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

Acknowledgements go here.

The class is a modification of the `cs4rep` style used in the computer science department until 1998-9.

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 Using Sections 2

1.2 Citations 2

1.3 Class Options 2

1.4 Restrictions 2

2 Response generation **3**

2.1 Bucketing algorithm 3

2.2 Strategies 4

2.2.1 Cooperative strategies 4

2.2.2 Non-cooperative strategies 5

2.3 Composing a response 6

2.3.1 Understanding the context 6

2.3.2 Text snippets 7

2.3.3 Finite state machines 8

2.3.4 An example message 9

2.3.5 Sending the reply 9

Bibliography **10**

Chapter 1

Introduction

The document structure should include:

- The title page in the format used above.
- An optional acknowledgements page.
- The table of contents.
- The report text divided into chapters as appropriate.
- The bibliography.

Commands for generating the title page appear in the skeleton file and are self explanatory. The file also includes commands to choose your report type (project report, thesis or dissertation) and degree. These will be placed in the appropriate place in the title page.

The default behaviour of the class is to produce documents typeset in 12 point, and appropriate for doubled sided printing (all new chapters appearing on the first clear right-hand page). Regardless of the formatting system you use, it is recommended that you submit your thesis printed (or copied) double sided.

NB please note that the report should be printed single-spaced. Previously advertised policy of printing in double space has changed as of November 24th 1999 and is no longer valid. Space recommendations are revised as follows: the dissertation should be around 40 sides in single space printing. The page limit is 60 sides in single space printing. Appendices are in addition to the above and you should place detail here which may be too much or not strictly necessary when reading the relevant section.

1.1 Using Sections

Divide your chapters in sub-parts as appropriate.

1.2 Citations

Note that citations (like [Chang and Keisler, 1990] or [Arimura, 1997]) can be generated using `bibtex` or by creating `thebibliography` environment. This makes sure that the table of contents includes an entry for the bibliography. Of course you may use any other method as well.

1.3 Class Options

The only class option available is `parskip`. It alters the paragraph formatting so that each paragraph is separated by a vertical space, and there is no indentation at the start of each paragraph. This option is used in the current document. See `documentclass` in the skeleton file for usage.

1.4 Restrictions

The class does not allow the use of `listoffigures` or `listoftables`.

Chapter 2

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.

2.1 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.

The bucketing algorithm 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 $B_4 \coloneqq B_c \cup \emptyset$.

2.2 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.

2.2.1 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.

2.2.2 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.

2.3 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.

2.3.1 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.

2.3.2 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.

2.3.3 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 $p = \frac{1}{3}$.

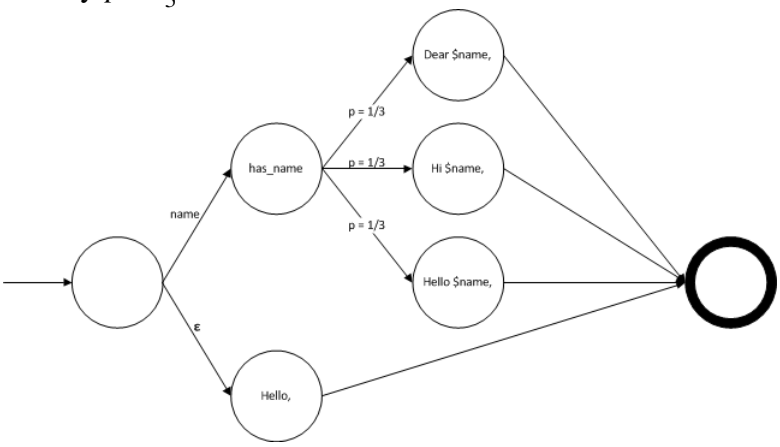
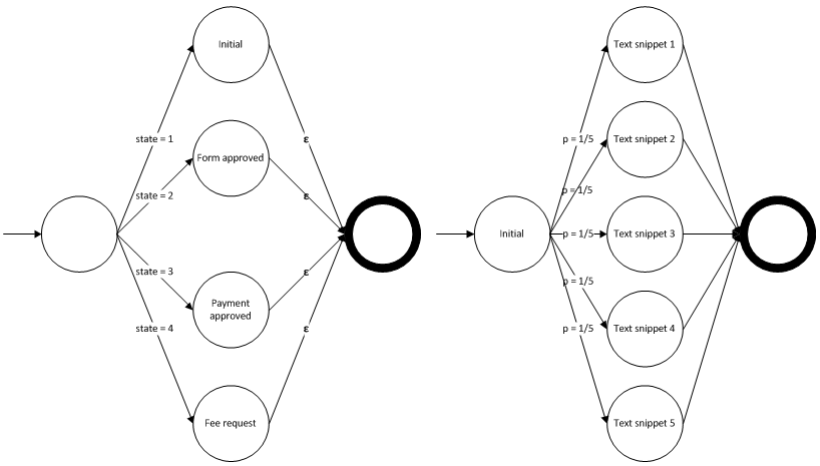


Figure 3 (missing) illustrates the PFSM which generates the content body of a message. It takes in as inputs the results of classifier, information extraction and identity generation tasks. Its main purpose is to select strategies and create placeholders for lower-level PFSMs. Strategies are selected 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.

2.3.4 An example message

[FIGURE]

2.3.5 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.

Bibliography

[Arimura, 1997] Arimura, H. (1997). Learning acyclic first-order Horn sentences from entailment. In *Proceedings of the International Conference on Algorithmic Learning Theory*, Sendai, Japan. Springer-verlag. LNAI 1316.

[Chang and Keisler, 1990] Chang, C. and Keisler, J. (1990). *Model Theory*. Elsevier, Amsterdam, Holland.