Hashing, Pooling and Coding:

Towards Optimal Embedding in Linguistic Steganography



Alex Wilson

Worcester College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2016

Abstract

The goal of steganography, the art of hiding information, is to send hidden messages without revealing their existence to outside observers. So-called *cover* objects are modified to convey the *payload*, the hidden message. In the majority of literature, covers are images; this work is concerned with the sub-field of *linguistic steganography*, where covers are pieces of natural language.

This area has received little attention, and as a result has lagged behind the state-of-the-art approaches present in image steganography. This work identifies the following weaknesses of the current linguistic steganography literature: methods to assign values to cover sentences are inefficient; the amount of data hidden in covers (*capacity*) is low; security is rarely tested. The first two weaknesses are linked, low capacity caused, in part, by inefficient value assignment techniques. The last weakness is an inexcusable flaw; the evolution of linguistic steganography is impossible without thorough security analysis.

This work applies concepts from image steganography and steganalysis in order to address these flaws, and to provide a robust framework for future linguistic steganography research. A novel linguistic stegosystem is developed, utilising *hashing* to assign values to cover objects, and manual input to ensure fluency of generated steganography. Security of the system is evaluated twice: first against a human attacker, an approach uniquely appropriate to the linguistic setting; then against a novel automated attack, utilising the paradigm of *pooled* steganalysis.

The automated attack is powerful against the generated steganography, until *adaptive embedding*, a technique well known in image steganography, is applied to the linguistic setting for the first time. This utilises *coding* to minimise *distortion* (a measure of the detectability of generated steganography). Four linguistic distortion measures are described, the first of their kind. The resulting steganographic objects are shown to be secure both against humans, and the developed attack.

Contents

1 Introduction **1**

1.1 The Art of Hiding Information 1
1.2 The Thesis 3
1.3 Linguistic Setting 4
1.3.1 The Censor 5
1.4 Roadmap 7
1.5 Terminology 8

2 The Three Core Tasks **11**

2.1 A Brief History of Steganography Research 11
2.2 The Cover Source 12
2.3 Problem 1: The Transformation 13
2.3.1 Kerckhoffs' Principle 17
2.4 Problem 2: Value Assignment 18
2.4.1 Linguistic Steganography as Graph Labelling 20
2.5 Problem 3: Coding 23
2.5.1 Linguistic Steganalysis 25

3 The CoverTweet System **26**

3.1 Cover Source: Twitter 26
3.1.1 Data 28
3.2 Manual Steganography 29
3.3 Transformation 31
3.3.1 The Paraphrase Database 31
3.3.2 Implementation 33
3.4 Value Assignment 34
3.4.1 Hash Functions 35
3.4.2 Implementation 38

3.5 Ordering: Language Modelling . . . 39
3.5.1 Statistical Language Modelling . . . 41
3.5.2 Implementation . . . 44
3.6 Human Filtering . . . 45
3.7 Experimental Design and Results . . . 47
3.7.1 Evaluation of Stegosystem Security . . . 49
3.7.2 Evaluation Results . . . 51
3.8 Summary . . . 55

4 Attacking The System **57**
4.1 Pooled Steganalysis . . . 58
4.2 Proposed features . . . 59
4.2.1 Basic: 130 features . . . 60
4.2.2 Probability features: 45 features . . . 61
4.2.3 Paraphrase features: 28 features . . . 64
4.2.4 Pooling the features . . . 68
4.3 Experimental Design and Resources . . . 68
4.3.1 Data . . . 68
4.3.2 Feature Extraction . . . 73
4.3.3 Classifier . . . 74
4.3.4 Experimental Design . . . 75
4.4 Results . . . 75
4.4.1 Classifying Individual Tweets . . . 75
4.4.2 Pooled Steganalysis . . . 78
4.4.3 T-Lex . . . 83
4.4.4 Feature Evaluation . . . 86
4.5 Summary . . . 90

5 Coding **91**
5.1 Non-Shared Selection Channel . . . 91
5.1.1 Syndrome Coding . . . 93
5.2 Minimising Distortion . . . 95
5.3 The Optimal Embedding Paradigm . . . 97
5.3.1 Simulating Optimal Embedding . . . 98
5.4 Distortion Measures for Linguistic Steganography . . . 99
5.5 Data . . . 102
5.5.1 Manual CoverTweet . . . 102

5.5.2 Automatic CoverTweet . . . 104
5.5.3 T-Lex . . . 106
5.5.4 Pooling . . . 107
5.6 Results . . . 108
5.6.1 Automatic CoverTweet . . . 108
5.6.2 Manual data . . . 110
5.6.3 T-Lex . . . 117
5.7 Square Root Law . . . 118
5.8 Summary . . . 121

6 Conclusions **122**
6.1 Applications of Linguistic Steganography . . . 124
6.2 Further Work . . . 126
6.2.1 Embedding . . . 126
6.2.2 Attacking . . . 131
6.2.3 Coding . . . 134

Appendices **135**

A Instructions **136**

B CoverTweet Output **139**

Chapter 1

Introduction

It starts like this:
"Look: Billy Pilgrim has come unstuck in a timely manner."

Kurt Vonnegut, Slaughterhouse-Five
via CoverTweet

1.1 The Art of Hiding Information

Two prisoners, locked in separate cells, are allowed to write letters to one another. They want to plan their escape, but the prison warden is onto them, and carefully reads every letter before passing it on. Cryptography is out of the question, the warden will surely destroy any letter containing an obviously encrypted message. The prisoners must find a way to plan secretly, while appearing to be writing perfectly innocent messages to one another. The solution: *steganography*, the art of hiding information.

The idea of hiding messages is ancient. Its roots stretch at least as far back as Ancient Greece: Greek historian Herodotus relays stories of hiding news of invasion behind the wax on writing tablets, and tattooed under the hair of a slave [Kahn, 1996]. The name for the art itself (derived from the greek terms for 'covered' and 'writing') comes from the work *Steganographia*, written in 1499 by German occultist Johannes Trithemius. After nearly 500 years, tables of numbers in the work (ostensibly about magical communication with spirits) were decoded [Reeds, 1998], and found to be humdrum messages: a Latin sentence using every letter in the alphabet; the start of a psalm.

The origin of modern steganography is less remote. [Simmons, 1984] introduced the above prisoner scenario, and the idea of "the subliminal channel": a communica-

tion channel hidden from outside observers, and what we know now as the steganographic channel. With the explosion in amounts of digital media (ideal covers for hidden messages) shared over the internet, steganography emerged as a serious scientific field of study.

Section 1.5 provides the important definitions for understanding this work, but we summarise some basic terminology here [Pfitzmann, 1996]. The *steganographer* (sometimes called *Alice*) is the actor wishing to send hidden messages to the *receiver* (or *Bob*). *Cover* objects (pre-existing digital objects, e.g. images) are modified to contain a *payload*: the hidden message. The modified object is known as *stego*; the modification process is *embedding*. Objects have a *capacity*, which is the amount of data they can carry, usually measured in bits.

The *attacker* (or, borrowing from the prison metaphor, the *warden*) is observing the channel over which the steganographer is sending these objects. In this work we are only considering a *passive warden*, who can read all messages, but not change, divert, or delete messages. The goal of the warden is to identify which objects contain payload; to distinguish innocent cover objects, from modified stego ones. The alternative is an *active* warden, who can modify objects in an attempt to disrupt the steganographic channel.

Cover objects are images, in various file formats, in the majority of research. An early method for hiding in bitmap images is *Least Significant Bit* (LSB) embedding, which stands as a simple example of steganography in practice [Fridrich, 2010]. To hide the payload, least significant bits of pixel values are altered, so the parity of each pixel matches bits of the payload. These small changes are invisible to human eyes, but were found to be detectable by simple statistical attacks [Lu et al., 2004].

Over time, the focus of the literature has shifted. Starting as clumsy heuristic approaches to hiding and detection, it has slowly evolved to include robust steganography theory. This brings us to the focus of this work, which is still firmly in the realm of the clumsy and the heuristic: *linguistic steganography*.

In this area, the steganographic channel is not images, or digital files, but *natural language*; cover objects are pieces of text[1]. Compared to image steganography, linguistic steganography has received very little attention. This has left the area lagging behind the state-of-the-art, something we intend to address here.

[1]Strictly speaking, cover objects need not be text in linguistic steganography; they could take the form of spoken word recordings, for example. This has never been tackled in any literature, but would likely make little difference in practice.

Note that our above descriptions assume the *cover modification* paradigm of steganography. Cover *selection* is an alternative, where covers already containing the desired payload are chosen from a large set. For any reasonable payload, this requires a vast collection of covers, and to the best of our knowledge has never been applied to linguistic steganography. Another alternative is cover *synthesis*, where covers containing the desired payload are created from scratch. We will describe the problems with this paradigm in Chapter 2.

1.2 The Thesis

By virtue of receiving the majority of attention, image steganography represents the state-of-the-art, and the area has benefitted from a number of recent technologies and methods. Modern image steganography is concerned with minimising the *cost* of embedding, achieved by indirectly transmitting payload through the application of *coding*. This same technology allows for the solution of the *non-shared selection channel* problem (discussed in Chapter 5), which arises when the receiver does not know where the steganographer could have hidden their message.

In addition to practical solutions, work on steganographic theory has found theoretic bounds on capacity, and embedding efficiency; a result known as the *square root law* shows that there is a sub-linear relationship between the total payload and the number of cover objects, subject to a detection bound [Ker et al., 2008].

Often, separate steganographic media are studied as separate topics of research, with separate research cliques, and separate avenues for publication. As a result, less popular research areas are relearning important findings from image steganography years later.

We contend in this work that the important findings from image steganography are equally applicable to linguistic steganography, and we are applying them here. In particular, the field has been continually hounded by the problem of the non-shared selection channel; assigning values in an unambiguous way is difficult in linguistic steganography, and certain methods result in the receiver not knowing which objects contain payload. The result in the literature is severe restrictions on linguistic embedding, in order to avoid this, and disappointing results for steganographic capacity. We present a solution for unrestricted embedding, and describe the application of coding to handle a non-shared selection channel, while simultaneously minimising the *distortion* caused by embedding.

Work on linguistic steganalysis has been lacking: both in content, and quantity. In addition, many published stegosystems decline to thoroughly test security, often conflating language *fluency* with security. This is especially true of systems employing manual input to guarantee fluency, as we will be doing here. Linguistic steganography can simply not improve without robust security analysis; the idea of improving the field without knowing the current state of security is ludicrous.

Once again, we contend modern ideas from image steganalysis are appropriate for linguistic, and are applying them here. We have produced the first serious work on linguistic steganalysis, using both human attackers and an automated attack utilising large feature sets, explicit acknowledgement of *Kerchkoffs'* principle [Kerckhoffs, 1883], and *pooled steganalysis* [Ker, 2006]. The last idea, which sees the warden attempt to identify guilty actors instead of steganographic objects, has yet to see successful and widespread application, even in image steganalysis.

We believe the following will prove true:

1. Linguistic steganography produced with manual input will be undetectable to human judges.

2. Automated attacks may work against linguistic steganography when using pooled steganalysis.

3. Coding will be of significant help to linguistic stegosystems.

4. Minimising distortion will reduce the detectability of generated stego.

A general goal of this work was to put linguistic steganography on a sound steganographic footing, bringing the literature in line with the state-of-the-art from image steganography. Aside from the novelties present in our approach to linguistic embedding and steganalysis, we hope to provide a template for linguistic steganography research. Note that we will not solve linguistic steganography: we expect our methods to be beaten by future work.

1.3 Linguistic Setting

We make use of a single linguistic setting here: the social networking site Twitter. Twitter is a popular site (∼310m active users [Brandwatch, 2016]) where registered users can post public messages (*tweets*) limited to 140 characters. While we will give full description and justification of the setting in Chapter 2, there are two important points to make here.

First, we wish to dispel any thoughts that this decision was borne of frivolity; rather, the strict definition of a realistic setting allows us to make a strict definition of what security means in this context, which we will give soon. Prior work has given little thought to where covers will be sourced from, which we believe to be an oversight. The security of a linguistic stegosystem likely relies heavily on what the covers are: good steganography requires noisy covers, which Twitter easily provides by being rife with typing and spelling errors.

Secondly, despite the setting, all techniques presented in this work are widely applicable. We have deliberately abstained from the use of Twitter-only methods, despite there being a great deal of potential in the area; any Twitter specific possibilities are relegated to the discussions in Chapter 6. In applying these techniques to a new setting, a re-evaluation of security would be required.

1.3.1 The Censor

A revised prisoner scenario for the 21st Century: Alice and Bob live in China, communicating only with Sina Weibo, the Chinese micro-blogging site. They wish to share rebellious messages with one another, but the channel is being monitored by censors; any noticeable sedition could have consequences. The two users must find a way to share their messages secretly. This scenario is entirely realistic, and we will use it to inform our model of the attacker.

Sina Weibo was launched in 2009; since then it has amassed approximately 100m daily users, and has 100m messages posted each day. The site is heavily censored; some users attempt to circumvent this by using particular phrases, arguably a crude form of steganography [Independent, 2015]. One example is the use of the phrase "hide-and-seek" to mean someone has died in police custody. These phrases are only useful while their true meaning is not widely known; once they are discovered, messages containing them are retroactively removed. Note that this same behaviour has been observed amongst teenagers on the social networking site Facebook; some researchers have suggested it be called *social steganography* [Boyd and Marwick, 2011].

In a study on Sina Weibo censorship, [Zhu et al., 2013] noted that the majority of filtering happened within 5-10 minutes of posting. Given the volume of messages, it is clear that automatic systems must be in use. The work identified that certain keywords (relating to controversial topics) triggered three types of censorship, depending on the exact word used. The first explicitly informs the user that the content is "against the rules"; the message is never posted to the site. The second delays the posting of the message, claiming server issues; the authors believe this is to allow

time for manual checking. The final type renders the message invisible to all but the original user, without informing them of this fact. This is an example of *shadow banning*, a common method employed by message boards against spam, or abusive users [Atwood, 2011].

In this work, we will assume the presence of censors on Twitter, attempting to identify steganography in use. In this scenario we pretend that steganography is against the terms of service. We recognise two models for adversary: a human censor, and an automated censor.

Here the two are disjoint; we will not consider the possibilities of combining the two until discussions of future work in Chapter 6.

Human Censor

The goal of the human censor is to read individual tweets, and attempt to identify those that are steganographic. State-of-the-art computational linguistics cannot yet compete with human level language skills, and the danger posed by the human is one unique to the linguistic setting; humans pose no threat at all to image steganography. Despite this, prior literature has never (with the exception of some work on cover generation [Orgun, 2009]) explicitly employed humans to detect steganographic text.

From an abstract, dehumanised perspective, this censor has vast language skills, but low computational power: low memory, and unable to handle too many operations. It is our intention to evaluate whether these language skills are enough to spot linguistic steganography; as a consequence, this censor will fall short of being a full Kerckhoffs' attacker (an attacker who knows all aspects of the system).

The specific knowledge of the attacker is this: they know the general mechanics behind linguistic steganography, but not the exact transformations used by the system we will be evaluating. This is perhaps representative of a censor who is looking for steganography, without knowing the exact systems in use on the platform.

Automated Censor

The human censor has limits. We know from the later work in Chapter 3 that it takes a human approximately 36 seconds to judge a tweet; there are 6000 tweets posted *every second*. Assuming 8 hour work days, 600k people would need to be employed to monitor the entire of Twitter. This number is prohibitive, and is certainly also the case on Sina Weibo: the number of messages sent is far too many for manual checking, so automated systems are used.

The automated censor is the antithesis of the human: relatively poor language skills, but significant computational power. This increase in computational power gains the automated censor two advantages:

1. The censor can reasonably keep track of statistics and abnormalities beyond the scope of the human.

2. It is feasible for a censor to compare new tweets to earlier ones from the same user.

In this work we will exploit the first by giving the automated censor full knowledge of the system in use, including the exact details of possible changes during embedding (the automated censor will be a full Kerckhoffs' attacker). We exploit the second by utilising the paradigm of *pooled steganalysis*: instead of simply judging tweets individually, the censor will pool together evidence from the same user, and attempt to detect the presence of any steganography in any of the tweets. It is our belief that such a technique is beyond the scope of the human's computational prowess.

1.4 Roadmap

Chapter 2 breaks the task of linguistic steganography down into three problems, giving a thorough definition of each one, while simultaneously acting as a literature review. Viewing the task as three, mostly independent, problems is beneficial from the point of view of critiquing earlier work. To the best of our knowledge this breakdown is unique in the literature, and applicable to any linguistic stegosystem.

In Chapter 3, we present our novel linguistic stegosystem *CoverTweet*. This is adapted from our paper "Linguistic steganography on Twitter: hierarchical language modelling with manual interaction" [Wilson et al., 2014], and details the system more thoroughly than in the original work. One important novelty of our system is the use of *hashing* to assign values, gifting our system the ability to change covers in any way, without worrying about *desynchronisation*, where the receiver is looking in the wrong place for the message. We evaluate the system for security against human judges, assuming the first of our censor types, as defined above.

In Chapter 4, we take the role of the warden, and develop an attack against CoverTweet. This chapter applies standard elements of image steganalysis to the linguistic setting for the first time: a large feature set used to train a classifier, and the explicit application of Kerckhoffs' law in our attack. We utilise the paradigm of *pooled steganalysis*, a relatively recent concept, which has yet to see successful research in any

steganography setting, let alone linguistic [Ker et al., 2013]. This attack assumes an automated censor, and is adapted from our short paper "Detection of steganographic techniques on Twitter" [Wilson et al., 2015].

In Chapter 5 we return to the role of the steganographer, and apply *coding* to linguistic steganography. This allows the transmission of un-changeable cover objects, and lets us minimise the *distortion* made during embedding. This is adapted from our paper "Avoiding detection on Twitter: Embedding strategies for linguistic steganography" [Wilson and Ker, 2016]. This work is important to the linguistic steganography field, as it solves the *non-shared selection channel* problem, which has recurred throughout previous literature, and reduces the detectability of stego objects by minimising distortion.

Chapter 6 concludes the work, and discusses avenues for future research.

1.5 Terminology

Listed here is a collection of basic steganographic terminology. In addition to these, we will use the acronym *bpt* for *bits per tweet*; this is in lowercase to match the convention of *bpp* for *bits per pixel*.

Cover The object that is modified in order to convey the secret message. In linguistic steganography this is some form of text.

Stego The resulting modified object, which carries the payload.

Cover Source The source of covers.

Payload The name for the data that we are trying to hide.

Capacity The amount of data that can be hidden in a cover; this is equal to the entropy of the cover.

Embedding The act of modifying a cover in order to convey the payload. In this work we will only be considering this type of steganography, known as *cover modification*.

Extraction The act of extracting the payload from the received stego object.

Embedding Efficiency The amount of payload conveyed per change made to the cover.

Steganographer, or Alice The person performing the steganography.

Key The key shared by Alice and Bob, to allow Bob to extract the message.

Receiver, or Bob The recipient of the stego object, their only task is to extract the message.

Stegosystem The name for a particular method of steganography, encompassing the cover source, the embedding and extraction algorithm, and the channel used to exchange data between Alice and Bob.

Selection Channel The subset of the covers where embedding changes are allowed; when this is not known by both Alice and Bob, it is *non-shared*.

Kerckhoffs' Principle The principle that we should assume any attacker knows the workings of the system.

Warden or Attacker The party attempting to spot the use of steganography. We assume a *passive warden* throughout, who is looking to prove that steganography is in use. The *active warden* is attempting to close the steganographic channel.

Pooled Steganalysis The paradigm of steganalysis (the study of steganography detection) where the warden is attempting to spot guilty actors, as opposed to individual steganographic objects.

Distortion or Cost The cost of embedding: how much the act of embedding has moved the distribution of stego objects away from the distribution of cover objects.

Adaptive Embedding The form of embedding that aims to minimise the distortion caused.

Transformation The method utilised by linguistic stegosystems to modify cover text.

Value Assignment The method utilised by linguistic stegosystems to assign values to elements in the cover.

Coding A method of conveying the message indirectly, granting a boost to embedding efficiency, and fixing the problem of non-shared selection channels.

Hash Function One-way functions that map input data of any size to data of a fixed length.

n-gram A sequence of n words or tokens.

Language Model A model which provides probability estimates for language. A common type is n-gram models, which estimates the probability of n-grams.

False Positive Rate The proportion of cover objects incorrectly labelled as stego by a classifier.

False Negative Rate The proportion of stego objects incorrectly labelled as cover by a classifier.

Equal Prior Error The equal prior error $P_E = \frac{FP+FN}{2}$, where FP is the false positive rate, and FN the false negative rate.

Chapter 2

The Three Core Tasks

Return my calls, Ishmael

Herman Melville, Moby-Dick
via CoverTweet

Though not currently acknowledged in the literature, every cover-modification based linguistic stegosystem can be broken down into three main components: transformation, value assignment, and coding. This chapter will step through the anatomy of a linguistic stegosystem, discussing the approach taken by the literature at each point.

2.1 A Brief History of Steganography Research

As noted in Chapter 1, sending secret messages has a storied history, but steganography as an area of academic research is still relatively young, having been introduced by [Simmons, 1984] as "the subliminal channel". In the early years, the literature was concerned with finding out *where* a message could be hidden, in a variety of image file formats. An example of this is *least significant bit* (LSB) embedding [Ker, 2007], which was a widely known technique.

Eventually it became apparent that regardless of the actual changes made during embedding, each one had a *cost* or *distortion* [Crandall, 1998]. It was not known how this fact could be exploited, so the community turned to coding and communication theory for a solution. This led to techniques which allowed certain parts of the cover to be left unchanged, without having to tell the recipient of this fact [Fridrich et al., 2005, Fridrich et al., 2006].

With the development of *syndrome trellis coding* [Filler et al., 2011] (see Section 2.5), it became possible to minimise the distortion while embedding (known

as *adaptive embedding*). This ushered in the latest change in focus: the literature is now concerned primarily with accurately modelling cost.

This description of the shifting research focus in image steganography is relevant here, because linguistic steganography has stalled in that first stage: the literature is entirely concerned with finding *places to hide*. We are attempting to bring linguistic steganography in-line with the state-of-the-art of steganography. As part of this, we hope to demonstrate that the *linguistics* aspect of linguistic steganography is largely separate from the actual act of embedding, and that the field has more in common with image steganography than previously thought.

It is worth noting that linguistic steganography is not alone in its embryonic state: *network* steganography [Lubacz et al., 2012] is a recent addition to the steganography canon (attempting to hide in network communications), and is likewise still just trying to find places to hide.

2.2 The Cover Source

Explained in Chapter 1, the cover is the original object that the system embeds information into. Depending on the system, a cover might be as general as a sentence or paragraph, or a specific piece of text such as an email or a text document.

There is one cardinal rule to consider: the original cover must not be available to an outside observer. If a system attempts to hide in news articles sourced from the internet or famous novels, the system is destined to fail. An attacker can distinguish stego from cover just by knowing the existence of the original text (cf. the epigraph). Though not a strict requirement, a general solution is for the steganographer to write the cover themselves.

Though also true for images, it is worth noting the effect this has on the *bandwidth* (measured in payload bits per second, as opposed to payload bits per cover byte) of the channel. Text takes significantly longer to write than digital images are to take: even if a sentence could hide as much as a large image (we will see that this is certainly not the case in published systems), the bandwidth of the linguistic steganographic channel would still be significantly smaller than for images.

We will not be considering the source-less *cover generation* methods here. Although many published works tackle this paradigm, all contain the same fatal security flaw: the generated covers are utter nonsense. Some systems, such as Neko [Grosvald and Orgun, 2011], use humans to create passable text on a sentence level, but the generated covers are still nonsense on the global level.

2.3 Problem 1: The Transformation

Given a source of covers, we can think about how to hide within them. Regardless of the type of cover (e.g. individual sentences, paragraphs, e-mails, text documents, etc.), the first step is always to apply some *transformation* to the text, in order to generate the possible options for the stego.

According to the literature, the only requirement of the transformation is that it preserves the meaning of the original cover. Justification for this requirement has never been given; it has been passed on from publication to publication without question. Potentially it arose organically as a way to separate cover *modification* methods from cover *generation*. The requirement was notably not included in a list of objectives for linguistic stegosystems, given by [Bergmair, 2004], a thorough early work on linguistic steganography; its absence suggests that not all researchers believe the requirement, though it is nonetheless upheld by all published methods.

We are not convinced by the requirement, although the system we propose in Chapter 3 adheres to it. We discuss possible alternatives to the restriction in Chapter 6.

Some of the work discussed in this section was designed for the goal of linguistic *watermarking*, rather than steganography. Both fields aim to embed messages in text, but watermarking embeds with the goal of message *robustness* in mind, instead of *covertness*: it does not matter if people can detect the embedded messages, as long as they cannot *remove* them. On the whole the systems are not compatible, although approaches to the transformation are applicable to both areas.

Note that there is confusion in the field, and some linguistic steganography systems are mislabelled in the literature as watermarking (e.g. [Halvani et al., 2013]); this overlap is possibly another reason for the semantic preservation requirement, as it is certainly a true requirement of watermarking.

Substitution

The earliest transformation (and the one best represented in the literature) is *synonym substitution* which generates stego options by swapping individual words for their synonyms. This method was first proposed in [Bender et al., 1996], and first implemented in the *Tyrannosaurus Lex* (T-Lex) stegosystem [Winstein, 1999]. T-Lex uses a subset of WordNet[1] [Miller, 1995] as a substitution source. The set of

[1]WordNet is an English language lexical database, containing 155,287 words grouped into sets of synonyms (*synsets*).

he	had	a	*statuette*	on his	*mantlepiece*
			figurine		*mantel*
					mantelpiece
					chimneypiece

Figure 2.1: Synonym substitutions allowed with T-Lex.

```
lost my eyeglasses again
somebody's not acquiring a present
tantalum for the lift to school
```

Figure 2.2: Example tweets transformed by T-Lex.

possible stego objects is generated by substituting each word in the cover with an alternative from this substitution source.

In the example shown in Figure 2.1, we are given the cover "`he had a statuette on his mantlepiece`". T-Lex's substitution source gives us synonyms for "`statuette`" and "`mantlepiece`"; we can generate the stego options by replacing the original words with these, e.g.:

```
he had a figurine on his mantel
he had a statuette on his chimneypiece
...
```

It will likely be clear to a native English reader that not all substitutions are made equal, and T-Lex is prone to outputting obviously *odd* text, as in the examples in Figure 2.2, generated from real-world tweets. Note that the system is particularly fond of chemical element substitutions, shown by the use of `tantalum` (the original used the informal word `ta`, meaning thanks).

T-Lex's set of substitutions is severely limited, to allow for an easy solution to the problem of value assignment (see the next section for more on that), only containing ∼9k synonym sets. This gives a low capacity of approximately 0.14 bits per sentence. Many systems have improved upon this transformation, using different (and larger) substitution sources, and additional tools such as *part of speech* (POS) taggers and language models [Bolshakov, 2004, Taskiran et al., 2006, Topkara et al., 2006a].

[Chang and Clark, 2010] extended this type of transformation to work with *phrasal* substitution. The proposed system made use of a paraphrase dictionary, automatically generated by exploiting parallel corpora: pairs of corpora containing the same content in multiple languages. Using machine translation techniques, words and

phrases in one language were aligned with those in the other. When two English phrases were aligned with the same non-English phrase, these were assumed to be paraphrases of one another (the system we develop in Chapter 3 uses a source generated using the same method).

Original	Transformed
`the end of this year`	*later this year*
	the end of the year
	year end

Figure 2.3: Example entry in the generated paraphrase dictionary from [Chang and Clark, 2010]

Figure 2.3 shows an example entry from the generated dictionary. This dictionary is directional: one phrase maps to multiple paraphrase options, but none of the suggested paraphrases have their own entries in the dictionary. The example demonstrates that the generated paraphrases are not appropriate in all situations: "`year end`" refers to the end of the financial year, and cannot be used as a general substitution for "`the end of the year`". To make sure paraphrases are appropriate, the system attempts to automatically check the fluency of each paraphrase before changing the cover, using a two step approach.

First, the Google n-gram corpus [Brants and Franz, 2006] is used to check each paraphrase in context. The corpus is a huge collection of n-grams (sequences of n words) along with the number of times that n-gram was observed on public web pages. This corpus is huge, containing almost 1 billion trigrams (and over 1 billion 4- and 5-grams). A paraphrase passes this step if the frequency counts for the potential paraphrase in context are non-zero. Consider the example "`We're going to Edinburgh at` *later this year* `for a holiday`". The system checks frequency counts for a selection of sub-strings to the right and left of the substitution. In this example, the corpus might have non-zero counts for "`year for a holiday`", "`this year for a`" and "`later this year for`", but it would hopefully fail on "`at later this year`".

The second step is based on a syntactic filter. Combinatory Categorial Grammar (CCG) is a grammar formalism where words in sentences are assigned labels expressing information about their categories (e.g. noun, verb, etc.). The paraphrasing stegosystem assigns CCG lexical labels to each word in the cover sentence (using the parser described in [Clark and Curran, 2007]) before and after making a substitution: the paraphrase is allowed if there is no change in category assignment after transformation.

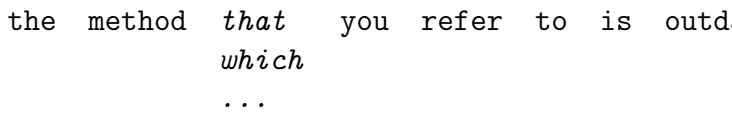


Figure 2.4: Example of a relative clause.

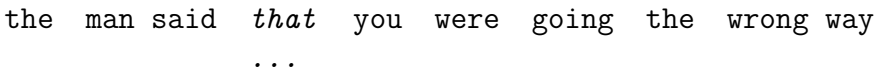


Figure 2.5: Example of a complement clause.

Having determined the appropriate paraphrases, the cover can be transformed by replacing phrases with entries in the dictionary with one of the allowed paraphrases. This system is possibly designed only for sentences with exactly one changeable phrase; there is no described method for when a sentence contains multiple transformable phrases.

There are a number of substitution based techniques that hide in things other than synonyms and paraphrases: [Wang et al., 2009] hides by modifying the choice of emoticon at the end of a sentence; [Topkara et al., 2007] simulates typing mistakes (e.g. substituting `the` for `teh`), and [Shirali-Shahreza and Shirali-Shahreza, 2007] uses variation in so-called *text speak* (a shorthand way of typing messages on a phone). The security of these is untested, and the latter is arguably out-of-date since the rise of predictive typing and full 'QWERTY' keyboards on mobile phones.

Syntactic Methods

An alternative class of transformation exploits syntactic features of language. [Atallah et al., 2001] suggested a number of these in the early days of linguistic watermarking, but implementations came in later work.

[Murphy, 2001] and the follow up [Murphy and Vogel, 2007] presented a system for hiding in relative and complement clauses. Relative clauses modify noun phrases to specify additional details about the subject. In the example in Figure 2.4, "`that you refer to`" is the relative clause. The transformation used to hide information involves changing the *relativiser*: the connecting word between the noun phrase and the clause. In the majority of relative clauses this can vary between `that`, `which`, or be dropped entirely.

Complement clauses act as arguments to verbs, taking the place of noun phrases. With these, the transformation drops the *complementiser* `that`, as seen in Figure 2.5.

```
I  have a  red  car
          ...
```

Figure 2.6: Example of a deletable adjective.

[Zhang et al., 2005] and [Meral et al., 2006] introduced similar syntactic transformations, but for Chinese and Turkish respectively.

Other Methods

There are some more eclectic transformations found in the literature. One example is [Chang and Clark, 2012], which transforms sentences by identifying and removing deletable adjectives. These adjectives can be identified using the Google n-gram corpus and a similar contextual check to the one used in [Chang and Clark, 2010] to identify appropriate paraphrases. Figure 2.6 shows an example of a deletable adjective.

2.3.1 Kerckhoffs' Principle

We have questioned the need for meaning preservation in transformations, but there is an important requirement that *must* be observed: *Kerckhoffs' principle* [Kerckhoffs, 1883].

Well known and adhered to in other security fields (such as cryptography), Kerckhoffs' principle states that the security of a system must not rely on the secrecy of its workings. Systems should remain secure even if knowledge of their steganographic algorithm should fall into the hands of the enemy; history tells us to be cautious on this front. We refer to enemies with knowledge of the system as *Kerckhoffs' attackers*.

This is a wise practice when designing proprietary stegosystems, but ignoring it in published research is inexcusable: if your system is not secure against Kerckhoffs' attackers, *the act of publication has demolished its own security*[2]. Unfortunately, linguistic steganography is often mistakenly conflated with *text* steganography, a long standing branch of steganography that has *no security at all* against such attackers.

This surprisingly persistent strain of steganography hides in text formatting, rather than the language itself. For example, messages are embedded by tweaking character encodings (e.g. [Por et al., 2012]), or word and line spacings (e.g. [Shirali-Shahreza and Shirali-Shahreza, 2006, Por et al., 2008]). These features typically do not vary much (or at all) within a document, so a knowledgeable attacker can trivially identify steganographic documents.

[2] An alternative argument is that it had no security to begin with.

The weakness of text steganography transformations can perhaps be quantified with reference to the part of steganographic theory relating to capacity. The theory tells us that the capacity of an object is its entropy[3] (or, the entropy of the feature we intend to hide in). Without fail, text steganography systems are attempting to hide in a feature with a negligible entropy (zero, in most cases).

Some linguistic transformations are guilty of the same flaw. Already mentioned, [Shirali-Shahreza and Shirali-Shahreza, 2007] behaves like other synonym substitution methods, but restricts the substitutions to 'SMS' or 'text-speak'. An example synonym set is `sorry`, `sry` and `sory`. Intuition tells us that the true entropy of this feature is low: most people have their own preferred set of abbreviations, and constant variation is rare. [Topkara et al., 2007] hides in typing mistakes; this is vulnerable in the same way.

These vulnerabilities are intuited, rather than mentioned in any publications; the literature in this field is particularly weak with regards to evaluating security. The message-hiding capability of any feature ultimately lies with accurate distortion estimation, something lacking from the literature.

2.4 Problem 2: Value Assignment

Having generated the set of stego options, the system must assign each one a value, which will be used to convey the payload. In digital media this is a straightforward step (e.g. the parity of pixel values in an image), but the problem of assigning values to language has plagued the linguistic steganography community since its inception.

Given a set of n stego objects, it should be possible to send $\log_2 n$ bits of payload (the simplest method being numbering alphabetically). The problem is this: how can you assign values to language in a way that lets a recipient unambiguously extract the same value?

For synonym substitution methods, T-Lex and many subsequent systems assign values on a word by word basis by numbering the synonyms alphabetically. In the example shown in Figure 2.1, `statuette` would be changed to `figurine` to convey a 0, or left unchanged to convey a 1. The receiver of the text "`he had a figurine on his mantel`" can use T-Lex to extract the message in binary: T-Lex has no options for `he`, `had` or `a`, so these convey no message; as we know, `figurine` conveys a 0; `on` and `his` again have no alternatives; `mantel` is in a set of 4, and is the 2nd alphabetically, so conveys 01. The total message is therefore three bits: 001.

[3] see Section 2.5 for more on this.

The difficulty in synonym substitution based systems comes from words with multiple senses or meanings (*polysemes* or *homographs*, depending on the reader's preferred definitions). Synonym sets containing such words are *non-disjoint*, and numbering alphabetically is no longer a straightforward solution. Consider the word `bank`, which has a multitude of meanings. If the cover sentence is "`she walked to the riverbank`", `riverbank` might be in a set containing `bank`, `edge`, and `embankment` (numbered alphabetically, conveying 00, 01 and 10 respectively). The word `bank` has a different set of synonyms (containing alternatives for all meanings of the word), and thus a different alphabetical numbering. Upon receiving "`she walked to the bank`", which numbering should be used to extract the message?

As already mentioned, T-Lex's solution is to simply remove all words with multiple meanings from the substitution set, leaving ∼9k synonym sets each containing only words with unambiguous meaning.

More advanced synonym substitution systems (such as [Bolshakov, 2004] and [Chang and Clark, 2014], previously described) allow the use of words with multiple meanings by using contextual checks. These approaches have similarities with the computational linguistics problem of *word-sense disambiguation*. This task aims to identify which sense of a word is being used in a given sentence, and is difficult, even to humans.

With perfect disambiguation (if such a thing is possible), a system could minimise the number of non-disjoint synonym sets; total eradication in most scenarios is likely not possible. Disambiguation techniques are crucial to ensure quality steganography (a system employing such a technique would surely not output T-Lex's `tantalum` gaffe), but value assignment does not become trivial in their presence. Consider an artificial scenario of a system utilising proper noun substitutions, along with standard synonym substitutions:

$$\left\{\begin{matrix} 0 : \texttt{Sainsbury's} \\ 1 : \texttt{supermarket} \end{matrix}\right\} \quad \left\{\begin{matrix} 0 : \texttt{supermarket} \\ 1 : \texttt{Tesco} \end{matrix}\right\}$$

We have the same problem of non-disjoint sets as before, and no word-sense disambiguation can help us in the sentence "`I just got back from the supermarket down the road`". The first proper solution to non-disjoint transformations came in [Chang and Clark, 2014], which developed a technique that utilises graph labelling to assign values to synonyms: see Section 2.4.1.

Alphabetical labelling is not possible for systems using phrasal substitutions, nor in most non-substitution based systems. In order to hide with paraphrasing and by deleting adjectives, [Chang and Clark, 2012] and [Chang and Clark, 2010] assign

values by splitting the cover (assumed to be text containing multiple sentences) into chunks containing a fixed number of sentences. This number is set so that each chunk contains at least 1 changeable element: either a deletable adjective, or a paraphrasable phrase.

To convey a 1 in a chunk, all changeable elements within the chunk are changed; to convey a 0, the elements are left unchanged. In the case of adjectives, these are deleted, so the receiver will extract a 1 if there are no deletable adjectives left in a chunk (and a 0 otherwise). In the case of paraphrases, the paraphrase source used by the system only allows substitutions in one direction: once a phrase has been changed for a paraphrase, the system cannot change it back. The receiver will extract a 1 if no paraphrasable phrases remain in the chunk.

The fixed number of sentences in each chunk must be shared with the receiver in order to facilitate extraction. Note that the capacity of these systems depends on this value.

In practice, all systems have faced a trade-off between limiting the transformation, or limiting the values assigned (whether this is implicit or explicit). Even in our hypothetical single word substitution system employing "perfect" word-sense disambiguation, we contend the transformation is implicitly limited by not using phrasal substitution.

2.4.1 Linguistic Steganography as Graph Labelling

[Chang and Clark, 2014] presents a synonym substitution system that uses a method other than alphabetical value assignment. Having filtered the synonyms using a contextual checking method, the remaining synonyms are used to build a graph, like the example in Figure 2.7. Values are then assigned by finding a graph labelling such that each maximal clique of the graph contains all symbols of a particular base. Synonyms in multiple synonym sets are in overlapping maximal cliques, but the labelling assigns them the same value in each set.

The exact method has two limitations: it is designed to work only with binary symbols, rather than maximising the base of the alphabet, and it is only designed to work with single word substitutions. To demonstrate the problem of the binary symbols, consider the example in Figure 2.7 again. The published technique would assign each node a single bit, but assigned ternary symbols, each node conveys $\log_2 3$ bits (approx. 1.58 bits).

