Spam Companion - working title

Ivan Trendafilov

4th Year Project Report

Computer Science

School of Informatics

University of Edinburgh

2012

Abstract

A really good abstract.

Acknowledgements

I would like to thank my supervisor, Charles Sutton, for his advice and support throughout this project. His suggestions and feedback were invaluable to building the system to its current state.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ivan Trendafilov*)

HI MUM.

Table of Contents

1 Introduction **1**

1.1 Overview of advance fee fraud 1

1.2 Project rationale 2

2 Background **4**

2.1 Inside a 419 scam 4

2.2 Related work in advance fee fraud prevention 6

2.3 Related work in conversational agents 8

3 Architecture overview **10**

3.1 Principles 10

3.2 Overview 11

3.3 Honeypot environment 13

4 Collector **14**

4.1 Motivation 14

4.2 Implementation 15

4.3 Producer modules 15

4.3.1 419eater.com crawler 15

4.3.2 Forwarded messages collector 16

4.3.3 Replies collector 16

5 Information extraction **17**

5.1 Overview 17

5.2 Implementation 17

5.2.1 Dealing with dirty data 18

5.2.2 Named-entity recognition and emails 19

6 Classification **21**
6.1 Overview . . . 21
6.2 The 419 corpus . . . 21
6.3 Methodology . . . 22
6.3.1 Maximum Entropy model . . . 22
6.3.2 Feature selection . . . 23
6.3.3 Training the classifiers . . . 23

7 Response generation **24**
7.1 Bucketing algorithm . . . 24
7.2 Strategies . . . 25
7.2.1 Cooperative strategies . . . 25
7.2.2 Non-cooperative strategies . . . 26
7.3 Composing a response . . . 27
7.3.1 Understanding the context . . . 27
7.3.2 Text snippets . . . 28
7.3.3 Finite state machines . . . 29
7.3.4 An example message . . . 30
7.3.5 Sending the reply . . . 30

8 Evaluation **31**
8.1 Terminology . . . 31
8.2 Intermediate results . . . 32
8.2.1 Scam variation classifier . . . 32
8.2.2 PQ classifier . . . 33
8.2.3 Named-entity recognition classifier . . . 34
8.3 Conversation results . . . 34
8.3.1 Methodology . . . 34
8.3.2 Bouce rate . . . 35

9 Conclusions **37**

Chapter 1

Introduction

The goal of this project is to explore the feasibility of building an autonomous conversational agent to sustain a dialogue with a person perpetrating advance fee fraud over email. We begin our discussion by introducing the key concepts about advance fee fraud in Section 1.1. Once we have set the context, we outline the rationale of our project in Section 1.2.

1.1 Overview of advance fee fraud

Advance fee fraud (AFF) emails are identical or nearly identical messages sent in bulk to a large number of recipients. They are considered to be a subset of email spam, however, unlike traditional spam messages, their purpose is not to advertise products, but to convince the recipient to advance sums of money in the hope of realizing a significantly larger gain later. Because of this, AFF emails can be defined at the intersection of spam and social engineering.

AFF emails have another important distinction from generic spam messages. Typically, replying to a generic spam message serves no purpose to spammer or the target. The headers of these messages are most often spoofed and the *From* and *Reply-To* addresses point to non-existent email boxes. In contrast, AFF messages explicitly require the target to respond and express interest in the proposed opportunity. Once a scammer receives a reply, he continues the exchange for as long as it is necessary to convince the target of the veracity of the opportunity. Finally, the scammer introduces a delay or monetary hurdle that prevents the deal from occurring as planned. This hurdle is set up so it can be overcome by paying a fee for some service – e.g., a bribe for a bank official, a processing fee for a transaction, legal fees, etc. If the target advances the

requested fees, the scammer will proceed to introduce additional hurdles, in attempt to extract further fees. At this stage the scammer is even more likely to succeed, as the target is already emotionally invested in the deal [REF]. The exact details of these exchanges vary considerably, however scammers often employ deception and a set of other manipulation techniques [REF].

The modern advance fee scam originated in Nigeria in the 1980s, as the country's oil-based economy declined and left millions of people unemployed [REF]. As a result of this, several enterprising individuals took the idea of the older Spanish Prisoner scam and extended it to various electronic means – such as fax, telex and, eventually, email. The boom of email and advances in software, such as web crawlers and email address harvesters, helped significantly lower the cost of sending scam letters. Due to the low costs involved in running the scam and its proportionally high payout, the idea quickly spread to other regions of Africa, as well as parts of Europe and the United States. Despite its global reach, advance fee fraud is often referred to as "419 fraud", after article 419 of the Nigerian Criminal Code (Chapter 38: Obtaining Property by false pretences; Cheating) [REF].

419 scams have a significant impact on the wider economy. Chatham House Research estimates that Nigerian-style advance fee fraud costs the UK economy £150 million annually and the average victim loses £31,000 [REF]. Between 2000 and 2003, the National Criminal Intelligence Service has found over 78,000 different advance-fee-style letters, faxes and emails in London alone [REF]. It is possible that these figures underestimate the total financial loss incurred by businesses and individuals, as victims are likely to feel ashamed or may be convinced that they have also done something illegal and are, thus, less likely to report their losses. Whilst this is anecdotal, it would help explain the relatively large reported loss per victim, as scammers typically aim to extract smaller amounts from a larger pool of victims, rather than vice versa.

1.2 Project rationale

As we discussed earlier, a major difference between generic and advance fee spam is the active involvement of the scammer in the former. An interesting approach to dealing with it would be to build a conversational agent to emulate a victim with the goal of maintaining an email exchange for as long as possible.

The core assumption of this approach lies in the cost of human effort. It is time consuming to read and respond to emails. By introducing automatically generated

email threads into a scammer's workflow, we are increasing the time he has to spend reading, replying, and monitoring the state of conversations that cannot possibly lead to a positive outcome. This helps dilute the pool of potential victims and lowers the overall utility of his actions.

Another advantage of this approach is scalability. Since the proposed agent is fully autonomous, it can sustain conversations with a large number of scammers simultaneously (bounded by available system resources). As we will see, the proof of concept implementation in [CHAPTER] maintained conversations with 45 scammers over the course of a 10-day period. Further scalability can be achieved by deploying multiple instances of the agent.

Chapter 2

Background

This chapter sets the context for our discussion. We start by exploring the dynamics of advance free fraud scams. Then we outline some of the most prominent work in the area of AFF prevention. At the end of this chapter, we explore notable conversational agents and their design.

2.1 Inside a 419 scam

To create an optimal system, it is important to understand how 419 scammers operate. In order to do this, we set up a simple experiment. We randomly selected 5 email messages that had been recently posted on 419eater.com, a popular anti-scam web site, and replied to all them from an anonymous email account, with the intention to keep the conversation going for as long as possible and try out different conversational tactics. Over the course of a week, we had received a total of 37 emails. The main observations from these conversations are summarized below:

1. Requests for detailed personal information, such as full name, address, telephone number, occupation, age, gender, passport photograph are very common.
2. Scammers commonly use templates written ahead of time over the the first two email exchanges.
3. It is common for scammers to share their targets' email addresses with other scammers – e.g., via a mailing list.
4. Often, scammers send their targets counterfeit documents – e.g., certificates, passports, etc.
5. Scams are often operated by more than one person ("actor") or, possibly, the same person assuming multiple identities.
6. Scammers sometimes insist on phone numbers and attempt to call the target.

