposed research question was “How can steganalysis and machine learning techniques be used to identify the existence of steganographic messages in image or video files?” This project explored the use of steganalysis and machine learning techniques and found them to be very useful in identifying files that potentially contain steganographic messages, with the prototype system created for analysing steganalysis techniques averaging around 58% accuracy in identifying files affected by steganography. It is clear that machine learning lends itself very well to media file steganalysis, where any file changes made by steganography can often be difficult to manually identify from media file analysis alone, however it is also evident that the machine learning algorithm used for classifications does have a vital role in affecting the accuracy of the overall system.

The aim of this project was to develop a prototype system for use in analysing the operation and effectiveness of various steganalysis techniques against image and video files. The prototype system that was created ultimately examined the effectiveness of one method of image steganalysis and one method of video steganalysis, implemented alongside SVM and LR machine learning classifiers, in making predictions on whether or not target image and video files contained steganographic messages.

The project did attempt to investigate a further three feature extraction algorithms for image steganalysis, however these could not be implemented and fully examined due to issues with incompatible and outdated module code. Had these algorithms been implemented, the difference in performance between feature sets for each media type could have been examined in detail, and the effect of feature size on the effectiveness of steganalysis could have been explored.

5.3. Challenges

The main challenge in effectively creating the prototype system was the lack of publicly-available steganalysis algorithms. The PySteg (2012b) package that was ultimately implemented was written in Python 2.7, a language which has been slowly fazed out of use in recent years and will no longer be supported after 2020. As the package has not been

updated since it was written and released in 2012, many of its functions had outdated coding syntax incompatible with newer versions of the required modules, which meant it was difficult to successfully implement the package without having to make changes to the code. Due to confusion surrounding the licensing of the package, wherein the author mistakenly believed that the package's license prohibited any modifications to the code, it was not possible to implement another method of image feature extraction until too late in the project, and so the actual performance of three of the four planned image steganalysis techniques could not be properly investigated.

6. Conclusion

In conclusion, there is a wide range of steganalysis techniques currently in use against digital image and video files, each of which target difference aspects of the files in an attempt to find evidence of steganographic content. Machine learning has also been implemented alongside steganalysis in recent research, to make predictions on the existence of steganography within digital files.

A prototype system was created to further investigate several current steganalysis techniques, and to examine the use of machine learning in aiding these steganalysis techniques. Although there were some difficulties in implementing the steganalysis algorithms, the Farid features (Schaathun, 2012b) of image files and NPELO features (Zhang, 2019) of video files were successfully extracted and used to train SVM and LR machine learning classifiers which were then, in turn, used to make predictions on the existence of steganography against input files of unknown classifications.

The machine learning classifiers created for the project were also tested and evaluated, to determine their accuracy in correctly predicting the classification of input image and video files. The implemented SVM and LR classifiers were trained using 400 image files and 80 video files, with approximately half of the dataset having been affected by steganography. When used against new input data, the SVM classifiers for both image and video files were found to be more effective than their LR counterpart; the SVM classifiers were around 60% accurate at classifying files correctly on average, while the LR classifiers were only around 45% accurate on average.

Overall, the project was successful in answering the research question: the project results show that steganalysis certainly can be very effective at identifying files affected by steganography, and that machine learning can be easily integrated into the steganalysis process. However, the project was only somewhat successful in meeting the project aims and objectives, as only one steganalysis technique was investigated for each type of expected input file. Three further algorithms for image steganalysis were explored but, unfortunately, could not be implemented within the project timeframe.

6.1. Future Work

The prototype system created for this project could be extended to examine a range of other steganalysis algorithms. In particular, the AoSO and IDFB methods (Zhang, 2019) of feature extraction for video files could be examined as possible alternatives to the NPELO feature extraction currently implemented in the system, as their effectiveness at correctly classifying video files was not assessed. The associated research paper for the IDFB method has also not yet been published, but may offer new insights into the topic at hand. In terms of image steganalysis, modifications could be made to the syntax of the broken PySteg (Schaathun, 2012b) modules in order to fix the issues caused by the outdated code, and allow for easier implementation of the algorithms that were not tested in this project. As Python 2.7 will no longer be supported after 2020, the entire PySteg package could be overhauled and re-written in Python 3; this would remove the current issues in the code and make it easier to implement code fixes in the future. However, such an extensive change to the code would likely require the permission of the original author as the code is freely-available but not open sourced.