It is possible to generalise their presented solution to give a more formal definition of the entire problem of value assignment in linguistic steganography. We can do this

by defining each cover element (this might be a word, a phrase, or a whole sentence, depending on the system) as a node on a graph. Each node shares an edge with every node it can be substituted for, in order to hide information. We will call this the *transformation graph*, and every maximal clique on the graph a *transformation set*: a set of cover elements that can all be substituted for one another.

To assign values, we want to label the graph, ideally numbering each transformation set with symbols of a base as close to the size of the set as possible. Imagining this graph for T-Lex, each node is a single word, and the graph connects words which are synonyms of each other. It numbers disjoint transformation sets alphabetically, as in Figure 2.7.

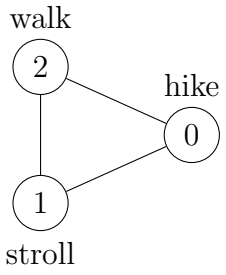


Figure 2.7: An example synonym graph.

The capacity for any node is its entropy. Assuming all words have equal probability, the entropy of each node in a disjoint transformation set T is $\log_2 |T|$. In Figure 2.7 each node is conveying its full payload with a ternary symbol. Problems arise when transformation sets are non-disjoint. For synonym substitution systems like T-Lex, this occurs when words have multiple meanings. In Figure 2.8, `hike` is now in two transformation sets. The new set contains only two nodes, so has been numbered with binary symbols; however, we now do not know the base of the 0 assigned to `hike`.

We have three options. The first option is to number the graph with symbols of a base equal to the size of the smaller set, but then the larger set is conveying a sub-optimal payload. The second option is to assign symbols of a larger base, but then the smaller set is not capable of conveying the full range of values (which will be a problem for embedding).

The final option is to make alterations to the graph. We can cut the edges connecting the overlapping sets (or remove the offending nodes). This is analogous to limiting the transformation itself. In Figure 2.9, the edge between `hike` and `raise`

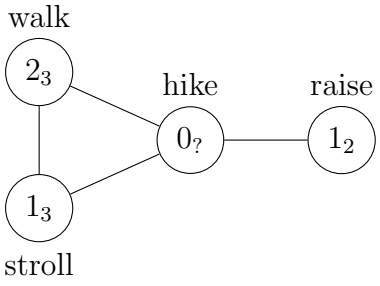


Figure 2.8: A numbered synonym graph, with non-disjoint transformation sets.

has been removed, leaving `raise` unable to carry any payload. T-Lex takes the extreme option of removing *all* non-disjoint sets, so none of the nodes would be able to carry payload.

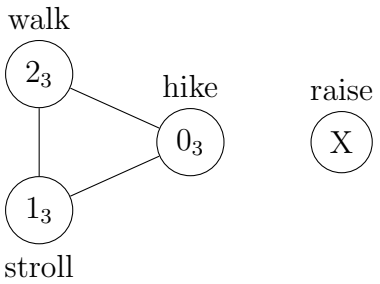


Figure 2.9: A numbered synonym graph

Having labelled the graph, the final step is to *traverse* the graph, and change the elements in our cover for those with values matching our desired payload. In most published systems this is a trivial step; things get harder when changes have a *cost* or *distortion*, and the edges of the graph are *weighted*. We will cover this problem in Section 2.5.

Under this formulation of linguistic steganography, each system must perform the following to embed in a cover:

1. **Build** the graph of cover elements, using a chosen transformation.
2. **Label** the graph.
3. **Traverse** the graph, making changes to the cover to convey the desired payload.

How difficult each step is depends on the transformation: different transformations produce graphs with different properties and parameters. It is possible (though

sometimes a bit forced) to define *every* published linguistic stegosystem in terms of how it approaches these problems. Note that some systems produce graphs where traversal is only possible in one direction, such as [Chang and Clark, 2012]'s adjective deletion system.

2.5 Problem 3: Coding

Having assigned values to the possible stego objects, it might be natural to assume that our job is finished: we just need to transmit the option with our desired payload. However, what happens if we have multiple options with the same value? Which do we send? What happens if we have *no* options with the desired value? The answer to these questions lie in the task of *coding*, which is well known in other cover media (particularly in images), but has never previously been applied to linguistic steganography. Here we will describe the theory behind it, and discuss approaches taken in image steganography, with the intention of highlighting the importance of the task for our linguistic setting.

When we perform embedding on cover objects, any changes to the cover have a *cost* or *distortion*. We saw this in a number of the transformation examples: some substitutions were worse than others, and so their cost is higher. In terms of the graph formulation of steganography, each edge has a cost to travel along it (in fact, two costs: one in either direction). Coding spreads payload out across multiple covers, with the aim of minimising the overall cost. This has an additional benefit, because it allows us to avoid 'missing' values.

There are a number of potential solutions to value assignment that result in some symbols being unavailable. Consider the graph in Figure 2.10, which contains two maximal cliques, numbered with ternary symbols. The smaller transformation set is only capable of conveying two of the three possible values: the remaining value is missing from the graph. We define any missing values as having an *infinite* cost; coding allows us to avoid them.

We denote the cost of embedding a symbol j in a cover i as d_{ij}, and denote the probability of this change being made by the steganographer as p_{ij}. The average capacity of object i is the entropy of p_i

$$H(p_i) = -\sum_j p_{ij} \log_2 p_{ij}, \tag{2.1}$$

and the average cost is

$$E(d_i) = \sum_j d_{ij} p_{ij}. \tag{2.2}$$

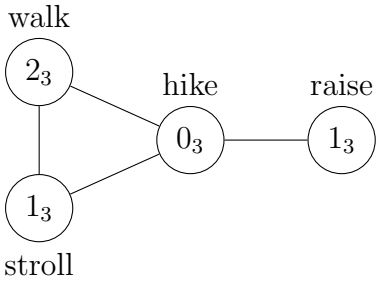


Figure 2.10: A numbered synonym graph, with non-disjoint transformation sets.

There are two ways to frame the task of minimising the embedding distortion:

- Payload Limited Sender (PLS): embedding a fixed average payload m while minimising average distortion, $\min E(d)$ s.t. $\sum_i H(p_i) = m$.
- Distortion Limited Sender (DLS): fixing the average distortion to a value D while maximising average payload, $\max H(p)$ s.t. $\sum_i E(d_i) = D$.

Both options have the same optimal solution for p_{ij} [Fridrich and Filler, 2007b]:

$$p_{ij} = \frac{\mathsf{e}^{-\lambda d_{ij}}}{\sum_j \mathsf{e}^{-\lambda d_{ij}}}, \tag{2.3}$$

for some constant λ, determined by either m or D. Binary search can be used to find the distribution, as $E(d)$ and $H(p)$ are monotone with respect to λ.

For binary symbols, *syndrome trellis coding* (STC) gives a method of embedding close to the theoretical bound [Filler et al., 2011]. Unfortunately, in linguistic steganography we are often dealing with alphabets much larger than binary, and the complexity of syndrome trellis coding grows exponentially with alphabet size. There is a partial solution known as *nested STCs* [Filler and Fridrich, 2010b], though there exists little work on this.

We will avoid the problem altogether by *simulating* perfect coding. By calculating the optimal solution for p_{ij}, we make changes to the cover with this probability. In Chapter 5 we will show how this technique can be used to evaluate the security of linguistic transformations completely divorced from the problem of value assignment.

Note that almost all work on coding assumes *additive* and *independent* costs, which is certainly not the case in linguistic steganography[4]. [Filler and Fridrich, 2010a]

[4]Nor images either, in fact.

presents a method which allows for arbitrary distortion measures (including non-additive), at significant cost to speed; faster solutions for minimising non-additive distortions are not forthcoming.

2.5.1 Linguistic Steganalysis

Recalling the prisoner's problem from Chapter 1, so far we have been acting as the prisoners in the scenario: hiding messages from the prison warden. In *steganalysis* we attempt the opposite, taking the role of the warden and attempting to spot steganographic objects. In order to break the security of a stegosystem the warden need only identify the *presence* of a hidden message.

This section on linguistic steganalysis will be unavoidably brief: to the best of our knowledge there have been just five previous pieces of work on the detection of cover modification based steganography. Each one has attacked either T-Lex or an equivalent proprietary system.

[Taskiran et al., 2006] was the first, using 8 n-gram language models (trained on the same data, with different training parameters) to extract a set of features, including: the number of *out-of-vocabulary* (OOV) words; number of words in the sentence; zero probability words; the mean, variance and maximum n-gram probabilities for a range of n. Training a *support vector machine* (SVM) on features extracted from a training set, this attack achieved a false positive rate of 0.62 and a false negative of 0.15, giving an overall accuracy of 61.5%.

The remaining four works extract a single feature, the mean and variance of which is used to train a classifier: [Xin-guang et al., 2006] uses a feature based on the distribution of characters; [Yu et al.,] and [Chen et al., 2011] estimate a score for *suitability* of the words in a sentence, based on their context; [Xiang et al., 2014] compares the frequency of certain synonyms within cover and stego text.

The evaluation of these works is unfortunately lacking, and occasionally bizarre in experimental procedure ([Chen et al., 2011] creates a testing set entirely out of the works of Charles Dickens). Additionally, it is an unfortunate fact that even beyond steganalysis, very few proposed linguistic stegosystems have been satisfactorily evaluated for security. The majority of work makes no distinction between evaluating for fluency (using human judges) and evaluating for security: the latter is seen to come from the former, which is a flawed assumption. The sentence "`I shall see you anon`" is fluent English, but it might be obviously steganographic in certain contexts (`anon` is unlikely to be a common word for teenagers to use on Twitter, for example).

Chapter 3

The CoverTweet System

> It was the most beautiful of moments, it was the most serious of moments
>
> *Charles Dickens, A Tale of Two Cities*
> *via* CoverTweet

In this chapter we will introduce and describe our system for linguistic steganography, CoverTweet. Figure 3.1 shows the steps involved in embedding steganographic payload in our covers. In-depth descriptions of each step follow.

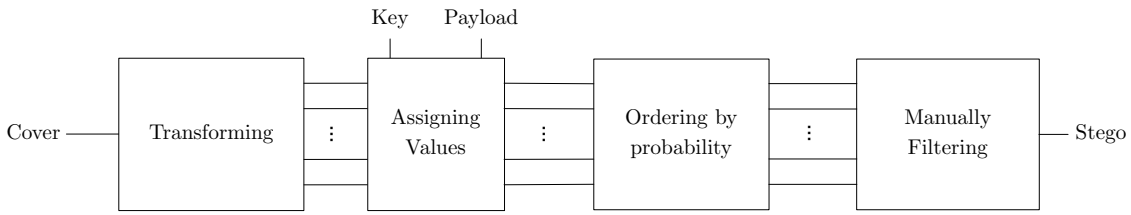


Figure 3.1: The CoverTweet pipeline.

3.1 Cover Source: Twitter

In Section 2.2 we noted the only firm rule in sourcing covers is that the original is unavailable to an outside observer, and that the simplest solution to this is for the steganographer to generate these covers themselves. Although some publications suggest idiosyncratic systems which rely on the steganographer having a specific job (e.g. maths teacher in [Desoky, 2012]), the most likely result of this criterion is that the cover source will be some form of personal communication: e-mails, text messaging, internet comments and posts on social networking sites.

We have a number of options for the last. Facebook is a social networking site with 1.71b monthly active users, and allows for publicly (or privately) visible posts of an arbitrary length. In addition, the site has an instant messaging component, allowing conversations to take place between users who are registered as 'friends' on the site. Google+ is a site with 300m monthly active users, focusing on enabling conversations between people sharing interests. These are both viable settings for steganography, however we will focus on another site: *Twitter*.

Twitter is a social networking website created and launched in 2006. Registered users can post publicly visible messages, limited to 140 characters. These messages are known as *tweets*. Since launch, the site has accumulated 310m active users, and 6000 tweets are posted *every second* [Brandwatch, 2016]. Twitter was one of the first examples of a 'microblogging' site: a broadcast medium that specialises in frequent but short posts. A study on usage of social networking sites [Pew Research, 2015] found that 23% of all internet users in America claimed to use Twitter in some way.

Each Twitter user has a username (up to 15 alphanumeric characters, including certain punctuation symbols), and messages can be directed to other users by including that username prefixed by '@' within the tweet. Twitter includes the ability to *re-tweet* a message: re-posting another user's tweet in order to spread it, echo the sentiment, or add an additional comment. These tweets are marked either by placing `RT` at the start of the copied tweet, or simply by placing the original in quote marks. A further social aspect is the ability for users to subscribe to other user's tweets by 'following' them; the popularity of Twitter pages is usually measured according to the number of followers they have.

Early Twitter lacked a robust search function, so users began categorising tweets using the hash symbol (`#`) followed by a categorising word or phrase (without spaces). For example, users wishing to tweet about the UK general election in 2015 ended tweets with `#ge2015`. These so-called *hashtags* have since been officially adopted by the platform. Hashtags can be appended, or can appear within the body of the tweet, forming part of the sentence. Figure 3.2 shows an example tweet from the University of Oxford Computer Science department.

Twitter has a number of features that make it desirable as a setting for steganography research. Tweets are publicly visible by default, meaning we do not have to concern ourselves with establishing communication channels between the steganographer and the intended recipient (subsequently referred to as Alice and Bob). Twitter is a noisy platform, thanks to the prevalence of hashtags, typos, elongated words and emoticons. Finally, we can assume a reasonable approximation of independence



Figure 3.2: An example tweet from the user `@CompSciOxford`, as it appears on Twitter.

between tweets. For the purposes of minimising distortion (see Chapter 5) this is crucial, as existing coding methods rely on additive distortion measures.

In this chapter, we will assume the following use case: Alice has a Twitter account, and she wishes to send steganographic tweets. She will write the tweet, embed the hidden message by modifying it, then immediately post it to Twitter. In later chapters we will consider an alternative, where the user will write multiple tweets before posting any, for the purposes of minimising distortion.

We will be assuming a human censor when evaluating security, as described in Section 1.3.1. This censor is reading individual tweets, and attempting to spot those that have been modified. As already discussed, this has precedent in the real world: the Chinese microblogging site *Sina Weibo* is subject to monitoring by human censors.

3.1.1 Data

For both implementation and evaluation of CoverTweet, we required a large corpus of tweets. This was provided by the Harvard TweetMap [Mostak, 2013]. This corpus consists of 72m tweets from 1.8m users (in 5 different English speaking countries). We analysed this corpus to find some tweet statistics: the average number of words per tweet is 12.0; the average unique number of words per tweet is 8.6. The entropy of tweets is of interest, given the capacity of a cover is bounded by this. [Neubig and Duh, 2013] estimated the entropy of English language tweets is approximately 250 bits per tweet.

3.2 Manual Steganography

In image steganography, steganographic techniques can easily be invisible to the human eye, but trivially detectable to a statistical attack (such as LSB-based methods). This is likewise the case for text steganography methods, such as adjusting whitespace of a document (see Section 2.3.1 for more). Unlike these, it is impossible to perform 'invisible' linguistic steganographic transformations. This gives rise to a unique weakness of the area: human attackers. Even with state-of-the-art computational linguistic techniques, it is unlikely we can guarantee 'perfect' output: text free of odd phrasings, misuses of words and ungrammatical sentence structures. These imperfect techniques cannot compete with humans' in-built sense of language.

In settings without security concerns, this is excusable; in steganography, it is potentially fatal to security. A single mistake of this nature could signal to a human Kerckhoffs' attacker that the text has been changed by an automatic system. As noted in the previous section, human attackers are plausible in the Twitter domain.

This presents a hurdle to linguistic steganographic research. If a computational linguistic system is incapable of guaranteeing a certain level of fluency, then stego generated with it is likely weak to a human attacker. Consider the example stego texts produced by T-Lex, shown in Figure 2.2; it is difficult to imagine these slipping past a human sentry. Serious linguistic steganalysis work is potentially made irrelevant by this trivial, human-driven detection.

However, our setting offers a viable solution to this problem. We have assumed that the steganographer will generate the cover, which will be modified by the system before being posted to Twitter. The lack of delay between writing and modifying means it is reasonable to utilise the steganographer during the embedding, to do a form of *manual* steganography. Using a human 'in-the-loop' like this, we hope to remove human detectable mistakes from the output.

Manual steganography is not a new idea; it has been used in cover generation systems (e.g. [Grosvald and Orgun, 2011]), and notably in a system that hides information in *Internet Relay Chat* (IRC) channels [Wyseur et al., 2008]. This uses a human to remove incorrect synonyms suggested by the system. This work shares a number of similarities with CoverTweet, however it neglects to describe the workings behind the transformation system, and contains no evaluation of security. The idea of using a human was also mentioned, but not implemented, in [Bergmair, 2004].

In order to differentiate CoverTweet from this earlier work, it is important to outline the specific assumptions we are making about the human in-the-loop. These

are mostly derived through intuition, but we have provided our reasoning.

The first assumption is that the human is *expensive* in terms of time and resources. As such the human should not be used to perform the transformation *in toto*[1]. To understand the issue, consider a human rewriting a cover until it conveys the desired payload. If the payload is randomly assigned to the text, the human cannot direct the generation towards the desired value. We can consider the number of times a rewrite is required as a geometric random variable. For binary alphabets, the probability of success $p = 1/2^b$, where b is the number of bits we are hiding in each message. The expected number of rewrites required is therefore $1/p = 2^b$, so the number grows exponentially with payload size. The cost of the human means that this is out of the question for a practical system. The informal impact of this assumption is: we cannot ask too much of the human.

The second assumption is that the human is capable of identifying *suitably* fluent text with perfect accuracy. The distinction between *fluent* and *suitably fluent* is an important one. In the context of steganography, we take suitably fluent to mean text that is *at least as fluent* as other output from the same user. The argument behind this is straightforward: if the steganographic text reads fluently to the steganographer (who is also the cover source), then the text must be as fluent as other covers from the same source. This means that if the human writes and speaks in very poor English (or any language), then the manually generated steganography is of no worse quality.

The final assumption is that the human is *not* capable of accurately quantifying how similar generated text is to the original cover text. Alternatively: the human is incapable of accurately gauging the impact embedding is making on their own language distributions. The argument here is two-fold: first, generated steganography can shift statistics of the user's language that the user is unaware of (but that are detectable to statistical attacks); second, seeing the generated steganographic options may encourage subconscious decisions on the part of the human. An example of the latter might see the human accepting text that is *more* fluent than the average output from that user, something which is potentially damaging to security. Combined, the second and third assumptions suggest that we can view the human as a coarse distortion measure.

We will investigate in this chapter whether this manual steganography is impervious to a human attacker. In the following chapter, we will evaluate both automat-

[1]Subsequent work [Clarke et al., 2014] has attempted this, but in our view this is not advisable. This work is flawed in many ways, notably misunderstanding fundamental ideas about steganography.

`constituent` | `source` | `target` | `features` | `alignment`

Figure 3.3: The rule structure from the Paraphrase Database. `source` and `target` denote the substitutable strings.

ically generated and manual steganography against statistical attacks. By neglecting to evaluate the generated stego, previous work utilising humans has implicitly supported the view that fluency (given by our second assumption) is equivalent to security; subsequent chapters will empirically show this to be false.

3.3 Transformation

3.3.1 The Paraphrase Database

The prior art dissected in Chapter 2 employed singular transformations (e.g. synonym substitution, adjective deletion) to allow for straightforward value assignment. However, we will see in Section 3.4 that CoverTweet's approach to value assignment is independent to the transformation, leaving CoverTweet free to use any combination of transformations.

As with previous work, we aim to produce a stego object with the same semantic content as the cover; discussions of transformations that do not conform to this are confined to Chapter 6. With this restriction, the act of transformation becomes exactly equivalent to the computational linguistic task of *automatic paraphrasing*. This task aims to produce semantically equivalent text for a number of purposes, including text summarisation, sentence compression, or text normalisation [Madnani and Dorr, 2010]. These applications each aim to generate paraphrased text with particular features, e.g. shorter sentences for summarisation and compression. Here, we are aiming to generate paraphrased text that contains a particular payload.

In order to achieve this, CoverTweet utilises the *Paraphrase Database* (PPDB) [Ganitkevitch et al., 2013]. Released in 2013, the PPDB is a huge database of over 100m paraphrase rules. Each of these rules contains a source string, a target string, features of the rule, and a constituent label for the paraphrase pair (e.g. noun phrase, adjective). Figure 3.3 shows this rule structure, and Figure 3.4 an example rule. The features include the translation probability $\Pr(T|S)$ for a target string T and a source string S. The alignment describes how the words in the source string map to words in the target string, which is only required in a subset of the database.

NN | ill-treatment | mistreatment | -logp(source | target)=1.52
-logp(target | source)=3.68 | 0-0

Figure 3.4: An example rule from the Paraphrase Database, with a reduced feature set. This rule shows that the negative log likelihood of substituting `ill-treatment` for `mistreatment` is 1.52.

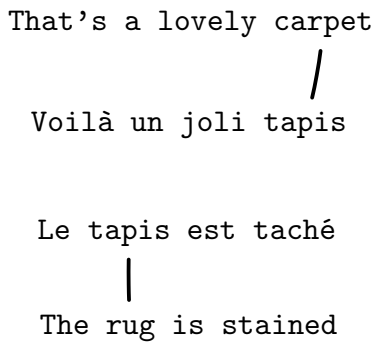


Figure 3.5: An example of the pivoting technique used to generate the Paraphrase Database.

The PPDB was generated automatically from large numbers of bilingual corpora. Each bilingual corpus contains pairs of sentences in different languages, which share a meaning (e.g. the Canadian Hansard, which is available in both English and French).

First, each corpus is *aligned*, an automatic process that connects the words in one language to the translated words in the other. Paraphrase rules are extracted by looking for English phrases which are aligned to the same foreign language phrase; a technique known as *pivoting* [Bannard and Callison-Burch, 2005]. Figure 3.5 shows an example of this. Each sentence pair is partially aligned: in one, `tapis` is aligned with the English word `carpet`; in the other, `tapis` is aligned with `rug`. Because of the alignment to the same French word, we could extract a paraphrase rule linking `rug` to `carpet`. Once extracted, the PPDB rules were scored according to *distributional similarity* in monolingual corpora (i.e. the similarity of the contexts in which the phrases appear).

Although now on version 2.0 (with improved features, and expanded scope), all results described here are based on the first version of the database. This original release is available in 16 languages, and 6 sizes (from 'S' to 'XXXL'). The entries in each size are organised according to confidence score: the smallest set contains only the highest scoring paraphrase rules; the largest contains every extracted rule. Further, the rules are split into three types (see Figure 3.6 for an example of each):

```
VB | ill-treatment | mistreatment
S/. | you 're kidding | you gotta be kidding
NP/NP | [NP/NP,1] [NP/NP,2] collaboration with | [NP/NP,1] [NP/NP,2]
partnership with
```

Figure 3.6: Three paraphrase rules. The first is a *lexical* rule, the second *phrasal* and the third *syntactic*. We have left the use of this final set for further work.

lexical, with only single word substitution rules; *phrasal*, with multi-word phrase substitution rules; *syntactic*, with rules containing non-terminal symbols.

3.3.2 Implementation

For practical reasons, CoverTweet uses the large ('L') database of lexical and phrasal paraphrase rules. This size of set was chosen in order to speed up the embedding process, and to reduce the burden on the human (more rules results in an increase in options for the stego object). We would expect that a larger set would produce similar results to those in Section 3.7; the only advantage would be a reduced number of tweets unable to embed information, which we will see is a problem in the next section.

For a number of reasons, we do not use the syntactic rules. Using this set would require additional resources, such as a parser; also, by only using lexical and phrasal paraphrases, CoverTweet is more directly comparable to earlier systems such as T-Lex. The effect of syntactic rules on security is certainly of interest – in particular the effect of these rules on the performance of the statistical attack described in Chapter 4 – but the application of them is left for further work. Without the syntactic rules, or rules employing deletion, all transformations applied by the implementation of CoverTweet are symmetric.

Prior to embedding, CoverTweet loads the PPDB into memory. A dictionary maps possible phrases (including single words) to a list of rules for that phrase. Each rule contains both the substitution and the probability of that substitution $\Pr(s|c)$ (where s is the stego substitution and c is the cover phrase), which is provided by the rule features in the PPDB. Other features are discarded.

Given a cover tweet, the system processes it: it is made lowercase; URLs are replaced with generic tokens (`URL`); it is tokenised, using the Treebank Tokeniser available in the Natural Language Toolkit (NLTK) [Bird, 2006]. This tokenisation step splits the tweet on words and most punctuation characters. Common contractions are separated (e.g. `don't` is split into `do` and `n't`). Hashtags (words preceded

```
initialise table
for i ← 0, |tweet| do
    for j ← 1, |tweet| − i do
        phrase = tweet[i : i + j]
        table[i, j] ← PPDB(phrase)
    end for
end for
return table
```

Figure 3.7: Pseudocode for the generation of the substitution table, where **PPDB** is a function that takes a sequence of tokens and returns the list of subtitutions for it.

by a hash symbol) are left as single tokens. In practice we would have to reverse this canonicalisation after embedding, but we felt it unnecessary to do so here.

CoverTweet works from left to right, searching the PPDB for paraphrase rules for each subsequent word or phrase in the tweet. Pseudocode describing this process is given in Figure 3.7. The system populates a table mapping index number and phrase length to the list of substitutions for that index.

An example phrase table is shown in Figure 3.8. The phrase `my night` starts at index 3, and is two tokens long, so the table entry for $(3, 2)$ is the substitution `tonight`. Not shown is the probability of that substitution, also given by the table. Note that when the number of possible substitutions for a word is greater than zero, the table also contains the identity probability: the probability of the word remaining unchanged.

CoverTweet augments the rules with one transformation unique to the Twitter setting. When multiple hashtags and usernames appear in sequence, it is assumed these can be permuted without loss of meaning. For example, if a tweet reads "`@Alice @Bob, pub tonight?`", it is equally likely to start "`@Bob @Alice`".

Constructing this substitution table is trivially quick, proportionally; in an experiment based on 1000 tweets, the average time per tweet spent building this sub table was 1.6ms, while the entire process (up to the filter - Section 3.6) took an average of 1s.

3.4 Value Assignment

In the three tasks of linguistic steganography presented in Chapter 2, each task feeds into the next. The design choices made in the transformation step affect the parameters of the value assignment problem, and the value assignment solution affects

i	don't	want	my	night	to	end	.
i do not		want to see my		evening		finish	..
i really don't want			tonight			conclude	...
i just don't want					to be finished		

Figure 3.8: A phrase table produced with the PPDB. This example has been significantly reduced from the full table.

the coding step. CoverTweet is not restricted with regards to transformation; we therefore need to use a value assignment that works with any transformation.

As noted, all value assignment methods are equivalent to labelling a *transformation graph*, but the complexity of this task depends on the transformation used. The more transformations applied, the larger the graph becomes, and the more computationally expensive finding an optimal labelling is. Most methods are predicated on the receiver being able to construct the same transformation graph as the sender. These matching graphs can then be labelled with the same values. However, some transformations (such as those based on word *deletion*) do not allow for such a solution, as the receiver is unable to build the same graph without access to the original cover.

For CoverTweet we use a solution that assigns values without requiring matching graphs, and without performing costly labelling: *hash functions*.

3.4.1 Hash Functions

Hash functions are one-way functions that map input data of any size to data of a fixed length. This fixed length output is known as the *hash*. We can assign fixed length values to stego sentences by using this generated hash. This idea is not new to the field, having first been proposed for steganography in [Anderson and Petitcolas, 1998], and used in the linguistic sub-field of *translation steganography* [Grothoff et al., 2005]. Translation steganography embeds data while translating from one language to another; the choice of translation affects the embedded payload.

Unlike many other security settings, we do not require properties of cryptographic hash functions such as *pre-image resistance* or *second pre-image resistance*; nor do we expect them, given the length of the hash values we will be using. All we require is that the hash function assigns values that are *suitably random*, with small changes

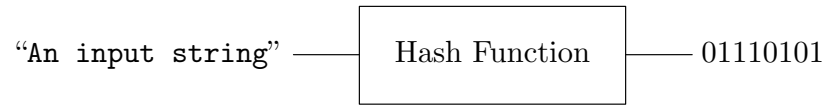


Figure 3.9: The hash function assigns a pseudo-random fixed-length string of bits to an arbitrary length input text string.

to the text resulting in output uncorrelated to the original; we are using the hash function simply to assign pseudo-random values to text strings.

By assigning values pseudo-randomly, we can embed and extract messages without considering the transformation graph at all. If we wish to embed 4 bits per tweet (as we will be doing in this chapter), there are 16 possible values that can be assigned by the hash function; to ensure we are able to embed any value, we need at least one stego option to be assigned each value. This is likely as long as the PPDB produces enough fluent stego options, and our hash function assigns values near uniformly.

Note that in practice, it is likely we would want the hash function to be *keyed*. The sender and receiver would share a short key, meaning only they could extract the correct messages.

There are a number of advantages to such a scheme. The sender and receiver can embed and extract the same values with confidence, without having to construct and label a transformation graph, and without having to apply the transformation in reverse (some schemes require this). In fact, the receiver barely needs to know the workings of the system: they need only know the hash function, and the key. Because we do not need to build the graph, there are no restrictions on the transformation used: any transformations, in any combination, can be freely applied with such a system. This is a significant advantage over earlier work, which relied on low capacity individual transformations in order to send messages.

The scheme is not without downsides. Without constructing the transformation graph, we do not know the capacity of each cover. This means that we are forced to assign the same length hash to *every* stego tweet, regardless of the true capacity.

Consider a tweet with four options for stego object; Figure 3.10 shows the transformation graph for one word from this hypothetical tweet. With the four options we should be capable of transmitting two bits, but the system does not know this without the graph.

On the left, we are assigning a one bit hash to the graph; we have options for both hash values, but are embedding at less than full capacity. On the right, we are assigning a three bit hash to the graph; although each node is conveying a separate

value, we have no options for four of the possible hash values. We say that these values are *missing*, and the system is unable to embed them.

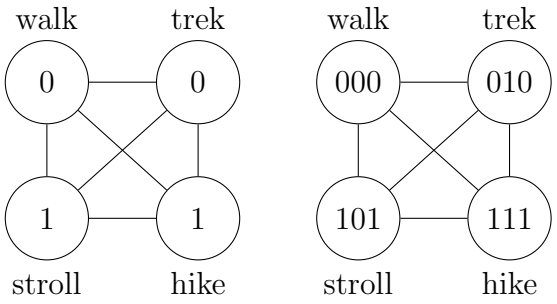


Figure 3.10: Two graphs, labelled with 1 and 3 bit hashes respectively. On the left we are embedding under capacity; on the right, we are missing four values.

These missing values are also caused by *collisions*: the hash function assigning the same value to multiple nodes. Figure 3.11 illustrates this. We are embedding two bit hashes, which we know the tweet has capacity for, however two nodes have been assigned the same hash of 01, and 00 is missing. Note that this is related to the famous *coupon collector's problem*: assuming uniformly distributed hash output, the mean number of distinct stego options required to cover all values of a 4 bit hash (the length of hash used in our experiments in Section 3.7) is approximately 55.

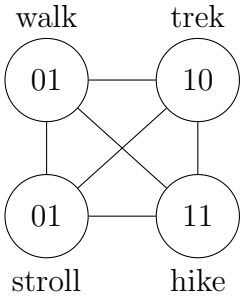


Figure 3.11: A synonym graph, labelled with 2 bit hash values. The value `00` is missing, because two nodes have the value `01`.

Although exacerbated by it, it is important to note that the problem of missing values is not solely caused by hash functions. Even if we were capable of constructing and labelling the graph, CoverTweet uses a human to manually filter the generated stego options (described in Section 3.6), which equates to nodes being removed, as shown in Figure 3.12. We have embedded at full capacity, with no collisions; however,

the human has ruled the node assigned the hash of 10 (`trek`) out, meaning the value is missing.

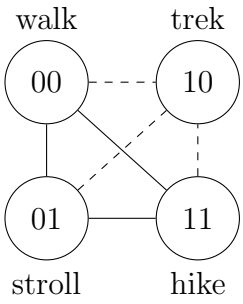


Figure 3.12: A synonym graph, labelled with 2 bit hash values. The human has filtered `trek` out, leaving the value `10` missing.

These missing values are a symptom of a well known problem in steganography (the *non-shared selection channel*). In Chapter 5 we will discuss this further, and address these missing values using coding. Our approach for this initial work is described in Section 3.7.

3.4.2 Implementation

In the transformation stage, CoverTweet populated a phrase table with possible substitutions for each substring. In total, tweets have a vast number of stego options: an average of 3 million (before filtering non-fluent options). However, only a fraction of this total will contain the desired value (on average, $1/2^n$, where n is the length of the hash in bits). Generating each of the options, only to discard the majority, is a significant investment. Instead, CoverTweet's hash function is designed so that only the options with the desired payload are constructed.

For a string s, made up of j tokens $s_0, \ldots, s_j$, the n bit hash assigned to s is given as

$$h(s_0, \ldots, s_j) = \bigoplus_{i=0}^{j} \text{Hash}(s_i) <<< i, \tag{3.1}$$

where $<<<$ denotes circular bitwise rotation, and Hash denotes a base hash function used by the system. In implementation, CoverTweet uses the Python HMAC module, which provides implementations of a number of keyed hash digests of length n; we use the MD5 hash.

There are two important aspects of our hash: by XORing the hash of individual words, we only have to calculate the MD5 hash digest once for each word in the

phrase table; by using bitwise rotation, we ensure that permuting tokens in the tweet changes the final value.

We can define this hash function recursively

$$h(s) = \begin{cases} \text{Hash}(s), & \text{if } |s| = 1 \\ h(s_0, \ldots, s_{|s|-1}) \oplus (\text{Hash}(s_{|s|}) <<< |s|), & \text{otherwise} \end{cases}. \tag{3.2}$$

Let $h(s)_w$ denote a partial hash value $h(s_0, \ldots, h_w)$, and $\text{sub}(i, j)$ the set of possible substitutions for the phrase beginning at index i and ending at index $i + j$. Some substitutions contain multiple words, which means the index of words in the stego option are not correlated with indices in the original cover (which affects the circular rotation), so for convenience we will define Hash′ as

$$\text{Hash}'(x, y) = \bigoplus_{i=0}^{|x|} \text{Hash}(x_i) <<< i + y. \tag{3.3}$$

CoverTweet constructs a dictionary of partial hash values for each index, working from left to right. This dictionary maps tuples containing index i, hash value h and word count w (due to multi-word substitutions, word count is not relative to index) to a list of tuples, each containing a phrase $p \in \text{sub}(i, .)$, the previous hash value $h' = h \oplus \text{Hash}'(p, w)$ and the previous index $i - |p|$.

To initialise the dictionary d, for every substitution $x \in \text{sub}(0, .)$, the system adds the tuple $(x,\ 0,\ -1)$ to the list stored in $d(|x| - 1, h = \text{Hash}'(x, 0), |x|)$ (creating the list if it does not already exist).

To update the dictionary for an index i, for every $x \in \text{sub}(i, .)$, and every key $(i - 1, h', w')$ (for any value of h' and w'), the system adds the tuple $(x,\ h',\ i - 1)$ to the list stored in $d(i + |x| - 1, h = h' \oplus \text{Hash}'(x, w'), w' + |x|)$.

Once constructed, CoverTweet uses the dictionary to generate each possible sentence recursively. Pseudocode for this process is given in Figure 3.13. Based on a sample of 1000 tweets, constructing all options (with the correct 4 bit hash) in this manner takes an average of 0.24s per tweet (including the 7ms to create the hash dictionary); constructing every stego tweet before calculating the hash value takes an average of 3.77s. The longer the payload, the more time we would expect to save.

3.5 Ordering: Language Modelling

The system now has a list of possible stego tweets, all sharing the same hash value. The length of this list depends on a number of factors, including: the length of the

```
function CONSTRUCT(i, h, w)
    partialSentences ← []
    for (p, h′, i′) ∈ d[(i, h, w)] do
        if i′ < 0 then
            add p to partialSentences
        else
            partialSentences′ = CONSTRUCT(i′, h′, w − |p|)
            for p′ ∈ partialSentences′ do
                add p′ + p to partialSentences
            end for
        end if
    end for
    return partialSentences
end function
```

Figure 3.13: Pseudocode for the generation of all stego options with the desired payload. This function is called with the final index, the desired payload, and all possible word counts, and returns the list of all sentences with that payload.

original tweet; the number of bits being embedded per tweet; the number of options in the PPDB for each word and phrase. Some short tweets have no options; conversely, longer tweets with multiple rewritable substrings have millions. On a sample of 1m tweets (from 1000 randomly selected users), only 0.6% of tweets had no options when embedding a randomly chosen 4 bit payload; 65% of tweets had over 1000 options (28% with over 1m).

So far, the intention has been to use the steganographer to generate steganography free of odd linguistic blemishes. However, as already stated, we are assuming humans are *expensive*; certainly, it should be clear that the time taken for a human to read 1000 tweets is prohibitive to a practical system (let alone 1m tweets). To reduce the workload of the human, we wish to order these options before presenting them to the steganographer. This has an additional benefit: any method used for ordering doubles as an initial step towards a *distortion measure* for linguistic steganography, a necessary prerequisite to developing mature coding methods in this domain.

We look to statistical machine translation to order our stego options. We wish to model $\Pr(e|f)$, the probability that a target sentence e (commonly denoted for 'English') is a translation of a source sentence f (for 'French', or 'foreign'). In the steganography domain, the target sentence is the stego text (s), and the source is the original cover text (c), so we will write this probability as $\Pr(s|c)$. Modelling this

probability directly is difficult [Brown et al., 1993], so instead we apply Bayes' law,

$$\Pr(s|c) = \frac{\Pr(c|s)\Pr(s)}{\Pr(c)}. \tag{3.4}$$

The denominator, $\Pr(c)$, is constant for any given cover, so we can ignore it when using the probability to rank the options:

$$\Pr(s|c) \propto \Pr(c|s)\Pr(s). \tag{3.5}$$

$\Pr(c|s)$ is known as the *translation model probability*. These are dependent on the transformation source, and here are provided by the PPDB in the features of each paraphrase rule. The features are given as log likelihoods, so we calculate this for the full stego tweet by summing the values for each substitution made. This assumes all substitutions are independent, which is an approximation. When a word has no entry in the PPDB, the log probability defaults to 0; when a word or phrase has an entry, but has been left unchanged, the PPDB provides identity probabilities (the probability of the phrase translating to itself).

The second factor $\Pr(s)$ is the *language model probability*. We want to model the language of our stego objects, which should be as close to that of the cover source as possible; when the distribution of stego objects is identical to the distribution of cover objects, the stego is undetectable. Language models have been previously used in a number of linguistic watermarking systems (e.g. [Topkara et al., 2006b]), but have not seen much application in the steganography domain. Where models have been used for information hiding, the models have been general English models. For a system with a specific cover source like Twitter, this is not good enough.