7. Sometimes scammers do not respond to their target, despite initiating the conversation.

Obs. (1) shows that our natural language model needs to take into account these requests for information and deal with them effectively. There are a few possible ways to do that.

One approach would be to attempt to avoid or deflect these questions. The benefit of this approach is that it spares the agent from the requirement to store identity information about itself and does not actively deceive the scammer. This simplifies our design and implementation. However, a substantial drawback is that scammers use these questions as a signal of established trust. From the point of view of the scammer, if the target fails to comply with these requests, this is a strong indication to simply move on to the next viable target.

A second approach is to let the agent invent and maintain an identity throughout the entire conversation thread. This is guaranteed to pass the initial screening. These strategies are discussed in more detail in [CHAPTER]

Obs. (2) shows the scale of AFF operations. It is clearly inefficient to compose each response individually, so scammers commonly prefer reusing templates as much as possible, whilst still making a few modifications – e.g., adding the name of the target, changing aliases and altering country-specific details. Therefore, we will create strategies that aim to force the scammer to manually compose as many responses as possible. This is a very challenging task as it requires the agent to present the scammer with unexpected scenarios. Furthermore, it is not clear whether there is a metric which we can use to measure our success [Chapter].

Obs. (3) suggests a degree of organization in AFF operations. It is reasonable to suggest that groups of scammers work in coordination. The details of a victim, which is already proven to be susceptible, are valuable and appear to be traded among groups of scammers. In our experiment above, we received 3 additional messages which did not belong to any of the conversation threads we initiated. Whilst it is unfortunate that vulnerable people are repeatedly targeted, over an extended period this practice could supply our agent with additional scam instances.

Obs. (4) refers to situations in which the scammer sends out pictures of counterfeit documents to his target. The presence of an attachment in a document is a signal for the state of the conversation. The extension of the attached file makes it easy to determine whether the document is a picture or not, however objection recognition is required to determine if the picture contains a document, a person, or neither. This

task requires the addition of a image processing component to our system. Because this functionality is not central to our project, we cho

[HAHHA]

This task would require the addition of image processing capabilities to our system. Because this functionality is not central to the project, we have chosen not to implement it and instead rely on other heuristics, as described in [CHAPTER]

Obs. (5) means that conversation threads can involve multiple people (or multiple aliases of the same person). In order to continue the conversation, the system must be able to cluster the involved parties together into the correct thread, as well as handle the transition from addressing person *A* to addressing person *B*. Failure to do so means we have to abandon any state information for that thread. We will discuss the solution to this problem in [CHAPTER].

Obs. (6) describes variation of email 419 scams that transitions to a different medium – the telephone. There is no useful way for us to receive or process phone calls, therefore we would prefer to either avoid or completely ignore these requests.

Obs. (7) has impact on how we can interpret our results during evaluation. Some email accounts get disabled, deleted, or become abandoned before the scammer has sent replies to all potential targets. Therefore, we will need to determine a baseline performance. We will discuss that in detail in [CHAPTER].

2.2 Related work in advance fee fraud prevention

Generic email spam has been the subject of a lot of research, including analyses of its economics and legality. A lot of effort has been spent on developing anti-spam techniques. Some of the most widely used techniques are client-side challenge/response systems, authentication and reputation mechanisms, DNS blacklists, pattern detection, rule-based filtering and statistical filtering. [REF] [REF] [REF]

Popular state of the art email services use a combination of these techniques to protect their users. An illustrative example is Yahoo! Mail. The service uses a rule-based filter on the sender side. Should a message trigger the filter, the user is prompted to respond to a CAPTCHA correctly before he can proceed with sending a message. On the receiver side, Yahoo! Mail combines its rule-based filter with a Bayesian filter, in addition to DNS blacklisting and domain authentication.

Yahoo!, as well as the other top email providers Hotmail and Gmail, does not publish accuracy statistics of their spam filters. Nevertheless, the independent competitive

insights firm Cascade Insights provides the most recent study to attempt to measure the accuracy of these providers. According to their methodology, they set up email accounts with these three email services, as well as an unfiltered account with a web hosting company. Next, they aggressively seeded these accounts by posting their addresses on blogs, public walls on social networks, various other target web sites, etc. Finally, they subscribed these accounts to a set of solicited, opt-in activities. At the end of the 5-week study period, they found that 58.33

Common spam filtering techniques do not differentiate between generic spam and advance fee fraud spam. Nevertheless, there are methods designed specifically to combat AFF scams and are most often employed by online communities. Some popular communities have colorful names such as: 419eater.com, The Scambaiter, 419 Hell, the Artists Against 419. Their members describe their own actions as scam baiting and often consider the act a form of Internet vigilantism.

The most common approach used by scam baiting communities is similar to the one discussed in this project emulate a potential victim and attempt to keep the scammer occupied for as long as possible. Additionally, many scambaiters will attempt to find out personal information and give it to the authorities. Frequently, scambaiters will also contact hosting or Internet service providers and urge them to suspend accounts and fraudulent web sites. These approaches are somewhat successful, but limited in scale. Each scambaiter has to spend proportional time writing and responding emails to that of the scammer. As such, its impact is limited by the time and effort scambaiters are able to donate to the activity. Interestingly, scam baiting appears to also provide entertainment value, which would explain its steady popularity, in spite of its obvious limitations.

A more radical approach to fighting AFF scams used to be popular with scam baiting communities until recently [REF]. This approach relied on the scambaiter's ability to obtain a scammer's IP address. Then, through assistance of the community, the scambaiter would proceed by initiating a Distributed Denial of Service (DDoS) attack on the subnetwork. Many variations of this approach are possible, including targeting phising web sites and open relay SMTP servers. Whilst this approach is effective in temporary disrupting a scammer's operation, it suffers from a few major drawbacks. First, scammers often host their operations on compromised servers. In these cases, DDoS attacks cause additional problems to network administrators uninvolved in AFF at all [REF]. Second, the practice is considered illegal in many countries, including the UK (as of 2006) [REF].

2.3 Related work in conversational agents

The origin of conversational agents can be tracked back to the 1960s. The most known early example of simple natural language processing is the computer program ELIZA. The program can be described as a simple rule-based engine which attempts to mimic a Rogerian psychotherapist. It is implemented via a simple parser and substitution algorithm, which draws from a script of canned phrases. ELIZA has no knowledge of state, context, reasoning, relationships, or the world [REF].

A more recent example is the computer program A.L.I.C.E., which won the 2004 Loebner Prize for most human-like chatterbot. Nearly half a decade later, A.L.I.C.E. is still based on pattern matching and word substitution into canned responses. The main improvements over ELIZA lie in its engineering. A.L.I.C.E. supports its own markup language AIML, which is used to input knowledge into the system. In its essence, AIML is similar to regular expressions matching, in that it provides a pattern and a set of possible responses. A simple clustering algorithm allows similar entities to be grouped together e.g. mother and family. Finally, a spelling and grammar correction algorithm attempts to transform user input before it is passed to the system's pattern matching engine. In its essence, A.L.I.C.E. can be thought of as a well-engineered extension of the principles set up by ELIZA.