The algorithms provided by Fridrich, Holub and Denemark's (2018) MATLAB implementations of feature extractors for image steganalysis were completely different from the algorithms provided by PySteg, and so a potential future area of research would be to re-write the Python system in MATLAB and examine its performance when used with those different feature sets.

Furthermore, the system could be extended to look at other media file formats such as the image file types that were not tested during this project (e.g. TIFF or ICO files), video file types of a format other than raw H264 bitstreams, or even audio files, which were not the focus of this project.

Deeper investigation could also be conducted in to the ways in which machine learning could be utilised as part of steganalysis; currently, the prototype system only implements two forms of supervised machine learning – this could be extended to examine other supervised machine learning algorithms, such as random forests or convolution neural

networks, or even to examine the use of alternative types of machine learning algorithms, such as unsupervised or semi-supervised machine learning. This would be an interesting investigation into which type of machine learning algorithm would be the most appropriate method to use with steganalysis.

New research and discoveries are constantly being made in the field of steganalysis – even over the course of this project, new research has been conducted and released that is highly relevant to the field, such as Chutani and Goyal's (2019) research into current techniques in forensic steganalysis, Sun *et al*.'s (2019) research into the use of deep neural networks for detecting steganography, and also the StegoAppDB database of clean and stego images from Newman *et al*. (2019). In the future, new techniques of steganalysis could be analysed using similar methods to those described in this project.

Appendices

A. Statistics from Testing Classifier Accuracy

Table 7: Accuracy of classifier with training data split as 85% training and 15% testing

85% Training & 15% Testing								
File Type	SVM classifier				LR classifier			
	Attempt 1	Attempt 2	Attempt 3	Average	Attempt 1	Attempt 2	Attempt 3	Average
Images	0.63333	0.61111	0.58889	0.61111	0.61111	0.60000	0.55556	0.58889
Videos	0.54689	0.53965	0.55748	0.54801	0.56807	0.57143	0.56368	0.56773

Table 8: Accuracy of classifier with training data split as 80% training and 20% testing

80% Training & 20% Testing								
File Type	SVM classifier				LR classifier			
	Attempt 1	Attempt 2	Attempt 3	Average	Attempt 1	Attempt 2	Attempt 3	Average
Images	0.62500	0.65000	0.59167	0.62222	0.60833	0.60833	0.56667	0.59444
Videos	0.53846	0.55861	0.55028	0.54912	0.56849	0.57159	0.57838	0.57282

Table 9: Accuracy of classifier with training data split as 75% training and 25% testing

75% Training & 25% Testing								
File Type	SVM classifier				LR classifier			
	Attempt 1	Attempt 2	Attempt 3	Average	Attempt 1	Attempt 2	Attempt 3	Average
Images	0.64000	0.60667	0.57333	0.60667	0.52000	0.51333	0.60667	0.54667
Videos	0.54727	0.54665	0.56091	0.55161	0.56711	0.56804	0.57021	0.56845

Table 10: Average accuracy of trained classifiers by percentage of data used for training

Classifier	Data split (% training to % testing)			Average across all splits
	75 to 25	80 to 20	85 to 15	
SVM images	0.606667	0.622222	0.611111	0.613333
SVM videos	0.551612	0.549118	0.548007	0.549579
LR images	0.546667	0.594444	0.588889	0.576667
LR videos	0.568454	0.572822	0.567726	0.569667
All classifiers	0.56835	0.584652	0.578933	0.577312

References

Al-Jarrah, M. (2018). 'RGB-BMP steganalysis dataset', Mendeley Data, v1. doi: 10.17632/sp4g8h7v8k.1

Avcibas, I. *et al.* (2005) 'Image steganalysis with binary similarity measures', *EURASIP Journal on Advances in Signal Processing,* 2005(17), pp. 2749-2757. doi: 10.1155/ASP.2005.2749

Avcibas, I., Memon, N. and Sankur, B. (2003) 'Steganalysis using image quality metrics', *IEEE Transactions on Image Processing,* 12(2), pp. 221-229. doi: 10.1109/TIP.2002.807363

Babu, J., Rangu, S. and Manogna, P. (2017) 'A survey on different feature extraction and classification techniques used in image steganalysis', *Journal of Information Security,* 8(3), pp. 186-202. doi: 10.4236/jis.2017.83013

Blundenson, S. J. and Fisher, R. B. (2010) 'The BEHAVE video dataset: ground truthed video for multi-person behavior classification', *Annals of the BMVA,* 2010(4), pp. 1-12.