3.5.1 Statistical Language Modelling

CoverTweet uses a common type of language model known as an n-gram model. We want to model the probability of a tweet s, made up of a sequence of tokens $s_1, \ldots, s_k$ ($\Pr(s_i, \ldots, s_k)$). We can decompose this probability using the chain rule, giving:

$$\Pr(s_i, \ldots, s_k) = \Pr(s_1) \prod_{i=2}^{k} \Pr(s_i|s_1, \ldots, s_{i-1}). \tag{3.6}$$

It is simply not practical to know all the conditional probabilities; we have to limit the *history* (the conditioning context $s_1, \ldots, s_{i-1}$) to only a few words. For example, when the history is limited to a single word, we approximate the probability as:

$$\Pr(s_1, \ldots, s_k) \approx \Pr(s_1) \prod_{i=2}^{k} \Pr(s_i|s_{i-1}). \tag{3.7}$$

Models that provide estimates of $\Pr(s_i|s_{i-1})$ are known as *bigram* models; they provide probabilities for pairs of words (bigrams) appearing in sequence. For a sequence of words of arbitrary length n (known as an n-gram), these are called n-gram language models, where $n-1$ is the number of words used for the history. A general formula for these models is:

$$\Pr(s_1, \ldots, s_k) \approx \Pr(s_1) \prod_{i=2}^{n} \Pr(s_i|s_{i-1}, \ldots, s_1) \prod_{i=n}^{k} \Pr(s_i|s_{i-1}, \ldots, s_{i-n+1}). \tag{3.8}$$

Readers more familiar with steganography than computational linguistics will likely recognise these as $(n-1)^{\text{th}}$ order Markov models.

In order to approximate the probability of each n-gram, $\Pr(s_i \mid s_{i-1}, \ldots, s_{i-n+1})$, we can use maximum likelihood estimation (MLE):

$$\Pr_{MLE}(s_i|s_{i-1}, \ldots, s_{i-n+1}) = \frac{\text{count}(s_{i-n+1}, \ldots, s_i)}{\text{count}(s_{i-n+1}, \ldots, s_{i-1})}, \tag{3.9}$$

where $\text{count}(s_{i-n+1}, \ldots, s_i)$ is the number of times the sequence of words $(s_{i-n+1}, \ldots, s_i)$ has been seen in training text. For example, for the bigram ($n = 2$) model,

$$\Pr(s_1|s_0) = \frac{\text{count}(s_0, s_1)}{\text{count}(s_0)}. \tag{3.10}$$

Smoothing

Using the MLE for approximating n-gram probabilities results in significant over-fitting to the training data. We can never hope to have enough data to adequately approximate the probability for all n-grams; in practice, there will always be n-grams that are unseen in the training data. With the MLE, these unseen n-grams are given probabilities of 0, which is almost certainly unrealistic. The standard approach to this over-fitting is to apply *smoothing*, adjusting the counts of each n-gram in order to estimate the probabilities of unseen n-grams.

A number of smoothing techniques exist [Chen and Goodman, 1999]. The simplest approach is *add-one* or *additive* smoothing, where we add 1 (or some other constant δ) to every count. For a bigram model, adding one gives:

$$\Pr(s_2|s_1) = \frac{1 + \text{count}(s_1, s_2)}{|V| + \text{count}(s_1)}, \tag{3.11}$$

where $|V|$ is the size of some given vocabulary V (the set of possible words). This method is crude, and performs very poorly in practice [Gale and Church, 1994].

An example of a more sophisticated method is that of Good-Turing smoothing [Good, 1953]. For an n-gram seen r times, we adjust the count to r^*:

$$r^* = (r+1)\frac{n_{r+1}}{n_r}, \tag{3.12}$$

where n_r is the number of n-grams that are seen r times in the training data. The probabilities are then estimated with:

$$\Pr(s_i|s_{i-1},\ldots,s_{i-n+1}) = \frac{r^*}{\sum_{r=0}^{\infty} n_r r^*}, \tag{3.13}$$

for an n-gram $s_{i-n+1},\ldots,s_i$ that has been seen r times. This reallocates the probability mass for n-grams seen $r+1$ times to those seen r times; giving us an estimate for those seen 0 times. Good-Turing has a number of issues (including when n_{r+1} is 0), but is the basis upon which newer methods are built.

Most smoothing methods combine information from higher order and lower order models. *Interpolated* methods include information from lower-order models, even for n-grams with non-zero counts. For example, a bigram interpolated model:

$$\Pr_{interpolated}(s_i|s_{i-1}) = \lambda \Pr_{MLE}(s_i|s_{i-1}) + (1-\lambda)\Pr_{MLE}(s_i), \tag{3.14}$$

where λ is an interpolation weight. For general n-gram models we recursively interpolate n^{th} order and $(n-1)^{\text{th}}$ order models, grounding the recursion with a uniform 0^{th} order model. Design of these smoothing methods focuses on appropriate values for λ, ideally making the weight dependent on the n-gram.

By contrast, *back-off* methods fall back on the lower-order estimates when the higher-order n-gram has a zero count. For example, when estimating the probability $\Pr(\texttt{dogs}|\texttt{love},\texttt{i})$, if the trigram "`i love dogs`" was not seen in the training data, the language model returns the bigram probability $\Pr(\texttt{dogs}|\texttt{love})$ instead. If this bigram was also not observed, it returns the unigram probability $\Pr(\texttt{dogs})$.

We will be using modified Kneser-Ney smoothing [Kneser and Ney, 1995], a commonly used method which incorporates back-off and interpolation with good performance [Chen and Goodman, 1999]. When estimating the probability of an unseen n-gram $\Pr(s_i|s_{i-1},\ldots,s_{i-n+1})$, Kneser-Ney takes into account the number of unique contexts the word s_i has been observed in. Intuitively, words which have appeared in many different contexts are more likely to appear in a new (unseen) context, and words observed in few unique contexts are unlikely to appear in a new one. For example, `Francisco` may have been observed in the training data many times, but

only ever following `San`. The probability of `Francisco` following another word in an unseen n-gram should be low.

We do not believe the choice of smoothing method will have much of an effect in this setting, given we intend to use the steganographer to choose the final steganographic output. As a result, we have chosen to omit the full technical details of Kneser-Ney smoothing, which is relatively complex.

Evaluating Language Models

We can evaluate the performance of a language model on test data. An English language model should assign a high probability to real English text; here, a Twitter language model should assign high probabilities to real tweets. A common metric for measuring how well a language model fits testing data is *perplexity*:

$$\text{perplexity}(W) = 2^{H(W)}, \tag{3.15}$$

where W is test data containing n words, and $H(W)$ is the estimate of cross entropy

$$H(W) = -\frac{1}{n} \log \Pr(W). \tag{3.16}$$

3.5.2 Implementation

From our Twitter corpus, we held back data for 10 users with between 500 and 1000 tweets (further selection criteria for these users is described in Section 3.7), for use in experimental evaluation. We were left with approximately 75m tweets from the remaining users.

We used the SRI Language Modelling Toolkit (SRILM) [Stolcke, 2002] to train a model on these remaining tweets. For training, we set n to 5, and the type of smoothing to modified Kneser-Ney.

Already noted, we should be attempting to model the cover source: a general English model is not suitable for the setting, which is the reason for training ours on Twitter data. However, we should arguably be attempting to model the language of the specific user who is embedding the data. This introduces the problem of *domain adaptation*: in-domain data (tweets from the steganographer Alice) is scarce and expensive; general data (other tweets) is easy to obtain. Extensive exploration of this problem is beyond the scope of this work, but CoverTweet makes a small step towards personalised models for steganography.

We split the data from each held back user into three: the set of covers, for evaluation, containing the 100 most recent tweets; a training set, containing 90% of

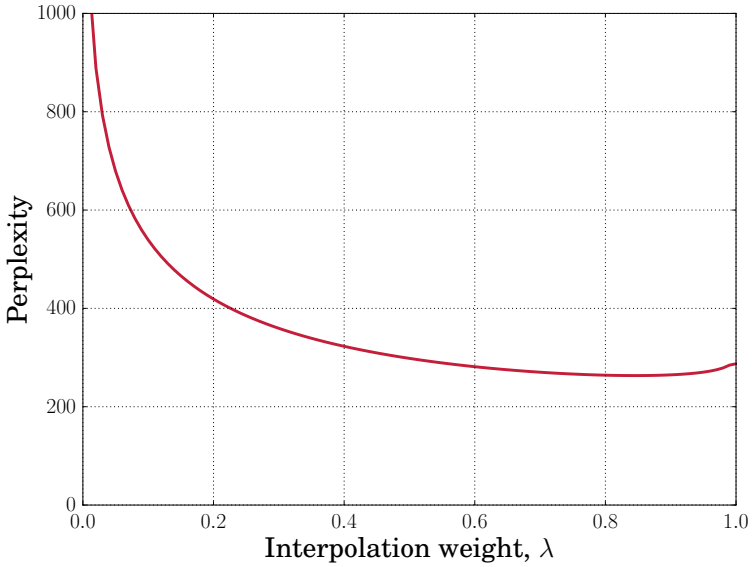


Figure 3.14: Average perplexity of each user's language model, as λ varies.

the other tweets; a tuning set, containing the remaining 10%. Small language models were trained on the middle data set, using SRILM. As we would expect, this set of data is too sparse for smoothing to help; the resulting models are poor. However, we can linearly interpolate the small personal model with the large general model:

$$\Pr(s) = (1 - \lambda) \Pr_{\text{general}}(s) + \lambda \Pr_{\text{source}}(s), \tag{3.17}$$

where λ is the interpolation weight, and $\Pr_{\text{general}}$ and $\Pr_{\text{source}}$ are the probabilities provided by the general and cover source language models respectively.

Figure 3.14 shows the averaged perplexity of these language model on the tuning set for each user, as λ varies. We see that the interpolated models have a slightly better fit (lower perplexity on the tuning set) than the general model, though the average interpolation weight that gave the lowest perplexity was higher than expected. This is most likely due to the very small number of tweets for each user. We do not expect this small reduction in perplexity to have any significant effect on security: especially not when we are performing manual filtering, and not when steganographic tweets are being judged individually as they are here.

3.6 Human Filtering

So far, we have generated steganographic options containing a desired payload, then ordered these according to a probability estimate. The final stage is to make use

of the human, in order to correct any mistakes made during the previous steps; the ultimate goal is for the human to select what they regard as the *best* stego tweet. There are two types of mistake: transformation, and ordering errors.

In the transformation step, poor substitutions may have been used to generate the stego options, meaning options need to be discarded by the human. The PPDB contains some poor quality substitutions, including a number of examples of foreign language words being used as substitutions for English words (e.g. `ecole` for `school`); we expect any instance featuring one of these will be removed by the human, although these should hopefully have a low probability anyway. In addition, PPDB rules have constituent tags, describing which part of a sentence the rule applies to. Many words have rules for multiple types of constituent part. For example, `cut` can be substituted for `haircut` when it appears as a noun, or `wound` when it appears as a verb. We are guaranteed a certain number of incorrectly applied rules, as CoverTweet makes no attempt to check constituent parts before applying them. This is by design: we are assuming the human can spot these mistakes, and we could otherwise expect to lose an amount of capacity to paraphrase rules incorrectly *not* applied.

In the ordering step, less fluent options may have been ranked above more fluent ones. We cannot expect the human to always fix this, as it is unlikely they will read every stego option.

CoverTweet gives the human access to two commands to speed up this task: the user can either remove all options containing a certain word or phrase, quickly removing all options with a bad substitution, or remove all options *without* a certain word or phrase, for cases where the original cannot be changed (e.g. proper nouns). The system shows the user 20 tweets at a time, in order of probability, and the user can request to see additional sets of 20 tweets. If there are no acceptable options, the user can indicate this, and CoverTweet will give up on embedding in the current tweet.

Figure 3.15 shows CoverTweet displaying the top 20 stego options for the input cover "`I don't like my hair cut`" (with a URL, which the system ignores). This contains a number of transformation errors: "`haircut cut`", "`doing n't like`", "`ai n't likes`". Figure 3.16 shows the list after the human has utilised the filtering commands to remove some of these. There are a number of options the human could choose, for example:

"`I don't really love my hair cut`"
"`I just do not like my hair cut`"
"`I don't really like my hair cut`"

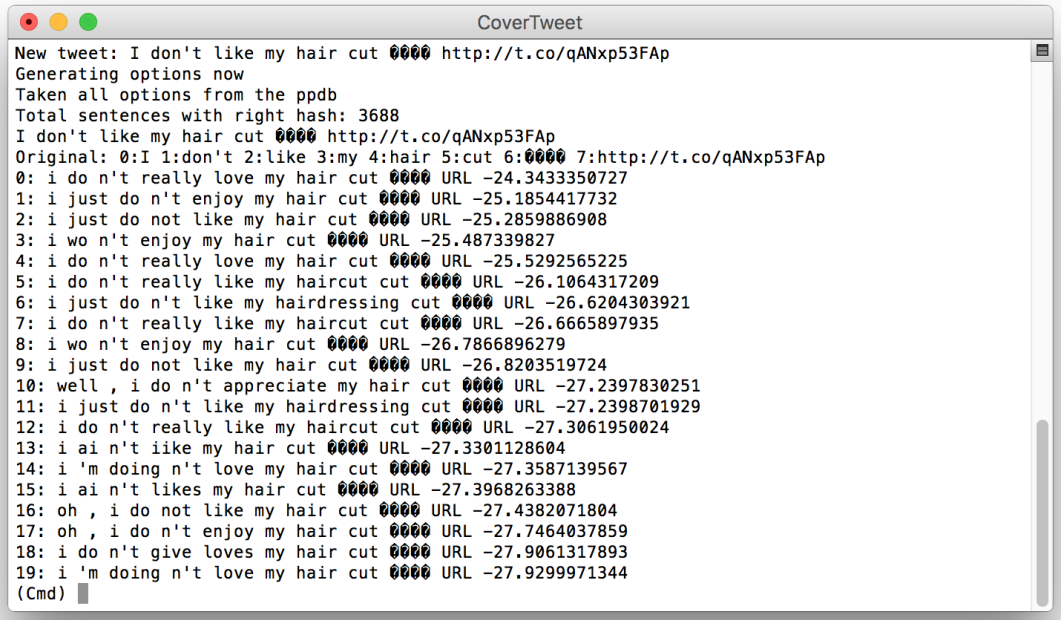


Figure 3.15: A picture of CoverTweet displaying the top 20 stego options for the given cover tweet.

Recalling our assumptions about the human, we can rely on them to choose text which is at least as fluent as the original, but not to be aware of how they are altering the distribution of their tweets with their choice. Given at least three technically fluent stego options, the human cannot be expected to know which is the most detectable, whether to a human censor (as in this chapter), or an automatic one (as we will see in Chapter 4).

This tweet had 3688 options for the desired hash, and it is not guaranteed that the language model will have ranked all the fluent options high enough that the human will ever read them. That said, we can see a clear overall reduction in quality as we look further down the list. Figure 3.17 shows the lowest ranked stego options, highlighting the importance of the ordering step.

3.7 Experimental Design and Results

From our Twitter corpus, we selected ten users at random, meeting the following criteria:

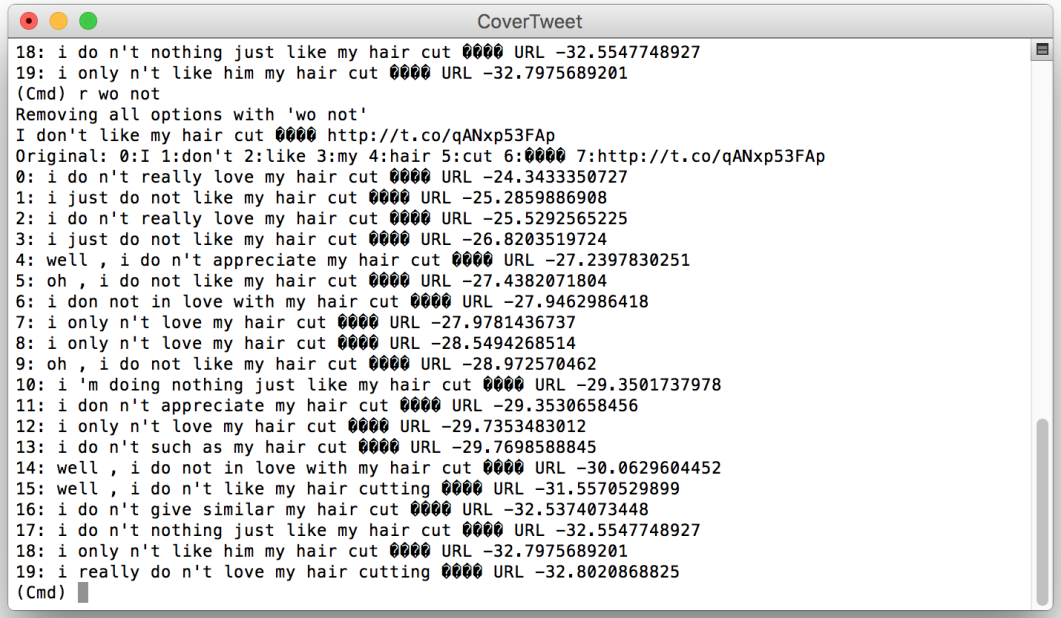


Figure 3.16: A picture of CoverTweet displaying stego options, which have been filtered by the human.

1. They had posted (during the month for which the corpus was collected) between 500 and 1000 tweets. This was sufficient to train and tune a language model and few enough to filter spam accounts.

2. Their average number of words per tweet was at least 9. This removed users who frequently posted messages too short to contain payload.

3. The size of their vocabulary per tweet was approximately equal to 8.5. This also removed spam accounts and collected 'typical' users.

When collecting tweets from users, any tweet containing `RT`, indicating that it was a re-tweet, was ignored.

As described in Section 3.5.2, we split the tweets for each user into three sets: a set of 100 cover tweets; 90% of the remaining tweets for training; the final 10% for tuning the interpolation weight. We trained individual language models for each user, and a general language model on the remaining 75m tweets. For each user, the optimal weight was found that minimised the perplexity of the interpolated model on the user's tuning set.

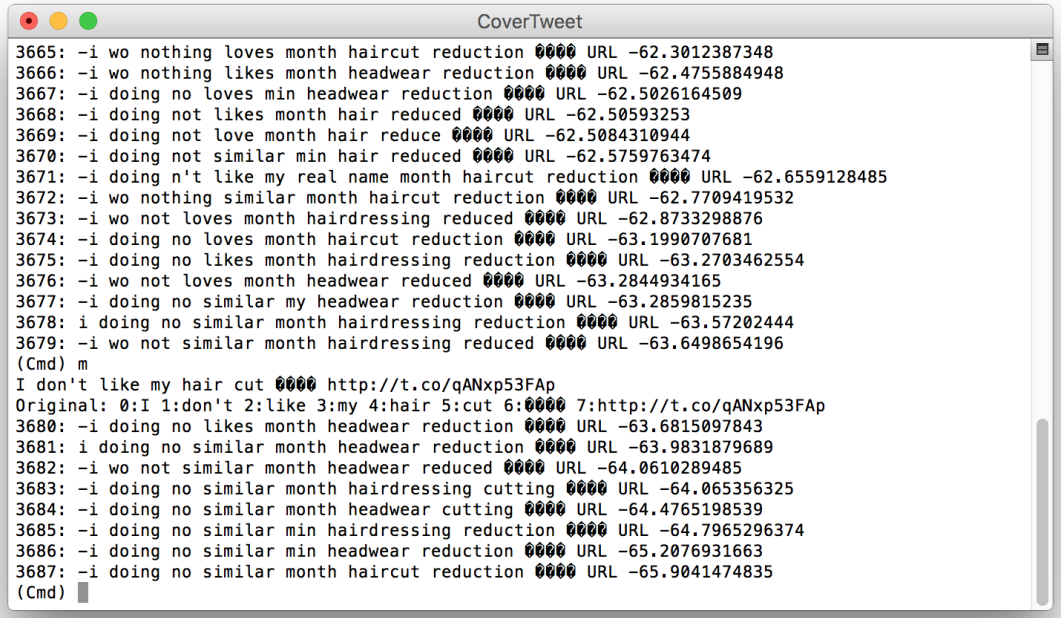


Figure 3.17: A picture of CoverTweet displaying the lowest ranked ranked stego options for the given cover tweet

3.7.1 Evaluation of Stegosystem Security

A goal of CoverTweet was to be secure against human wardens, while still embedding a reasonable amount of data. In order to evaluate how successfully the system achieved this goal, we designed an experiment that used human judges to attempt to spot stego tweets.

We used the CoverTweet system to generate 20 stego tweets containing 4 bits of randomly generated payload, for each of the 10 users. Subsequently we will denote the amount of payload in each tweet in *bits per tweet* or *bpt*; lowercase is used to match the convention of bits per pixel (bpp) from image steganography. We used the interpolated models for ordering, and the author substituted for Alice to perform filtering. The author's involvement in the data generation is an unfortunate necessity: filtering steganographic tweets is time consuming, and expensive, which ruled out the use of external humans. Though a potential cause of bias, this does at least guarantee consistency throughout the manual data (including the data generated in the subsequent chapters). Further discussion of this subject is found in Section 6.2.2.

4 bits is a significant increase in payload over previous systems (almost all hiding 1 bit per sentence, or less), however due to human filtering and our use of hash functions, sometimes the desired random payload was missing; in these cases, we skipped the tweet and moved on to the next. On average it took 50 tweets from the set of covers to find 20 usable tweets for each user. This should not be interpreted as a suggestion that each cover is only capable of embedding an average of 1.6 bits, as some tweets may be able to convey all symbols other than the desired one. We will calculate the average number of available options in Chapter 5, where we will be tackling the *non-shared selection channel* problem. This is the problem of Alice knowing more about where she can hide than Bob.

For the evaluation, we gathered twenty human judges, who were each given an exercise: having been shown 40 known 'innocent' tweets (taken from the training and tuning sets), they were presented with 20 unclassified tweets from a particular user. Of these unclassified tweets, 10 are stego (picked at random from the 20 stego tweets generated), and 10 are innocent: the task was for each judge to identify the 10 stego tweets. The innocent tweets were capable of embedding the 4 bit payload, so the judges were not able to simply choose the longer tweets, or tweets containing particular features that allow for easy changes (such as multiple usernames). Judges were asked to assign a score (1-10, to denote how sure they were about a particular tweet being stego) to the 10 tweets they felt were steganographic.

After assigning 10 scores, the judges were given another 40 innocent tweets, and were asked whether this affected their scores at all: this was to test whether having more history would affect the judge's ability to detect steganography. Each judge repeated this task for 5 of the users; the users were divided up so that each was seen by the same number of judges. We then measured the ability of the judges to distinguish cover and stego tweets, and whether this ability differed significantly between judges and between twitter users.

Figures 3.18 and 3.19 show part of a task presented to a judge; the former shows some of the innocent tweets from a given user, the latter shows the 10 steganographic and 10 innocent tweets which the judge is asked to classify. Figure 3.20 shows an example stego tweet with the original text above.

The judges were recruited from students and staff at Oxford University. Not all were computer scientists, and none were computational linguistics experts. They were aware of the general steganographic aim of the CoverTweet system, and briefly told how it worked; they did not see any contents of the PPDB, nor have access to the system itself. This mimics a real-world scenario where a human censor monitors

Twitter traffic without targeting a specific stegosystem. Appendix A includes the instructions given to each judge.

3.7.2 Evaluation Results

It turned out that the judges' performance was in no (statistically significant) way affected by seeing only 40 of the full 80 innocent tweet history of the users they were scoring. Therefore we include only the results for the latter, which is the more stringent testing environment for the security of our stegosystem.

The accuracy of each identification of the stego tweets, for each user individually, is displayed in Figure 3.21. The same results broken down by judge are in Figure 3.22. Overall, out of 2000 classifications, 1037 were correct and 963 were incorrect.

Overall accuracy of human judges

These 2000 answers are not entirely independent because the judges were told to identify exactly 10 guilty tweets out of 20 from each actor: we can account for this dependence by measuring their accuracy only on true guilty tweets (of necessity their true negative rate equals their true positive rate). Our judges collectively identified 515 out of 1000 guilty tweets correctly. A simple Z-test allows us to test the null hypothesis that judges are guessing randomly, verses a one-sided alternative that they are answering correctly more often. There is insufficient evidence to reject the null hypothesis of random guessing ($p = 0.180$).

Part of the security stems from the frequency of spelling and grammar mistakes in the cover tweets: this randomness was one of the desirable features of Twitter as a setting.

Effects on accuracy

We wish to know whether some judges were (statistically significantly) more accurate than others, whether some users' tweets were more or less difficult to classify, and also whether identification with higher confidence scores tended to be more accurate. All of these can be accomplished in one go using an analysis of variance, the results of which are displayed in Table 3.1.

We conclude that neither the identity of the judge, nor the confidence with which they make judgements, significantly affects the accuracy of classification. We also tested separately whether higher confidence was associated with correct answers, using

Innocent tweets:

1. your avi @user41 >>>>
2. middle finger up to my competition
3. im gone download me some emojis #thestruggle lml
4. @user67 the one you wore lastnight to the fashion show thing? lml.
5. i got the jokes tonight
6. phone sticky <<<.
7. she on time to get off.
8. tssk, the struggle is real she was not gone talk to her until she seen me tweeting her smh.
9. but anywaysssssss ..
10. get of my tl please ya.
11. fake? haha lets not go there hun.
12. you change alot i dont even no who you is anymore.
13. don't start something you can't finish.
14. i don't argue over petty stuff ill just start to distance myself away from you ..
15. """@user84: fake girls <<<<<<<<<<<<<<<<<<<<<<<<<<<< no can do"""
16. i need to call daija asap.
17. modeling classes ova the summer >>>
18. leave me alone ..... lol.
19. lml.
20. i no she telling my other cousin a whole different story but i could careless.
21. my circle geting smaller everyday, this thing is geting worse. im cuting em' off.
22. im going to get off that case and im gone dismiss it, anit nobody got time for that. it is what it is.
23. you must care lol .. hold up? why am i even talking to you or responding to you?
24. from what i heard today i dont even wanna talk to her no more!
25. i shall return .. bbl.

Figure 3.18: 25 of the 40 innocent tweets from one of the Twitter users used for evaluation.

10 steganographic and 10 innocent tweets:

1. soon as i got home im gone go sleep i had a long day!!
2. i need to buy me some new headphones..
3. would beilive me if i said im inlove? baby ..
4. *my mama bout me another new pair of beats but... it's not the headphones it the ear ones..*
5. *people still wash there air force 1s shoes, i take it some people didnt learn the lesson.... if you wash air force 1s they will turn yellow!! lol*
6. *yall must have 11:11 set 1 minute early before yall tweet it, because soon as 11:11 hit yall don't wastes no time. lol*
7. i sleep with my earings on but tonight i can't .
8. #oomf toes is ugly .... woooo jesus everytime i see them things i run, i sweaaa!
9. *@user270 lol.*
10. *@user341 i'm remembering that time in 7th grade when ashley b made us get that f on that project in science class, lmao we been to mad!*
11. *most of the people who got mouth can't beat you .*
12. #weallknowthatonepersonwho hate on you, don't like you, envy you, laugh in ya face and then talk about you!!! i know em'
13. why my family choose to come my house this late at night? i can't deal with all that laughing right now .. lol
14. feel like going with my cousin today .. i need a laugh .
15. """@user163: that awesome moment when you say something really funny and everyone laughs, so you just sit there like a boss."""
16. *you can tell when somebody hating on you!*
17. *mfs be thinking i gaf about whatever they think, when really i don't .*
18. *people say ice not good thing for you, yet little do they no ice is just frozen water. people need to think before they speak.*

Figure 3.19: An example of the tweets shown to the judges. The steganographic tweets have been highlighted.

									I had			
soon	as	I	got	home	im	gone	go	sleep	*I had kind of*	a	long	day

Figure 3.20: Stego tweets with the embedding changes highlighted; the original words are displayed above.

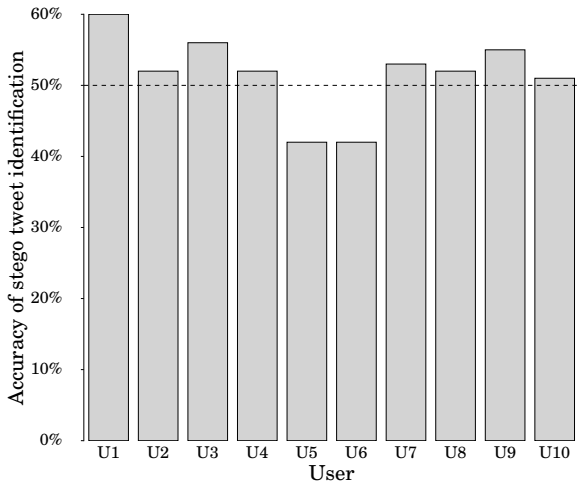


Figure 3.21: Accuracy of detection of stego tweets by human judges. Each user was scored on 200 tweets.

Factor	d.f.	deviance	p-value
Judge	19	14.1707	0.7737
User	8	19.6039	0.0112*
Confidence	1	0.5431	0.4612

Table 3.1: Results of variance analysis on our judge's scores.

a Wilcoxon rank sum test. This makes no assumptions about the distribution of the scores, but does use their ranking. Again no association was found ($p = 0.621$).

To investigate further the effect of the user, we performed a multivariate logistic regression, modelling the log odds of a correct judgement in terms of user and judge. We omit the regression table because only one factor was significantly different from 0: user 5, it turns out, was significantly more difficult to classify correctly than the rest.

We attribute this to the particular selection of unclassified tweets for this actor, though potentially exacerbated by a noticeable lack of consistency in the user's language. Two of actor 5's innocent tweets had features that a number of judges took as clear signs of guilt, but were just a product of this unpredictable nature. One tweet was part of a group of three tweets to the same two users, but the usernames of the recipients were in reverse order (see Figure 3.23). In this instance, the third tweet actually was stego, but the username order had not been changed: "`oh , i can't`" had been substituted for "`i can't`". Another tweet, shown in Figure 3.24, used both `you` and `u` within the same tweet. Again, this was taken as a sign of guilt, although in

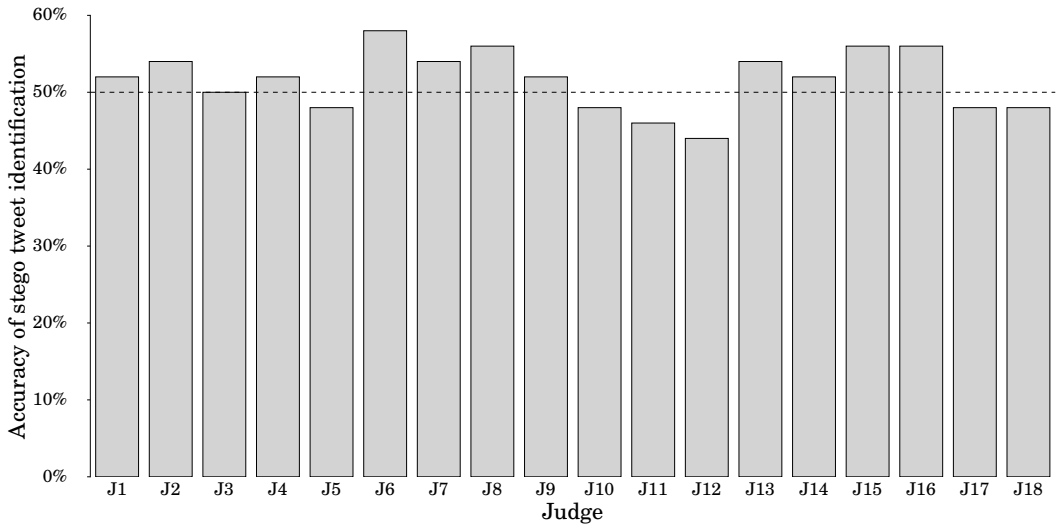


Figure 3.22: Accuracy of detection of stego tweets by human judges. Each judge scored a total of 100 tweets.

1. `@user123 @user48 not me!  i'm getting wastey pants`
2. `@user48 @user123 lol u guys can babysit me then for once!`
3. `@user123 @user48 lmao!  gee!  oh, i can't do nuthin *george lopez voice*`

Figure 3.23: Example tweets from Actor 5. The change in username order for the second tweet was taken as a sign of guilt, but it was in fact innocent.

fact the PPDB does not contain this substitution as a rule; arming the human judges with the PPDB would be of significant interest, though a large undertaking.

3.8 Summary

In this chapter we have described the construction of the stegosystem CoverTweet. This system has a number of novel (or previously unevaluated, security-wise) features: hash functions are used for value assignment, which allows for unrestricted transformations; lexical and phrasal substitutions provided by the Paraphrase Database are

`@user427 i know i 'm just teasing you, i have a lock on mine too , u just never know`

Figure 3.24: Example tweet from Actor 5. This was innocent, but had inconsistent usage of `you` and `u`.

used in combination; a human filters the output for fluency. Despite a record payload of 4 bits embedded in evaluation data, we found that steganographic tweets generated with CoverTweet were indistinguishable from innocent tweets to human judges.

Having established that humans are incapable of identifying individual tweets, we will move to the second adversary model: the automated censor. In Chapter 4 we will develop a machine learning based approach to the automatic detection of linguistic steganography.

Note that the CoverTweet system is not without problems: the amount of payload per tweet is uniform, despite tweets having vastly differing numbers of steganographic options. After filtering, it is common for tweets to not have an option for the desired payload; these tweets must be discarded. This is an issue for capacity, and in Chapter 4 we will evaluate whether this is also a security weakness when faced with an automated censor. In Chapter 5 we will look at solutions to the problem.

The uniform payload is one downside of using a hash function for value assignment: we no longer expect to know how much we can actually hide in a given tweet. Even if we could calculate this, and use hash values of the correct length, collisions cause an amount of lost capacity.

It is also important to note that hash functions do not come without some computational cost, as we have to construct a great deal of options to find the ones with the correct value. While the design of our hash function allows us to guide the construction somewhat, it may be infeasible to apply the technique to embedding in entire paragraphs, as opposed to independent tweets.

Chapter 4

Attacking The System

This is keeping an eye on you, child.

Casablanca
via CoverTweet

In Chapter 2 we summarised the previous work on linguistic steganalysis. This was a short discussion: there has been very little of it. There are a number of possible reasons for this. Firstly, the barrier for entry is somewhat high, with implementations of published systems unavailable and often too expensive or difficult to recreate; for example, when labelled data is required, as in [Chang and Clark, 2012]. Possibly as a result, almost all work has attacked the easily available (and *weak*) T-Lex [Winstein, 1999].

Further, many systems have either (a) produced steganography easily detectable to humans, or (b) leveraged a transformation deemed (prematurely, and likely incorrectly) undetectable; both scenarios mean the research effort is potentially written off as unnecessary. We certainly do not take this view. In the case of (a), we believe that the development and analysis of attacks against human-weak steganography are of vital interest to the field, particularly due to scenarios where human weakness is irrelevant, e.g. steganography in email spam [McKellar, 2000], or the twitter setting, when there are too many tweets for a censor to read. In the case of (b), claims of security are worthless without security analysis; it is very unlikely that an 'undetectable' transformation is anything of the sort.

Having verified the security of manually operated CoverTweet against human attackers, we are in a position to undertake the first serious linguistic steganalysis work. In this chapter we will apply the standard paradigm of image steganalysis to CoverTweet: extracting features from labelled data, and training a classifier to distinguish stego objects from covers. We will produce the first linguistic steganalysis

work that explicitly acknowledges Kerckhoffs' principle, and explicitly utilises the paradigm of *pooled steganalysis* (see the next section for a description of this).

In addition to attacking the manually filtered CoverTweet, we will attack an automatic version devoid of the human. For further comparison, we will also try the attack against T-Lex. Evaluation of other published systems would be desirable, but was ultimately beyond the scope of this work, due to the aforementioned cost of system reproduction.

4.1 Pooled Steganalysis

Our human judges in Chapter 3 were shown tweets one at a time, and asked to identify those they thought had been changed. This follows the traditional paradigm of steganalysis: the individual distinction of stego and cover objects. Individually however, tweets carry very little payload, especially when compared to digital media (such as images). When covers have low capacity, the steganographer requires multiple tweets to carry a reasonable payload; in addition, the *attacker* needs to see multiple tweets to be successful at an attack.

This brings us to a new paradigm: *pooled steganalysis*. Instead of identifying steganographic *objects*, we aim to identify guilty[1] *users*. First presented in [Ker, 2006], and further discussed in [Ker and Pevnỳ, 2014b, Pevnỳ and Nikolaev, 2015], pooled steganalysis has yet to receive successful research in the steganographic field [Ker et al., 2013].

We will gather evidence against each user, collating features across multiple tweets, and evaluate how the performance of our attack changes as we see more evidence. Ideally we would also test the performance of human judges against batches of tweets, but this is prohibitively expensive, especially compared to the experiments of the previous chapter. We would require more judges to read significantly more tweets. Consider that it took our judges approximately an hour to read and evaluate 100 individual tweets; when grouped into batches of 10, judges would have to read 1000 tweets in order to gather the same number of data points.

We can speculate on what human results would look like. Some features, undetected in single tweets, would potentially be noticeable to a human across multiple: high occurrences of unusual word choices, for example. However, we suspect humans

[1]Describing users as 'innocent' or 'guilty' has the potential to seem politically charged to readers outside the steganography field, but are in fact the standard terms used in steganalysis research. Consider that we are playing the role of warden, in a prison where steganography is against the rules. Steganographers are guilty of breaking those rules.

are incapable of spotting subtler trends across a large number of tweets, such as changes to the length and number of words, and irregular choice of punctuation.

The line between pooled and traditional steganalysis is slight in our Twitter setting, as it depends on what we regard as the linguistic steganographic channel. A steganographer could potentially have multiple Twitter accounts, each with a number of tweets: is the channel the individual tweets, or all usable tweets? Certain pieces of previous research (e.g. [Chen et al., 2011], which contains a poor evaluation of results) have attacked documents made up of many steganographic sentences; this obviously has parallels with attacking multiple tweets at once. Arguments about channel definition aside, as long as payload is spread out across multiple covers (tweets) or cover elements (sentences in a document), methods should be developed to attack these in unison.

In our adversary model presented in Chapter 1, we established that the adversary is a censor working to identify steganographic tweets. The high cost of performing pooled steganalysis experiments with human judges is mirrored by the practical costs of a human censor pooling evidence against Twitter users: upon receipt of a new tweet, such a censor would have to re-read older tweets from the same user in order to gather evidence. This is not a practical solution when considering the incredible volume of tweets posted every day. As the censor has failed to identify tweets one-by-one, they must change their tactics, and try machine learning based attacks instead. We suspect pooled steganalysis is only practically achievable by this sort of machine learning based detector.

4.2 Proposed features

We have designed a set of features for detecting the steganographic output of CoverTweet. These features come in three flavours: *basic*, simple statistical counts that may be changed through embedding; *probability*, features extracted using n-gram language models (such as those described in Chapter 3), and similar models for word length sequences; *paraphrase*, features extracted using CoverTweet's source of transformations.

This last set takes advantage of Kerckhoffs' principle, and the knowledge we (as a Kerckhoffs' attacker) know about the system. Certainly any such attacker must know the transformation mechanism (e.g. synonym substitution, paraphrasing, adjective deletion), but if a stegosystem relies on a source of data (as many, including CoverTweet, do), then we must also assume the attacker has access to this same

Cover: `i 'm going to stay in bed and never ever come out ever again .`
Stego: *`i just wanna`* `stay in bed and` *`never , never`* `come` *`outside`* `ever again` *`...`*

Figure 4.1: A cover tweet and the stego tweet produced by CoverTweet, with human filtering.

data source. To the best of our knowledge, this work is the first to explicitly apply Kerckhoffs' principle to linguistic steganalysis.

Below are full descriptions of the features, including the motivation behind them. In Section 4.4 we will evaluate the performance of each set of features (individually, and in combination) against both automatically generated and manually generated stego tweets.

Previous work ([Xin-guang et al., 2006],[Yu et al.,],[Chen et al., 2011], and [Xiang et al., 2014]) has focused on a single feature, in a way reminiscent of early image steganalysis (e.g. [Westfeld and Pfitzmann, 1999]). In designing these features we took inspiration from the types of features utilised in modern image steganalysis.