Another well-known chatterbot is Cleverbot. Its design and operation are similar to the agents we have discussed so far, however it is novel in the way it obtains its script. The agent preserves responses it has received from conversations with people and is said to have a knowledge bank of 50 million contextual responses [REF]. In addition to that, the Cleverbot team performs offline filtering to determine which learned responses are usable.

Finally, perhaps the most widely used service that includes a conversational natural language interface is Apple's Siri. Whilst Siri's main design goal is to serve as an intelligent personal assistant, it also incorporates a set of interesting techniques in its natural language generation and understanding components. The agent is capable of evidential and probabilistic reasoning, and uses a combination of machine learning and rules to determine the correct state of the conversation. Each conversation is modelled as a finite state machine e.g., instructing the agent to create a new appointment transitions the agent into the creating appointment state, which now allows transitions to the set date, set appointment name, set appointment location and cancel states. The agent also delegates different types of requests to various ontologies and backend service

providers and is capable of reasoning about basic relations between entities.

In summary, conversational agents use a wide range of techniques for natural language processing. State of the art systems most commonly use a combination of machine learning, text pattern matching, probabilistic reasoning and finite state models. Therefore, we believe it is possible to develop convincing conversational agents for domain-specific problems, such as advice fee fraud, by using a combination of the techniques described above.

Chapter 3

Architecture overview

This chapter can be regarded as a self-contained, high level overview of the implementation details of our system. We start by presenting the main principles we follow in our design. Next, we provide an overview of the system's architecture and the implemented functionality. Lastly, we outline the specification and setup of our honeypot environment.

3.1 Principles

In the implementation of our system, we follow three high-level principles:

1. We aim to keep each conversation thread going for as long as it is possible. To achieve this, we incorporate various machine learning and natural language processing techniques into the design of our system, and attempt to mimic human performance in the composition of outgoing messages as much as possible.

2. We aim for full system automation. No user interaction should be necessary for the system to function. In order to achieve this, we incorporate a collectors subsystem, classifier and probabilistic finite state models into our design, in order to automatically collect, categorize, and compose messages.

3. We aim to build a safe and secure system. In order to do this, we run our system from a secure remote server and use anonymous email accounts for sending and receiving emails. In addition, we ensure our natural language modules provide only fictitious details in conversations with scammers.

3.2 Overview

The main functionality of the system is split into five major components collectors, information extraction, classification, identity generation and response generation. These components operate together to obtain AFF emails, perform information extraction and classification tasks, generate and maintain the agent's identity and compose responses. This is illustrated in [FIGURE].

Collectors

The collector subsystem defines an interface for entering new messages into the system. This interface is thread-safe and allows for multiple collector modules to run simultaneously. It is also extensible, so creation of new collector modules is easy. For the purposes of this project, we have implemented three message collectors a crawler and scraper for 419eater.com, an identity email box collector, and forwarded messages collector. The crawler and scraper is designed to download surplus 419 scam letters from the aforementioned web site. The identity email box collector simply iterates over all email boxes maintained by our agent and downloads any received messages. Finally, the forwarded message collector monitors a mailbox, where people can forward their spam messages and the collector will automatically filter out any personally identifiable information before passing them into the system. These three collectors ensure a constant flow of new messages into the system and also collect responses to existing conversation threads. The collectors subsystem is explained in more detail in [CHAPTER].

Information extraction

The information extraction component provides a set of algorithms for extracting useful information from incoming messages. As this is the first step of processing after entry into the system, these algorithms are also equipped to deal with potentially dirty data e.g. malformed email headers, message body, etc. The main tasks performed by this component are named-entity recognition, email extraction, removing HTML, removing quoted messages, header parsing, relationships disambiguation. Some of these tasks are challenging e.g., given a set of emails and named entities, compute which email address belongs to whom. Others appear trivial, but are also difficult e.g., it is hard to separate a quoted message from the body of an email message, as there is

no universally observed standard on messages should be quoted on reply. We go into further detail on these problems and the proposed solutions in [CHAPTER].

Classification

The classification component provides a wrapper interface to an instance of the Stanford Classifier [REF]. The Stanford package is a Java implementation of a maximum entropy classifier. Maximum entropy models maximize the conditional likelihood of classes and their weights take feature dependence into account, which make them particularly suitable for text classification tasks. For the purposes of this project, we have trained two classifiers. Our first classifier is trained on over ? AFF messages from our 419 corpus and has a F1 score of 81.177

Identity generation

The identity generation component is tasked with providing the agent with a personality. In the context of AFF scams, this means a name, age, occupation, email address, postal address, country, marital status. This information is fictitious and may be used by the response generation component, depending on the state of the conversation and the current strategy. An identity is attached to each conversation thread in order to ensure consistency. The component provides the facilities required to obtain and manage identities. It is discussed in detail in [SECTION].

Response generation

The response generation component takes in as inputs the result of the information extraction tasks, an identity, the message, and its class. It then composes a response and sends it to the scammer. This is accomplished in three steps. The first step is to determine whether the incoming message belongs to a thread. We define a thread as a logical unit of all messages related to a single instance of a scam. Because a thread can contain more than two participants, we develop a bucketing algorithm to group messages together. Second, once a thread is selected (or a new thread is created), based on the recorded state of the conversation, message class, PQ classifier, and class-specific rules, a multi-layer finite state machine generates a response. The response is generated in multiple stages. First a top-level FSM builds a template, which is then filled out by lower-level probabilistic finite state machines (PFSMs) with snippets of

text or another set of PFSMs. In the final step, the component opens a connection to a SMTP server and pushes out the message. We provide lower level detail on response generation in [CHAPTER].

How these components interact together is illustrated in [FIGURE]. Graph of the circle Graph of how the feed looks

3.3 Honeypot environment

To satisfy the personal safety requirements of this project, we require a customized setup that does not expose the author's IP address, identity or a link to the University of Edinburgh.

For these reasons, we run our prototype system on a leased virtual private server. The virtual machine features 4 GB RAM, Intel Xeon 2.80 GHz CPU, CentOS 6 (Linux kernel 3.0.22) and is secured through a firewall that blocks all incoming traffic, except secure shell access. The physical is located in a data center in London and is managed by clustered.net. For sending and receiving emails, we use free anonymous email accounts set up on Yahoo! Mail and GMX. Connections to the POP3 and SMTP servers are secured with SSL. The minimum deployment environment requirements for the prototype system are: Python ¿= 2.7.2, Oracle Java ¿= 1.6.0, Beautiful Soup, Stanford Classifier ¿= 2.1.3, Stanford NER ¿= 1.2.2.

Chapter 4

Collector

In this chapter we outline our system's mechanism for collecting new messages. We first set up the context and motivation of the problem. Then, we explore the implementation of our solution. Lastly, we describe three specific modules which implement the collection interface.

4.1 Motivation

The collector component provides the entry point for messages into the system. There are two parts to this problem. First of all, the component provides the required functionality to retrieve replies from all current conversation threads. Second, it also provides a mechanism to obtain new instances of the AFF scam and pass it to the conversational agent for further procesisng. Both mechanism ensure full automation, so no user interaction is required for system operation.