Canonical Ltd. (2019) *Ubuntu.* Available at: https://www.ubuntu.com/ (Accessed 25 February 2019).

Chutani, S. and Goyal, A. (2019) 'A review of forensic approaches to digital image steganalysis', *Multimedia Tools and Applications,* pp. 1-36. doi: 10.1007/s11042-019-7217-0

Coates, A., Lee, H. and Ng, A. Y. (2011) 'An analysis of single layer networks in unsupervised feature learning', *AISTATS*.

Deepa, G. M. *et al*. (2012) 'Face recognition using spectrum-based feature extraction', *Applied Soft Computing,* 12(9), pp. 2913-2923. doi: 10.1016/j.asoc.2012.04.015

Deng, J. *et al.* (2009) 'ImageNet: A large-scale hierarchical image database', *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA, 20-25 June, pp. 248-255. doi: 10.1109/CVPR.2009.5206848

Deng, Y. *et al*. (2013) 'Digital video steganalysis based on motion vector statistical characteristics', *Optik,* 124(14), pp. 1705-1710. doi: 10.1016/j.ijleo.2012.05.014

Donges, N. (2018) 'The random forest algorithm', *Towards Data* Science, 22 February. Available at: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd (Accessed 1 April 2019).

EmbeddedSw (2018) *OpenPuff.* Available at: http://www.embeddedsw.net/openpuff.html (Accessed 20 March 2019).

ffmpeg (2019) *ffmpeg.* Available at: https://ffmpeg.org/about.html (Accessed 20 March 2019).

Fridrich, J., Holub, V. and Denemark, T. (2018) 'Feature extractors for steganalysis*'.* Available at: http://dde.binghamton.edu/download/feature_extractors/ (Accessed 28 March 2019).

Goljan, M., Fridrich, J. and Holotyak, T. (2006) 'New blind steganalysis and its implications', *Electronic Imaging 2006*, San Jose, CA, USA. doi: 10.1117/12.643254

Gomis, F. K. *et al.* (2018) 'Multiple linear regression for universal steganalysis of images*',* *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, Fez, Morocco, 2-4 April 2018. doi: 10.1109/ISACV.2018.8354060

Hashemipour, S. M. and Rahmati, M. (2012) 'A DCT statistics-based universal image steganalysis', *2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing,* Piraeus-Athens, Greece, 18-20 July, pp. 94-97. doi: 10.1109/IIH-MSP.2012.28

Hetzl, S. (2003) *steghide*. Available at: http://steghide.sourceforge.net/ (Accessed 19 April 2019).

JetBrains (2019) *PyCharm.* Available at: https://www.jetbrains.com/pycharm/ (Accessed 14 January 2019).

Jordan, M. I. and Mitchell, T. M. (2015) 'Machine learning: trends, perspectives, and prospects', *Science,* 349(6245), pp. 255-260. doi: 10.1126/science.aaa8415

Judge, J. C. (2001) 'Steganography: past, present, future*', SANS Institute*. Available at: https://www.sans.org/reading-room/whitepapers/stenganography/steganography-past-present-future-552 (Accessed: 9 October 2018).

Kancherla, K. and Mukkamala, S. (2012) 'Block level video steganalysis scheme'*, 2012 11th International Conference on Machine Learning and Applications*, Boca Raton, FL, USA, 12-15 December, pp. 651-654. doi: 10.1109/ICMLA.2012.121

Kodovsky, J., Fridrich, J. and Holub, V. (2012) 'Ensemble classifiers for steganalysis of digital media', *IEEE Transactions on Information Forensics and Security,* 7(2), pp. 432-444. doi: 10.1109/TIFS.2011.2175919

Li, B. *et al*. (2011) 'A survey on image steganography and steganalysis', *Journal of Information Hiding and Multimedia Signal Processing,* 2(2), pp. 142-172.