Our basic and paraphrase features contain a number of counts and heuristic measures: similar to those found in the PEV274 feature set for JPEG steganalysis [Pevnỳ and Fridrich, 2008]. Our probability features also have significant similarities with the PEV274 set, and the SPAM set [Pevnỳ et al., 2010a]. Both sets contain features derived by modelling sequences as Markov processes: adjacent pixels in the former, adjacent DCT coefficients (for JPEG steganalysis) in the latter. We will model sequences of words (n-grams) and word lengths. These probabilities also have parallels with features utilising *residuals* in image steganalysis [Ker, 2008]. Residuals measure the difference between an observed stego object and an estimated cover; large residuals correspond to unlikely images.

As a running example, we will compare the features of the cover and stego tweet pair shown in Figure 4.1.

4.2.1 Basic: 130 features

These are the most basic features; a collection of statistics extracted from individual tweets.

Word count The number of tokens in the tweet, k.

Word length The mean and variance of the number of characters in each token.

Stop word count The total number of stop words in the tweet; that is, common, short function words such as `is`, `the` and `it`. The list of stop words is taken from a set provided by the *Natural Language Toolkit* (NLTK) [Bird, 2006], which contains 127 such words. The full list can be found in Table 4.1.

Individual stop word counts For each individual stop word in the list, we use the number of times they occur in the tweet as features.

These features are designed to capitalise on a number of rules within the PPDB which insert, or remove, words from phrases. Stop words are particularly common additions. One example rule replaces the phrase "`I like`" with "`I just like`". In our manually generated data containing 4 bpt (described in Section 4.3) we observed that the average number of words in cover tweets increased fractionally from 10.7 to 11.1 in stego tweets, and the average stop word counts from 4.12 to 4.22. This difference in values does not tell the entire story: the average number of some individual stop words is reduced during embedding (e.g. `have`, `but`); for some the number increases (e.g. `just`, `am`, `the`); others are unchanged (e.g. `been`).

Table 4.2 summarises the basic features for the example cover and stego pair, with the exception of the individual stop word counts. The majority of these counts are 0. For the cover, the counts for `I`, `to`, `in`, `and`, `out` and `again` are all 1. For the stego, the counts are 1 for `I`, `just`, `in`, `and` and `again`. Due to the substitution of `out` for `outside`, the stego tweet actually has fewer stop words than the original.

4.2.2 Probability features: 45 features

We use two types of probability estimates for features, the first using an n-gram language model, the second using a model of word lengths. Where the n-gram model provides probability estimates for a sequence of words (e.g. $\Pr(\texttt{like}|\texttt{I}, \texttt{just})$), the word length model gives estimates for sequences of character counts (e.g. $\Pr(4|1, 4)$).

n-gram features

For n from 1 up to $o + 1$, where o is the order of the given n-gram language model, the mean, variance and total log likelihood of the n-grams in the tweet.

word length features

For n from 1 up to $o + 1$, where o is the order of the given *word length* model, the mean, variance and total log likelihood of the word length sequences in the tweet.

i	me	my	myself	we
our	ours	ourselves	you	your
yours	yourself	yourselves	he	him
his	himself	she	her	hers
herself	it	its	itself	they
them	their	theirs	themselves	what
which	who	whom	this	that
these	those	am	is	are
was	were	be	been	being
have	has	had	having	do
does	did	doing	a	an
the	and	but	if	or
because	as	until	while	of
at	by	for	with	about
against	between	into	through	during
before	after	above	below	to
from	up	down	in	out
on	off	over	under	again
further	then	once	here	there
when	where	why	how	all
any	both	each	few	more
most	other	some	such	no
nor	not	only	own	same
so	than	too	very	s
t	can	will	just	don
should	now			

Table 4.1: List of stop words used to extract features, taken from the Natural Language Toolkit.

	Cover	Stego
Token count	15	15
Word length mean	3.2	3.7
Word length variance	1.76	2.46
Stop word count	6	5

Table 4.2: Basic features for the example tweets.

The impetus behind the former is clear: fluent users should prefer high probability sentences. In our manual data with 4 bpt, 91% of steganographic tweets have a lower total log likelihood than the corresponding cover (ignoring the stego objects which are unchanged from the cover).

The mean and variance are intended to capture any substitutions that are unusual in the context, with the total log likelihood intended to catch the fact the average probability is lower in steganographic tweets. These features are a subset of those used in [Taskiran et al., 2006]; that work differed in the use of multiple language models (trained with different parameters) to extract features. While the use of multiple models is certainly of interest, the full exploration of language model choice for steganalysis (along with how the performance of the attack changes according to model size and order) is left for further work. In experiments we use a single 5-gram model.

The word length features have the same reasoning as the n-gram set. When a substitution replaces a common word (with high probability) for a rare word (with low probability), on average the number of characters will increase; this is a natural feature of language. In addition, PPDB rules can replace single long words with multiple short, or the reverse. Changes to word length distributions might be noticeable, especially if a user has a bias towards short words, or a predilection for sesquipedalianism. The training of the word length model is described in Section 4.3. Because the vocabulary size for word lengths is far smaller than for words (for our Twitter setting it has an upper limit of 140), the model order was increased to 10.

Table 4.3 shows the unigram and 5-gram probability features for the example pair. All total log likelihoods (for n-gram and word length) are lower for the stego tweet, but the particular highlight is the 5-gram log likelihood variance, which is 1.09 for the cover and 8.68 for the stego. This large change can be attributed to the substitution of "`I'm going to`" with "`I just wanna`": the value of $\log \Pr(\texttt{wanna}|<\texttt{s}>, \texttt{I}, \texttt{just})$ is estimated as -12.74, while it is estimated as -1.8 for the original $\log \Pr(\texttt{going}|<\texttt{s}>, \texttt{I}, \texttt{'m})$ (note that the token $<\texttt{s}>$ denotes the start of the sentence).

Compared to n-gram probabilities, the difference between word length features was slight. This is to be expected, as the predicted effects of embedding on word length are far subtler. The largest difference between individual word length sequence probabilities is between $\log \Pr(3|3, 5, 4, 4)$ and $\log \Pr(7|5, 1, 5, 4)$ (-0.79 and -1.42, respectively) which correspond to the sequences ending in `out` and `outside`.

		Cover	Stego
n-gram	Total unigram log likelihood	-39.94	-49.14
	Unigram log likelihood mean	-2.66	-3.28
	Unigram log likelihood variance	0.39	4.19
	Total 5-gram log likelihood	-21.44	-39.45
	5-gram log likelihood mean	-1.43	-2.63
	5-gram log likelihood variance	1.09	8.68
word length	Total unigram log likelihood	-12.43	-13.33
	Unigram log likelihood mean	-0.83	-0.89
	Unigram log likelihood variance	0.02	0.03
	Total 5-gram log likelihood	-11.65	-13.34
	5-gram log likelihood mean	-0.78	-0.89
	5-gram log likelihood variance	0.04	0.04

Table 4.3: A sample of the probability features for the example tweets, using unigram and 5-gram probabilities. The full set includes probabilities for every size n-gram up to the order of the models used.

Even without the conditioning context, the probability for a word with 7 characters is less than one with 3: shorter words are more common than long.

4.2.3 Paraphrase features: 28 features

As noted, the paraphrase, or PPDB, features assume adherence to Kerckhoffs' principle, and make use of the attacker's access to the steganographer's data source: the Paraphrase Database. We expect these features to perform best in our attack against CoverTweet. By querying the PPDB we know exactly where data *could* be hidden (i.e. the words and phrases that have entries in the database): we exploit this knowledge by recalculating some of the basic and probability features, ignoring the n-grams that cannot have been changed during embedding.

We can also use the PPDB to apply CoverTweet's transformations, and generate all possible rewritten versions of a tweet. While the design of CoverTweet allows for asymmetric transformations, recall that the implementation uses only symmetric ones (due to the subset of the PPDB used). This means that, if a tweet is steganographic, one of the generated tweets *must* be the original cover. Along with the fact that 91% of the modified tweets are less likely than the original, our data also showed that 63% of cover tweets had no option of a higher probability, compared to 8% of stego tweets.

To aid definition, we will fix some notation. Let $\mathbf{t}$ be a tweet (which could be cover or stego), made of k tokens $\{t_0, \ldots, t_{k-1}\}$. The number of characters in the i^{th} token is given with $|t_i|$.

For convenience, $\mathbf{t}_{ij}$ denotes the sequence of words $\{t_i, \ldots, t_j\}$. Let $\mathbf{P}_{ij}$ denote the set of possible substitutions for the phrase $\mathbf{t}_{ij}$, and $\mathbf{P}_i$ as the union of all possible paraphrase rules for token t_i, taking into account the context:

$$\mathbf{P}_i = \bigcup_{l=0}^{k-1} \bigcup_{m=\max(0,i-l)}^{\min(i,k-l-1)} \mathbf{P}_{m(m+l)}. \tag{4.1}$$

Where we say a word t_i is 'in' the PPDB, we specifically mean that $|\mathbf{P}_i| \geq 1$.

Finally, $\mathbf{P_t}$ is the set of all possible rewritten versions of $\mathbf{t}$ produced by applying paraphrase rules in the PPDB; we generate this set in order to calculate certain features.

We make use of the *Iverson bracket* $[P]$ in definitions, which is defined as 1 if P is true, and 0 otherwise.

Proportion of PPDB entries

For phrase lengths $j = [1, 5]$, the proportion of j-grams in the tweet that have paraphrase rules in the PPDB. The upper limit of j is set somewhat arbitrarily, based on the fact that only 3% of PPDB rules are for phrases with more than 5 words.

$$\frac{\sum_{i=0}^{k-j}[|\mathbf{P}_{ij}| \geq 1]}{k-j}. \tag{4.2}$$

We also use the proportion of tokens in the tweet which are in the PPDB; this will increase whenever the PPDB introduces words to the tweet.

$$\frac{\sum_{i=0}^{k}[|\mathbf{P}_{i}| \geq 1]}{k}. \tag{4.3}$$

Word length

The mean and variance of the number of characters in each word that features in at least one rule in the PPDB. For example, the mean:

$$\frac{\sum_{i=0}^{k}|t_i|[|\mathbf{P}_{i}| \geq 1]}{\sum_{i=0}^{k}[|\mathbf{P}_{i}| \geq 1]}. \tag{4.4}$$

This feature also appears in the basic set, without the added PPDB knowledge. The inclusion of this feature with additional focus does not make the earlier one redundant, as we may be able to identify stego objects by comparing the two features.

n-gram probabilities for possible substitutions

The n-gram feature set, only taking into account the n-grams in the tweet that could have been modified by a paraphrase rule (that is, at least one word in the n-gram is in the PPDB).

Likelihood of most probable substitute This is the log likelihood of the most likely paraphrase, extracted by generating $\mathbf{P}_t$, then finding the most probable sentence in the set according to the language model. Once again, note that one sentence in $\mathbf{P}_t$ must be the original cover.

$$-\log(\max_{s \in \mathbf{P_t}} \Pr(s)). \tag{4.5}$$

Replacement score

One at a time, each word and phrase in the PPDB is replaced with its most likely substitution (according to the language model); a set **Substitution** is made of each of these sentences:

$$\mathbf{Substitution} = \bigcup_{i=0}^{k-1} \bigcup_{j=i}^{k-1} \{(\mathbf{t}_{0(i-1)}, x, \mathbf{t}_{(j+1)k}) | \arg\max_{x \in \mathbf{P}_{ij}} \Pr(\mathbf{t}_{0(i-1)}, x, \mathbf{t}_{(j+1)k})\}. \tag{4.6}$$

We calculate the log likelihood of each sentence in **Substitution**, and use the max, min, mean and variance of these.

Note that our attack here uses the exact subset of the PPDB used to generate experimental data. Left for further work is the exploration of how differences between the attacker's and the steganographer's transformation source might affect security. The idea of an enriched feature set using multiple transformation sources (and language models) is enticing, but beyond the scope of this work.

These features are specifically designed to work against CoverTweet; how might we expect them to work against another system? While they could be trivially modified to utilise another transformation source, this may not be strictly necessary. We can reasonably expect any source of sufficient size and quality to subsume smaller sources from other systems (excepting truly bad rules, which should obviously be discarded, and would hopefully be detectable for other reasons); we can think of these as approaching a platonic ideal of transformation source.

	Cover	Stego
Proportion of unigrams with substitution rules	0.67	0.73
Proportion of bigrams with substitution rules	0.64	0.57
Proportion of trigrams with substitution rules	0.23	0.15
Proportion of tokens included in any applicable substitution rule	1.0	0.93
Log likelihood of best substitution	-21.44	-21.44
Substitution score minimum	-21.44	-24.88
Substitution score maximum	-21.44	-39.45
Substitution score mean	-21.44	-37.66
Substitution score variance	0.0	19.58

Table 4.4: A sample of the PPDB features for the example tweets.

This is not the case with T-Lex however, the transformation source of which only shares 2472 rules with the PPDB (approximately 15% of all T-Lex rules). Despite this small proportion, we will see in Section 4.4 that the inclusion of PPDB features still improves the detection rate against steganography produced by T-Lex. Due to the large number of obscure words found within the T-Lex dictionary (e.g. `nitrofurantoin`, `hagiolatry`, `clinocephalism`), we speculate that the 15% of rules shared by the PPDB are the most likely T-Lex rules to ever be applied.

Table 4.4 shows a selection of these features for the example tweets. Not included are recalculations of probability features, and the mean and variance of word lengths with the added PPDB knowledge; in this instance, all words in the cover (and all but one in the stego) are in the PPDB, so there is little to no difference in the values. That the proportion of tokens in the PPDB went down during embedding is interesting, and shows that the feature is not monotonic with embedding. The cause in this instance is the phrase "`come out`". CoverTweet's subset of the PPDB contains a rule for substituting `out` for outside, which has been applied during the embedding. There are substitution rules for "`come out`", but none for "`come outside`", nor for `come` on its own. As a result, `come` is paraphrasable in the cover, but not in the stego.

The benefit of having access to the PPDB is made clear by the substitution scores and the log likelihood of the best substitution. For the original cover, there is no alternative sentence of greater probability: the log likelihood of best substitution is the log likelihood of itself, and the substitution score variance is 0. For the stego, the best substitution is (unsurprisingly, but not inevitably) the cover; the substitution score variance is 19.58.

4.2.4 Pooling the features

Defining our features is the first step; we are now in a position to perform traditional steganalysis. To do this, we can train a classifier on features extracted from individual tweets. Before we can perform *pooled* steganalysis, we must define the way we will gather evidence from multiple tweets. The best method for doing this is an open question in steganography [Ker et al., 2013].

For each user, we will pool together batches of tweets, for a range of batch sizes. There are three ways in which we expect features for steganographic tweets to differ from their corresponding cover: some features will tend to extreme values in certain steganographic tweets (such as the substitution score for particularly dubious tweets); some features will have small consistent changes between steganographic and innocent tweets (such as the probability features); finally, some features will become inconsistent for steganographic tweets (such as word length variance). To capitalise on each of these, we combine the individual features within the batches in four ways.

To help with detection when a feature takes an extreme value, we take the maximum and minimum of the features. We take the mean to help detection with features that change consistently. Finally, we take the variance to help with inconsistent features.

Before pooling, we have 203 features. After, we have a final count of 812 features per batch.

4.3 Experimental Design and Resources

4.3.1 Data

Once again, the tweets gathered from the Harvard TweetMap will be used in the experiments (see Section 3.7). In the earlier chapter we selected 10 users with certain characteristics, and generated (with manual filtering) 20 steganographic tweets for each. Here we will add to that set, and select a further 990 users for a total of 1000.

A new language model was trained (with the previous parameters) on the remaining users. Originally the paraphrase options were ordered, before human filtering, by a large language model interpolated with a smaller personal one. Given the fairly small decrease of perplexity observed when using the interpolated models, we will eschew the interpolation and use only the large language model for ordering.

Manual

For 10 of the 1000 users, we employed CoverTweet to generate 300 steganographic tweets with manual filtering: 100 each for three sizes of payload. Whereas before we embedded only 4 bpt, here we embedded 1 bit, 2 bits and 4 bits. Apart from the sizes of payload, and the choice of language model, the generation of steganographic options was otherwise the same as in the previous experiment (see Section 3.7). We will refer to this data as *Manual CoverTweet* (M-CT).

Recall that we assign each tweet a uniform length hash value, and tweets may be missing values for particular payloads (especially when they have few steganographic options). When a tweet is missing an option for the desired value, we refer to them as 'unusable' (though noting that they may have options for other payload values); we must consider how to handle unusable tweets in the evaluation of our attack. There are multiple scenarios to consider, none perfectly realistic: one supposes a naive steganographer, the others either an impossibly resourceful steganographer or attacker.

In the case of the naive steganographer, tweets unable to convey the desired payload are discarded, as the receiver would have no way to distinguish these tweets from those hiding information. From the view of training a classifier, we remove these tweets from the stego data sets, but they remain in the cover sets. This is a potential weakness of the system: for example, very few tweets with no options at all will appear in the stego sets, but will appear freely in the cover sets. This behaviour is representative of this stage in development, as we have not yet discussed alternatives to discarding unusable tweets; we can state however that alternatives do exist (Chapter 5 is dedicated to them), so it is potentially preferable to assume a sophisticated steganographer would be using a better solution.

An alternative is to simulate a situation where the steganographer can send these unusable tweets. We are pretending that the sender and receiver have some technology which signals the tweets capable of hiding the payload. The steganographer can transmit them as *padding*, and the receiver ignores them when extracting the message. As a Kerckhoffs' attacker, we likely also have this technology: if we do, we can strip the padding from both the cover and stego sets; if we do not, it is left in both.

Note that this experimental approach is useful while we have not yet handled the *non-shared selection channel problem* (see Section 5.1), but is less than perfect. The hypothetical technology employed is pure fiction: it would require all parties to know both the original cover and the intended message, along with prescient knowledge of which options the human filter will remove.

In summary, we consider three scenarios:

Unpadded Alice must discard unusable tweets; they are removed from the stego data sets, but not the cover sets. We refer to this as M-CT-U.

Padded Alice sends unusable tweets as padding, and the attacker cannot identify them; the tweets are included in both stego and cover sets. We refer to this as M-CT-P.

Stripped Alice sends unusable tweets as padding, and the attacker can identify them; the tweets are removed from both stego and cover sets. We refer to this as M-CT-S.

For the manual data at 1 bpt, 11% of tweets needed to be discarded, implying an average payload across *all* tweets of 0.89 bpt; at 2 bpt, this average payload was 1.59; at 4 bpt, it was 1.9 bpt.

Automatic

The generation of this manually filtered data is time consuming and expensive, and the amount we have is not enough to perform a full and thorough evaluation of our attack. We therefore look to an alternative, cheaper source: we remove the human filter, and generate steganographic tweets by relying solely on the probability estimates to choose the best option. Given an input cover tweet, the system generates the list of potential stego tweets with the desired hash. This list is then ranked according to translation probability, as normal. Bypassing the human, the option with the highest probability is transmitted as stego. We call this mode of operation, and the resulting data set, *Automatic CoverTweet* (A-CT).

For each of our 1000 users, we generated 1000 tweets with Automatic CoverTweet, for each of the three payload sizes. Without the human filter, encountering unusable tweets is significantly less common. When embedding 4 bpt automatically it occurred for 33% of tweets, compared to 53% when embedding manually. As with M-CT, we will consider the same three scenarios: A-CT-U, where the stego set is unpadded, and the unusable tweets discarded; A-CT-P, where the classifier is trained and tested on data including the unusable tweets; A-CT-S, where the classifier is trained and tested on data excluding the unusable tweets.

This automatic data does not, of course, carry the same guarantee of fluency that our manual data has. By extension, it is not guaranteed to be secure against a human attacker. In fact, a cursory glance at the data, especially at high payloads, suggests it

is almost certainly weak to a knowledgeable human. Figure 4.2 shows some examples. Some are good ("`your whole style dated`" would likely pass as innocent, given the fluency of the rest of the tweets), but there are multiple egregious occurrences of `-i`, which the PPDB contains as a possible substitution for `i`. Another bad substitution is `rues` (the original is `streets`) and `gout` (the original is `drop`).

Despite this, the automatic data is indispensable. It allows us to evaluate fully how our attack performs against different sizes of payload, and varying amounts of pooled data. This will let us speculate how the attack might work if we had sufficient manual data.

We must also recognise that linguistic steganography cannot become reliant on the manual filter. It is a useful tool while computational linguistics cannot guarantee fluency, but ultimately the future of linguistic steganography (if it is to escape the confines of low capacity) lies in the improvement of automatic embedding and the gradual decrease in manual intervention. The evaluation of an attack against this A-CT data forms a base-line, with which we can judge future transformation and embedding methods.

T-Lex

To provide a comparison with another system, we also generated steganographic tweets with T-Lex's transformation source. Because of the diminished size of this source, we were limited in terms of how much data we could generate for the 1000 users, and the amount of payload we could embed.

For each tweet from the selected users, we applied the substitution rules from T-Lex to generate the possible steganographic options. A 1 bit hash was assigned to each of these, and the option with the desired randomly generated hash value was chosen. The unchanged original was output if it had the desired value, otherwise the output was chosen randomly from the set of possibilities (T-Lex does not order options with a language model). Unlike with the CoverTweet data, we discarded any tweet that could not be changed by T-Lex's transformation source, even if it had the correct hash without changing. This was done due to the high percentage of tweets without options (78% of tweets); had we kept tweets with the correct random hash, but without any steganographic options, it is conceivable that we would have ended up with a set of entirely unchanged tweets[2] A tweet without any options has a

[2]Whether a set of entirely unchanged tweets is detectable is an interesting question (and puts us purely in the realm of cover selection), but not one we are studying here. If we exclude the possibility of exhaustion attacks, then certainly the answer is 'no' while we are assuming independence between tweets, but such independence is unlikely in reality. Note that if the attacker can recognise when

1. USER lol we was juss *talkin'* about you
2. USER he asked if your phone had money on it , and he said he 'll be out in 6 months hopefully
3. *anytime* i come around
4. opm >>>
5. USER : long *alive* them cali players
6. your whole *style* dated
7. number 8 jordon *is* unlaced wit the two straps
8. *-i* know you go slow when you tryna get me sprung
9. *-i* wan na watch you shop
10. i bet you ever knew that
11. tell yo baby mama what it do
12. before thy negative energy curve imma cut you off
13. deep inside my mind are buried *offences* you ca n't imagine
14. guns are the ones that talk now
15. never let anything off track you from your goals
16. never heard of you , i bet you heard of me
17. *gout*
18. summertime during the lbc gets hectic
19. got ta stay focused , do n't let these *rues* distract you
20. USER lmao

Figure 4.2: A sample of A-CT data, containing 1 bpt. The changes made during embedding are highlighted.

	T-Lex	Manual	Automatic
Number of users	1000	10	1000
Tweets per user	∼153	100	1000
Payload sizes	1 bit	1, 2 and 4 bits	
Batch sizes	2, 5, 10, 50, 100		+200, 500, 1000

Table 4.5: Summary of generated steganographic data.

capacity of zero; it is arguably correct that such tweets should be discarded, even for CoverTweet data (we will discuss this a bit more in Section 4.4.2, and significantly more in Chapter 5).

After discarding unusable tweets, we ended up with ∼154k steganographic tweets for our 1000 users, an average of 154 per user. This equates to approximately an average of 0.15 bpt.

4.3.2 Feature Extraction

For extraction of features, we use the same language model as CoverTweet uses to generate the tweets: a 5-gram model trained on approximately 72m tweets (all tweets apart from those by the 1000 held-back users), with Kneser-Ney smoothing. For vocabulary this model knows every word in the PPDB (and the T-Lex transformation set), every word in the training set, and every word used by the reserved users. We do not expect this to provide an unrealistic advantage to the censor, as any word not in the PPDB cannot hide information.

For the word length model, we trained a 10-gram model on the same data, but with each word replaced by its character count. The increase to 10-gram was due to the vastly reduced vocabulary count. For simplicity, we used the same smoothing technique, although there are likely better options for the word length domain. Design and selection of optimal word length models for the task was not in the scope of this work.

After generating features for all individual tweets, we pooled them in batches. For each data set, and each user, we divided the feature instances up into batches of varying size: from 2, up to the maximum possible for the set, which is 100 for M-CT, 100 for T-Lex, and 1000 for A-CT. The maximum, minimum, mean and variance was taken for each feature in the batch. Table 4.5 summarises the data, and includes all

they have extracted the correct message (for example, if the embedded message is unencrypted text), then they can search for stego by trying each extraction key, if such an endeavor is computationally feasible.

the batch sizes used in experiments. We limited the batch size to 100 for T-Lex data, given the variance in number of steganographic tweets we had for each user.

For the experiments where we include unusable tweets in the training and testing data, we must consider how we go about this. For individual tweets, we can simply append the unusable tweets to the cover sets (for unpadded), or to both cover and stego sets (for padded). When creating batches for the padded experiments, we randomly disperse the unusable tweets across the cover and stego sets, inserting the same unusable tweets into both sets, in the same locations. For the unpadded batches, where the unusable tweets are only included in the cover sets, this is not a viable option. Dispersing the tweets throughout just the cover set would result in unsynchronised cover and stego batches (the tweets contained in batches will be different for cover and stego data), which is thought to be a problem when training classifiers for steganalysis [Kodovsky et al., 2012]. To create the unpadded cover batches, we randomly replaced tweets in the cover data with unusable tweets, maintaining the correct proportion of unusable tweets to cover.

4.3.3 Classifier

In the experiments, we use an ensemble linear classifier made specifically for the sorts of large feature set used in steganography, and designed for speed [Kodovsky et al., 2012]. Prior work has found that linear classifiers achieve a similar performance to non-linear classifiers (e.g. Gaussian SVMs) given sufficient real world data [Lubenko, 2013]. While this result is based on image steganalysis, the focus of this thesis is not on machine learning; the effect of classifier choice on linguistic steganalysis is left for future work.

The ensemble consists of multiple Fisher Linear Discriminant (FLD) base learners, each trained independently on a randomly chosen subset of the features. Bootstrap aggregating (or *bagging*) is used: each base learner is trained on a data set generated by drawing from the full training data, with replacement. The decision threshold of each base learner is set to minimise the *equal prior error*, P_E, on the unused training data, defined as

$$P_E = \frac{FP + FN}{2}, \tag{4.7}$$

where FP is the false positive rate, and FN the false negative rate. When classifying an unseen object, the classifier aggregates the decisions of each individual base learner.

We will use the equal prior error on testing data as our main performance metric in experiments. Some literature finds a threshold on the testing data which minimises

this error; this requires unfair knowledge of the testing data however, so we will not repeat this.

4.3.4 Experimental Design

We have a wealth of users for the A-CT and T-Lex data, so we will split them: train the classifier on tweets from one half of the users, test the classifier on tweets from the other. In this way we will only be classifying batches of tweets by users for whom the classifier had no prior knowledge. We will repeat each experiment with different random user splits: all error rates will be averaged over 10 splits.

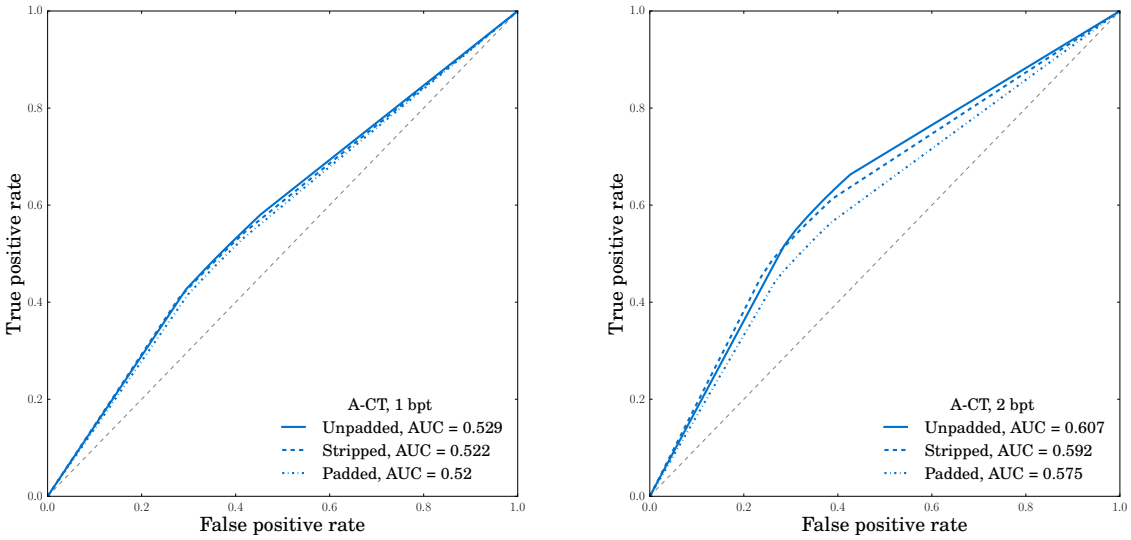
For the M-CT data, where we only have 10 users, we will perform 10-fold cross validation. The classifier will be trained on 9 users, and tested on the remaining one. This will be repeated 10 times, testing on a different user each time: the error rates will be averaged over the 10 folds. We have a great deal of additional covers (from the A-CT data) with which we could augment the training and testing data. However, Automatic CoverTweet deemed far fewer tweets unusable than Manual CoverTweet, so we cannot do this for the unpadded or stripped experiments without biasing the results. We chose to leave these additional covers out of the padded experiments as well, to enable a fair comparison. We will make use of the additional covers in Chapter 5, where unusable tweets are not an issue.

4.4 Results

4.4.1 Classifying Individual Tweets

Initially we will look at the results before pooling is introduced, where the classifier has been trained on features extracted from single tweets. Figure 4.3 shows the ROCs for classifiers trained and tested on individual A-CT and T-Lex tweets; Figure 4.4 show the same for M-CT data. Each figure shows results for unpadded, padded, and stripped data. Equal prior error rates and AUCs for all data types and payload sizes are shown in Table 4.6.

In all cases, increasing payload size makes tweets more detectable (P_E decreases). This is no surprise, as the system has fewer options to choose from at higher rates of payload, and must make do with options which are, on average, poorer quality. In addition, the chances of the cover having the desired hash decreases as payload size increases (1/2 for 1 bpt, 1/16 for 4 bpt), so more tweets will need to be modified to convey the message.



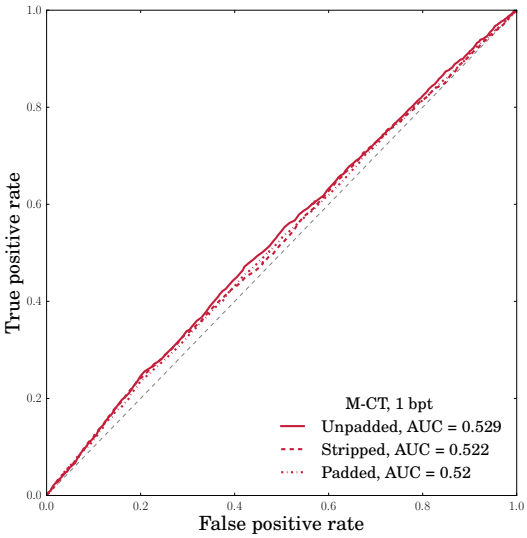
(a) A-CT, 1 bpt

(b) A-CT, 2 bpt

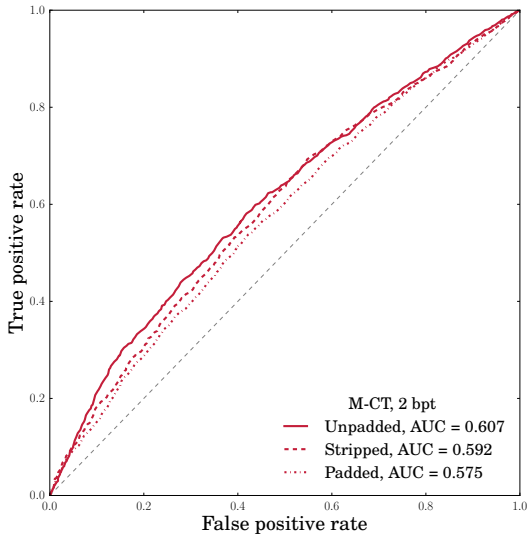
(c) A-CT, 4 bpt

(d) T-Lex, 1 bpt

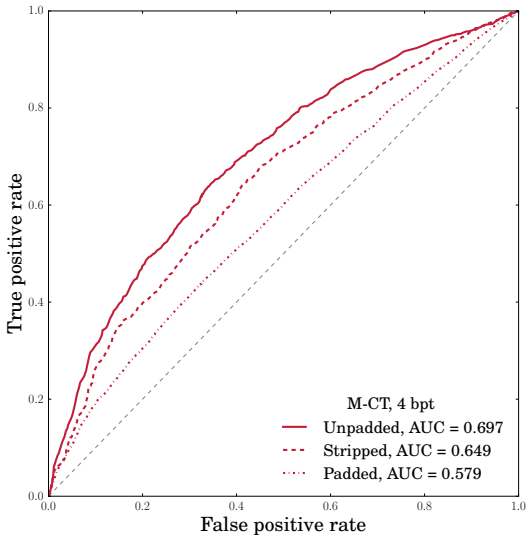
Figure 4.3: ROCs for classifiers trained on individual A-CT and T-Lex generated tweets, for each payload size.



(a) M-CT, 1 bpt



(b) M-CT, 2 bpt



(c) M-CT, 4 bpt

Figure 4.4: ROCs for classifiers trained on individual M-CT generated tweets, for each payload size.

At 1 bpt, there is no discernible difference between results for A-CT-S, A-CT-U and A-CT-P (stripped, unpadded and padded respectively). This is most likely due to the small number of discarded tweets at this size of payload (8% of the total), and the fact that half of the zero capacity tweets will contain the desired payload: there is very little difference between the three data sets. At larger payloads (where there are more discarded tweets), results for A-CT-S and A-CT-U remain similar, but A-CT-P is harder to detect. Individually, there is no way for the classifier to distinguish padding in the cover set and padding in the stego set, so P_E increases.

As with A-CT, there is no significant difference between the three types of M-CT data at 1 bpt; in this case, the classifier is unable to detect any of them. At larger payloads, detection increases, and differences between the types emerge. At 4 bpt, the unpadded data is the easiest to detect ($P_E = 0.36$), followed by stripped ($P_E = 0.39$), then finally padded ($P_E = 0.44$). That results for M-CT-S and M-CT-U differ (when they did not for A-CT) is likely because of the much larger number of tweets discarded in the generation of M-CT data.

The result for M-CT-S is of particular interest: data generated by the same method, containing the same payload size, with no discarded tweets, was undetected by human judges in the previous chapter. Our automated attack has an advantage in the form of explicit PPDB knowledge, but even without these features the classifier achieves a P_E of 0.42. We reiterate that it would be of interest (but ultimately not explored in this work) to see how arming a human judge with the PPDB would affect detection. It is worth noting that while the human filter is capable of removing human-detectable hallmarks of steganography, they are unaware of the majority of statistics our features are aiming to exploit.

The T-Lex data proved more detectable than CoverTweet data with comparable payload, though recall that the average payload across all tweets is far lower, given the number of tweets without any steganographic options in T-Lex. A-CT data (and unpadded M-CT data), containing 4 bpt proved easier to detect, but the attack is specifically designed to work against CoverTweet, something we will address in Section 4.4.3.

4.4.2 Pooled Steganalysis

As predicted, the performance of our attack against individual tweets was reasonably poor, even against the A-CT data. The steganographic signal is too weak to detect in such a small sample; this is especially the case when we are hiding only 1 bit. Figure 4.5 shows a comparison of ROCs for a classifier trained to identify individual

	bpt	Unpadded AUC	Unpadded P_E	Padded AUC	Padded P_E	Stripped AUC	Stripped P_E
A-CT	1	0.575	0.433	0.565	0.441	0.572	0.436
A-CT	2	0.635	0.379	0.599	0.409	0.625	0.384
A-CT	4	0.704	0.312	0.630	0.381	0.701	0.316
M-CT	1	0.529	0.483	0.520	0.486	0.522	0.490
M-CT	2	0.607	0.422	0.575	0.444	0.592	0.432
M-CT	4	0.697	0.357	0.579	0.441	0.649	0.395
T-Lex	1	Not Applicable				0.674	0.362

Table 4.6: Summary of AUCs and equal prior error rates for classifiers trained to identify individual tweets.

A-CT-U tweets (containing 1 bpt), versus one trained to identify batches of 100 tweets of the same. The difference is vast: the classifier achieved an error rate of 0.09, compared to 0.43 for individual tweets. To see the progression, Figure 4.6 shows the error rate versus batch size for A-CT and T-Lex data. At the largest batch sizes, A-CT detection is near perfect, for all payload sizes.

As with individual tweets, there is very little difference between A-CT-U and A-CT-S when pooling is in use; A-CT-P is harder to detect on larger payloads, but still approaches near perfect detection at the highest batch size.

Figure 4.7 shows the error rate versus batch size for M-CT data. With the exception of M-CT data with 1 bpt, this system is also vulnerable to pooled attacks. The data containing 1 bit remains close to undetectable until the largest batch size, when P_E drops to 0.33 for unpadded, 0.38 for stripped, and 0.39 for padded. Other payload sizes show a clear drop as we increase batch size.

There are a number of unusual results present, such as P_E being higher for M-CT-U (2 bpt) at batch size 10 than at batch size 5. We attribute these anomalies to under-training of the classifier, rather than embedding curios. As batch size increases, our small set of training data rapidly shrinks: at batch size of 100, we are training on only 18 instances each fold (9 each for stego and cover), and testing on 2; the same effect is seen on the T-Lex data. To confirm it as such, we repeated the experiments for M-CT-U and T-Lex, keeping the number of training instances constant across each batch size. Figure 4.8 show the results for this; curious results are absent, which seems to confirm the previous effect as a symptom of under-training. Note that we started at a batch size of 2, as comparison with individual tweets is unfair given the difference in feature space dimensionality.