The solution to the first problem is trivial – it is briefly described in [SECTION]. In contrast, the problem of obtaining new AFF email messages is more unusual. One common approach is to plant a set of target email addresses in online guestbooks and forums of web sites with high PageRank score. These pages are frequently crawled by email harvester bots. The extracted email addresses are then added to spammers' mailing lists and eventually start receiving spam. Whilst this approach ensures a constant flow of new spam messages, it suffers from two major drawbacks. First of all, it requires manual entry, which is against our principle of full automation. Second, it attracts all types of spam, not advance fee fraud scams specifically. Instead, it is more effective for us to build our own crawler and scraper for AFF scams, as discussed in [SECTION].

4.2 Implementation

The implementation of the collector component can be described as a variation of the classical producer-consumer pattern, for N producers and a single consumer, using a FIFO queue. The consumer module defines a uniform, thread-safe interface for accepting messages. Similarly, a set of producer modules define methods for obtaining new messages from various sources. The set of producers run concurrently and implement the consumer interface.

Communication between the consumer and the producers is accomplished through the file system. When a producer starts to fetch a message, it creates a unique temporary buffer on the file system. Once the message is fully downloaded, its contents are flushed to the buffer. Finally, the file is renamed with an extension, which signals it is ready to be consumed. This is safe, because the rename operation is atomic on POSIX compliant operating systems. Concurrently, the consumer spinwaits for new incoming files. If multiple new files are available, it processes them in FIFO order. This approach allows the implementation to be extended easily with new producer modules, without having to change the consumer or any of the existing producers.

[FIGURE of collectors]

4.3 Producer modules

For our prototype system, we have implemented three producer modules. These fall into two categories: collectors of new scam instances and collectors of replies to existing threads.

4.3.1 419eater.com crawler

The 419eater.com crawler is an example of a producer which collects new scam instances from the aforementioned web site. 419eater.com is a popular online scam baiting community. Its members use a message board to exchange various information – scammer names, techniques, etc. Most notably, the message board features a section titled: “Surplus 419 scam letters” which receives between 20–25 new AFF scam posts per day. This section is a good source of new scam instances.

The producer crawls the section to build an index of available posts. It then crawls each post and scrapes the message content by matching tags in the HTML parse tree. This is illustrated in the algorithm in [FIGURE].

$index \leftarrow$ index of all posts
$visited \leftarrow$ index of crawled posts
$messages \leftarrow \emptyset$
for all $post \in index \setminus visited$ **do**
    $tree \leftarrow$ HTML parse tree
    for all $node \in tree$ **do**
        if $\exists text \in node$ **then**
            $messages \leftarrow text \cup messages$
        end if
    end for
    $visited \leftarrow post \cup visited$
end for

4.3.2 Forwarded messages collector

The forwarded messages collector was a commonly requested feature in project meetings and is another example of a producer designed to gather new scam instances. Users can configure their email client to forward the contents of their spam folder to an email account monitored by this provider. The collector will then automatically fetch all messages, strip any personally identifiable information and deliver them to the agent for further processing. Depending on the outcome of the classification task, the agent may initiate a series of new conversation threads.

4.3.3 Replies collector

The replies collector interfaces with the identity generation component to fetch new messages from all email accounts maintained by the agent. This is a very simple proces – the collector obtains the current list of email accounts, iterates over the list and downloads any new messages from the associated POP3 server and account. Notably, the collector also observes a limit to the number of consecutive connections allowed to the same POP3 server. This is currently set conservatively to one connection within a 15-minute period, in order to avoid triggering any potential anti-abuse measures.

Chapter 5

Information extraction

This chapter discusses the implementation of the information extraction component. In [SECTION], we briefly motivate its functionality. In [SECTION], we describe some of the challenges of dealing with non-homogeneous input data. Lastly, [SECTION] outlines our approach to named-entity recognition and the relationship between named entities and email addresses.

5.1 Overview

The goal of the information extraction component is to obtain useful information from incoming messages. The main tasks performed by this component are named-entity recognition, email extraction, HTML removal, header parsing, quoted text removal and relationship disambiguation. Because this component is the first processing step after entry into the system, it is equipped with methods to deal with potentially dirty data [SECTION]. Once relevant information is extracted, the result is passed as input to the response generation component and is used to generate a convicing, human-like response. For example, the outputs of the named-entity recognition (NER) task might be used to compose a personalized greeting in the beginning of the message – e.g., "Hello *John*".

5.2 Implementation

In order to extract the information we are interested in, we apply a series of transformations to each incoming message. These steps are summarized in the algorithm below:

1. remove HTML
2. remove any quoted messages
3. cleanse headers
4. extract *From, Reply-To, Subject, To* headers
5. extract message body
6. perform NER on the message body
7. extract all emails from the message body
8. compute any relationships between emails and named-entities

The implementation of steps1–4 and how we deal with dirty data is discussed in [SECTION]. Similarly, in [SECTION] we outline our approach to named-entity recognition and the computation of the named-entity, email pairs.

5.2.1 Dealing with dirty data

As we explained in [CHAPTER], a web crawler is one of the main sources of new email messages for our system. Whilst this aids automation, it also creates a number of problems. First, because messages are posted by human forum participants, they rarely conform to MIME or any other RFC memoranda. The lack of standartization makes it impossible to rely on a standard, compliant parsers to extract the headers and payload of our messages. Under MIME, the body part of a message is separated from the headers via a blank line [REF]. This is often omitted in human posts. Furthermore, email clients and servers attach custom headers to processed email messages, so there is no standard set of headers we can match against. Finally, some of the messages downloaded by the crawler are not emails at all, so we also need a way to discard them.

We observe that correct headers follow a set pattern: *"Header: content"*. Furthermore, there is at most one header per line and headers are placed on consecutive lines in the message. Using that observation, we propose a best-effort algorithm to address the problem above. We use a vocabulary V of 144 verified headers (compiled ahead of time). Then, for each incoming message M, we use a regular expression to match the contents of the message against the standard set of headers S, such that $S = \{Subject, To, From, Reply\text{-}To\}$. If we fail to match at least three elements of S, we discard the message, as it is not a valid email. If we find a match, we then iterate over all lines in the message to find L_{vmax}, defined to be the last line that contains a header $\in V$. L_{vmax} allows us to split the message into two segment – headers and body. Lastly, we iterate over all lines in the headers segment and match each line against our

header heuristic. If there is a match, we augment V, which gradually improves the performance of the algorithm.

Another problem with parsing incoming email messages is HTML. Popular web-based email services regularly use HTML to format outgoing messages. As these messages are multi-part, some are encoded in both *text/plain* and *text/html*, whilst others only offer *text/html*. For messages available only in HTML format, we construct a HTML parse tree and remove all nodes that do not contain text. Finally, we strip all tags and convert any ampersand character codes to ASCII, where possible – e.g., *&* to *&*.