Li, H. (2017) 'Which machine learning algorithm should I use?', *The SAS Data Science Blog*, 12 April. Available at: https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/ (Accessed 1 April 2019).

Liu, S. *et al.* (2009) 'Universal steganalysis based on statistical models using reorganization of block-based DCT coefficients*', 2009 Fifth International Conference on Information Assurance and Security,* Xi'An China, 18-20 August, pp. 778-781. doi: 10.1109/IAS.2009.185

Lubenko, I. and Ker, A. D. (2011) 'Steganalysis using logistic regression*', Proc. SPIE 7880, Media Watermarking, Security, and Forensics III*, 78800K, 10 February. doi: 10.1117/12.872245

Luo, X. Y. *et al.* (2010) 'Image universal steganalysis based on best wavelet packet decomposition', *Science China Information Sciences,* 53(3), pp. 634-647. doi: 10.1007/s11432-010-0044-6

Lyu, S. and Farid, H. (2003) 'Detecting hidden messages using higher-order statistics and support vector machines' In: Petitcolas, F. A. P. *et al.* (eds). *Information Hiding.* Berlin, Heidelberg: Springer, pp. 340-354. doi: 10.1007/3-540-36415-3_22

Meghanathan, N. and Nayak, L. (2010) 'Steganalysis algorithms for detecting the hidden information in image, audio and video cover media', *International Journal of Network Security & Its Applications (IJNSA),* 2(1), pp. 43-55.

Newman, J. *et al.* (2019) 'StegoAppDB: a steganography apps forensics image database', *IS&T Int'l. Symp. on Electronic Imaging, Media Watermarking, Security, and Forensics 2019,* Burlingame, CA, 19 April.

Ofcom (2018) *A decade of digital dependency*. Available at: https://www.ofcom.org.uk/about-ofcom/latest/media/media-releases/2018/decade-of-digital-dependency (Accessed 3 May 2019)

Oh, S. *et al.* (2011) 'A large-scale benchmark dataset for event recognition in surveillance video', *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Colorado Springs, CO, USA, 20-25 June. doi: 10.1109/CVPR.2011.5995586

Oliphant, T. E. *et al.* (2019) *NumPy*. [Python Module]. Available at: https://pypi.org/project/numpy/ (Accessed 2 February 2019).

Paliienko, M. (2018) *Fleep.* [Python Module]. Available at: https://pypi.org/project/fleep/ (Accessed 2 February 2019).

Pedregosa, F. *et al*. (2011) 'Scikit-learn: machine learning in Python', *JMLR,* 12, pp. 2825-2830.

Pevny, T., Bas, P. and Fridrich, J. (2010) 'Steganalysis by subtractive pixel adjacency matrix', *IEEE Transactions on Information Forensics and Security,* 5(1), pp. 215-224. doi: 10.1109/TIFS.2010.2045842

Raschka, S. (2016) 'When does deep learning work better than SVMs or random forests?', KDnuggets. Available at: https://www.kdnuggets.com/2016/04/deep-learning-vs-svm-random-forest.html AAccessed 1 April 2019).

Sadat, E., Faez, K. and Saffari Pour, M. (2018) 'Entropy-based video steganalysis of motion vectors', *Entropy,* 20(4), p. 244. doi: 10.3390/e20040244

Schaathun, H. G. (2012a). *Machine learning in image steganalysis.* Chichester, West Sussex, England: IEEE Press : Wiley.

Schaathun, H. G. (2012b) *PySteg.* [Python Package]. Available at: http://www.ifs.schaathun.net/pysteg/ (Accessed 28 Januray 2019).

scikit-learn developers (2018) *Preprocessing data.* Available at: https://scikit-learn.org/stable/modules/preprocessing.html (Accessed 20 April 2019).

Shi, Y. Q. *et al.* (2005) 'Image steganalysis based on moments of characteristic functions using wavelet decomposition, prediction-error image, and neural network*', 2005 IEEE International Conference on Multimedia and Expo,* Amsterdam, The Netherlands, 6 July, pp. 269-272. doi: 10.1109/ICME.2005.1521412

Sun, Y. *et al.* (2019) 'Deep neural networks for efficient steganographic payload location', *Journal of Real-Time Image Processing,* pp. 1-13. doi: 10.1007/s11554-019-00849-y

Team Anaconda (2019) *End of Life (EOL) for Python 2.7 is coming. Are you ready?* Available at: https://www.anaconda.com/end-of-life-eol-for-python-2-7-is-coming-are-you-ready/ (Accessed 20 April 2019).