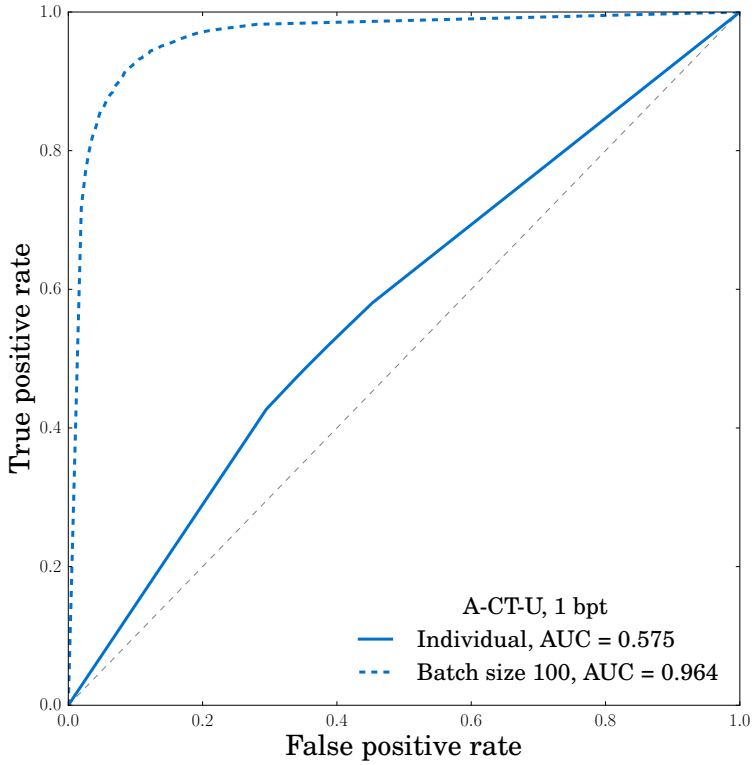
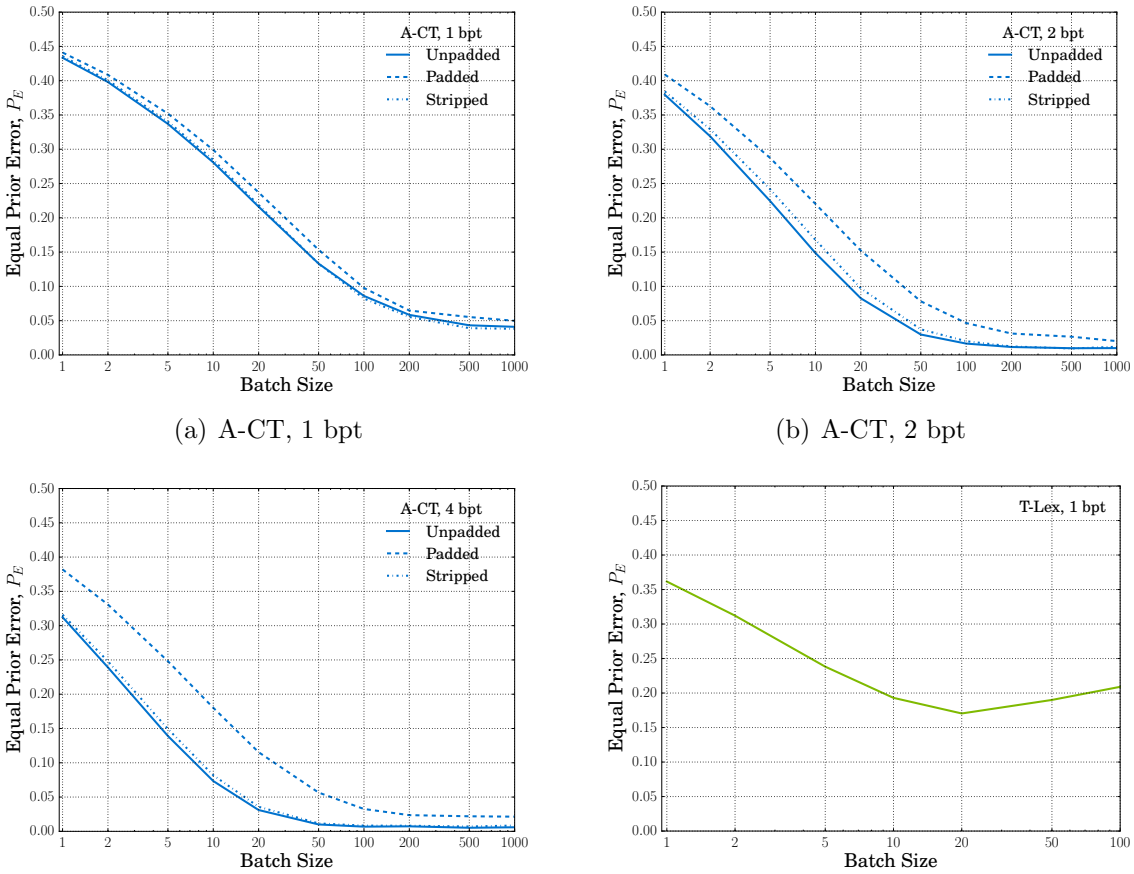


Figure 4.5: ROCs for classifiers trained on individual A-CT generated tweets, and batches of 100 A-CT tweets.

The differences between padded, unpadded and stripped M-CT are most noticeable at 4 bpt: M-CT-P is significantly harder to detect than M-CT-U. At batch size of 100, P_E is 0.05 for the unpadded data, and 0.25 for padded. We conclude that having to discard tweets has a significant effect on security.

As already mentioned, tweets containing no options should likely never be used to hide any payload: they should instead be transmitted and ignored by the receiver and attacker. It would be somewhat desirable to see how the attack would fare if these zero capacity tweets were removed from both the cover and stego data sets; whether the inclusion of tweets rendered unusable by human filtering, or by a low (but greater than zero) number of stego options, would still aid detection to the same degree. While we do not have results for this, the point is somewhat moot: the next chapter will address the use of covers with no capacity, and dispense with the act of discarding unusable tweets.



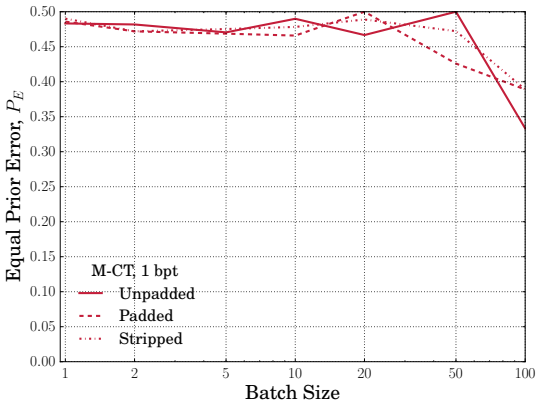
(a) A-CT, 1 bpt

(b) A-CT, 2 bpt

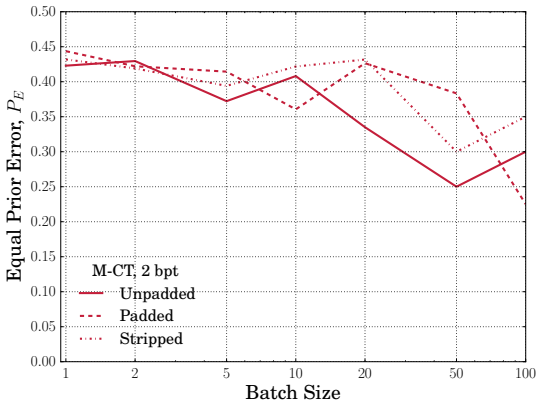
(c) A-CT, 4 bpt

(d) T-Lex

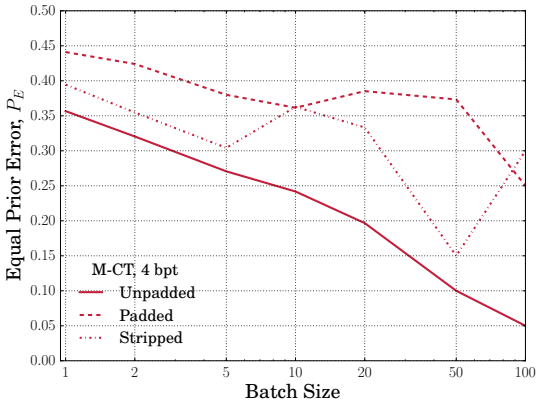
Figure 4.6: Equal prior error rates of classifiers trained on A-CT and T-Lex tweet batches of increasing size. Note that the range of batch sizes for T-Lex is different to that for A-CT.



(a) M-CT, 1 bpt



(b) M-CT, 2 bpt



(c) M-CT, 4 bpt

Figure 4.7: Equal prior error rates of classifiers trained on M-CT tweet batches of increasing size.

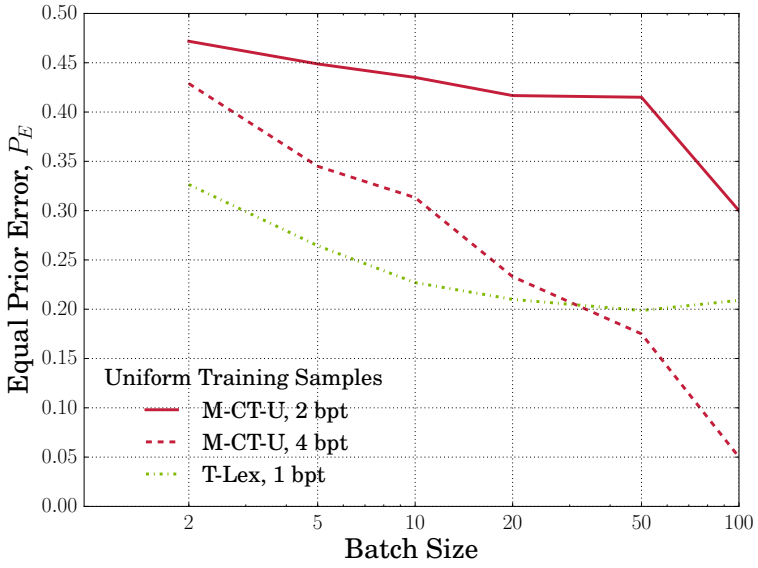


Figure 4.8: Equal prior error rates of classifiers using the same number of training samples at each batch size.

4.4.3 T-Lex

Individually, and at small batch sizes, T-Lex data is consistently easier to detect than A-CT data containing the same payload size, as seen in the previous figures. However, it appears to plateau at a higher error rate than A-CT data, and is harder to detect than A-CT data containing more payload. This is perhaps not surprising, as the features are explicitly designed to detect output from CoverTweet. Note that the results on T-Lex data are not completely comparable to those on CoverTweet data, because tweets without any substitutable words were discarded immediately from the T-Lex data sets, whereas with CoverTweet these are kept if they contain the desired payload.

To avoid giving the impression that T-Lex is a secure system (and justifying swathes of future steganalysis literature dedicated to attacking it), we extracted an additional set of features using the T-Lex data source instead of the PPDB. With the exception of the features for "proportion of PPDB entries" (the equivalent with T-Lex is useless: tweets with zero entries were discarded, and T-Lex only performs single word substitutions), all other features were calculated by querying the set of T-Lex substitutions instead of the PPDB. We expected this would be a powerful attack, as the set contains a significant number of non-fluent substitution rules. Figure 4.9 shows the equal prior error rate versus batch size for this new data: there can be no

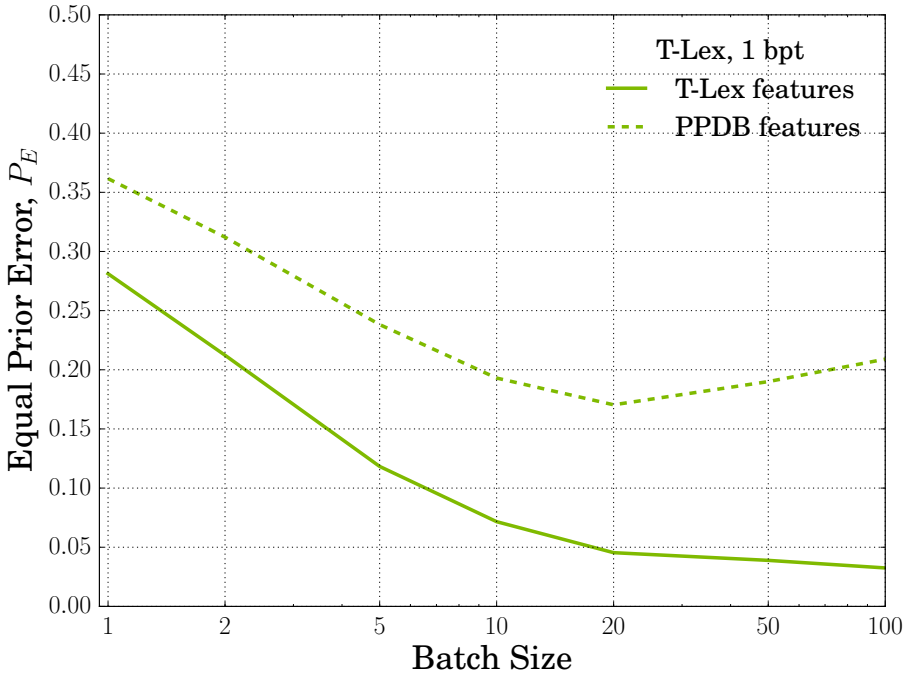


Figure 4.9: Equal prior error rate P_E of classifiers trained on T-Lex tweet batches of increasing size, with the basic and probability feature sets, along with either the PPDB features, or T-Lex features.

question that T-Lex is easy to detect.

There are likely multiple weak aspects of the system, but one weakness can be summarised by two points: only 5% of cover objects have any stego options with a higher probability; transformation sets are disjoint. Given a tweet which might have been modified, the latter point allows a Kerckhoffs' attacker to construct the exact set of options the stegosystem could choose from; the former point means that if this set contains an option with a higher probability, the tweet is almost certainly steganographic.

Our features capture this, by including the log likelihood of a given tweet t, and the log likelihood of the most probable modified option t'. The value of

$$\log \Pr(t) - \log \Pr(t'), \tag{4.8}$$

is less than 0 for 97% of the modified tweets in our data sets, and 5% of the unmodified tweets.

This is a strong enough signifier of steganography that all other features are superfluous. We can build an incredibly simple detector which actually achieves lower error rates at the largest payload size.

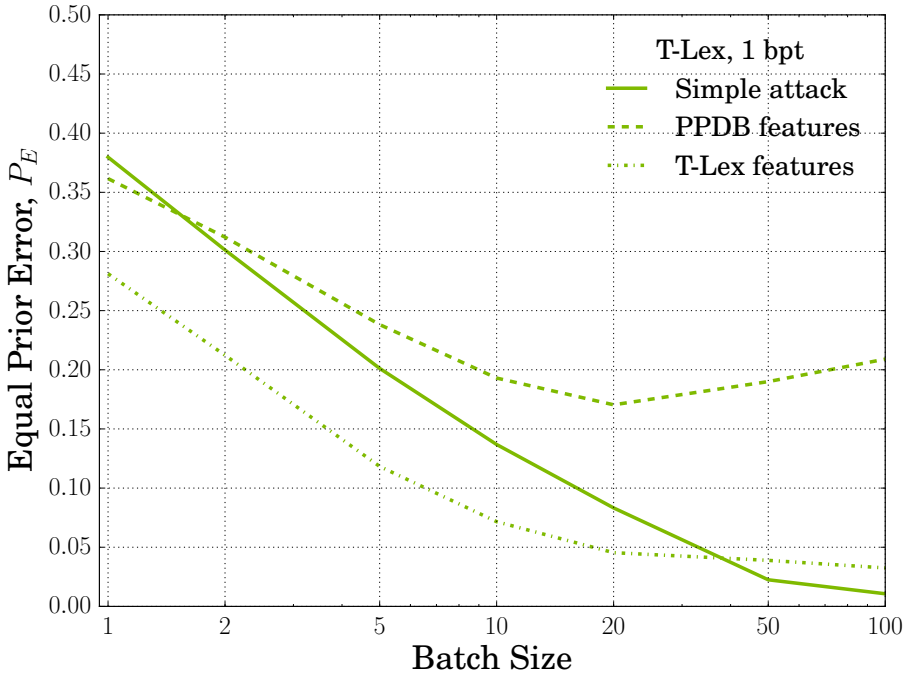


Figure 4.10: Equal prior error rates for the simple statistical detector of T-Lex.

This detector works as follows. On individual tweets, it denotes a tweet as stego if $\log \Pr(t) - \log \Pr(t')$ is less than 0, and cover otherwise. To pool this value for batches, it counts how many tweets within a batch it thinks are stego. On training data, it calculates what the average count of this is, in both stego and cover sets; it then sets a decision threshold at the average of these two counts. When classifying an unseen batch, if the number is less than the decision threshold, it marks it as cover; otherwise, it marks it as stego.

Figure 4.10 shows the results for this detector, compared to the other attacks; though the full attack with T-Lex features achieves better detection at lower batch sizes, at the largest batch size the simple detector achieves a P_E of 0.01 (the full attack achieves a P_E of 0.03 at this size). This sort of simple statistical detector is reminiscent of early steganalysis; it is fitting that an old stegosystem should be so weak to an old technique.

The fact that the full attack is beaten by the simple detector at the largest size is potentially down to the pooling used: the attack uses the mean, variance, maximum and minimum of $\log \Pr(t)$ and $\log \Pr(t')$ across the tweets in the batch, but not the exact feature $(\log \Pr(t) - \log \Pr(t'))$ employed by the simple detector.

CoverTweet itself is not impervious to this attack, but is not weak to the same

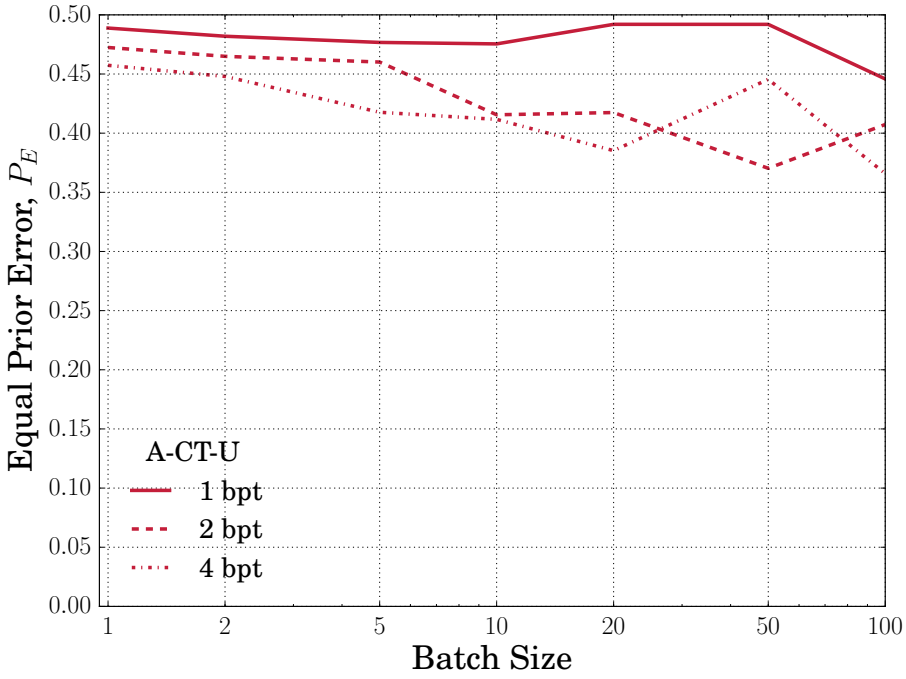


Figure 4.11: Equal prior error rates for the simple statistical detector on unpadded M-CT data.

degree as T-Lex. The simple detector performs worse than the full attack at all batch sizes and payload rates; Figure 4.11 shows P_E against batch size for this simple detector on M-CT-U. The advantage CoverTweet holds over T-Lex is likely in its non-disjoint transformations, meaning the attacker cannot construct the exact set of possible stego options.

4.4.4 Feature Evaluation

To establish which class of features had the biggest effect on detection, we trained models on each combination of features, using all data types, and batches of 10 tweets. A selection of the results for padded data are shown in Table 4.7; unpadded and stripped are not included, but are comparable. M-CT with the largest payload is shown; results with the 1 bit data would be useless for comparison purposes, as it is nearly undetectable. Direct comparison of systems is not entirely encouraged on these results, because of the different payload sizes.

We can see that for A-CT, the PPDB features easily outperform others, with a P_E of 0.217 when used alone; the warden's knowledge of the system is predictably powerful. The second best on A-CT data is the n-gram set, followed by basic, then

	b	n	p	w	bn	np	pw	bnp	npw	all
A-CT	0.345	0.311	0.217	0.376	0.266	0.204	0.203	0.180	0.191	0.169
T-Lex	0.448	0.226	0.273	0.317	0.226	0.202	0.218	0.198	0.168	0.168
M-CT*	0.431	0.337	0.225	0.357	0.386	0.233	0.229	0.372	0.277	0.345
M-CT†	0.407	0.448	0.406	0.396	0.367	0.362	0.373	0.343	0.362	0.345

* 4 bpt.

† 4 bpt, padded with random features.

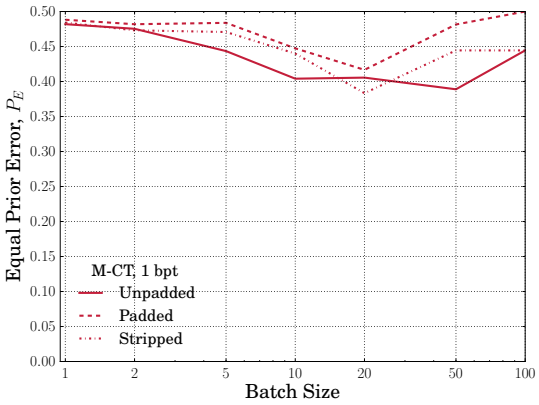
Table 4.7: Equal prior error rates for models trained on different combinations of feature class, for a batch of 10. M-CT data is at 4 bpt, A-CT and T-lex at 1 bpt; all data sets are stripped. The feature sets are as follows: basic (b), n-gram probabilities (n), PPDB (p) and word length probabilities (w).

word length. Given the simplicity of the set, it is somewhat of a surprise that the basic features perform so well ($P_E = 0.345$) on their own.

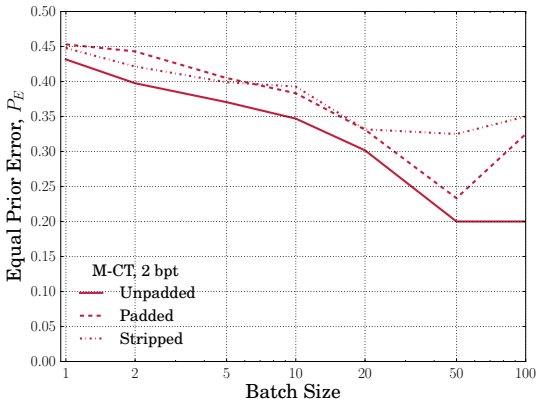
For T-Lex, the n-gram set is the best on its own; this is likely due to poor substitutions contained in the T-Lex transformation source. The second best is the PPDB set; remember that a number of the PPDB features are based on the n-gram features, which is likely the main contribution in this case. It is clearly not the only contribution however: combining the PPDB and the n-gram set reduces the error rate, despite the mismatch of substitution source. Word length features are third best, and are actually much better for T-Lex than for CoverTweet, which we did not predict. Finally, basic features are very poor: 0.448 error rate. The basic features are aimed at changes in word count and stop word usage, neither of which are affected by T-Lex substitutions. Not included in the table are the modified T-Lex features, which on their own achieved an impressive error rate of 0.09 (the addition of the other feature sets only managed to drop this to 0.07).

The comparison for M-CT reveals an unusual result: the classifier trained with all features is outperformed by classifiers trained with certain subsets. Most notably, the PPDB features on their own achieve a P_E of 0.225, compared to 0.362 with all features. We suspect this is another symptom of under-training; the dimensionality of the feature space is too large for the number of training samples we have. To confirm this, we tried each combination again, keeping the number of features constant by padding with random numbers. These results can also be seen in the table, and are more consistent with our expectations.

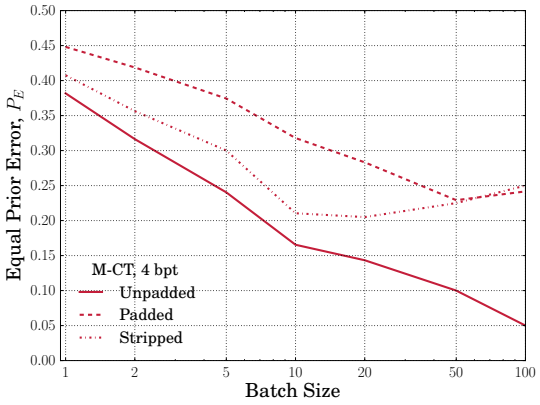
Having discovered that the PPDB features potentially work best on their own, we have repeated the pooling results for M-CT data using only these. Figure 4.12 shows the results of this. When using just the PPDB features, the results show the



(a) M-CT, 1 bpt



(b) M-CT, 2 bpt



(c) M-CT, 4 bpt

Figure 4.12: Equal prior error rates of classifiers trained on M-CT tweet batches of increasing size, using only PPDB features.

same general trends as when using all features: unpadded data is the most detectable; increasing batch size and payload rate decreases P_E. On the original results, 1 bpt data was essentially undetected until the largest batch size; with the reduced feature set the classifier achieved a P_E of 0.4 at a batch size of 10.

As a final investigation into feature performance, we selected a pair of batches (the stego and the cover) from the M-CT data, which were correctly identified by our attack (using all features). Figure 4.13 shows the batches; the stego contains 4 bits per tweet, and the batch size is 5. We calculated the mean and standard deviation for each feature across all the M-CT cover data (for the same payload size). By seeing how many standard deviations the features in the selected stego batch are away from the mean, we can get an idea of which features are having the most impact in this specific case.

In the basic set, the pooled features relating to the stop word `very` were the most

Cover batch:

1. `USER not really i 'm actually pretty loud i mean like`
2. `ew creeper alert .`
3. `thanks but no thanks .`
4. `‘ b cups are the best , they be like everyone calm down ’`
5. `‘ you just obsess over boys from afar ’ omg stop`

Stego batch:

1. `USER not really i 'm actually pretty loud i mean` *`loves`*
2. `ew creeper alert .`
3. `thanks but no` *`thankyou`* `.`
4. `‘ b cups are the` *`very`* `best , they be like everyone` *`just`* `calm down ’`
5. `‘ you just obsess over` *`lads`* `from afar ’ omg stop`

Figure 4.13: A pair of detectable M-CT batches, containing 4 bpt.

standard deviations (8.16) away from the cover mean; the stop word only appears once in the set, but it was indeed inserted by CoverTweet ("`the best`" was substituted for "`the very best`").

In the n-gram set, the features the most standard deviations away from the mean were all variance of n-gram means (for $n > 1$). No word length features appeared to stand out on this batch. The main features to stand out in the PPDB set are the n-gram features focused only on changeable n-grams.

Of the five stego tweets, only one was unchanged during embedding. One was changed through the insertion of two stop words (`very` and `just`), one by replacing `boys` with `lads` and the remaining two by taking advantage of the overall noisiness of the user: an incongruous `like` was changed for an equally out of place `loves`; `thanks` was changed for the odd `thankyou`. That last change is one of the two tweets giving the batch away as stego, the other being the tweet with inserted stop words. In this case, the other tweets can be reverted to their original cover, and the batch is still detectable as stego; conversely, reverting either of these tweets back to cover, and the batch becomes undetectable. This is a clear demonstration of the fact that some changes are more detectable than others. In the next chapter, we will look at techniques which prioritise the application of better quality changes (so-called *adaptive embedding*).

4.5 Summary

We have created a large feature set for the automatic detection of linguistic steganography. Steganographic tweets are not easily detectable individually, but neither are they capable of conveying large payloads on their own. Pooling, both for the steganographer and the steganalyst, is essential. Applying the pooled steganalysis paradigm allows for near perfect classification of automatically generated CoverTweet and T-Lex stego data, and good classification of manually generated CoverTweet data.

We evaluated the separate feature sets, and found that all features contribute something to the attack. However, vastly outperforming other features were those extracted using the PPDB, CoverTweet's transformation source. Using the same principles, features extracted using T-Lex's transformation source were employed very successfully against T-Lex generated tweets. The lesson learned from both cases: Kerckhoffs' attackers have a lot of power. We believe this is the first work to explicitly apply Kerckhoffs' principle to linguistic steganalysis, and have left it for future work to investigate weaker attackers, without such total knowledge of the steganographer's system.

The results showed that some changes are easier to detect than others. We could potentially change the PPDB to generate tweets which are harder to detect, selectively filtering the paraphrase rules to remove the ones we know result in easier detection. However, it is better to instead introduce an estimate of *cost* per change, and aim to minimise this cost during embedding. We will look at approaches to this in the next chapter.

Currently, the steganographer discards tweets without options for the payload they wish to embed. While we already knew this was wasteful with regards to capacity (often good tweets will be discarded, despite being able to convey some payload values), we showed here it is also harmful to security. We will also address this in the next chapter.

When evaluating the attack against M-CT data, we came across a number of signs of under-training, caused by a lack of sufficient data. This is unfortunately unavoidable at present: manual data is slow and expensive to produce. Overcoming this cost of data generation is an important task for future work.

This chapter was adapted from the paper "Detection of steganographic techniques on Twitter" [Wilson et al., 2015]. The feature set and simple detector designed for use against T-Lex are new to this work.

Chapter 5

Coding

> What we have in this case is an inability to communicate.
>
> *Cool Hand Luke*
> *via* CoverTweet

In Chapter 3, we introduced a system for producing steganographic text with an unrestricted transformation step, which was shown to be secure against human attackers. In Chapter 4, we switched to the role of the warden, developing an attack against the generated text. We will now enter the steganographer's shoes again, and address the problems we have been avoiding thus far: what should we do with unusable tweets, and how should we deal with the fact that some changes are more detectable than others?

These problems already have solutions in image steganography, through the application of *coding*. Sections 5.1 and 5.2 describe the prior art; Section 5.3 onwards describes the deployment of coding in our linguistic setting (some of which was described in brief in Chapter 2). This is the first work on linguistic steganography to use this technology, and could have a significant impact on the future of research in the area.

5.1 Non-Shared Selection Channel

The *selection channel* is the subset of the cover where embedding changes can be made [Fridrich, 2010]; a *non-shared* selection channel is one which is known by the steganographer, but not by the receiver.

Chapter 2 gave a detailed discussion of how value assignment can be studied as a form of graph labelling, and how the majority of published stegosystems assign values

in a way that requires the receiver to build the same graph as the sender. Ostensibly the purpose of this was to allow unambiguous embedding (and therefore extraction) of messages, but by extension it also meant the receiver always knew *where* the message could be embedded: the selection channel was always shared.

Most systems cripple themselves to achieve this: e.g. single non-disjoint substitutions in T-Lex; dividing the cover into chunks containing at least one deletable adjective in [Chang and Clark, 2012]; etc. CoverTweet's hash function provides an easy solution (without curtailing transformation freedom) to unambiguous value assignment, but it also means the selection channel is no longer shared: the receiver cannot reliably identify tweets capable of embedding messages. This is not the only part of CoverTweet responsible for a non-shared selection channel: the human filter, which we have seen to be a powerful resource, cannot be used in any system without making the selection channel non-shared[1].

Our approach so far has been to discard covers incapable of embedding the correct payload. This is both a problem for security (seen by the results in Chapter 4), and for capacity: when embedding 4 bpt, a tweet with 15 options has at least a 1/16 chance of being discarded, as though it cannot carry any payload at all.

The problems caused by a non-shared selection channel are not unique to linguistic steganography. One example arises when modifying DCT coefficients while embedding in JPEG images[2] [Westfeld, 2001]. It is commonly thought that modifying zero coefficients (that is, coefficients with a value of zero) during embedding causes easily detectable steganographic images (by causing visible artefacts in the images), so they should not be used to convey payload. However, embedding can introduce *new* zero coefficients: how can the receiver differentiate between a coefficient that was originally zero (and therefore not used for embedding), and one which was made to be zero during embedding?

An early solution: if a zero coefficient is introduced when embedding a bit of payload, the same payload bit is re-embedded in the next coefficient (this is repeated until the result is a nonzero modified coefficient). This increases the number of zero coefficients (an effect known as *shrinkage*), which makes detection easier; the number of modified coefficients also increases. This has echoes of our approach to unusable tweets, both in its inefficiency, and in the weakening of security.

[1]Certain transformations would also imply a non-shared selection channel (e.g. adjective deletion), but the subset of the PPDB used in this work does not contain any of these transformations.

[2]Knowledge of DCT coefficients is unneccessary to understand this example: JPEG images have them; the coefficient values can be modified to hide information.

A better solution: apply ideas from coding theory to allow for the embedding of payload without the shrinkage effect. Instead of sending the desired payload bits directly, we indicate them indirectly. This is related to communication over a noisy channel, where codewords are sent to indirectly represent the desired message. When the number of bits being hidden is less than the full capacity of the cover, there is a choice of coded message; we can choose one that leaves unusable tweets (or unusable DCT coefficients) unchanged.

5.1.1 Syndrome Coding

Syndrome coding is one possible method [Crandall, 1998, van Dijk and Willems, 2001, Galand and Kabatiansky, 2003].

Assume we have a set of n cover tweets, and we want to send b symbols from a payload alphabet Σ (of size q). In earlier chapters we were embedding binary strings of various lengths; here each tweet conveys a single q-ary symbol from the alphabet. Embedding 4 bpt (as done in previous chapters) is equivalent to embedding a single hexadecimal ($q = 16$) symbol per tweet.

To code, sender and receiver share a q-ary matrix $\mathbf{H}$ (called the *parity check matrix*) with n columns and b rows. This matrix is large, but the sender and receiver can share it by generating it pseudorandomly with their key, or by using a structured matrix and permuting it with the key. Let $\mathbf{c}$ be the vector of q-ary symbols conveyed by the set of covers (one symbol per tweet), and $\mathbf{e}$ the vector representing changes to be made to the cover tweets. The set of symbols conveyed by the produced stego tweets is defined as $\mathbf{s} = \mathbf{c} + \mathbf{e}$. The aim is to produce a set of stego tweets so that $\mathbf{s}$ meets:

$$\mathbf{Hs} = m, \tag{5.1}$$

where m is the desired message ($m \in \Sigma^b$). Solutions to this require finding a change vector $\mathbf{e}$ s.t. $\mathbf{He} = m - \mathbf{Hc}$; then, to decode, we multiply the generated $\mathbf{s}$ by the parity check matrix:

$$\mathbf{Hs} = \mathbf{H}(\mathbf{c} + \mathbf{e}) = \mathbf{Hc} + \mathbf{He} = \mathbf{Hc} + m - \mathbf{Hc} = m. \tag{5.2}$$

When $b < n$, there are multiple $\mathbf{e}$ that work as solutions; we can solve the non-shared selection channel problem by finding an $\mathbf{e}$ which does not change tweets to symbols they cannot convey [Fridrich et al., 2005]. The probability of there being such an $\mathbf{e}$ depends on the number of missing symbols we have, compared to the size of the cover, and the amount of payload we wish to hide. For a random $\mathbf{H}$, the

probability of there being no solution is $o(q^{-(n-b-w)})$, where w is the number of tweets missing symbols. In other words, there is very likely a solution, as long as $b+w<n$. Note that this result is based on the probability of a random q-ary b by $(n-w)$ matrix having full row rank:

$$\Pr(\text{Full row rank}) = 1 - \prod_{k=0}^{b-1} 1 - \frac{q^k}{q^{n-w}}. \tag{5.3}$$

When $b \ll n-w$:

$$\Pr(\text{Full row rank}) \approx \sum_{k=0}^{b-1} \frac{q^k}{q^{n-w}} \leq \frac{q^b}{q^{n-w}}. \tag{5.4}$$

Consider a parity check matrix

$$\mathbf{H} = \begin{pmatrix} 0 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \tag{5.5}$$

and a set of cover objects (e.g. tweets) conveying ternary symbols

$$\mathbf{c} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}. \tag{5.6}$$

If we wish to send the message

$$m = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \tag{5.7}$$

we need to make changes to the cover objects so that $\mathbf{Hs} = m$, where $\mathbf{s}$ is the vector of symbols conveyed by the resulting stego objects. One solution is to change the middle object so that it conveys a 0, which would give us

$$\mathbf{Hs} = \begin{pmatrix} 0 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}. \tag{5.8}$$

If the middle object cannot convey a 0 (in the case of CoverTweet, if there is not an option with a hash value of 0) we could alternatively change the first and last cover objects to both convey 0, and leave the middle object unchanged.

It is important to note that message symbols no longer directly correspond to individual stego objects, but are effectively spread out across all of them.

As well as solving the non-shared selection channel problem, there is an additional benefit: we can simultaneously boost *embedding efficiency*, the amount of payload conveyed per change made to the cover. Every change has an effect on security, so minimising changes (and maximising efficiency) is theoretically a good idea.

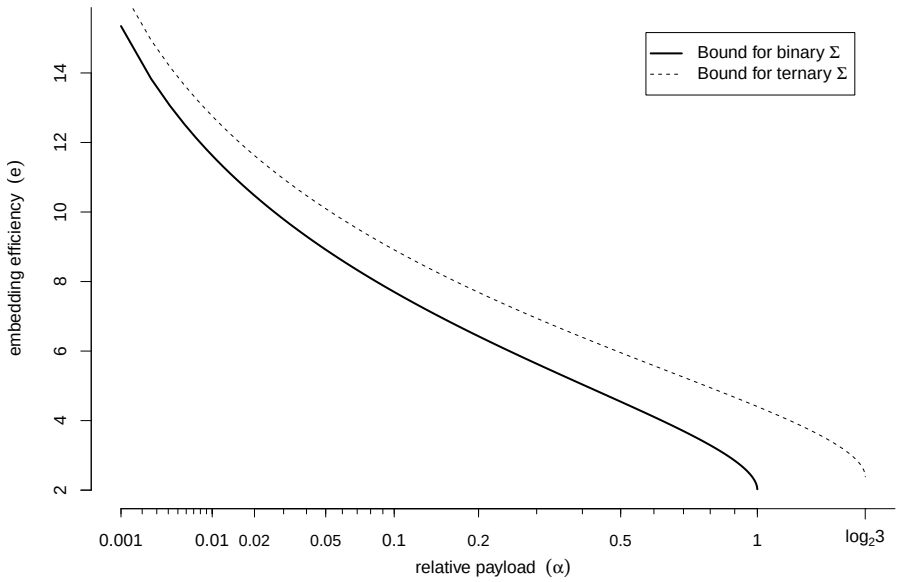


Figure 5.1: Embedding efficiency versus relative payload. Reproduced with permission from [Ker, 2015].

Embedding efficiency has a theoretical upper bound, derived as

$$e \leq \frac{\alpha}{H_q^{-1}(\alpha)}, \tag{5.9}$$

where relative payload $\alpha = \frac{b}{n}$ and H_q^{-1} is the inverse of the q-ary entropy function [Ker, 2015]. Figure 5.1 shows this bound for binary and ternary alphabets; random codes (with pseudo-random $\mathbf{H}$) approach this bound [Fridrich and Filler, 2007a].

When using syndrome codes, we boost efficiency by finding

$$\min w(\mathbf{e}) \quad \text{s.t.} \quad \mathbf{H}(\mathbf{c}+\mathbf{e}) = m, \tag{5.10}$$

where w is the number of non-zero values in a vector (the *Hamming weight*). Each non-zero value in $\mathbf{e}$ corresponds to a change made to the cover, so minimising Hamming weight maximises embedding efficiency.

5.2 Minimising Distortion

Section 2.5 defined the term *distortion* as a measure of substitution quality. Every change made to a cover has a distortion, or cost, which reflects how much the change moves the distribution of stego objects away from the distribution of cover objects.

The larger the distance between the stego and cover object distributions, the more detectable the stego objects are.

Coding gives us a way to avoid our first problem of unusable tweets, and as an added advantage, we get a boost to embedding efficiency. However, maximising embedding efficiency is not enough for security, unless all changes have an equal cost; we know this is not the case.

Consider the batch of tweets examined in Section 4.4.4 (Figure 4.13). Analysis of the features for the batch suggested that it was the stop word `very` in "`the very best`" (from the fourth tweet), and the use of the unusual token `thankyou` (in the third tweet) that gave the batch away: reverting either of the tweets back to cover resulted in batches which were no longer detectable. This is indicative of changes with high distortion, and if possible we should avoid making these. By comparison, the substitution of `lads` for `boys` (in the fifth tweet) is an example of a low distortion change.

Instead of just minimising the number of changes, systems should strive to minimise distortion during embedding. This is known as *adaptive embedding*. We can do this with syndrome coding, if we are able to find $\mathbf{e}$ with the lowest distortion (instead of the lowest Hamming weight).