5.2.2 Named-entity recognition and emails

Named-entity recognition (NER) is an information extraction task which seeks to locate and classify entities in text into a set of predefined categories – e.g., *Persons, Locations, Organizations*. For our prototype system, we are interested only in the *Persons* category, although other categories might have applications for certain scam variations. By performing NER on the message body, we aim to determine the names of all actors in the scam with high accuracy and pass that information to the response generation component. This is a challenging problem, but there are systems available which offer high performance on these tasks.

One example is the Stanford Named Entity Recognizer [REF]. It is a Java implementation of a linear chain Conditional Random Field (CRF) classifier [REF, REF] and comes with large pre-trained models on the CoNLL 2003, MUC-6, MUC-7 and ACE named entity corpora. In this project we are specifically interested in the provided 3 class model, as it is reported to have a F_1 score of 90.36 for the *Persons* class on the CoNNL 2003 dataset. Its features are presented in [TABLE]:

[PLACEHOLDER FOR BIG TABLE]

Whilst the aforementioned entity corpora provide a robust coverage of the English language, we have also measured the performance of the model on AFF-specific emails on a small validation set. The results are available in [CHAPTER].

Another important problem is to determine the relationship between named entities and extracted email addresses. As we explained in [BACKGROUND], a single scam message may involve multiple actors and contain multiple email addresses. It is necessary to pair actors with the correct email addresses when composing a reply. Failure to do so may result in replying to a non-existing address or addressing the scammer

under the wrong name. Formally, given a set of named entities $N = \{n_1, n_2, ..., n_m\}$ and a set of email addresses $E = \{e_1, e_2, ..., e_p\}$, compute all pairs (n, e), such that each pair belongs to a unique actor in the scam. This is a difficult problem, but we can observe that in AFF scams email addresses are often a variation of a person's real name. Alternatively, they are located in proximity to a named entity.

We propose a two-round algorithm. In the first round, we remove all domain names from $\forall e \in E$ and compute string similarity using bigrams $\forall (n, e) \in N, E$. Next, we rank all pairs by similarity score *Sc*, remove the top pairs for each named entity from N, E and place them in a bucket B, provided $Sc > minconst$, where *mincost* is the minimum bigram similarity score required ($minconst = 0.40$ in our implementation). In the second round, we rank all remaining pairs (n, e) by word distance (measured by the number of words between n and e). Finally, for each remaining $n \in N$, we add the (n, e) with the lowest word distance score to B. The bucket now contains all valid (n, e) pairs.

Chapter 6

Classification

6.1 Overview

The classification component serves two purposes. First, it provides a Python wrapper interface to an instance of the Stanford Classifier [REF] – a Java implementation of a Maximum Entropy classifier. Second, it provides two trained models to solve text categorization problems. The first model – the scam variation classifier – helps us place an incoming message in one of 22 recognized classes of advance fee fraud scams. The second model – the PQ classifier – is a binary classifier that recognizes instances of personal questions in the body of a message. Both classifiers are trained on a corpus of AFF messages we have constructed specifically for this project. Their outputs are used to generate relevant responses, as shown in [CHAPTER].

6.2 The 419 corpus

In this section we introduce the 419 corpus – a collection of $32,000$ advance fee fraud email messages in 22 classes. We have constructed this corpus from freely available data on antifraudintl.org over the course of a week using a crawler, a scraper and a set of preprocessing techniques. It takes up 134 MB of disk space uncompressed and the data span a period of 5 years – between January 2007 and January 2012. Each message contains the following information:

- A minimal set of headers – *Subject*, *From* and/or *Reply-To* address.
- An extended set of headers – these vary, but may contain *Return-Path*, *Received*, *Date*, *X-Originating-IP*, etc.

- A message body – the contents of the email message
- Scam class – e.g. *Lottery*, *Romance*, etc. See table [TABLE].

The original data has been preprocessed heavily to construct the corpus. Using the information extraction techniques in [CHAPTER], we have discarded over $11,000$ malformed messages – i.e., messages that do not contain the minimal set of headers, a message body or are not emails. This represents 25.6% of the original data. In addition, we have grouped similar categories together (e.g., *Romance scam* and *Russian romance scam*) and reduced the number of possible classes from 31 to 22. Because the data has been labelled by third parties – the moderators and users of antifraudintl.org, we have randomly sampled 380 messages to measure the labelling error rate. We define this rate as the percentage of email messages assigned a wrong class over the total number of messages in the sample. With 95% confidence, we estimate a 0.79% labelling error rate in the data. Whilst this is not insignificant, given the total size of the corpus, it is unlikely that classification accuracy will be significantly degraded due to the error rate.

32000 922 Romance 263179 870 ATM Card 546786 638 Western Union & Moneygram 204352 628 Misc 192875 603 Widow 555 Church and Charity 5473 Next of kin 454 Military 444 Refugees 4347 Lottery 405 Delivery company 389 Mystery shopper 382 Employment 3659 Government 276 Commodities 2615 Business 1779 Banking 1742 Compensation 1625 Dying people 1608 Fake cheques 1582 Loans 1104 Orphans

6.3 Methodology

In this section we briefly describe the maximum entropy model. We then outline our feature selection and methodology for training the classifiers.

6.3.1 Maximum Entropy model

The Maximum Entropy model is a statistical classification model which seeks to optimize for the probability distribution with the maximum entropy, subject to the constraints of the training set . It is a an exponential model of the following form:

$$p(a|b) = \frac{\prod_{j=1}^{k} \alpha_j^{f_i(a,b)}}{\sum_a \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}} \tag{6.1}$$

where $p(a|b)$ denotes the probability of predicting a, given b, α_j is the estimated weight of feature f_i, and $\sum_a \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$ is the normalization factor.

By combining evidence into a bag of features, the Maximum Entropy model allows us to represent knowledge about a problem in the form of contextual predicates. Also, in contrast to Naïve Bayes, another commonly used classifier for text categorization [REFERENCE Mitchell, 1997)], the Maximum Entropy model does not assume conditional independence between observed evidence. This allows us to incorporate overlapping features, such as whole words and corresponding n-grams, and typically achieve better probability estimates than possible with Naïve Bayes. [REF]

6.3.2 Feature selection

Optimal feature selection for supervised learning problems is a challenging task. Exhaustive enumeration of all feature sets is computationally infeasible, so popular approaches often rely on greedy algorithms such as forward selection and backwards elimination. Nevertheless, building the best possible text categorization model is not the main goal of this project, so we have opted to use n-gram based features, which have shown satisfactory performance in the literature [REF][REF]. The tables [FIGURE], [FIGURE] illustrate the feature selection for the scam variation classifier and the PQ classifier, respectively.

Scam variation classifier - N(-)Grams (length 1–4) - Prefix and Suffix NGrams - lowercase words - lowercase ngrams PQ classifier: - lowercase words - use ngrams - yes - no prefix or suffix ngrams

6.3.3 Training the classifiers

We have trained both classifiers on data from the 419 corpus. The scam variation classifier is trained on a 80

In order to estimate the performance of our predictive models, we have performed cross-validation. The methodology and results are presented in [CHAPTER, SECTION].

Chapter 7

Response generation

This chapter details the functionality and implementation of the response generation component. As emphasized in sections [SECTION], [SECTION], response generation depends on the outputs of the information extraction, classification, and identity generation components.