Wang, K., Zhao, H. and Wang, H. (2014) 'Video steganalysis against motion vector-based steganography by adding or subtracting one motion vector value', *IEEE Transactions on Information Forensics and Security,* 9(5), pp. 741-751. doi: 10.1109/TIFS.2014.2308633

Wang, P., Cao, Y. and Zhao, X. (2017) 'Segmentation based video steganalysis to detect motion vector modification', *Security and Communication Networks,* 2017, pp. 1-12. doi: 10.1155/2017/8051389

Warkentin, M., Bekkering, E. and Schmidt, M. (2008) 'Steganography: forensic, security, and legal issues', *Journal of Digital Forensics, Security and Law,* 3(2), pp. 17-34. doi: 10.15394/jdfsl.2008.1039

WineHQ (2019) *WineHQ.* Available at: https://www.winehq.org/ (Accessed 15 April 2019).

Xia, Z. *et al.* (2016) 'Steganalysis of LSB matching using differences between nonadjacent pixels', *Multimedia Tools and Applications,* 75(4), pp. 1947-1963. doi: 10.1007/s11042-014-2381-8

Xie, X., Lai, J. and Zheng, W. (2010) 'Extraction of illumination invariant facial features from a single image using nonsubsampled contourlet transform', *Pattern Recognition,* 43(12), pp. 4177-4189. doi: 10.1016/j.patcog.2010.06.019

Xu, G., Wu, H. and Shi, Y. (2016) 'Structural design of convolutional neural networks for steganalysis', *IEEE Signal Processing Letters,* 23(5), pp. 708-712. doi: 10.1109/LSP.2016.2548421

Xu, X., Dong, J. and Tan, T. (2012) 'Universal spatial feature set for video steganalysis', *2012 19th IEEE International Conference on Image Processing,* Orlando, FL, USA, 30 September - 3 October, pp. 245-248. doi: 10.1109/ICIP.2012.6466841

Xu, X. *et al.* (2013) 'Video steganalysis based on the constraints of motion vectors', 2013 *IEEE International Conference on Image Processing,* Melbourne, Australia, 15-18 September, pp. 4422-4426. doi: 10.1109/ICIP.2013.6738911

Yadollahpour, A. and Naimi, H. M. (2009) 'Attack on LSB steganography in color and grayscale images using autocorrelation coefficients', *European Journal of Scientific Research,* 31(2), pp. 172-183.

Yang, X. *et al*. (2009) 'Universal image steganalysis based on wavelet packet decomposition and empirical transition matrix in wavelet domain*', 2009 International Forum on Computer Science-Technology and Applications,* Chongqing, China, 25-27 December, pp. 179-182. doi: 10.1109/IFCSTA.2009.165

Yu, L. (2018) 'Image steganalysis based on wavelet packet decomposition*', Proceedings of the 2018 3rd International Workshop on Materials Engineering and Computer Sciences (IWMECS 2018),* Jinan, China. doi: 10.2991/iwmecs-18.2018.78

Zhang, H. (2019) *Feature-Extractors-for-Video-Steganalysis.* [GitHub Repository] Available at: https://github.com/zhanghong863/Feature-Extractors-for-Video-Steganalysis (Accessed 1 April 2019).

Zhang, H., Cao, Y. and Zhao, X. (2017) 'A steganalytic approach to detect motion vector modification using near-perfect estimation for local optimality' *IEEE Transactions on Information Forensics and Security,* 12(2), pp. 465-478. doi: 10.1109/TIFS.2016.2623587

Zhang, R. *et al.* (2018) 'Efficient feature learning and multi-size image steganalysis based on CNN'. Available at: https://arxiv.org/abs/1807.11428 (Accessed: 7 February 2019).

Zong, H., Liu, F. and Luo, X. (2012) 'Blind image steganalysis based on wavelet coefficient correlation', *Digital Investigation,* 9(1), pp. 58-68. doi: 10.1016/j.diin.2012.02.003