Let d_{ij} be the cost of changing the i^{th} tweet to embed symbol $j \in \Sigma$. If the value of $\mathbf{c}_i$ is already j, the cost is 0; if there is no option for j (the value is 'missing'), the cost is infinite. If i has no stego options, $d_{ij} = \infty, \forall j \in \Sigma \setminus \mathbf{c}_i$.

We want to find

$$\min \sum_{i=0}^{n} d_{i(\mathbf{c}_i + \mathbf{e}_i)} \quad \text{s.t.} \quad \mathbf{H}(\mathbf{c} + \mathbf{e}) = m. \tag{5.11}$$

In general solving this is an NP-complete problem [Berlekamp et al., 1978]. However, if the parity check matrix has a particular form, then we can find the near-optimal solution in linear time using a version of the Viterbi algorithm. This is called *syndrome trellis coding* [Filler et al., 2011].

The parity check matrix $\mathbf{H}$ is constructed by repeating a smaller submatrix $\hat{\mathbf{H}}$ along the main diagonal (see Figure 5.2). The width w of $\hat{\mathbf{H}}$ depends on the desired relative payload; the choice of height h of $\hat{\mathbf{H}}$ affects the speed and efficiency of the algorithm. The embedding solution can be found with time and space complexity $O(nq^h)$: smaller h means finding a solution is faster, but it is less likely we will get near to the optimal solution.

Many published linguistic stegosystems could immediately benefit from this existing technology (any with a small q, as with binary changes). However, these systems

$$\mathbf{H} = \begin{pmatrix} \boxed{\hat{\mathbf{H}}} & 0 & 0 & 0 & 0 \\ & \boxed{\hat{\mathbf{H}}} & & 0 & 0 \\ 0 & 0 & \boxed{\hat{\mathbf{H}}} & & \\ 0 & 0 & 0 & 0 & \end{pmatrix}$$

Figure 5.2: The structured partity check matrix $\mathbf{H}$ used in syndrome trellis coding.

were built with a shared selection channel in mind, and their transformations are purposefully restrictive in scope.

By comparison, applying syndrome trellis codes to CoverTweet would be much harder, given the growth of algorithm complexity with q. In the work so far, we have used an alphabet with a maximum size of 16, and this is without knowing what the *actual* alphabet size should be (we will see that for some tweets it should be vastly larger).

It would be possible to use *nested*, or *multi-layered*, syndrome trellis coding [Filler and Fridrich, 2010b], which decomposes the non-binary embedding problem into multiple binary embeddings, but each binary embedding required increases the *coding loss*: the difference between the achievable payload and the maximum theoretical payload. The larger q, the more individual binary embeddings required, the larger the coding loss.

Note that syndrome trellis coding relies on two related assumptions: changes to covers are independent, and distortion is additive. This is an approximation in images, where distortion is measured per pixel or cover element. Our distortions are per *tweet* (which might contain multiple changes), making this assumption slightly more accurate; nonetheless, it is still an approximation. Gibbs embedding [Filler and Fridrich, 2010a] is a method that allows for arbitrary distortion measures (including non-additive), but it is very slow, so has not yet seen widespread application.

5.3 The Optimal Embedding Paradigm

Having looked at a practical solution for minimising embedding distortion, we return to the theoretic definition of the problem that was briefly discussed in Chapter 2. Recall that given a distortion d_{ij} of changing tweet i to symbol j, and a probability p_{ij} of making that change, the average capacity is the entropy of p_i:

$$H(p_i) = -\sum_j p_{ij} \log_2 p_{ij}, \tag{5.12}$$

and the average cost is:

$$E(d_i) = \sum_j d_{ij} p_{ij}. \tag{5.13}$$

We can either minimise the average distortion while embedding a fixed average payload of size b (Payload Limited Sender: PLS):

$$\min \sum_i E(d_i) \quad \text{s.t.} \quad \sum_i H(p_i) = b, \tag{5.14}$$

or maximise average payload while fixing the average distortion to value D (Distortion Limited Sender: DLS):

$$\max \sum_i H(p_i) \quad \text{s.t.} \quad \sum_i E(d_i) = D, \tag{5.15}$$

and both options have the same optimal solution for p_{ij} [Fridrich and Filler, 2007b]:

$$p_{ij} = \frac{\mathsf{e}^{-\lambda d_{ij}}}{\sum_j \mathsf{e}^{-\lambda d_{ij}}}, \tag{5.16}$$

for some constant λ, determined by either b or D. So-called *optimal embedding*[3] is achieved when changes are made according to this optimal distribution.

At present, no coding methods exist which completely fit our requirements; syndrome trellis codes could be used, but are likely too expensive for the alphabet sizes required in linguistic steganography. This does not mean we cannot evaluate CoverTweet as though we were using coding: we can *simulate* optimal embedding.

5.3.1 Simulating Optimal Embedding

We will generate a large number of stego options for each cover object, using Manual and Automatic CoverTweet, and T-Lex (see Section 5.5). Then, we will estimate distortions for each of these options (see Section 5.4). Finally, we will simulate the production of optimally embedded tweets, under the PLS paradigm.

Given a set of tweets from a particular user, we can simulate optimal embedding (or *perfect coding*) by finding the λ which gives us a distribution p_i s.t. $\sum_i H(p_i) = b$, for a desired average payload b.

When λ is 0, distortion is not taken into account, changes are equally likely, and entropy of p_i is maximum:

$$p_{ij} = \frac{\mathsf{e}^{-0 d_{ij}}}{\sum_{k \in \Sigma} \mathsf{e}^{-0 d_{ik}}} = \frac{1}{|\Sigma|}. \tag{5.17}$$

[3]Note that the literature refers to this idea as optimal embedding, but it should not be taken as a guarantee that resulting embeddings are actually optimal.

When $\lambda > 0$, and a symbol j is missing for tweet i, the probability of making that change is 0:

$$p_{ij} = \frac{\mathsf{e}^{-\lambda\infty}}{\sum_{k\in\Sigma}\mathsf{e}^{-\lambda d_{ik}}} = \frac{0}{\sum_{k\in\Sigma}\mathsf{e}^{-\lambda d_{ik}}} = 0. \tag{5.18}$$

The entropy of p_i is monotonic with respect to λ: as λ increases, the average capacity decreases. As such, we can use binary search to find the correct value for λ. Once found, we step through the data set, changing each cover tweet i to its j^{th} option with probability p_{ij}. We will do this with different values for b, and with different estimates of distortion d_{ij}.

Simulating optimal embedding is common in the literature. It gives an estimate of the lower bound of detectability; actually performing coding can only increase detectability. Simulating coding here isolates the evaluation of our distortion measures from weaknesses inherent in any actual coding scheme.

5.4 Distortion Measures for Linguistic Steganography

In order to practise coding, methods to estimate distortion are required. Four are defined here. We will generate batches of steganographic tweets, minimising each of these measures; we will re-evaluate our attack against these distortion minimised tweets.

For each measure, the distortion for changing a tweet to an unembeddable symbol is infinite, and 0 for leaving a tweet unchanged.

Binary A simple binary distortion measure: 0 for an unchanged tweet, 1 otherwise. This measure clearly does not get us the benefits of distortion minimisation: fluent changes are no more likely than non-fluent. However, we get the boost to embedding efficiency that coding grants us, and it solves the non-shared selection channel.

Edit Distance The minimum number of word-level edits (deletions, insertions and substitutions) required to turn the cover object into the stego object. For example, turning "`those are nice socks`" into "`those are not nice gloves`" has an edit distance of 2: one insertion (`not`) and one substitution (`socks` for `gloves`). This measure is based on the assumption that minimising the number of changes within a tweet is beneficial to security. Here, the

quality of each individual word-level edit is treated as equal. As a side effect, minimising this measure favours single word substitutions (each with a cost of 1), over phrasal changes.

Probability The log likelihood ratio of the cover object probability $\Pr(c)$ and the stego object probability $\Pr(s)$. Probabilities are estimated using an n-gram language model. The distortion is

$$d_s = -(\log \Pr(s) - \log \Pr(c)). \tag{5.19}$$

In the rare case that the stego is more likely than the cover object, the distortion is corrected to 0. Note that negative distortions are possible (the result is that the change is more likely than leaving the object unchanged), but we chose to avoid them here[4].

Recalling the description of n-gram models from Chapter 3, the probability for a string of tokens $\Pr(s_0, \ldots, s_k)$ can be decomposed as

$$\Pr(s_0, \ldots, s_k) = \Pr(s_0) \prod_{i=2}^{T} \Pr(s_i | s_0, \ldots, s_{i-1}). \tag{5.20}$$

If it were possible to estimate this directly, then any substitutions made within a tweet would not be treated as independent. However, the n-gram model approximates this, by limiting the conditioning context to $n-1$ tokens. In this scenario, changes far enough apart (not contained within a single n-gram) are treated as independent.

A more ideal distortion measure might be the likelihood that a tweet has been modified, given tweet t, based on language model and paraphrase probabilities: $\Pr(\text{changed}|t)$. Applying Bayes':

$$\Pr(\text{changed}|t) = \frac{\Pr(t|\text{changed}) \Pr(\text{changed})}{\Pr(t|\text{changed}) \Pr(\text{changed}) + \Pr(t|\text{unchanged}) \Pr(\text{unchanged})}. \tag{5.21}$$

[4]It is reasonable to argue that, if we trust the positive distortion values, we should also trust the negative. However, we know that the costs are only an estimate; trusting the positive distortions is better than the alternative (making changes at random), but it is conservative to assume that not making a change is always better than making one.

The probability of $\Pr(t|\text{unchanged})$ is just the language model probability $\Pr(t)$, and $\Pr(t|\text{changed})$ can be modelled as:

$$\Pr(t|\text{changed}) = \sum_{c \in C} \Pr(c) \Pr(t|c), \tag{5.22}$$

where C is the set of all possible covers. The subset of C for which $\Pr(t|c)$ is not zero can be produced by using the PPDB, but unfortunately this is expensive to calculate, and infeasible to do for every possible stego tweet arising from a given cover.

Feature The Euclidean distance between the feature vectors (for the previously defined features) of the stego object and the cover object. Features are extracted from each stego option for a given tweet. These features are standardised: made zero-mean (by subtracting the mean of each feature) with unit variance (dividing by the standard deviation of each feature).

Some of the features cannot be extracted for every stego object in reasonable time. The substitution score (from the PPDB feature set) is therefore not used. The likelihood of the most probable paraphrased sentence for each possible stego tweet is approximated; we instead used the likelihood of the most probable stego option for each *cover* tweet, meaning each stego object for a given cover has the same value for this.

This distortion explicitly avoids making the changes which are detectable by the attack developed in Chapter 4, although it treats all features as equally important to detection. We expect that minimising this distortion will produce steganography which is the hardest to detect out of the four measures. This is inspired by the HUGO image stegosystem which minimises changes to pixel co-occurrences [Pevnỳ et al., 2010b].

We expect the performance of the measures, judged by how well they mitigate the attack (and which we will see in Section 5.6), to follow the order in which they are described here. The first three measures follow a particular design trajectory: binary treats all stego objects as equal; edit distance differentiates stego objects, but treats all word changes as equal; probability differentiates all changes using language models.

The feature distortion stands somewhat apart from the others, by requiring explicit knowledge of the attack. It can be expected to perform best here, but not necessarily in hypothetical future work utilising a different attack.

These measures represent the start of potential distortion estimation in the linguistic domain. All are heuristic, which is the standard approach in steganographic literature at present. What an ideal distortion function should measure is an open question; in linguistic steganography, approaching the decisions made by the human filter would be a start.

5.5 Data

Once again we return to the Harvard TweetMap data. We use the tweets from the same users described in Section 4.3: 10 for manual, 1000 for automatic. The previously generated data sets contained a single steganographic tweet for each cover: the best option for a randomly generated payload, chosen either by human or language model (or randomly, for T-Lex data). Here we generate all (or for the automatic data, *a lot of*) steganographic options for each cover, so that we can minimise the embedding distortion.

5.5.1 Manual CoverTweet

We have always maintained that the human should be used sparingly: they are slow and expensive. Presently, the system queries the PPDB to get the substitutions which can be made to the cover; these substitutions are used to generate all stego sentence options; the options are filtered by hash, then ordered; the human filter is the final step. The human is shown all options for the desired payload, and has the ability to remove tweets that include (or exclude) particular phrases. Once an option is visible which is "good enough" (according to the human's personal opinion), it can be output.

Using a human to generate *all* fluent steganographic options for a cover, regardless of the hash value, requires a revisiting of this design. This list of options needs to be free of bad substitutions, and it is not at all feasible to require the human to read every single stego option (which would be required by the original design).

We solve this by adding an additional filter to the transformation step. When the system queries the PPDB, the human approves or disapproves of each substitution rule returned. Bad rules are completely discarded, and not used to generate sentences. Other rules are denoted as either always applicable, or *sometimes* applicable. This

booze make you feel alright
get *feeling*

Figure 5.3: A tweet with two substitution rules which are only applicable in combination.

filtered set of rules is used to generate the possible stego options. Sentences generated using only rules marked as 'always' applicable are added to the list of fluent options immediately. Sentences generated with rules which are 'sometimes' applicable are shown to the human again, who can filter them according to the original design.

This change to CoverTweet has an effect on the output. Before, CoverTweet occasionally generated tweets which were fluent through a fortuitous combination of otherwise non-fluent paraphrase rules. For these tweets to be generated now, the human must consider potential combinations which would make a bad rule acceptable; if the human thinks (or knows) such a combination is possible, they can mark the rule as 'sometimes' applicable, and filter any options without the allowable combination at the end. This is not foolproof, and sometimes options will be discarded which could have been used in particular scenarios. Additionally, sometimes options will be included which cannot be used in particular scenarios, through human error; the guarantee of fluency, that our human previously granted, is slightly weakened on certain tweets. Figure 5.3 shows an example of rules which are only fluent in combination. Note that this tweet is also an example of how fluency depends on the context; the sentence "`booze get you feeling alright`" is not grammatically correct, but it is as least as fluent as the original cover "`booze make you feel alright`".

We used this modified CoverTweet to generate all options for 100 tweets, for 10 users. Tweets with no options (other than the unchanged cover) are still included in the set, and during the coding simulation they will be left unchanged, and treated as though they carry no payload. Note that the generation of this data involves an implicit distortion minimisation step, provided by the human filters. The human is essentially assigning 0 and ∞ costs to each possible stego option.

As in the previous chapters, the author was directly involved in this step, due to the prohibitive cost of data generation; we acknowledge this as a limitation of the work, and discuss it further in Section 6.2.2.

Approximately 20% of tweets had no options after manual filtering. The average number of options was 76, or 95 when excluding tweets which cannot be changed. Some tweets were particularly high capacity: "`i never thought i 'd give keenan`

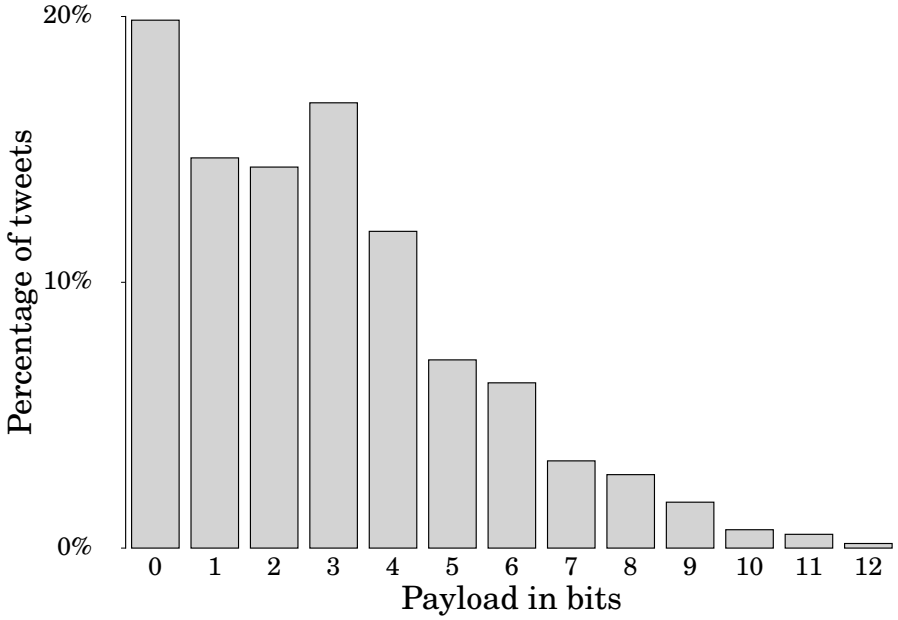


Figure 5.4: Percentage of M-CT tweets with given capacity, rounded to the nearest bit.

`another chance , but i 'm glad i did .`" had 7000 options, thanks to substitutions of "`...`" for the other punctuation, and lots of possible substitutions for "`i never thought`" ("`i did n't think`", "`i just never figured`", etc.) and "`i 'd give`" ("`i would have given`", "`i was gonna give`"). Figure 5.4 shows the distribution of tweet capacity, when rounding capacity to the nearest bit.

Using the generated stego options, we created two sets of stego tweets for each user: one with an average of 1 bpt, another with an average of 2 bpt; only one of our users had enough capacity to hide significantly more than this. The number of tweets changed during the embedding depends both on payload rate, and on the distortion measure used. The changes made in each data set are given in Table 5.1. Using binary distortion, where the only goal is to minimise the number of changes, only 0.11 changes were made per tweet in our set. The more sophisticated measures make more changes than this, as they are attempting to minimise overall distortion.

5.5.2 Automatic CoverTweet

As in Chapter 4, we have generated a large amount of automatic data for use in coding experiments. For the set of 1000 users, we employed CoverTweet to generate sets of steganographic options for 1000 tweets per user.

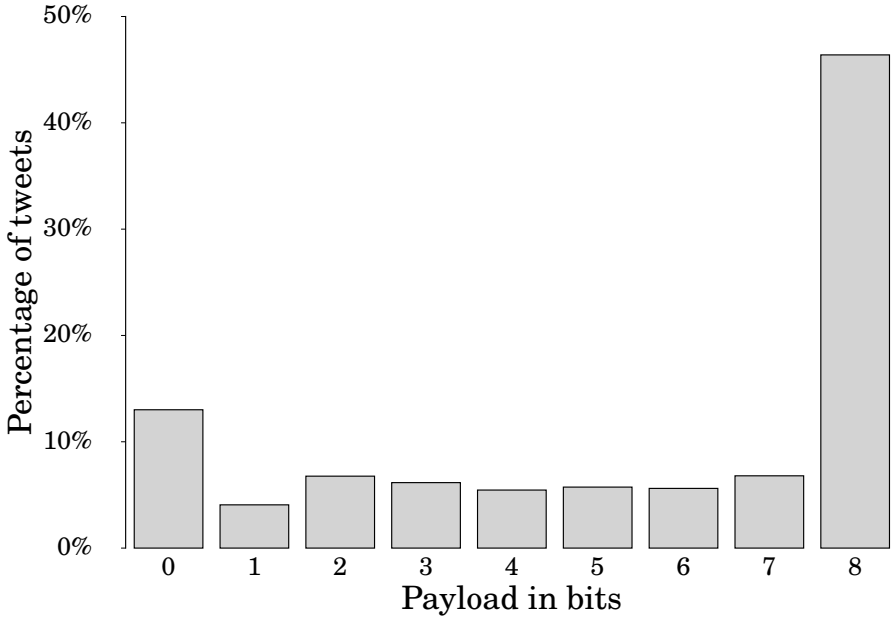


Figure 5.5: Percentage of A-CT tweets with given capacity, rounded to the nearest bit.

Without the human filter, the number of options for certain tweets grows to an enormous size (billions, in some cases), the vast majority of which are unfluent. Ideally, we would generate and store all of these options, relying on the distortion measure to avoid the bad ones. The requirements for storage and distortion estimation time make this untenable. As such, a cap was set at 256 options: we used Automatic CoverTweet (Section 4.3.1) to generate the best option for each possible 8-bit hash value.

In the generation of this data, 13% of tweets had no options; without the human filter, these are tweets that have no substitutable words in the PPDB. Figure 5.5 shows the distribution of tweet capacity for A-CT data. Given the cap on number of options, 46% of tweets can convey 8 bits. We know from Section 3.5 that the average number of total options in the PPDB is 190k; if we could use these, tweets could convey an average of 17 bpt (without a shred of security). Given the larger capacity than M-CT, embedding 1 bpt using binary distortion only required 0.08 changes per tweet.

Embedding Strategy

When embedding in tweets, the steganographer can either spread the payload out across all cover objects, or embed a higher average payload in fewer of the objects.

The former grants a greater boost to coding efficiency [Filler et al., 2011], but the latter potentially gives fewer opportunities to make a non-fluent error (depending, in part, on the distribution of non-fluent options). The steganographer may also wish to adopt the latter if they want to use their Twitter account to send non-steganographic tweets, or in an attempt to confuse the attacker.

In the context of batch steganography in images, this same question (which strategy is better) was studied in [Ker and Pevnỳ, 2014b]. We will study the question in the context of linguistic covers. This is partly a sanity check: with a working distortion measure, we expect that allowing payload to be spread out across all covers to be the superior choice; by allowing the use of all the cover for embedding, the system is able to choose for itself the best places to hide.

We created three sets of stego data, each time hiding an average of 1 bpt, but with different proportions of tweets to be hidden in: 100%, 50% and 25%. For each user, we selected the changeable tweets at random, setting the distortion of changing all other tweets to infinite. The result is equivalent to hiding an average of 1 bpt in all tweets, 2 bpt in half the tweets, and 4 bpt in a quarter of them.

5.5.3 T-Lex

For the same cover tweets used in the generation of automatic CoverTweet data, we generated all stego options using T-Lex. Due to the diminished size of T-Lex's substitution set, a cap was not generally required on the maximum number of tweets, although an exception had to be made for four of the tweets. These tweets had a vast number of stego options, due to the repetitive use of words with multiple options; in one, the user simply repeated the word `hi` twelve times. T-Lex contains three substitutions for this (`hello`, `howdy` and `hullo`), so the result was ∼16m stego options (4^{12}).

Unlike CoverTweet, T-Lex (by design) has a shared selection channel. Tweets with no steganographic options are easily identifiable by the receiver and attacker, and so are removed from our data sets. This reflects the fact that a Kerckhoffs' attacker would ignore such tweets when trying to detect guilty users. The same could be done to our CoverTweet data for the tweets without any PPDB options, but this is a much smaller percentage of total tweets (13% versus 78%), and CoverTweet's design allows transformations for which this cannot be done (though it does not use any).

In our experiments, we will once again be using the T-Lex centric feature set, as opposed to the PPDB based features. Our generated feature distortions take

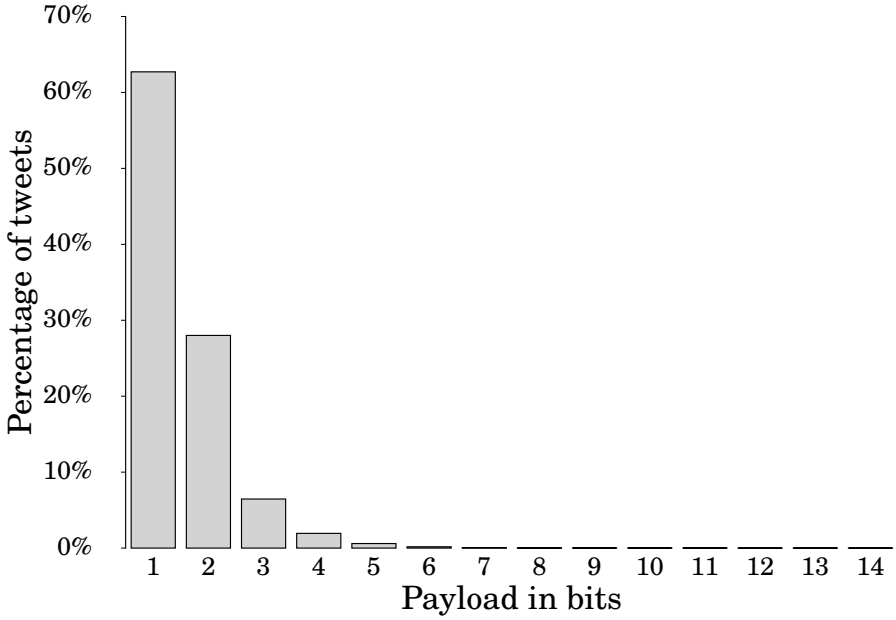


Figure 5.6: Percentage of T-Lex tweets with given capacity, rounded to the nearest bit.

this into account, using the distance between T-Lex feature vectors instead of the PPDB feature vectors. Owing to the vastly reduced transformation set of T-Lex, and the corresponding smaller sets of possible stego options, none of the features were approximated during distortion estimation.

Using the generated stego options, and the calculated distortions, we generated stego data hiding an average of 1 bpt (after removing those with zero capacity). Figure 5.6 shows the distribution of tweet capacity for T-Lex data, excluding tweets without any options. Despite no cap on the number of options, 63% of tweets can only convey 1 bit with T-Lex.

5.5.4 Pooling

For all data sets, we extracted the features described in Section 4.2, then pooled them for a variety of batch sizes. The final data sets are summarised in Table 5.2. Note that, although the number of tweets per user for T-Lex was 1000, only 22% of these were actually capable of hiding payload; those incapable were not included in training and testing data for the classifier.

Other than this data generation, the experimental procedure is the same as that described in Section 4.3.4.

	M-CT		A-CT			T-Lex
bpt	1	2	1	2	4	1
Binary	0.11	0.29	0.08	0.19	0.44	0.23
Edit Distance	0.22	0.42	0.14	0.29	0.56	0.24
Probability	0.24	0.44	0.21	0.36	0.59	0.29
Feature	0.19	0.39	0.18	0.34	0.58	0.28

Table 5.1: Summary of changes made per tweet in the generation of each data set.

	T-Lex	Manual	Automatic
Number of users	1000	10	1000
Tweets per user	1000	100	1000
Average bpt	1	1 and 2	1
Proportion of tweets hidden in	100%	100%	100%, 50% and 25%
Batch sizes	2, 5, 10, 50, 100		+200, 500, 1000

Table 5.2: Summary of generated steganographic data.

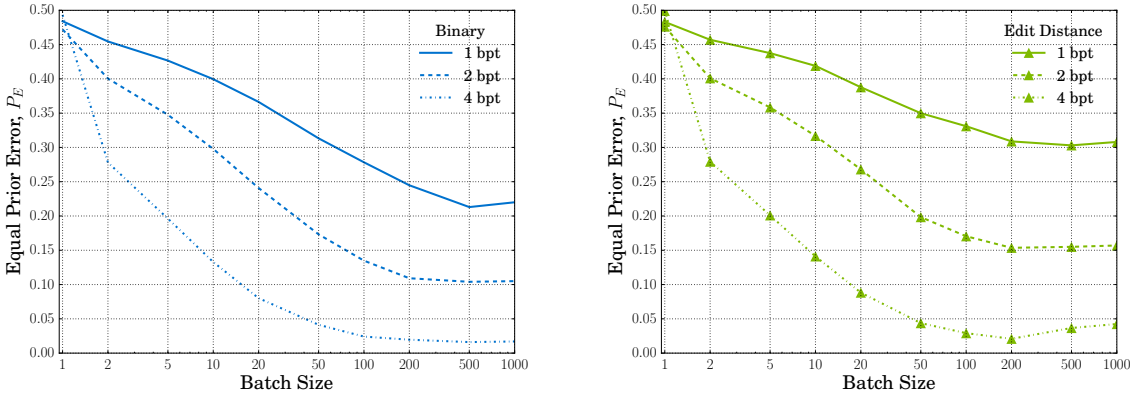
5.6 Results

5.6.1 Automatic CoverTweet

Figure 5.7 shows the results for the experiment on A-CT data generated to minimise each distortion measure, at 1, 2 and 4 bpt. Each graph shows the equal prior error rate P_E of the classifier (trained and tested on data produced with the given distortion measure) against increasing batch sizes.

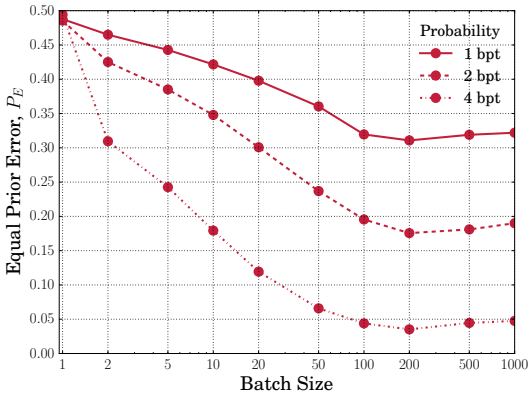
It is tempting to directly compare error rates here, to those in Chapter 4. Comparison with *unpadded* data would not be entirely fair however, as this assumed a very naive steganographer; though we note that coded data appears to be significantly harder to detect than this, as we would hope. We restrict direct comparisons to the *padded* data, which contained the unchangeable tweets in an attempt to mimic a solution to the non-shared selection channel. The application of coding actually solves it, so any difference must either be due to the boost to embedding efficiency (with binary distortion) or the effect of distortion minimisation (with non-binary distortion). The padded data actually has an unfair advantage, as zero capacity tweets were sent as though they contained payload, as long as they had the correct hash.

For all measures, detection is near random on individual tweets, regardless of distortion measure or embedding strategy. This is already an improvement over the results in the previous chapter. On padded A-CT data containing 4 bpt, the classifier achieved a P_E of 0.38 on individual tweets; just minimising binary distortion increases

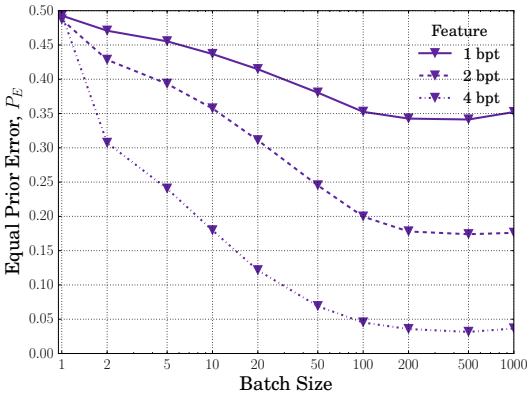


(a) Binary

(b) Edit Distance



(c) Probability



(d) Feature

Figure 5.7: Error rates for classifiers trained and tested on A-CT data, for each distortion, batch size and average payload.

the error rate to 0.49. The difference in P_E must solely be down to the boost to embedding efficiency the binary distortion gets us. Thanks to the efficiency gain, only 8% of tweets have to be changed to convey an average of 1 bpt.

There are similarities to the previous results: pooling causes the error rate to drop significantly; higher payloads are more detectable.

Figure 5.8 compares the error rates for each distortion measure at 1 bpt, allowing use of 100% of the data; the results for the padded A-CT data is also included. The difference between all coded sets and the (uncoded) padded data is significant. At the largest batch size, P_E was 0.05 on padded data, but even with binary distortion the P_E is 0.22 now. Predictably, minimising distortion estimated with refined measures (compared to binary distortion) makes the system significantly more secure. Data generated by minimising feature distortion proved the hardest to detect, due to the fact it explicitly minimises differences in the features that the attacker is looking for. As for the remaining measures, probability is marginally better than edit distance. The difference is slight however, which is notable when considering the overall simplicity of the edit distance measure. Note that in generating this data, we limited the number of allowed options, selecting the best option for each 8-bit hash value according to probability. It is possible that this is the reason the probability distortion does not perform much better than edit distance, as probability distortion has implicitly been partially minimised.

Finally, we investigate embedding strategy. Figure 5.9 shows the results for these tests. On all pooled results, for all measures, concentrating the payload made it slightly easier to detect, confirming our expectations about strategy. With binary distortion and batch size 100, for example, the error rate was 0.28 when allowing the use of 100% of covers, and 0.24 when the same total payload is concentrated into a random 25% of covers. By randomly reducing the number of places the system can hide, we are forcing it to make suboptimal choices for minimising distortion.

5.6.2 Manual data

We have seen how coding (and choice of distortion measure) can affect detection of automatically generated tweets, but the A-CT data is flawed: we assume some automatic tweets are easily detectable to a knowledgable human. Although we have been unable to formally test whether pooling can help a human judge (and we assume that it is never practical for them to do so), intuitively we suspect a human can spot stego tweets such as "`if i go thereafter she willingness go`" (originally "`if i go then she will go`"), and "`you should try to come rearward`

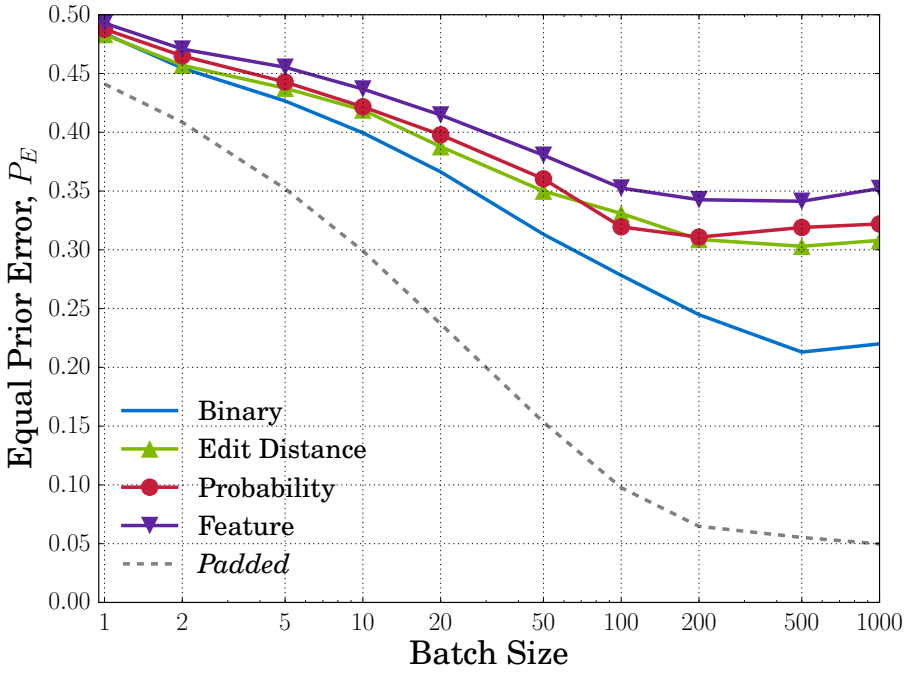


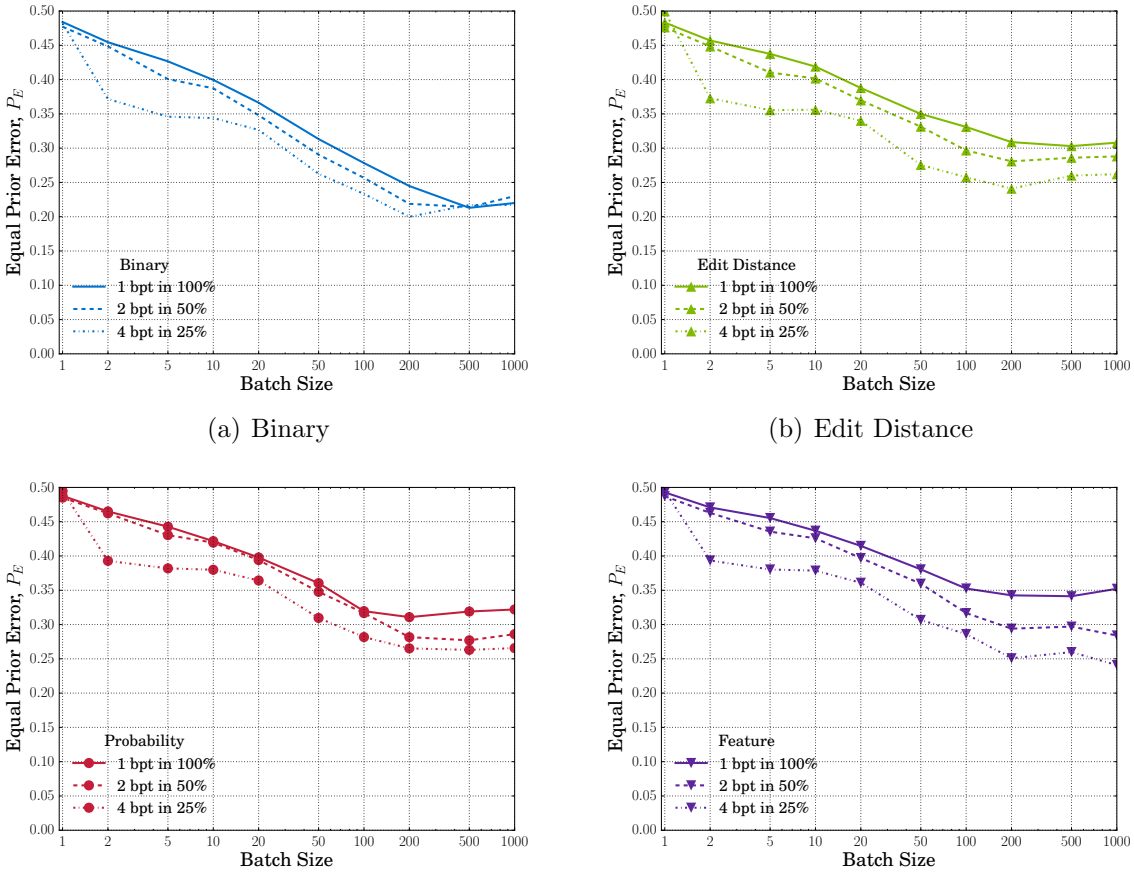
Figure 5.8: A comparison of the most secure embeddings for each distortion type, with A-CT data.

`into town...`" (`rearward` originally being `back`). Just one mistake like this is potentially enough to give away the user's guilt.

Because of this, we are most interested in the M-CT data, produced with the use of two human-driven filters. In Chapter 3 we showed that human filtered data was secure against human judges. Although the production of this data has been slightly altered with an additional filtering step, we believe it is safe to assume that this level of security still holds in the majority of cases.

As in Chapter 4, we trained the classifier on data from 9 users, and tested on the remaining 1. We augmented the training and testing data with additional covers from users not included in the M-CT data: this means we have a good estimate of false positive rate, though our false negative rate is still based on a very small number of testing samples. This was repeated 50 times, 5 times for each user, with a different randomly selected set of additional covers. The equal prior error rate was averaged across all data splits.