We begin by introducing a bucketing algorithm for grouping messages into conversation threads. Next, we discuss strategies for generating engaging replies. In section [SECTION], we go into detail on how responses are generated and provide examples. Finally, in [SECTION], we outline briefly the mechanism to send out emails anonymously.

7.1 Bucketing algorithm

The first step in generating a response to an incoming message is to determine whether the message belongs to a conversation thread. We define a thread as a logical unit of all messages related to a single instance of a scam. Furthermore, it is important to note that a single instance may involve multiple actors. Formally, given an incoming message M and a set of threads $T = \{T_1, T_2, ..., T_n\}$, determine if $\exists M \in T$.

This is a challenging problem. A naïve approach assumes that each conversation thread can contain at most two actors – the agent and the scammer. Therefore, keeping track of the From header is sufficient to maintain conversation state. Whilst this is true in some instances, we showed in [SECTION, Background] that many scams involve multiple actors. Another approach is to compute the thread with the largest word overlap for each incoming message. This is a better solution, but it makes the assumption that the message body always contains a quoted response. Due to the nature of AFF

scams, this is also ineffective.

We propose a bucketing algorithm which combines ideas from the aforementioned approaches with one important observation – the scammer always notifies the target in advance when he is being transferred to a third party – e.g. "*please contact the prize remittance manager, John Smith at john.smith@domain.com*". Therefore, an effective way to keep track of threads is to keep track of the mentioned emails in each message. In order to do this, we attach a bucket to each thread and collect email addresses. Once a new message comes in, we try to match it to a bucket. If successful, we update the bucket with any new email addresses. Otherwise, we create a new thread. This is illustrated by the pseudocode in [FIGURE].

[FIGURE - pseudocode]

Let us assume we have threads T_1, T_2, T_3 with corresponding buckets B_1, B_2, B_3, such that $B_1 = \{E_1, E_2\}$, $B_2 = \{E_3\}$, $B_3 = \{E_5\}$ where E_n is an email address. Message M_1 enters the system with a candidate bucket $B_c = \{E_3, E_6\}$. Then, for B_n in B, $\arg\max |B_n \cup B_c| = \{1\}$. Therefore, we attach M_1 to T_2 and update $B_2 \coloneqq B_2 \cup B_c$. Message M_2 enters the system with a candidate bucket $B_c = \{E_{10}\}$. For B_n in B, $\arg\max |B_n \cup B_c| = \{0\}$. Therefore, we create a new thread T_4 with $B_4 \coloneqq B_c \cup \emptyset$.

7.2 Strategies

The agent employs a set of strategies designed to occupy the scammer for as long as possible. These strategies can be divided into two categories – cooperative and non-cooperative. Cooperative strategies are mainly used to gain the scammer's confidence and are fulfilled by responding positively to requests. Non-cooperative strategies are used throughout each conversation thread and aim to deflect questions, drive the conversation to a different topic, or elicit extra work from the scammer.

7.2.1 Cooperative strategies

In our implementation, we use three different cooperative strategies.

The first strategy is to always express interest in any proposed scheme in the beginning of a conversation thread. The initial responses are always enthusiastic and reaffirm the premise set up by the scammer. For example, if the scammer's initial email claims we have won the lottery, we do our best to pretend that we did. This strategy helps create the impression that the agent is a viable target and encourages the

scammer to spend time replying back.

Another cooperative strategy involves responding to requests for personal information. As we discussed in [CHAPTER], scammers commonly request personal information as a way to establish the target's trust. If the target gives out personal details, it is considered much more valuable to the scammer, as it is very likely to cooperate with other future requests. Therefore, a useful strategy for our agent is to respond positively to these requests. If asked, our agent responds to these requests with personal details obtained from the identity generation component. This helps build up the scammer's perception that the agent is a viable target and makes him more invested in the conversation.

The final cooperative strategy is reaffirmation. Reaffirmation is used throughout each conversation thread to restate the agent's interest in the scammer's proposition. It is often used in conjunction with non-cooperative strategies to express that we are still interested in the scheme, despite any setbacks we might have introduced. For example, an example of reaffirmation is claiming – "*I look forward to working with you to process my winning*" at the end of the message, whilst asking many extra questions in the message body.

7.2.2 Non-cooperative strategies

We use four main non-cooperative strategies in the implementation of our agent.

The first non-cooperative strategy is deflecting questions. As we discussed in [SECTION], answering certain types of questions is beneficial, as it helps establish trust. However, complying with other types of questions can be dangerous or impossible. One example are requests for photo identification. Cooperating with these is clearly a bad idea. Instead, we choose to compose excuses – e.g. "*I am sorry, but I am bad with computers. Could you tell me how to put a photo in this letter?*" Excuses are usually effective at bypassing these questions altogether. Alternatively, the scammer has to spend the time to write a mini tutorial on how to work with email attachments.

Another non-cooperative strategy is asking questions about exceptional circumstances. As we observe in [SECTION], answering questions is a challenging NLP task. Instead, it is better to ask questions and attempt to drive the conversation. These questions are closely related to the variation of the scam in play and are designed to elicit extra work from the scammer. For example, in the context of a lottery scam, the agent might ask if the winnings are subject to any tax or whether they can be paid out

in Australian dollars.

Stories are the third non-cooperative strategy. They are employed in later stages of a conversation and are context independent. The main purpose of stories is to take up the scammer's time by having him read a large chunk of text, following a few initially promising exchanges. Because of this, each story is between 350 and 600 words long. At the end of a story, the scammer is asked whether he can relate to the situation described in the story. By requesting a comment, we validate that he has read the story.

Lastly, our final non-cooperative strategy is to prompt the scammer to resend messages. This behavior is elicited by composing replies that claim the scammer's previous message has been accidentally deleted, is garbled, or has never been received. This strategy has a single goal – to force the scammer to look through his inbox, find the correct message, and resend it. It is a simple way to get the scammer to do extra work.

7.3 Composing a response

Responses are generated from the outputs of the information extraction, identity generation, classification tasks and current conversation state via a multi-layer finite state machine. The process works as follows: the top layer FSM builds a template with placeholders for lower-level probabilistic finite state machines (PFSMs). The lower-level PFSMs emit text or another layer of PFSMs. At the bottom layer, all PFSMs generate text. This approach can also be described as top-down text generation.

7.3.1 Understanding the context

Understanding natural language is a very challenging task. Fortunately, advance fee fraud scams tend to follow a set pattern within a relatively narrow domain ([CHAPTER]). This makes it possible for us to use techniques such as machine learning, pattern matching rules and the conversation state to generate convincing replies.

Two maximum entropy classifiers are central to our effort to understand the contents of an incoming message. The the scam type classifier helps determine which one of the 22 known AFF variations is currently in play. For example, if we are dealing with an instance of a lottery scam, we will use the corresponding lottery PFSMs and text snippets to generate that part of the response. The second classifier helps determine whether the scammer asks us to provide any personal information. If so, similarly as before, we will use another set of corresponding PFSMs to generate that information.

Rules are the second method we use to understand the context of a message. Rules are specific to each AFF variation and allow us to look for patterns that are strong signals for conversation state. In the aforementioned lottery scam, we recognize four distinct states – *initial, claim form approved, payment approved, fee request*. Knowing the current state allows us to generate a reply with information specific to that state and creates the impression of a natural response.