We have not extended our exploration of embedding strategy to manual data: we have embedded an average of 1 bpt and 2 bpt in 100% of tweets. Seemingly due to the distribution of distortion, and the presence of a small number of very high capacity tweets (and perhaps the lack of a cap on the number of possible tweet options),



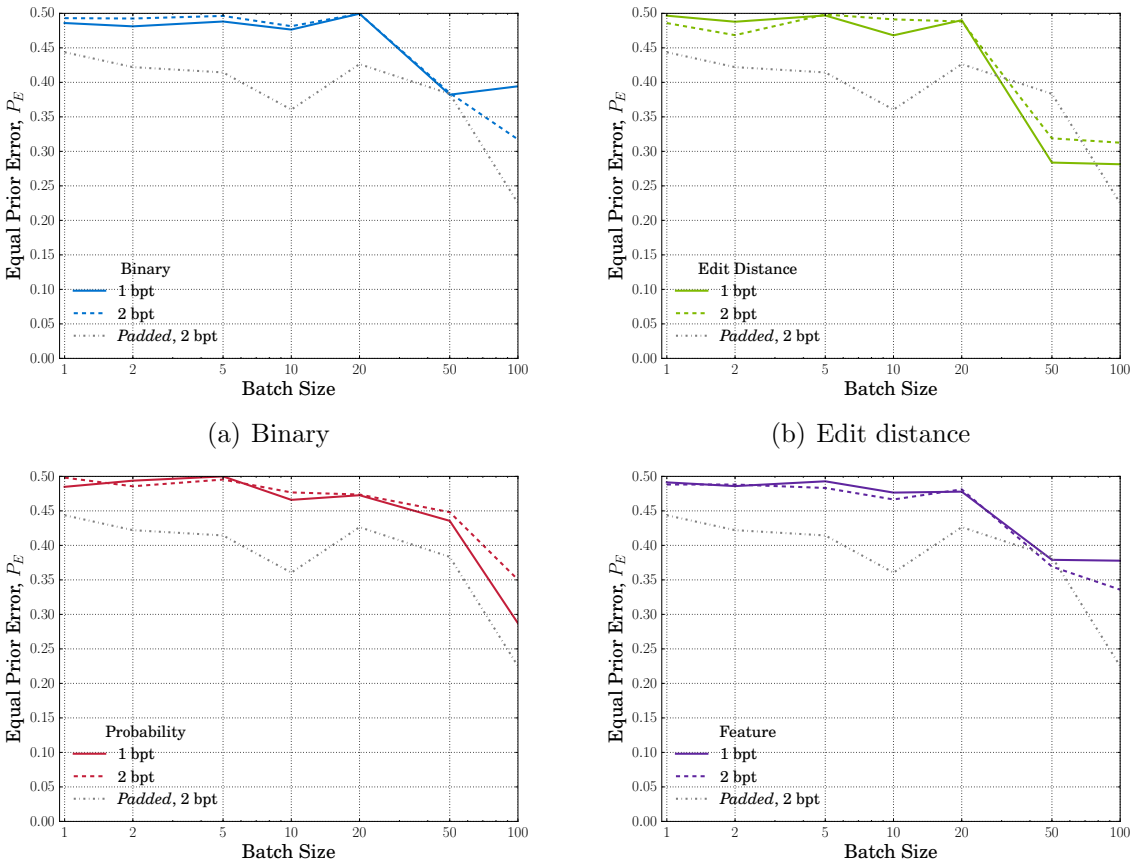
(a) Binary

(b) Edit Distance

(c) Probability

(d) Feature

Figure 5.9: Error rates for classifiers trained and tested on A-CT data, for each distortion and batch sizes. Each set used contained an average of 1 bpt, but in different percentages of the data.



(a) Binary

(b) Edit distance

(c) Probability

(d) Feature

Figure 5.10: Error rates for classifiers trained on M-CT data and tested on M-CT data, for each distortion type and range of batch sizes.

manual data with 1 bpt contains a very high percentage of unchanged tweets. Hiding an average of 2 bpt in a randomly selected 50% of the data actually *increases* the number of changes by a small amount; the strategy has no credibility in this situation.

In the previous chapter, we found that (presumably due to a lack of training data) using only the PPDB features achieved better results at certain batch sizes; this was not found to be the case here, which we speculate is due to the increase in security granted by coding. As such, we only show results for the classifier trained on all features. The results are shown in Figure 5.10.

The application of coding has rendered the manual data essentially undetectable until the largest batch sizes (50 and 100), regardless of distortion measure, or average payload rate. Our small amount of testing and training data leads to uncertainty in our average error rates: we do not expect batches of smaller sizes to be more detectable than those with larger sizes, nor that embedding a higher average payload

should be harder to detect (as the results show for edit and probability distance). We are hesitant to make many claims based on these results, barring two: coding has significantly increased the security of the system (compared to unpadded results in Chapter 4), and we do not have enough manually filtered training data.

Though not included on the figures, the padded M-CT data was comparable to these results at 1 bpt. However, the padded data at 2 bpt is consistently easier to detect than the coded data (barring a few results at the largest batch size, where the lack of data makes error rate unpredictable). We attribute this to the effect of minimising distortion (or maximising embedding efficiency for the binary measure).

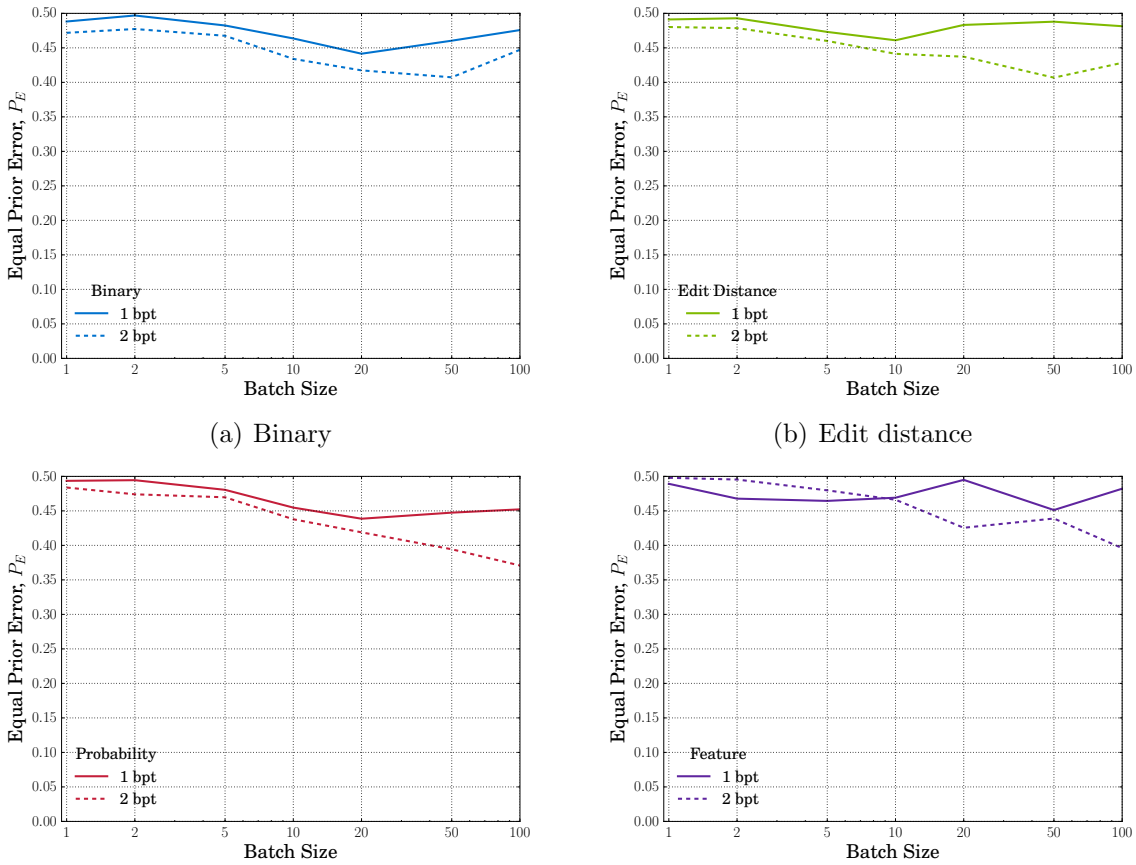
An interesting comparison can be made between our experiments on embedding strategy (Figure 5.9) and these results on manual data. In both cases, an implicit distortion step is applied in the generation of data, assigning either a distortion of 0, or ∞. For the embedding strategy data, this distortion measure acted randomly, and decreased security; in the manual data, the human assigned the distortions, and the security was increased significantly.

Over the course of the next few experiments, we will prod and poke the manual data in an effort to break it. Failing that, we may learn more about *why* it is (or seems) so secure, beyond the lack of data.

Utilising Automatic Data

First, we looked to the vast amounts of *automatic* data we have, in the hopes that a classifier trained with enough A-CT data will be capable of detecting M-CT. We trained the classifier on all A-CT data, minus the users for whom we had manual data. The testing set was made from all M-CT data, with the addition of all cover instances from the same users, giving 10000 cover tweets and 1000 stego tweets for testing. For each average payload and distortion type of M-CT data, we used the same for the A-CT training data (so we were training with tweets generated to minimise the same distortion that we were testing with).

Figure 5.11 shows the results of these experiments. Though results are still noisy (P_E increasing at larger batch sizes, for example), we can rule out under-training as a cause for security in this experiment. On data containing an average of 1 bpt, the classifier performs not much better than random guessing, even at the largest batch size (unlike when the classifier was trained and tested on M-CT data). We do see a clear difference between 1 bpt and 2 bpt data however, with the data containing the larger payload marginally easier to detect for all distortion measures.



(a) Binary

(b) Edit distance

(c) Probability

(d) Feature

Figure 5.11: Error rates for classifiers trained on A-CT data and tested on M-CT data, for each distortion type and range of batch sizes.

In these experiments we do not know to what extent the security is due to coding, or to the problem of *model mismatch* (also known as *domain adaptation*) [Ker et al., 2013, Ker and Pevnỳ, 2014a]. In-domain (manually filtered) data is prohibitively expensive to produce, but we have an abundance of automatic data, which is possibly too dissimilar to the manual data to be completely of help here. This is a problem well known to the steganographic community; finding a solution in this instance is beyond the scope of this work. The key to detecting manual data may lie in the generation of better quality automatic data.

Anti-Distortion

Despite our attempts to detect coded M-CT failing, except at the largest batch size, we can still check that the distortion measures perform as expected. To do this, we reversed each distortion, creating an *anti-distortion* measure. With this reversed

measure, the stego option with the highest distortion is given an anti-distortion of 0; the cover tweet is given this highest value as an anti-distortion.

New coded stego sets were produced with the measure, in an attempt to simulate the worst possible embedding by *maximising* distortion, rather than minimising. If a measure is a good estimate of distortion, then maximising it should reduce security of generated stego. If the security of M-CT data is entirely due to a lack of training data or a weak attack, we do not necessarily expect that maximising distortion would significantly damage security.

Figure 5.12 shows the results for this experiment, when trained and tested on distortion maximised M-CT data. There is a steady reduction in error rate as batch size increases; at batch size 50, the error rate is 0.16 for the anti-feature distortion. There is a difference, albeit a noisy one, between distortion measures: maximising the feature distortion appears to creates steganography that is easiest to detect, followed by probability. Maximising edit distance creates the least detectable steganography, not binary distortion, as we had expected. Naturally, signs of under-training are still present.

The success of this attack against distortion maximised steganography is a sign that not all human filtered data is simply undetectable by nature; perhaps, in large enough quantities, with a large enough payload, our classifier would be able to detect it even when minimising distortion.

Limiting Options

We limited the number of options allowed when generating the A-CT data, as an unfortunately necessary measure to avoid computationally infeasible amounts of data. We considered that by not limiting the number of options for M-CT data, we potentially gave the system the opportunity to convey the majority of payload in a few very high capacity tweets. If this were true, then it would be possible that limiting the number of options allowed by M-CT would produce steganography which is easier to detect.

In order to test this, we severely limited the number of M-CT options, initially to the same as A-CT (256 options), then to just 4 options. We generated stego containing an average of 1 bpt with these limited options. In either case, the results were the same: equally undetectable steganography (P_E of $\sim$0.48 at batch size of 20), whether the classifier is trained on M-CT data or A-CT. With more training data, it is possible that limited steganography would prove easier to detect.

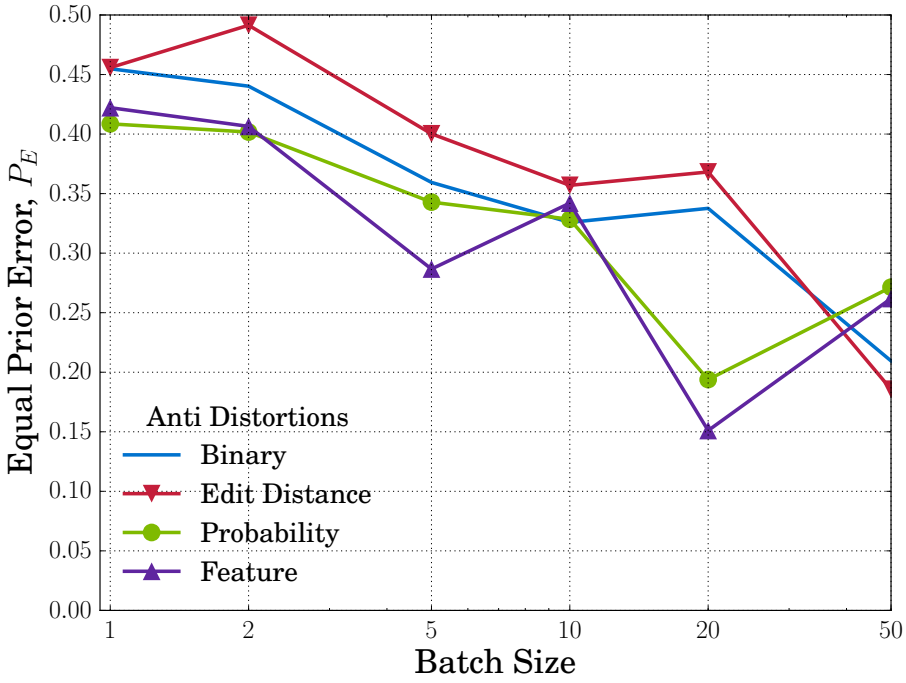


Figure 5.12: A comparison of error rates against M-CT steganography coded to maximise each distortion type.

Beyond merely seeing if we could weaken the M-CT data enough to detect it (we could not), this task mirrors practical coding considerations. Simulating optimal embedding has allowed us to use arbitrarily large payload alphabets (q), but at present there is no coding method which would allow us to do this in reasonable time. Recall that the space and time complexity of STCs is $O(nq^h)$; then note that one tweet in our data set has 15000 options. If generated steganography is equally undetectable when $q = 4$, using coding in practice (as opposed to simply simulating it) may be feasible.

5.6.3 T-Lex

Figure 5.13 shows the error rates of classifiers trained and tested on T-Lex data. While choice of distortion measure makes a slight difference to detectability, coding is unable to significantly help T-Lex when pooled steganalysis is deployed.

Once again, the simple detector described in Section 4.4.3 performs better than our full attack. Figure 5.14 shows the results for this detector with each distortion measure. Interestingly, binary distortion produces the most secure stego; this is likely because it aims specifically to minimise the number of changes, compared to other

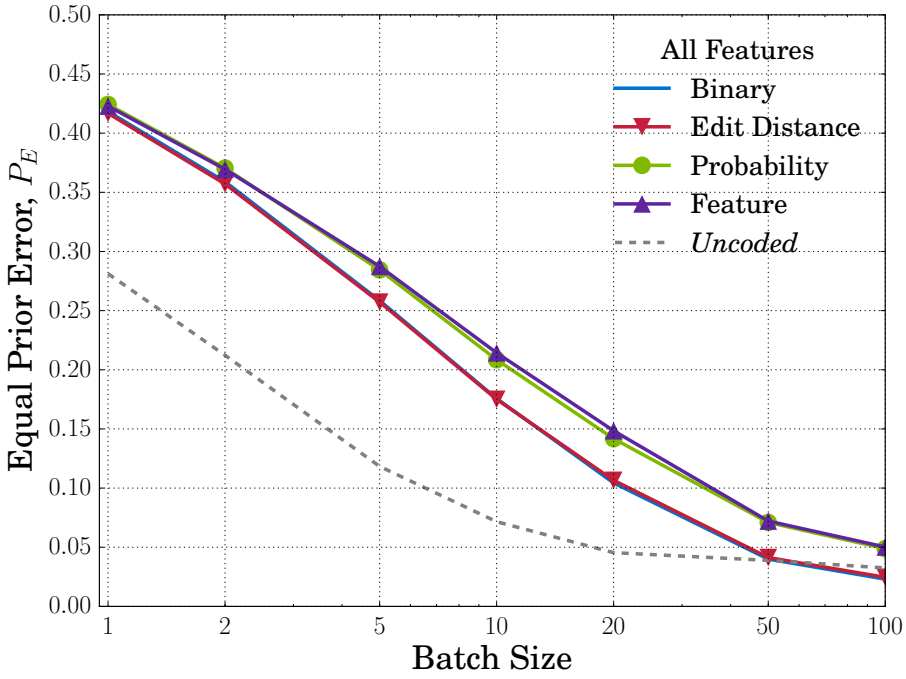


Figure 5.13: A comparison of error rates for the classifier trained on T-Lex data.

measures which make more, lower cost changes. Given almost all changes are marked as stego by this detector, more changes simply means the stego is more detectable. The second best distortion measure is edit distance, which will be equivalent to binary distortion on a large number of T-Lex tweets.

There is not much difference between feature and probability distortions, although the error rates for probability distortion are fractionally lower. We can possibly attribute this to the probability distortion making marginally more changes: 0.29 changes per tweet, versus 0.28. The binary and edit distance distortions made 0.23 and 0.24 respectively.

5.7 Square Root Law

It is known that there is a sub-linear relationship between the total capacity and the number of cover objects [Ker et al., 2008], as long as the steganographer cares about security. The *square root law* of steganography states that if a steganographer embeds a total of M bits into N cover objects, then:

- If $M/\sqrt{N} \to \infty$ as $N \to \infty$, then there is a pooled detector which, for sufficiently large N, comes arbitrarily close to perfect detection.

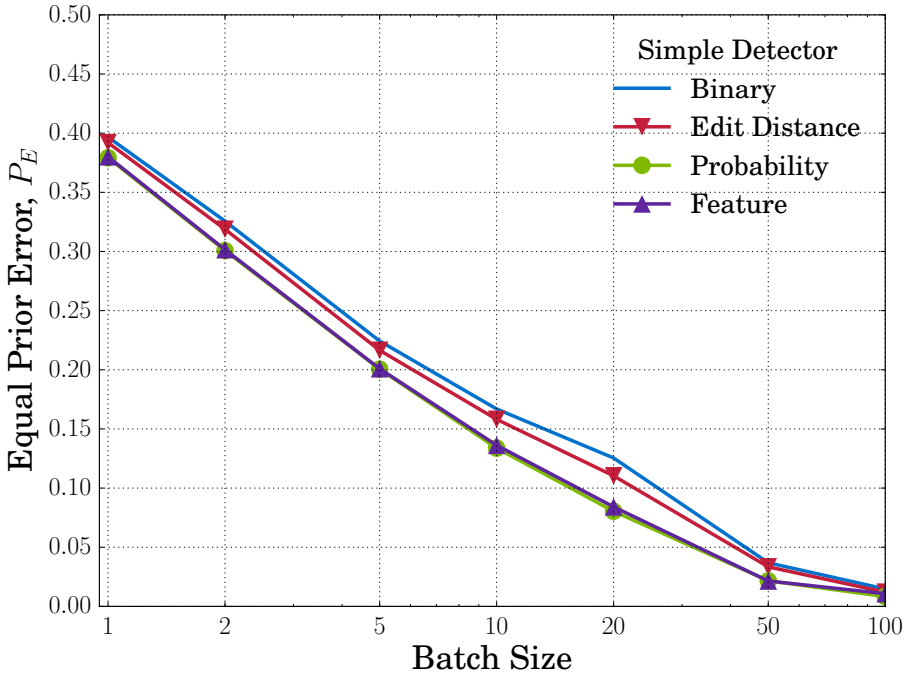


Figure 5.14: A comparison of error rates for the simple statistical detector of T-Lex.

- If $M/\sqrt{N} \to 0$ as $N \to \infty$, then the performance of any pooled detector must become arbitrarily close to random for sufficiently large N.

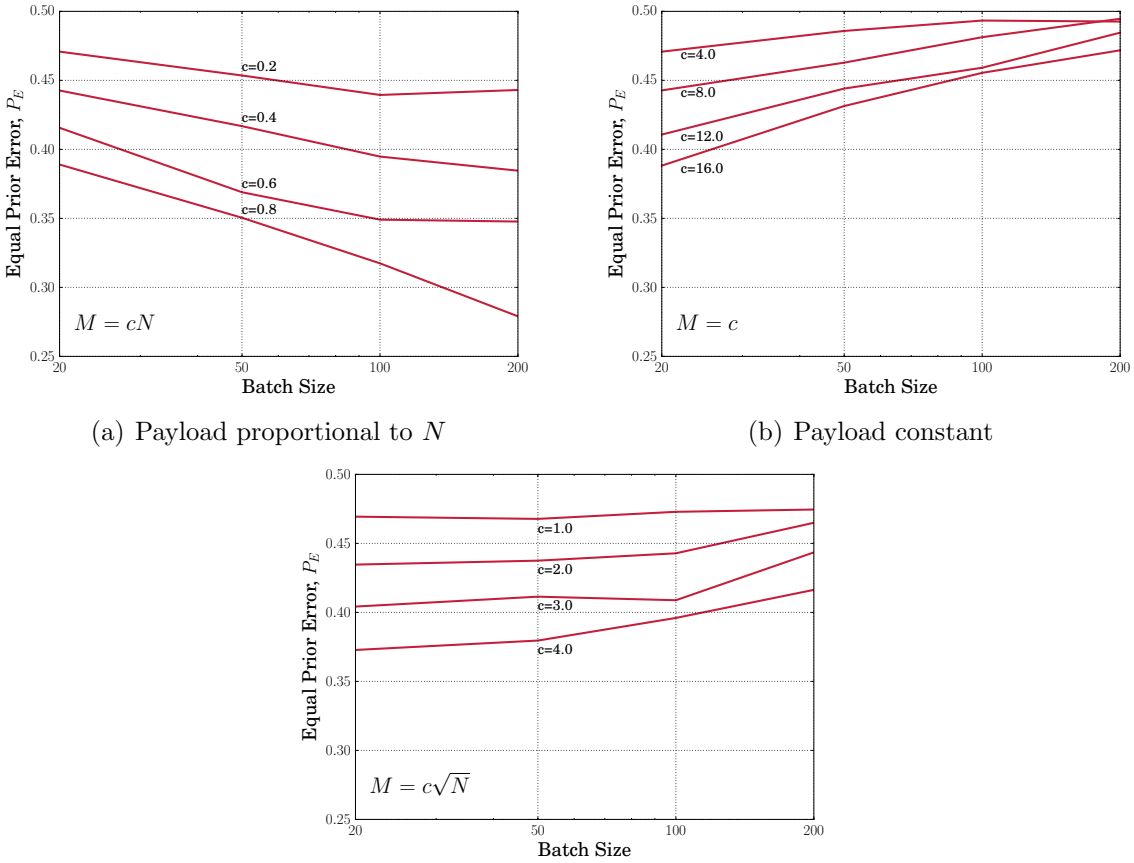
This is a theoretical result, but it has been observed robustly in image steganography. As a step toward serious linguistic steganalytic research, we will show empirically that this also holds also for our linguistic stegosystem, adding to the growing evidence for the square root law.

To show this, we create batches of our data containing N tweets, coded to convey a total payload size of M bits; for these experiments we used A-CT data, given M-CT data is currently undetected.

First, we let $M = cN$, for a range of constants c, and trained our classifier on these batches for an increasing size of N. With this linear payload, $M/\sqrt{N} \to \infty$ as $N \to \infty$, so if the theoretical result holds, detection should increase with N.

Second, we used a constant payload, $M = c$. Clearly, $M/\sqrt{N} \to 0$ as $N \to \infty$, so detection should decrease as N increases.

Finally, we set $M = c\sqrt{N}$. Note that the law applies to embedding changes, rather than raw payload. Here, where we are simulating the application of perfect coding, the two are not proportional; we know from Figure 5.1 that decreasing relative payload increases embedding efficiency, and we hide more information per change.



(a) Payload proportional to N

(b) Payload constant

(c) Payload proportional to $\sqrt{N}$

Figure 5.15: Equal prior error rate versus batch size for different amounts of payload M, set to be proportional to N in (a), constant in (b) and proportional to $\sqrt{N}$ in (c).

The additional information conveyed by each change comes from the following: when we have a choice of changes we could make (as we do when relative payload is less than 1), our *choice* of change also conveys information. As such, steganographic capacity is actually of order $\sqrt{N} \log N$, rather than $\sqrt{N}$ [Ker, 2010]. As we are setting $M = c\sqrt{N}$, we expect detection rate to decrease slightly with N.

Figure 5.15 shows the results, which confirm the theoretical predictions: when the payload size is proportional to total cover size, the detection rate increases with the batch size; when the payload is constant, detection rate decreases; finally, when the payload is proportional to the square root of the total cover size, detection rate stays approximately the same, albeit with a slight drop as expected.

5.8 Summary

In Chapter 4, our attack went up against a naive steganographer, who had to discard unusable tweets. This was found to be a significant weakness to the security of the system, as batches of covers were easily detectable due to the increased presence of tweets with few steganographic options. The cause of the problem was the non-shared selection channel; a problem well known in the image domain, solvable through the application of coding. We have applied coding to the linguistic domain for the first time, by simulating optimal embedding.

In addition to removing the need to discard tweets, coding also allows us to boost embedding efficiency to convey more information per change, and minimise *distortion*. This latter concept is known as *adaptive embedding*, and is a standard approach in image steganography. To practise it here, we created the first linguistic distortion measures.

In results, we saw that coding increased the security of the system. In the case of automatically generated A-CT data, the error rate of the classifier was significantly higher than on comparable uncoded data from the previous chapter, especially when utilising the feature distortion measure, which explicitly attempted to minimise the efficacy of the attack. In the case of manually generated M-CT data, the stego tweets were rendered essentially undetectable, regardless of payload rate, except at the largest batch size.

With more M-CT data, the attack may be successful; however, this data is expensive and slow to produce. Future work will need to find a solution before the field can continue to evolve. This may lie in the application of solutions to *domain adaptation* or *model mismatch*, the production of A-CT data closer in quality and content to M-CT data, or in a streamlined approach to manual data generation.

The application of coding to other linguistic stegosystems is of paramount importance to research. Systems with a small payload alphabet can implement coding simply, utilising *syndrome trellis codes* in conjunction with hash functions for value assignment. Alternatively, the simulation of optimal embedding can enable the evaluation of new stegosystems without worrying about implementation, the step that has seen many published systems throttle capacity to achieve.

This chapter was adapted from the paper "Avoiding detection on Twitter: Embedding strategies for linguistic steganography" [Wilson and Ker, 2016]. The use of anti-distortion and limited options against M-CT, along with the results for T-Lex, are new for this work.

Chapter 6

Conclusions

It concludes as follows:
"Poo-tee-weet?"

Kurt Vonnegut, Slaughterhouse-Five
via CoverTweet

Prior linguistic steganography literature has struggled with the same issues again and again: an inability to assign values without putting severe restrictions on the transformation; tiny payload size; human-detectable mistakes in transformation. In addition, the vast majority of publications have declined to evaluate security, neglecting the steganography aspect in lieu of evaluating stegosystems for computational linguistic merit. When the importance of security is remembered at all, it is often judged only by the author's intuition (e.g. [Desoky, 2012]), especially in the case of systems employing human input, or utilising "invisible" transformations.

We have attempted to rectify mistakes made in earlier work. The system we have developed uses *hashing* to allow for free transformation, in exchange for some computational cost (see Section 3.8), and we have utilised the steganographer themselves to provide a level of guaranteed fluency. Any system using hash functions for value assignment, a human filter, or non-symmetric transformations (which the system is capable of, though did not use in this work), is weak to the non-shared selection channel problem. Noting that this problem has already been solved in image steganography, we applied *coding* to solve it in the linguistic domain.

The security of our system was investigated three times, first by a human censor, then by an automated one, once more after using coding to minimise distortion. The human censor, investigating tweets individually, achieved a detection rate no better than random (Figure 3.22); utilising a human filter during embedding successfully disarmed the human attacker. The automated censor, with full knowledge of the

system, achieved what the human could not. *Pooling* the evidence against each user saw the successful detection of manually generated stego objects (Figure 4.7), and the near perfect detection of automatically generated ones (Figure 4.6). Kerckhoffs' attackers were found to be powerful adversaries; the early stegosystem T-Lex was decimated by a trivial attack (Figure 4.10), utilising knowledge of the system. The main weakness of T-Lex is possibly due to the fact its transformations are all disjoint, allowing the steganalyst to generate the exact set of steganographic options for a given object. This should be taken into account when designing future systems.

Having simulated *optimal embedding*, and minimising the first linguistic distortion measures, the manual data was rendered mostly undetectable without large pooled batches of stego objects (Figure 5.10). The failure of the detector is at least in part due to the small amount of manually filtered training data available; acquiring more was beyond the scope of this work. The expense of data generation is perhaps a unique aspect of linguistic steganographic security. It may be that the development of truly successful attacks relies, in part, on the design of intuitive interfaces for rapid data generation.

Despite years of separate research threads, we contend the majority of the underlying problems of linguistic steganography are fundamentally the same as image steganography; albeit with different parameters. The consequence: linguistic steganography can benefit significantly by applying the most important findings from image steganography, which we have done here.

With regards to our expectations given in Chapter 1:

1. Linguistic steganography produced with manual input was undetectable to human judges, at least when evaluated individually.

2. Automated attacks initially had success with detection, especially with large pooled batch sizes.

3. Coding allows the steganographer to send the previously un-sendable: objects incapable of conveying the desired payload. In addition, we can minimise distortion, and maximise embedding efficiency. Many existing linguistic stegosystems have restricted their transformation in order to avoid the non-shared selection channel problem, which coding fixes. Coding therefore has the potential to be of significant help to future, and existing, stegosystems.

4. Once the steganographer used coding to perform adaptive embedding, the steganography produced with manual input was undetected by our attack.

We hope that this work stands as a template for future research in the area. Consider the implementation of a new stegosystem based on a unique linguistic transformation. Recalling the three tasks of linguistic steganography (transformation, value assignment, and coding), the solution at present is to ignore coding, and implicitly solve value assignment by restricting transformation. The solutions presented here – hashing and coding – can be applied to any new transformation; indeed, simulating coding removes even the need to implement a full system for evaluation.

With regards to this evaluation, it is standard in image steganography to test new stegosystems against pre-existing attacks. A new system is hardly of worth if it is weak to known steganalysis methods. This concept has eluded the linguistic field, certainly in part due to the severe lack of appropriate steganalysis literature. New work could perhaps utilise the attack we presented in Chapter 4, as a step in the direction of sound steganographic research. Note that the PPDB-derived features (see Section 4.2) of our attack can be trivially altered to make use of another transformation source, if appropriate.

6.1 Applications of Linguistic Steganography

The elephant in the room: *what is linguistic steganography useful for?* Even when optimally embedding payload, our manually filtered tweets were capable only of 2 bits per tweets; if every single Twitter user was hiding in every tweet, the total global covert bandwidth of Twitter would only be approximately 1.5 kB/s.

While we have obviously not reached the limit of steganographic capacity in tweets (see Section 6.2.1 for transformation suggestions), recall that the estimate of English tweet entropy is 250 bits [Neubig and Duh, 2013]. If the conditional entropy of a tweet (which steganographic capacity is bound by) was equal to the full entropy of a tweet, the global covert bandwidth would still only be 187.5 kB/s.

A conclusion which could be drawn from our work is that linguistic steganography is a *high security*, *low capacity* channel, and could therefore see use in steganographic key exchange protocols. One hindrance is the requirement of a key in order to perform linguistic steganography in the first place; there exist public key steganography methods which would enable this [Craver, 1998], but these come with their own issues.

The claim of high security is a speculation based on our observed results in Chapter 5 (the inability of our attack to detect manually generated stego). Results from [Holub and Fridrich, 2015] show hiding 0.2 bits per DCT coefficient in JPEG images (with quality factor 75) is detectable with 40% accuracy by the JPEG Rich Model

[Fridrich and Kodovsky, 2012], a modern, but not state-of-the-art technique. A mean of 0.9 nonzero DCT coefficients per byte of compressed cover was observed in images taken from Facebook (which has a similar quality factor); this translates to a relative payload of approximately 0.02 bits per cover bit, after compression. Our tweets are hiding about half as much relative payload (2 bits in an object with an entropy of 250 bits), but are completely undetectable at present, even when pooling. The square root law makes this comparison dubious, however, as images are much larger than single tweets.

An idiosyncratic form of linguistic pseudo-steganography already in use is the burgeoning field of *automated plagiarism*. The task here is to pass purloined essays off as ones own, getting past automatic plagiarism detectors; successful detection of plagiarism is important in schools and universities, but also amongst commissioners of online articles (especially of the low quality variety used for *search engine optimisation* purposes).

The goal here is subtly different to true steganography: exact payload is unimportant, but the generated text must be sufficiently different to the original. This has similarities with the *distortion limited sender* paradigm of optimal embedding, where the payload is maximised such that the average distortion is equal to a variable D. The standard approach for plagiarising (embedding) is to employ simple synonym substitution. The following, generated with an online paraphrasing tool [Spinbot, 2014][1], may look both suspicious and familiar:

> "Two detainees, secured separate cells, are permitted to compose letters to each other. They need to arrange their departure, however the jail superintendent is onto them, and deliberately peruses each letter before passing it on. Cryptography is not feasible, the superintendent will most likely pulverize any letter containing any clearly scrambled message. The detainees must figure out how to arrange furtively, while seeming, by all accounts, to be composing consummately blameless messages to each other. The arrangement: steganography, the specialty of concealing data."

Simple plagiarism detection works by comparing text to a reference corpus [Barrón-Cedeño and Rosso, 2009] (or querying Google), to establish whether lines in the document have been sourced from an existing article. Detection of *paraphrase plagiarism* is an open problem [Potthast et al., 2009], and linguistic steganalysis is spectacularly similar to this task. Ignoring the use of a reference

[1]There are many other easily available tools, each performing the same task.

corpus, the question is the same: has this text been changed automatically? This similarity has previously been noted with regards to linguistic watermarking [Fu and Zhou, 2010], though we suspect the authors were making the common mistake of confusing linguistic watermarking with linguistic steganography. We speculate, based on what we have seen of automatically paraphrased text, that our attack could be successfully adapted for use in plagiarism detection, without the need for a reference corpus. Detection of plagiarism without this corpus is known as *intrinsic* plagiarism detection, and a number of techniques exist for this task (e.g. [Stamatatos, 2009, Zechner et al., 2009, Oberreuter et al., 2011, Stein et al., 2011]). Evaluation of these methods for linguistic steganalysis would certainly be of interest.

Viewed from a steganography perspective, the task of paraphrasing for plagiarism has an interesting detail: despite payload being irrelevant, *capacity* is still important. If a stolen sentence has zero capacity, it cannot be sent without partially signalling the document as plagiarism, so should be removed. Before we introduced coding, tweets incapable of conveying the correct hash were discarded; we observed that this action harmed the security of the system.

A final note on this topic is that the area is, like linguistic steganography, vulnerable to human attackers. This is a problem for plagiarisers in schools and universities where the warden (the teacher) can trivially spot odd language; the term *Rogeting* [Times Higher Education, 2014] (named after the thesaurus) was coined by one lecturer to describe the act, after noticing the substitution of "`sinister buttocks`" for "`left behind`" in an essay.

6.2 Further Work

Discussions of avenues for future research are grouped, roughly corresponding to the work in Chapters 3, 4 and 5.

6.2.1 Embedding

We intentionally avoided the use of any transformation which is only applicable to Twitter, or similar settings. This area is fertile ground however, with a number of possible transformation options. The capacity of linguistic stegosystems is directly linked with the transformations used, and in a practical setting stegosystems should likely use as many as possible (while minimising distortion). We suggest some here, but acknowledge that this is merely scratching the surface of possibilities.

Social Media Transformations

Twitter is not the only noisy setting for linguistic steganography. A number of techniques are applicable to essentially any personal communication channel in use by the correct demographic of steganographer. Tweets and other personal messages are rife with spelling and typing errors, along with intentional misspellings and abbreviations, all of which could be a source of transformations. For clarity, we define spelling errors as words which are spelled incorrectly through ignorance (e.g. `accomodate`, `apparantly`), and typing errors as words which are spelled incorrectly during input (e.g. `teh`). Intentional misspellings and abbreviations are tokens used consistently, and purposefully, by a user: one user in our data always used the token `miiyy` for `my` and `me`.

Exploiting these is not a new idea for steganography: [Topkara et al., 2007] and [Shirali-Shahreza and Shirali-Shahreza, 2007] use transformations based on typing mistakes and so-called "text speak", but do not take due care with these transformations. When exploiting intentional misspellings and abbreviations, a degree of *consistency* must be maintained, if the attacker is known to be utilising pooled steganalysis. Users tend to have a particular slang distribution, and this type of transformation must attempt to maintain this distribution for stego objects, as with any transformation. Maintaining consistency may prove too difficult for the area: this requirement places us firmly in the realm of non-additive distortion. Once a misspelling (or error) has been used once, the probability of the same word appearing correctly spelled, or with a different error, reduces drastically.

When exploiting spelling and typing errors, both consistency and *feasibility* must be maintained. Maintaining feasibility is easier: for typing errors imagine minimising a distortion based on *keyboard distance*, the physical distance between the correct spelling of a word, and mangled stego.

Note that the application of this transformation would likely be impeded in practice by the *autocorrect* methods employed by many modern text input devices. Autocorrect is acting somewhat like an unintentional *active warden* against steganographers wishing to hide in typing errors. Of course, autocorrect also makes mistakes, and the imitation of these is itself a viable hiding place (to be used sparingly, if the steganographer does not wish to be caught).

Another possible transformation would be that of *elongating words*. This is a common practice on Twitter, used for emphasis (e.g. `omgggg`, `yesss`). Transforming a tweet would involve simply adding or removing letters to words, though distortion would need to be estimated based on a model of the likelihood of repeating particular

letters, or elongating particular words; a character-level edit distance would likely be an appropriate starting point. This type of transformation has the potential for adding a lot of capacity to tweets, while likely being invisible to humans if implemented correctly. It equally has the potential to be misconstrued as "undetectable": once again we stress the importance of testing security on any new system. Features looking at the variance in number of letters at the end of a word may be of use in detection here. We also note that the exact set of possible stego objects is retrievable by the attacker; this was seen to be a significant weakness of T-Lex in Section 4.4.3.

Twitter Transformations

We suggest two systems exploiting features specific to Twitter: *hashtags* and *retweets*.

CoverTweet altered the ordering of hashtags (categorising tokens proceeded by the symbol `#`), having noted that order is unimportant in many cases. The system did not attempt to change the hashtags themselves, though we can imagine one which replaces, deletes, or appends additional hashtags. Distortion in this situation would require estimating the suitability of a particular hashtag in a given context. The manipulation of capitalisation within a hashtag is also a possibility; many hashtags are nearly equally likely all lowercase, or in camel-case. [Declerck and Lendvai, 2015] looked at the distribution of the token `#LondonRiots`, with different capitalisations. `#LondonRiots` and `#londonriots` were used nearly the same number of times, with a few users posting `#Londonriots`. Much like intentional misspellings, distortion here is likely to be non-additive. Once a hashtag has been used once with particular capitalisation, it should (at least if used in short succession) appear with the same capitalisation the next time.

Twitter offers users the ability to 're-tweet' an existing message, which reposts it so it appears on the user's list of messages. In Chapter 1, cover selection methods – where a cover with the correct payload is chosen from a large set – were dismissed, due to the vast set of covers required. However, a potentially viable scenario would be to utilise *retweets*, and selectively retweet messages with the correct hash; the user could continue to use their account as normal, as retweets are clearly marked as such. This is another steganographic technique which has the potential to be incorrectly thought of as undetectable. It is an example of a *rejection sampler* [Hopper et al., 2002], which has perfect security if tweets are independent, but we do not think they are; in practice, this would likely be detectable to a Kerckhoffs' attacker.