Finally, it is not always possible to understand the context of a message through a classifier or rules. Where this is not possible, we use the current state of the thread, as determined by the bucketing algorithm, and compose a reply using a non-cooperative strategy. For example, if the current state shows we are in the beginning of a thread, we will use the asking questions PFSM. In later stages, we will pick randomly between prompting the scammer to resend the message or the story strategy.

7.3.2 Text snippets

We use text snippets in the final states of the PFSMs to generate large paragraphs of text. These text snippets are written ahead of time and are stored in a hierarchical tree data structure. Top-level nodes represent specific scenarios – *photo request*, *lottery*, etc. Scenarios which describe scam variations have another set of mid-level nodes which list all recognized states of the variation. Finally, at the bottom of the tree, leaves contain a set of text snippets associated with their parent nodes. Text snippets are diverse, but usually consist of 1–3 sentences, with placeholder variables for conversation-specific information (e.g. the agent's identity). For current system prototype, we have defined 37 nodes and 169 text snippets. Extending the collection of text snippets is easy and requires no changes to the implementation – new snippets can be saved as text files in the file system.

Further to adding context, text snippets also provide variability to the generated responses. This is accomplished through redundant tree leaves. These leaves have similar sentiment, but are phrased in different ways. Let us illustrate this. Consider the example scenario of answering questions for personal information. The response paragraph is constructed from text snippets of the following bottom-level tree nodes: intro, name, age, occupation, location, postcode, contact details, closing. These nodes have 5, 7, 8, 5, 4, 2, 6, 6 leaves respectively. A PFSM crawls the tree and randomly selects a single leaf from each parent node. The outputs are then compiled into the final text paragraph. In this instance, the aforementioned process allows for $8 \times 7 \times$

$6^2 \times 5^2 \times 4 \times 2 = 403,200$ distinct paragraph compositions. As such, the probability of generating a duplicate response is very low. We use this approach to generate each paragraph of the outgoing message.

7.3.3 Finite state machines

Several layers of FSMs and PFSMs determine how each response is generated. We start by looking at the top-level FSM. We move on to a simple PFSM used for generating greetings. Finally, we illustrate the PFSMs that generate the content body.

The top-level FSM defines the high-level structure of our message – a letter. The Greeting, Body and Signature states represent a set of PFSMs, whilst Quoted_Text simply produces a quoted version of the incoming message. The FSM is illustrated below.

The greeting generation automata is an example of a simple bottom-level PFSM. Its task is to accept the name of the scammer and produce a greeting, as illustrated in [FIGURE]. The transitions between the has_name state and the text snippets have probability $p = \frac{1}{3}$.

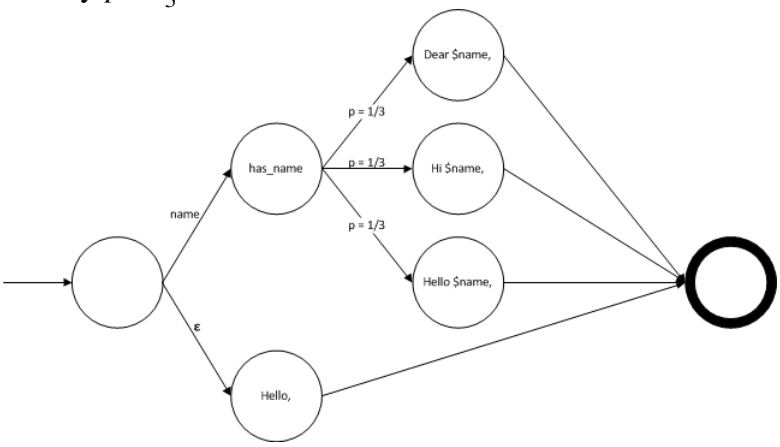
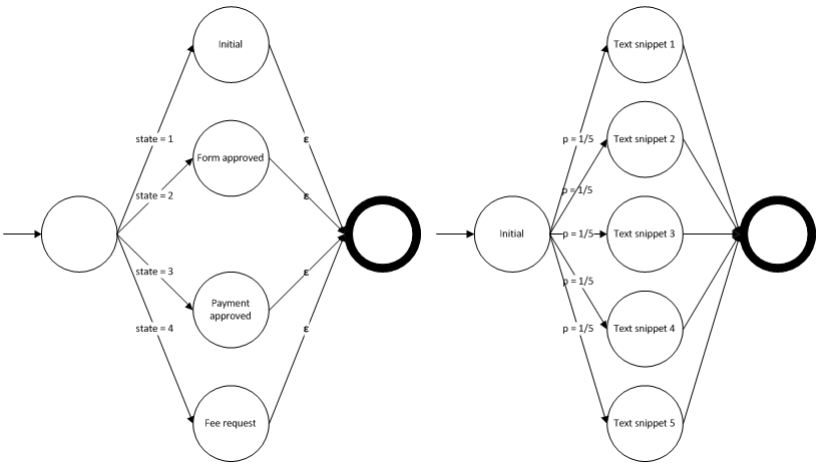


Figure 3 (missing) illustrates the PFSM which generates the content body of a message. It takes in as inputs the results of classifier, information extraction and identity generation tasks. Its main purpose is to select strategies and create placeholders for lower-level PFSMs. Strategies are selected based on the available information and conversation state. Please note, for clarify of the figure, we have omitted some of the notation.

Figure (a) illustrates a lower-level FSM for the lottery scam variation. The scam



(a) Lottery FSM (b) Snippet FSM

is modelled by four distinct states. Figure (b) illustrates a bottom-level PFSM for the same scam, after a transition to the initial state.

We follow the same approach when modelling other scam variations and conversation strategies.

7.3.4 An example message

[FIGURE]

7.3.5 Sending the reply

As we described in [SECTION], anonymous email accounts are used to send and receive emails. In order to ensure consistency, each thread has an associated identity and email account. Once a message is generated, it is transmitted via a SSL connection to the SMTP server

Through experimentation, we have established that sending 40 messages with less than a second delay between messages results in the termination of the associated email account by the provider. To mitigate this, we observe a random backoff period between consecutive SMTP connections. This is currently set to a minimum of 70 and a maximum of 170 seconds between connection attempts.

Chapter 8

Evaluation

Tchapter contrains the evaluation of our prototype system. In [SECTION .1] we define the terminology we use in our results. [SECTION 1.2] focuses on intermediate results, i.e., results of various probabilistic tasks. Finally, we provide a discussion on the meaning of these results.

8.1 Terminology

Throughout this chapter, for the sake of clarity and brevity, we use the following terminology to present our results.

F_1 The F_1 score (also F-measure) is the harmonic mean of precision (p) and recall (r), as defined by the formula: $F_1 = \frac{2rp}{r+p}$.

Macro F_1 The value is the result of averaging the F_1 scores for all classes.

Micro F_1 The value represents a global calculation of F_1, regardless of class.

Thread Each thread is defined as a logical unit of all messages related to a single instance of a scam. Threads are proportional and contain messages from both the agent and the scammer.

B_{rate} This is a measure of the bounce rate of our sample. It is defined as all threads containing messages that cannot be delivered (due to deleted email boxes, exceeded quota, etc.) over the total number of threads.

P_{rate} This is a measure of the participation rate of our sample. It is defined as all threads that contain at least one reply from a scammer, over the total number of threads in the sample (excluding bounced threads).

Other abbreviations For brevity, in some tables we use the abbreviations *TP, FN, FP, TN* to mean, respectively, *True Positive, False Negative, False Positive, True Negative*.

8.2 Intermediate results

The intermediate results presented in this section measure the performance of probabilistic tasks in the classification and information extraction component.

8.2.1 Scam variation classifier

The scam variation classifier is trained on a classical 80:20 split of data from the 419 corpus. Our model achieves Macro $F_1 = 0.81177$ and Micro $F_1 = 0.77134$ on the $5,996$ data points in the validation set. A breakdown of the results is shown in [FIGURE].

Class	**TP**	**FN**	**FP**	**TN**	**Precision**	**Recall**	**F_1**
ATM card	106	50	55	5755	0.658	0.679	0.669
Employment	35	22	17	5892	0.673	0.614	0.642
Next of kin	962	98	99	4807	0.907	0.908	0.907
Banking	206	132	146	5482	0.585	0.609	0.597
Fake cheques	233	31	30	5672	0.886	0.883	0.884
Orphans	153	60	55	5698	0.736	0.718	0.727
Business	287	211	234	5234	0.551	0.576	0.563
Government	552	137	122	5155	0.819	0.801	0.810
Refugees	51	36	29	5850	0.638	0.586	0.611
Church & Charity	79	26	15	5846	0.840	0.752	0.794
Loans	253	10	10	5693	0.962	0.962	0.962
Romance	134	14	26	5792	0.838	0.905	0.870
Commodities	35	18	11	5902	0.761	0.660	0.707
Lottery	801	30	53	5082	0.938	0.964	0.951
WU & MoneyGram	92	23	31	5820	0.748	0.800	0.773
Compensation	275	60	70	5561	0.797	0.821	0.809
Military	77	9	6	5874	0.928	0.895	0.911
Widow	55	61	36	5814	0.604	0.474	0.531
Delivery company	50	26	17	5873	0.746	0.658	0.699
Misc	82	41	34	5809	0.707	0.667	0.686
Dying people	270	24	25	5647	0.915	0.918	0.917
Mystery shopper	55	4	2	5905	0.965	0.932	0.948
						Macro F_1: 0.81177	
						Micro F_1: 0.77134	

8.2.2 PQ classifier

We evaluate the performance of the personal questions (PQ) classifier using 2-fold cross-validation on a set of 500 email messages. The averaged results are shown in [TABLE]. Note that since this is a binary classifier, the first two table rows are reciprocal.

Class	**TP**	**FN**	**FP**	**TN**	**Precision**	**Recall**	**F_1**
PQ	96	26	7	121	0.932	0.787	0.853
No PQ	121	7	26	96	0.823	0.945	0.880
						Macro F_1: 0.86800	
						Micro F_1: 0.86667	

8.2.3 Named-entity recognition classifier

As described in [SECTION], we use a CRF classifier with a pre-trained model on the CoNLL 2003, MUC-6, MUC-7 and ACE named entity corpora. In this section, we measure the model's performance on a validation set of 50 manually labelled messages (18655 words) from the 419 corpus. Whilst the sample is comparatively small, our time is limited and the model is already reported to achieve $F_1 = 90.36$ for the *Persons* class on the CoNNL 2003 dataset (see [SECTION]). The results are summarized in [TABLE]. Note that we do not include the *Locations* and *Organizations* classes, as they are not relevant to our use case.

Class	**TP**	**FN**	**FP**	**TN**	**Precision**	**Recall**	**F_1**
Persons	354	85	31	18185	0.919	0.806	0.859

8.3 Conversation results

In this section we present results of the conversations our system has maintained with scammers. We begin by reviewing our methodology. In Sections [SECTION][SECTION], we describe a number of interesting statistics.

8.3.1 Methodology

To measure the performance of our prototype system, we deployed it on a virtual private server ([SECTION]) and let it run continuously from 19 March to 28 March. Over the 10-day period, the system gathered and processed 348 scam instances in 19 classes. Of the 348 instances, 45 were in the set of currently supported classes – *Lottery, Orphans, Mystery shopper*. We present detailed statistics from these conversation threads in [SECTION]. To put our results in context, we provide several baseline measures, depending on the purpose of the experiment. For example, to measure the bounce rate of threads, we draw random samples from the remaining set of scam instances, in order to compare it with the sample obtained by the system.

Experiment overview	
Duration	10 days
Scam instances	348
Encountered classes	19
Threads	45

8.3.2 Bouce rate

We m

a 10-day period from a virtual private server ([SECTION environemtn]). Over the course

3. Conversation results i. Methodology 3 samples. sample 1 organic from the client. sample 2, random subset of 0. Setup. 10-days. 348 + 45. Another sample where we manually responded to 20 messages. Say time is limited. i. Bounce rate - bounce rate for thread - bounce rate for baseline ii. Participation rate - braindead rate - test sample - other sample iv. Distribution of scam

'church$_{a}nd_{c}harity'$: 3,$'next_{o}f_{k}in'$: 45,$'lottery'$: 41,$'fake_{c}heques'$: 4,$'business'$: 41,$'dying_{p}eople'$: 9,$'orphans'$: 3,$'government'$: 18,$'misc'$: 18,$'romance'$: 56,$'banking'$: 7,$'compensation'$: 2,$'delivery_{c}ompany'$: 14,$'commodities'$: 4,$'mystery_{s}hopper'$: 1,$'western_{u}nion_{a}n$
7,$'loans'$: 61,$'atm_{c}ard'$: 7,$'widow'$: 7

348 scam instances, 19 classes, 45 supported scam instances

v. Average thread length, words per thread - inclusive of all - existing emails - answered emails - human-sized sample 35/5 = 6.8

Possible other measures: In addition to average thread length, average number of words in thread? If it's easy to remove quoting etc., average

This chapter contains the evaluation

design and all probabilistic classification tasks. In

In addition, we also present results from all probabilistic classification tasks.

some intermediate results from

the evaluation of the performance of our system, as well as

This chapter contains the evaluation of the methods proposed in the previous part, chapter 4. In section 5.1, we present the data sets and the methodology used for testing. We give details about the parameter choices and set baseline scores obtained by either classical NCA or simple linear projections, such as PCA, LDA or RCA. Results for each individual method are presented in sections 5.3 and 5.4. A comparison of the methods is shown in subsection 5.3.4 using accuracy versus time plots. We should mention that we did not include all the results in this chapter to prevent cluttering. Further experimentations can be found in appendix B

This chapter details the functionality and implementation of the response generation component. As emphasized in sections [SECTION], [SECTION], response generation depends on the outputs of the information extraction, classification, and identity

generation components.

Chapter 9

Conclusions

This chapter details the functionality and implementation of the response generation component. As emphasized in sections [SECTION], [SECTION], response generation depends on the outputs of the information extraction, classification, and identity generation components.