Finally, the order tweets are sent in is often unimportant, so we can potentially *reorder* them to embed additional payload. Practical considerations likely rule this

out as a viable option; there is currently no coding method which could reasonably exploit this fact, without exponential increases in complexity.

One non-linguistic way to convey additional information on Twitter could be through the *timing* of posts. This has precedent in steganographic literature, having been used to convey information through the timing of packets across networks [Liu et al., 2010].

Beyond Semantic Preservation

As discussed in Chapter 2, the standard approach to linguistic steganography is to attempt to preserve the meaning of the original cover. This approach is often practical, but it is by no means a true requirement of the field (though some literature treats it as such, e.g. [Taskiran et al., 2006], [Yuling et al., 2007]). We can imagine the expansion of transformation scope, by allowing for broader changes to tweets.

Imagine the following artificial tweet:

> "went to the cinema with Kate tonight, had way too much popcorn"

At present, the majority of literature regards the set of possible stego objects as the set of sentences that have the same meaning, e.g.:

> "went to the cinema with Kate tonight, *ate* way too much popcorn"
> "went to the cinema with Kate tonight, had *far* too much popcorn"

Consider that we could add to this set the collection of tweets which contain less information than the original, or are *logically entailed* by it:

> "went to the cinema tonight, had way too much popcorn"
> "went to the cinema with Kate tonight"
> "had way too much popcorn tonight, I feel sick"

This type of transformation has already seen application in some specific areas (e.g. [Chang and Clark, 2012]). We can arguably also add to the original with the addition of previous unsaid information:

> "went to the cinema with Kate tonight, *to see the new star trek film*, had way too much popcorn"

[Vybornova and Macq, 2007] applies this idea to natural language watermarking, using *presupposition*. The system attempts to insert information which was implicit to the original cover. We propose that additional information need not be implicit; nor even true. The above tweet could state *any* film. There is potentially no limit to this style of transformation: we could freely add new information (false or otherwise), remove existing, or change details. For example:

> "went to the cinema *on my own* tonight, to see the new *jason bourne* film, had *a hotdog*"

Ignoring practical considerations (generating these options would be a serious undertaking, though could potentially make use of the human), evaluating the security of such a scheme is interesting. Once again, transformations cannot be assumed to be undetectable without analysis, and this is no exception. If we assume the system achieves perfect fluency, the distortion measure would be based on the semantic feasibility to a knowledgable attacker. If the warden knows the system employed, they know that details are being changed, and a knowledgable human detector might be suspicious of the following:

> "went to the moon with Kate tonight, had way too much cheese"

This example is perhaps slightly unrealistic, as it should stand out to a language model ("went to the moon" is presumably less likely than "went to the cinema"), as well as for reasons of believability.

The key when abandoning semantic preservation is that systems need not only move one "step" away from a cover. Iterative transformations could move a stego object semantically far away from the original, as long as the steganographer approves.

Note that any work hoping to exploit *lying* as a transformation would require careful consideration of what it is reasonable for the warden to know about the steganographer's life. Some things may be impossible to conceal: [Jaech and Ostendorf, 2015] predicted the gender of OkCupid and Snapchat users with 75% accuracy using only usernames.

Distortion

We presented the first linguistic distortion measures here, and we expect them to be beaten by future work. Our measures are specifically appropriate for CoverTweet's transformation (and our attack); new transformations may require new measures, such as the proposed 'keyboard distance' for typing error steganography.

Without better detection (see the next section), it is hard to evaluate the need for better distortion. However, if we view the human as an implicit distortion step in the generation of manual data, it is clear that better distortion measures are possible.

6.2.2 Attacking

While of mixed success, the results of our attack against CoverTweet made one thing abundantly clear: we need *more data*. We potentially also need a more powerful attack, which may come in the form of additional features; of course, as we saw in Chapter 4 (Table 4.7), the addition of features exacerbates the need for more data.

Data

The generation of quality data is paramount to further linguistic steganalysis work. This is true for both manual and automatic data.

Our manual data is likely good enough quality, but we require significantly more of it to mount a successful attack. It takes an approximate average of 2.4 minutes to generate all steganographic options of a single tweet; 40 hours to generate the 1000 steganographic tweets we have in our data set. It would take a single user 4.5 years (non-stop) to generate the same amount of Manual CoverTweet data that we have for Automatic CoverTweet. Though gathering the same volume of M-CT data as A-CT is infeasible, the generation of a sufficient amount may be possible through the use of a crowdsourcing site such as Amazon's *Mechanical Turk* [Amazon, 2005] or *CrowdFlower* [CrowdFlower, 2007].

Note that the author was directly involved in the generation of all manual data used in this work; the aforementioned crowdsourcing sites could allow future work to avoid this potential source of bias, but they should not be used without careful consideration. Mechanical Turk is popular for computational linguistics tasks [Callison-Burch and Dredze, 2010], but is ethically dubious [Fort et al., 2011], and the precious guarantee of fluency that our human granted could be eroded through its use (for example if the human is inattentive, or has not had the task adequately explained to them). Neither solution (crowdsourcing, or authorial involvement) is ideal;

humans are unlikely to agree on the suitability of steganographic tweet options, so perhaps the correct method of data generation is to employ genuine Twitter users to hide data within their own tweets. However, this would come with the same caveats as crowdsourcing, with additional time constraints.

One way to speed up the generation of manual data could be to include more automation within the system. Currently it uses few sophisticated techniques: no parser, no part-of-speech tagger. Early, automated detection of poor substitution options would alleviate the workload of the human. Any changes to the system would naturally require new security evaluation; it would be important to establish whether increases to detection rate are due to a weakened system, or an increase in training data.

For automatic data, there is no problem with quantity, but with quality. It is possible that improved automatic data could be used to aid the detection of manual stego; regardless, improving the generation of automatic data is surely a general goal of linguistic steganography. Recently, the PPDB was released as a tuned, prepackaged model for black box monolingual machine translation [Napoles et al., 2016]; this could be of great use to linguistic steganographic research. The trivial addition of a hash function essentially provides an easily available stegosystem. The model can be tuned for specific purposes; in steganographic settings, it could perhaps be tuned on *manual data*.

The generation of both data sets in Chapter 5 (M-CT and A-CT) utilised implicit distortion minimisation. The manual data used the human to assign some options infinite cost; the automatic data limited the number of options allowed, picking the best options based on probability. It would be of interest to see how the security of automatic data would be modified if this implicit distortion step minimised a different distortion (feature distortion, for example).

Note that some transformations may be applicable without human input (any "invisible" transformation, such as elongated words), but stego generated with these methods would need to be evaluated against human judges to confirm this. Any automatic transformation which is undetectable by a human likely does not require manual intervention during embedding.

The Attack

There are a number of ways in which our attack could be expanded, or altered. For example, our attack utilised only one language model, and one word length model. The attack could be extended by using multiple, trained on different data sets, with

different parameters. The idea behind this would be that different language models could detect different non-fluencies.

Another approach would be to investigate additional pooling techniques, beyond maximum, minimum, mean and variance. The best way to build a pooled detector is an open question in steganography [Ker et al., 2013], and alternative pooling methods could potentially achieve what ours could not.

This was the first work on linguistic steganalysis to use a large data set, and there are significantly more features which could be utilised (e.g. KL divergence, preposition usage). Compared to modern image steganalysis however, the feature set is tiny. For example, the set described in [Fridrich and Kodovsky, 2012] has 35k features; ours had 812, after pooling.

In this work, we assumed the automated censor knew every detail of the system in use. Future work could investigate detection when the warden's knowledge is less than perfect. For example: when the steganographer has a random subset of the PPDB; when the attacker only has access to an entirely different transformation source; when the attacker's language model is stronger than the steganographer's (or weaker). This work is of little interest at present however (apart from against automatic data), while the attack is incapable of detecting manual stego.

Note that the reliance of linguistic stegosystems on external data sources (e.g. paraphrase sources) makes them vulnerable to unique attacks from creative wardens. In the case of CoverTweet, the PPDB could potentially be modified in a way which would signal when it has been used for embedding. Poisoning the PPDB with obviously wrong synonyms would likely be noticed by the human filter, but subtle changes could go undetected; for example, tweaking the features so that words starting with an '`A`' are prioritised in substitutions. If the system were to also use an external language model (or a publicly available corpus for training a language model), further attacks may be possible. In general the steganographer should always be wary of trusting external resources.

The Human Warden

We only utilised the human censor and automated censor separately, the human relying only on their superior language skills for detection purposes. Giving the human censor access to the PPDB would certainly be of interest, and would likely result in significantly increased detection rates. However, the low speed and high cost of the human makes this an unlikely adversary in the real world.

A more realistic scenario would be to combine the two, using the automated censor to perform the bulk of detection, but querying the human as an oracle when necessary. This is exactly as observed on Sina Weibo, where messages containing certain banned key words are marked for manual checking. In making a decision, our automated system could show the human tweets or phrases it has decided are possible changes, to get the human's opinion; it could take this opinion into account during classification.

6.2.3 Coding

The final part of our work dealt with coding as a solution to minimising distortion, and the non-shared selection channel. The direction for future work is clear, as we did not actually perform any real coding, we merely simulated it. Syndrome trellis codes should be utilised; either using the *nested* construction, or severely restricting the size of the message alphabet. Determining the *coding loss* of linguistic steganography with existing coding methods is an important first step. Coding has applications beyond CoverTweet and Twitter of course; any linguistic stegosystem stands to gain through the use of coding.

Had we actually implemented STCs for CoverTweet, using the system to hide on Twitter would introduce *latency* to the tweets. To code, the steganographer must write all the tweets before any can be posted; to approach the efficiency bounds (Figure 5.1) the number of pre-written tweets must be large [Filler et al., 2011]. This might not be acceptable in practice, and may require new approaches to coding to reduce this latency. As it stands, we imagine a system where the steganographer enters the desired payload, and writes a number of tweets, selecting the transformation options which are applicable. Once done, the system would need to post the tweets periodically, so as not to raise suspicion.

Appendices

Appendix A

Instructions

The following was given to the volunteer judges before they performed the task described in Chapter 3.

5 twitter users are suspected of hiding dangerous information in their tweets, and it's up to you to catch them! You have been given 5 different word documents, each one containing tweets from one user. For each user, you will be required to do the following:

1. Carefully read 40 tweets from before they started hiding information - pay close attention to their language, and choice of words, to help you on the next step.

2. Read another 20 tweets. 10 of these have been changed (in a way described below), in order to hide information, and 10 of them have not.

3. Decide on the 10 you think have been changed, and assign them a score from 1-10, to show how strongly you think it's been changed (where 10 means you're completely sure it's been changed, and 1 means you're incredibly unsure). Feel free to go back to the first page and reread the unaltered tweets as much as you like, it might help!

4. When appropriate, circle, underline or otherwise mark the section of the tweet that tipped you off (this might not always be possible).

5. Read another 40 tweets from before they started hiding information - use this to gain an even better understanding of their language.

6. Having read even more tweets from them, update your scores by filling in the second score column of the table. It might be that reading more of their tweets will have made you even more sure of one of the changes, or made you change

your mind on which tweets have been changed. It's also ok if your scores haven't changed at all! If you mark a new tweet as having been changed, make sure you mark the one that you don't think has been changed any more.

How the tweets have been changed - the CoverTweet system

The users have changed the tweets using our CoverTweet system. This works in the following way:

1. The user writes a tweet.
2. The system generates as many alternatives for it as possible, assigning them all values
3. The user chooses the best of these alternatives, that has been assigned a particular value. If there isn't a good enough alternative, discard the tweet and start again.

The system generates the alternatives by:

- Replacing words with synonymous words (this includes punctuation)
- Replacing phrases with synonymous phrases (this also includes punctuation)
- Moving usernames and hashtags around

The system DOES NOT do the following:

- Change (or delete) hashtags or usernames (other than switching them around if there's more than one of them)
- Make any alterations that might change the meaning of the tweet
- Make any changes to whitespace - none of the spaces in the tweets have been altered
- Arbitrarily delete punctuation.
- Do anything with capitalisation - all the tweets should be in lowercase, if they aren't, I've made a mistake

- Do anything with emoticons, these haven't been touched (they might not even turn up properly on your computer!)
- Do anything with URLs (they should have all been anonymised, but some might have slipped through)

Note: some of the changes are essentially impossible to spot, don't worry if you find yourself making a few complete guesses! Just make sure to give the ones you're really unsure on a low score.

Good luck, and thanks!

Appendix B

CoverTweet Output

On the following pages are examples of steganographic output from Manual CoverTweet, coded to minimise the four different distortion measures. Tweets are all from one user in our data set, and have been processed according to the steps given in Section 3.3.2. Tokenisation has been reversed, to improve readability.

Interested readers are encouraged to compare the batches for each distortion type. Stego changes have been highlighted; we have removed any tweets which were unchanged in all sets (which obviously includes tweets without any steganographic capacity). At a glance, we can see a couple of details which fit with what we know of the distortion measures: steganography made to minimise edit distance uses mostly single word substitutions, and probability distortion *fixes* the double negative in the original cover (tweet 13).

Figure B.1: Cover tweets.

1. going to go hangout with my babe, since i'm trying to hide the fact we are together
2. hello
3. marcell (that's what came up when my dad said my name).
4. fresam - (what came up when my mom said my name).
5. USER see that's what i've been saying the whole entire time! running is great makes you feel grrreat!
6. USER it's okay, better then me. drank like 3 sprites today
7. little brother is fat i swear man, but yet probably only weighs 115 pounds lol
8. this is like his 3rd lunchable today
9. i was like another lunchable? he said yeah, idk why but i'm real hungry.
10. i said shoot that's probably cause your growing, then he just smiled and went in his room
11. ohh myy goshh! i graduate on saturday and i have yet to write a speech in like 2 days
12. USER alright, better get up too man! or else i'll cry and make a vine of it!
13. i didn't do nothing today but kind of clean
14. i think i need to go to the hospital
15. time to go make some friends
16. my mom wants to know if you guys USER USER want to help decorate?
17. have a few invites left
18. who to give them too.
19. USER ill ask my mom :)
20. USER USER think we are starting friday. but i will let you guys know
21. just found out i could of had the iphone 5 this whole time...
22. writing isn't really my thing
23. everyone's alseep at my house
24. have a feeling ill be up all night writing
25. don't think sketching counts haha
26. then scratched my shoulder up after taking the picture
27. then when i read previous speeches those speeches just talk about one thing throughout it.
28. like thanks to everyone
29. and i can't remember what that last speech was about /:
30. USER got the welcome part that's all that matters lol jp. not far
31. USER well if you do let me know ! lol i wish it could be that sweet & short! would make things a lot easier.
32. but it's not!

Figure B.2: Binary

1. going to go hangout with my *sweetheart* , *given that i keep* trying to hide the fact we*'re* together
2. hello
3. marcell (that's *just* what came up when my *father* said my name).
4. fresam - (what came up when my mom said my name).
5. USER see that's *exactly* what *i was* saying the whole entire time! running*'s awesome* makes you feel grrreat!
6. USER *that*'s ok, better then me. drank like 3 sprites today
7. *kid bro*'s fat i swear man, but *still* probably just weighs 115 lbs lol
8. *it's* like his *third* lunchable *now*
9. i *went* like another lunchable? he said *yes* , idk why but i'm *really starving...*
10. i *just* said shoot *it's gotta be* cause your growing, then he just smiled and went in his room
11. ohh myy goshh! i graduate on saturday and i have yet to write a speech in like 2 days
12. USER *ok* , *best wake* up too *dude*! *or i'll* cry and make a vine of it!
13. i didn't do nothing today but kind of clean
14. i think i*'ve gotta get* to the hospital
15. *it's time to* go make *a few* friends
16. my *mother* wants to find out if you guys USER USER *wanted* to help decorate?
17. *got* a few invites left
18. who to give them too.
19. USER ill *just* ask my mom :)
20. USER USER *guess* we are starting friday. but i will let you guys know
21. just found out i could of had the iphone 5 this *entire* time...
22. writing is *not* really my shit
23. *everybody is* alseep at my *place*
24. *got the* feeling ill be up *the whole night* writing
25. don't *suppose* sketching counts haha
26. then scratched my shoulder up after taking the picture
27. then when i read previous speeches those speeches just talk about one thing throughout it.
28. like thanks to everyone
29. and i *could*n't remember what that last speech was about /:
30. USER got the welcome part that's all that matters lol jp. not far
31. USER well if you do *i would like to* know! lol i wish it could be *this nice* & short! would make things a *great deal* easier .
32. but it *ain't* !

Figure B.3: Edit distance

1. going to go hangout with my *baby*, since i *am* trying to *disguise* the fact we are together
2. *ahoy*
3. marcell (*this is* what came up when my dad said my name)...
4. fresam - (what came up when my mom *says* my name)...
5. USER see *here*'s what i've been saying the whole entire time! running*'s* great makes you feel grrreat!
6. USER it *is* ok, better then me. drank like 3 sprites today
7. little brother is *big* i swear man, but yet probably only weighs 115 pounds lol
8. this is *just* like his 3rd lunchable today
9. i was like another lunchable? he said *yep*, idk why but i *am* real hungry.
10. i said shoot that's probably cause your growing , *so* he just smiled and went in his room
11. ohh myy goshh! i graduate on saturday and i have yet to write a speech in like 2 days
12. USER alright , *best* get up too man! or else i'll cry and make a vine of it!
13. i *haven't done* nothing today but kind of clean
14. i guess i*'ve gotta* go to the hospital
15. *it's* time to go make *a few* friends
16. my mom wants to know if you guys USER USER *wanna* help decorate?
17. *got* a few invites left
18. who to give them too...
19. USER ill ask my *mother* :)
20. USER USER think we*'re* starting friday. but i will let you guys know
21. just found out i could of had the iphone 5 this whole time...
22. writing is *not* really my *shit*
23. everyone's alseep at my *place*
24. *got the* feeling ill be up all night *long* writing
25. don't *suppose* sketching counts haha
26. then scratched my shoulder up after taking the picture
27. then when i read previous speeches those speeches just talk about one thing throughout it...
28. like thanks to *everybody*
29. ... and i can't remember what that last speech was about /:
30. USER got the welcome part that's all that *really* matters lol jp. not far
31. USER well if you do *keep me posted*! lol i wish it could be that sweet & *brief*! would make things a lot easier...
32. but... it's not!

Figure B.4: Probability

1. going to go hangout with my *sweetie because* i'm *just* trying to *disguise* the fact we are together
2. *morning*
3. marcell (that's what came up when my *father* said my name).
4. fresam - (what came up when my mom *says* my name)*...*
5. USER see that's what i've been saying the whole entire time! running is great makes you feel grrreat!
6. USER it *is good*, better then me. drank like 3 sprites today
7. little brother is *big* i swear *to god* man , but *still* probably only weighs 115 lbs lol
8. *it's just* like his 3rd lunchable today
9. i was *just* like another lunchable? he said yeah, idk why but i *am really* hungry*...*
10. i said shoot *it is* probably cause your growing, then he just smiled and went in his room
11. ohh myy goshh! i graduate on saturday and i have yet to write a speech in like two days
12. USER *ok*, better *wake* up too man ! *otherwise* i'll cry and make a vine of it!
13. i didn't do *anything* today but *sort* of clean
14. *i need* to get to the hospital
15. *it's* time to go make *a few* friends
16. my mom wants to know if you guys USER USER *wanted* to help decorate?
17. *got* a few invites left
18. who to give them too*...*
19. USER ill ask my *mother* :)
20. USER USER *guess* we are starting friday. but i will let you guys know
21. just *learned* i could of had the iphone 5 this whole time...
22. writing is *not* really my *shit*
23. everyone's alseep at my *place*
24. *got the* feeling ill be up all night writing
25. don't *suppose* sketching counts haha
26. then scratched my shoulder up after taking the *photo*
27. *and* when i read previous speeches those speeches just talk about one thing throughout it .
28. like thanks to *everybody*
29. and i *don't* even remember what that last speech was about /:
30. USER got the welcome part that is all that *counts* lol jp. not far
31. USER well if you do *tell me*! lol i wish it could be *this nice* & *brief*! would make things *so much* easier .
32. but *it ain't*!

Figure B.5: Feature

1. going to go hangout with my babe, since i'm *just* trying to hide the fact we are together
2. *hi*
3. marcell (that's what came up when my *father* said my name).
4. fresam - (what came up when my mom said my name).
5. USER see that's what i've been saying the whole entire time! running is great makes you feel grrreat!
6. USER it 's *fine*, better then me. drank like three sprites today
7. *kid* brother is *fat* i swear *to god dude*, but *still* probably only weighs 115 pounds lol
8. *it's* like his *third* lunchable *now*
9. i was *just* like another lunchable? he said yeah, idk why but i *am really* hungry*...*
10. i said shoot that's probably cause your growing, *so* he just smiled and went in his room
11. ohh myy goshh! i graduate on saturday and i have yet to write a speech in like *two* days
12. USER alright , better get up too *dude*! or i'll cry and make a vine of it!
13. i didn't do *anything* today but *sort* of clean
14. *i need* to *get* to the hospital
15. *it's* time to go make *a few* friends
16. my mom wants to know if you guys USER USER *wanted* to help decorate?
17. *got* a few invites left
18. who to give them too*...*
19. USER ill ask my *mother* :)
20. USER USER think we're starting friday. but i will let you guys know
21. just *learned* i could of had the iphone 5 this whole time...
22. writing is *not* really my *shit*
23. everyone's alseep at my *place*
24. *got the* feeling ill be up all night writing
25. don't *suppose* sketching counts haha
26. then scratched my shoulder up after taking the photo
27. and when i read previous speeches those speeches just talk about one thing throughout it.
28. like thanks to everyone
29. and i *don't even* remember what that last speech was about /:
30. USER got the welcome part that's all that *counts* lol jp. not far
31. USER well if you do let me know! lol i wish it could be *this nice* & short! would make things *so much* easier*...*
32. but it *ain't*!

Bibliography

[Amazon, 2005] Amazon (2005). Amazon mechanical turk. `https://www.mturk.com/`. Accessed: 2016-08-16.

[Anderson and Petitcolas, 1998] Anderson, R. J. and Petitcolas, F. A. (1998). On the limits of steganography. *IEEE Journal on Selected Areas in Communications*, 16:474–481.

[Atallah et al., 2001] Atallah, M. J., McDonough, C. J., Raskin, V., and Nirenburg, S. (2001). Natural language processing for information assurance and security: an overview and implementations. In *Proceedings of the 2000 Workshop on New Security Paradigms*, pages 51–65. ACM.

[Atwood, 2011] Atwood, J. (2011). Suspension, han or hellban? `https://blog.codinghorror.com/suspension-ban-or-hellban/`. Accessed: 2016-08-16.

[Bannard and Callison-Burch, 2005] Bannard, C. and Callison-Burch, C. (2005). Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 597–604. Association for Computational Linguistics.

[Barrón-Cedeño and Rosso, 2009] Barrón-Cedeño, A. and Rosso, P. (2009). On automatic plagiarism detection based on n-grams comparison. In *Proceedings of the European Conference on Information Retrieval*, pages 696–700. Springer.

[Bender et al., 1996] Bender, W., Gruhl, D., Morimoto, N., and Lu, A. (1996). Techniques for data hiding. *IBM systems journal*, 35:313–336.

[Bergmair, 2004] Bergmair, R. (2004). Towards linguistic steganography: A systematic investigation of approaches, systems, and issues. Master's thesis, University of Derby, Austria.

[Berlekamp et al., 1978] Berlekamp, E. R., McEliece, R. J., and Van Tilborg, H. C. (1978). On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24:384–386.

[Bird, 2006] Bird, S. (2006). NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics.

[Bolshakov, 2004] Bolshakov, I. A. (2004). A Method of Linguistic Steganography Based on Collocationally-Verified Synonymy. In *Proceedings of the 6th International Workshop on Information Hiding*, pages 180–191. Springer.

[Boyd and Marwick, 2011] Boyd, D. and Marwick, A. (2011). Social steganography: Privacy in networked publics. In *Proceedings of the International Communication Association*. International Communication Association.

[Brandwatch, 2016] Brandwatch (2016). Twitter stats 2016. `https://www.brandwatch.com/2016/05/44-twitter-stats-2016/`. Accessed: 2016-08-16.

[Brants and Franz, 2006] Brants, T. and Franz, A. (2006). Web 1T 5-gram version 1. `https://catalog.ldc.upenn.edu/LDC2006T13`. Accessed: 2016-08-16.

[Brown et al., 1993] Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19:263–311.

[Callison-Burch and Dredze, 2010] Callison-Burch, C. and Dredze, M. (2010). Creating speech and language data with amazon's mechanical turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pages 1–12. Association for Computational Linguistics.

[Chang and Clark, 2010] Chang, C.-Y. and Clark, S. (2010). Linguistic steganography using automatically generated paraphrases. In *Proceedings of the 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 591–599. Association for Computational Linguistics.

[Chang and Clark, 2012] Chang, C.-Y. and Clark, S. (2012). Adjective deletion for linguistic steganography and secret sharing. In *Proceedings of the 24th International Conference on Computational Linguistics*, pages 493–510. Association for Computational Linguistics.

[Chang and Clark, 2014] Chang, C.-Y. and Clark, S. (2014). Practical linguistic steganography using contextual synonym substitution and a novel vertex coding method. *Computational Linguistics*, 40:403–448.

[Chen and Goodman, 1999] Chen, S. F. and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13:359–394.

[Chen et al., 2011] Chen, Z., Huang, L., Miao, H., Yang, W., and Meng, P. (2011). Steganalysis against substitution-based linguistic steganography based on context clusters. *Computers & Electrical Engineering*, 37:1071–1081.

[Clark and Curran, 2007] Clark, S. and Curran, J. R. (2007). Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33:493–552.

[Clarke et al., 2014] Clarke, C. A., Pfluegel, E., and Tsaptsinos, D. (2014). Hide-as-you-type: an approach to natural language steganography through sentence modification. In *Proceedngs of 2014 IEEE Intl. Conf. on High Performance Computing and Communications, 2014 IEEE 6th Intl. Symp. on Cyberspace Safety and Security, 2014 IEEE 11th Intl. Conf. on Embedded Software and Systems (HPCC, CSS, ICESS)*, pages 945–952. IEEE.

[Crandall, 1998] Crandall, R. (1998). Some notes on steganography. *Posted on steganography mailing list*.

[Craver, 1998] Craver, S. (1998). On public-key steganography in the presence of an active warden. In *Proceedings of the Second International Workshop on Information Hiding*, pages 355–368. Springer.

[CrowdFlower, 2007] CrowdFlower (2007). Crowdflower. `https://www.crowdflower.com/`. Accessed: 2016-08-16.

[Declerck and Lendvai, 2015] Declerck, T. and Lendvai, P. (2015). Processing and normalizing hashtags. pages 104–109.

[Desoky, 2012] Desoky, A. (2012). *Noiseless steganography: The key to covert communications*. CRC Press.

[Filler and Fridrich, 2010a] Filler, T. and Fridrich, J. (2010a). Gibbs construction in steganography. *IEEE Transactions on Information Forensics and Security*, 5:705–720.

[Filler and Fridrich, 2010b] Filler, T. and Fridrich, J. (2010b). Minimizing additive distortion functions with non-binary embedding operation in steganography. In *Proceedings of the 2010 IEEE International Workshop on Information Forensics and Security*, pages 1–6. IEEE.

[Filler et al., 2011] Filler, T., Judas, J., and Fridrich, J. (2011). Minimizing additive distortion in steganography using syndrome-trellis codes. *IEEE Transactions on Information Forensics and Security*, 6:920–935.

[Fort et al., 2011] Fort, K., Adda, G., and Cohen, K. B. (2011). Amazon mechanical turk: Gold mine or coal mine? *Computational Linguistics*, 37:413–420.

[Fridrich, 2010] Fridrich, J. (2010). *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge University Press.

[Fridrich and Filler, 2007a] Fridrich, J. and Filler, T. (2007a). Binary quantization using belief propagation with decimation over factor graphs of LDGM codes. In *Proceedings of the 45th Allerton Conference on Coding, Communication, and Control*, pages 495–501.

[Fridrich and Filler, 2007b] Fridrich, J. and Filler, T. (2007b). Practical methods for minimizing embedding impact in steganography. In *Proceedings of SPIE volume 6505*. International Society for Optics and Photonics.

[Fridrich et al., 2005] Fridrich, J., Goljan, M., Lisonek, P., and Soukal, D. (2005). Writing on wet paper. *IEEE Transactions on Signal Processing*, 53:3923–3935.

[Fridrich et al., 2006] Fridrich, J., Goljan, M., and Soukal, D. (2006). Wet paper codes with improved embedding efficiency. In *Proceedings of SPIE volume 6072*. International Society for Optics and Photonics.

[Fridrich and Kodovsky, 2012] Fridrich, J. and Kodovsky, J. (2012). Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*, 7:868–882.

[Fu and Zhou, 2010] Fu, B. and Zhou, J. (2010). The application of information hiding technology to electronic assignment anti-plagiarism. In *Proceedings of the 6th International Conference on Wireless Communications Networking and Mobile Computing*, pages 1–4. IEEE.

[Galand and Kabatiansky, 2003] Galand, F. and Kabatiansky, G. (2003). Information hiding by coverings. In *Proceedings of the 2003 IEEE Information Theory Workshop*, pages 151–154. IEEE.

[Gale and Church, 1994] Gale, W. and Church, K. (1994). What is wrong with adding one. *Corpus-Based Research into Language*, 1:189–198.

[Ganitkevitch et al., 2013] Ganitkevitch, J., Van Durme, B., and Callison-Burch, C. (2013). PPDB: The paraphrase database. In *Proceedings of the 2013 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764. Association for Computational Linguistics.

[Good, 1953] Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, 40:237–264.

[Grosvald and Orgun, 2011] Grosvald, M. and Orgun, C. O. (2011). Free from the cover text: a human-generated natural language approach to text-based steganography. *Journal of Information Hiding and Multimedia Signal Processing*, 2:133–141.

[Grothoff et al., 2005] Grothoff, C., Grothoff, K., Alkhutova, L., Stutsman, R., and Atallah, M. (2005). Translation-based steganography. In *Proceedings of the 7th International Workshop on Information Hiding*, pages 219–233. Springer.

[Halvani et al., 2013] Halvani, O., Steinebach, M., Wolf, P., and Zimmermann, R. (2013). Natural language watermarking for german texts. In *Proceedings of the 1st ACM Workshop on Information Hiding and Multimedia Security*, pages 193–202. ACM.

[Holub and Fridrich, 2015] Holub, V. and Fridrich, J. (2015). Low-complexity features for JPEG steganalysis using undecimated DCT. *IEEE Transactions on Information Forensics and Security*, 10:219–228.

[Hopper et al., 2002] Hopper, N. J., Langford, J., and Von Ahn, L. (2002). Provably secure steganography. In *Proceedings of the Annual International Cryptology Conference*, pages 77–92. Springer.

[Independent, 2015] Independent (2015). How users of 'chinese twitter' sina weibo are beating state censorship. `http://www.independent.co.uk/news/media/online/how-users-of-chinese-twitter-sina-weibo-are-beating-state-censorship-10436302.html`. Accessed: 2016-08-16.

[Jaech and Ostendorf, 2015] Jaech, A. and Ostendorf, M. (2015). What your username says about you. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2032–2037. SIGDAT.

[Kahn, 1996] Kahn, D. (1996). The history of steganography. In *Proceedings of the First International Workshop on Information Hiding*, pages 1–5. Springer.

[Ker, 2006] Ker, A. D. (2006). Batch steganography and pooled steganalysis. In *Proceedings of the 8th International Workshop on Information Hiding*, pages 265–281. Springer.

[Ker, 2007] Ker, A. D. (2007). Steganalysis of embedding in two least-significant bits. *IEEE Transactions on Information Forensics and Security*, 2:46–54.

[Ker, 2008] Ker, A. D. (2008). Locating steganographic payload via WS residuals. In *Proceedings of the 10th ACM workshop on Multimedia and security*, pages 27–32. ACM.

[Ker, 2010] Ker, A. D. (2010). The square root law does not require a linear key. In *Proceedings of the 12th ACM workshop on Multimedia and security*, pages 213–224. ACM.

[Ker, 2015] Ker, A. D. (2015). *Advanced Security Notes*.

[Ker et al., 2013] Ker, A. D., Bas, P., Böhme, R., Cogranne, R., Craver, S., Filler, T., Fridrich, J., and Pevnỳ, T. (2013). Moving steganography and steganalysis from the laboratory into the real world. In *Proceedings of the 1st ACM Workshop on Information Hiding and Multimedia Security*, pages 45–58. ACM.

[Ker and Pevnỳ, 2014a] Ker, A. D. and Pevnỳ, T. (2014a). A mishmash of methods for mitigating the model mismatch mess. In *Proceedings of SPIE volume 9028*, pages 1601–1615. International Society for Optics and Photonics.

[Ker and Pevnỳ, 2014b] Ker, A. D. and Pevnỳ, T. (2014b). The steganographer is the outlier: realistic large-scale steganalysis. *IEEE Transactions on Information Forensics and Security*, 9:1424–1435.

[Ker et al., 2008] Ker, A. D., Pevnỳ, T., Kodovskỳ, J., and Fridrich, J. (2008). The square root law of steganographic capacity. In *Proceedings of the 10th ACM workshop on Multimedia and security*, pages 107–116. ACM.

[Kerckhoffs, 1883] Kerckhoffs, A. (1883). La cryptographie militaire. *Journal des sciences militaires*, 9:538.

[Kneser and Ney, 1995] Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Proceedings of the 1995 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 181–184. IEEE.

[Kodovsky et al., 2012] Kodovsky, J., Fridrich, J., and Holub, V. (2012). Ensemble classifiers for steganalysis of digital media. *IEEE Transactions on Information Forensics and Security*, 7:432–444.

[Liu et al., 2010] Liu, Y., Ghosal, D., Armknecht, F., Sadeghi, A.-R., Schulz, S., and Katzenbeisser, S. (2010). Robust and undetectable steganographic timing channels for i.i.d. traffic. In *Proceedings of the 12th International Workshop on Information Hiding*, pages 193–207. Springer.

[Lu et al., 2004] Lu, P., Luo, X., Tang, Q., and Shen, L. (2004). An improved sample pairs method for detection of LSB embedding. In *Proceedings of the 6th International Workshop on Information Hiding*, pages 116–127. Springer.

[Lubacz et al., 2012] Lubacz, J., Mazurczyk, W., and Szczypiorski, K. (2012). Principles and overview of network steganography. *IEEE Communications Magazine*, 52:225–229.

[Lubenko, 2013] Lubenko, I. (2013). *Towards Robust Steganalysis: binary classifiers and large, heterogeneous data*. PhD thesis, University of Oxford.

[Madnani and Dorr, 2010] Madnani, N. and Dorr, B. J. (2010). Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, 36:341–387.

[McKellar, 2000] McKellar, D. (2000). Spammimic. `http://www.spammimic.com/`. Accessed: 2016-08-16.

[Meral et al., 2006] Meral, H. M., Sankur, B., and Ozsoy, A. S. (2006). Watermarking tools for turkish texts. In *Proceedings of the 14th IEEE Conference on Signal Processing and Communications Applications*, pages 1–4. IEEE.

[Miller, 1995] Miller, G. A. (1995). Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41.

[Mostak, 2013] Mostak, T. (2013). Harvard tweetmap. `https://worldmap.harvard.edu/tweetmap/`. Accessed: 2016-08-16.

[Murphy, 2001] Murphy, B. (2001). Syntactic information hiding in plain text. Master's thesis, Department of Computer Science, Trinity College Dublin.

[Murphy and Vogel, 2007] Murphy, B. and Vogel, C. (2007). The syntax of concealment: reliable methods for plain text information hiding. In *Proceedings of SPIE volume 6505*. International Society for Optics and Photonics.

[Napoles et al., 2016] Napoles, C., Callison-Burch, C., and Post, M. (2016). Sentential paraphrasing as black-box machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 62–66. Association for Computational Linguistics.

[Neubig and Duh, 2013] Neubig, G. and Duh, K. (2013). How much is said in a tweet? A multilingual, information-theoretic perspective. In *Proceedings of the AAAI Spring Symposium: Analyzing Microtext*. Association for the Advancement of Artificial Intelligence.

[Oberreuter et al., 2011] Oberreuter, G., LHuillier, G., Rıos, S. A., and Velásquez, J. D. (2011). Approaches for intrinsic and external plagiarism detection. In *Working Notes for the CLEF 2011 Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse (PAN)*. Conference and Labs of the Evaluation Forum.

[Orgun, 2009] Orgun, C. O. (2009). The human attack in linguistic steganography. In *Handbook of Research on Social and Organizational Liabilities in Information Security*, pages 380–397. IGI Global.

[Pevnỳ et al., 2010a] Pevnỳ, T., Bas, P., and Fridrich, J. (2010a). Steganalysis by subtractive pixel adjacency matrix. *IEEE Transactions on Information Forensics and Security*, 5:215–224.

[Pevnỳ et al., 2010b] Pevnỳ, T., Filler, T., and Bas, P. (2010b). Using high-dimensional image models to perform highly undetectable steganography. In *Proceedings of the 11th International Workshop on Information Hiding*, pages 161–177. Springer.

[Pevnỳ and Fridrich, 2008] Pevnỳ, T. and Fridrich, J. (2008). Multiclass detector of current steganographic methods for JPEG format. *IEEE Transactions on Information Forensics and Security*, 3:635–650.

[Pevnỳ and Nikolaev, 2015] Pevnỳ, T. and Nikolaev, I. (2015). Optimizing pooling function for pooled steganalysis. In *Proceedings of the 2015 IEEE International Workshop on Information Forensics and Security*, pages 1–6. IEEE.

[Pew Research, 2015] Pew Research (2015). Demographics of social media users. `http://www.pewinternet.org/2015/08/19/the-demographics-of-social-media-users/`. Accessed: 2016-08-16.

[Pfitzmann, 1996] Pfitzmann, B. (1996). Information hiding terminology. In *Proceedings of the First International Workshop on Information Hiding*, pages 347–350. Springer.

[Por et al., 2008] Por, L. Y., Delina, B., Li, Q., Chen, S. Y., and Xu, A. (2008). Information hiding: A new approach in text steganography. In *Proceedings of the 7th WSEAS International Conference on Mathematics and Computers in Science and Engineering*. World Scientific and Engineering Academy and Society.

[Por et al., 2012] Por, L. Y., Wong, K., and Chee, K. O. (2012). UniSpaCh: A text-based data hiding method using unicode space characters. *Journal of Systems and Software*, 85:1075–1082.

[Potthast et al., 2009] Potthast, M., Stein, B., Eiselt, A., Barrón-Cedeno, A., and Rosso, P. (2009). Overview of the 1st international competition on plagiarism detection. In *Proceedings of the 3rd PAN Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse*, pages 1–9. Spanish Society for Natural Language Processing.

[Reeds, 1998] Reeds, J. (1998). Solved: the ciphers in book III of Trithemius's steganographia. *Cryptologia*, 22(4):291–317.

[Shirali-Shahreza and Shirali-Shahreza, 2006] Shirali-Shahreza, M. H. and Shirali-Shahreza, M. (2006). A new approach to persian/arabic text steganography. In *Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse*, pages 310–315. IEEE.

[Shirali-Shahreza and Shirali-Shahreza, 2007] Shirali-Shahreza, M. H. and Shirali-Shahreza, M. (2007). Text steganography in chat. In *Proceedings of the 3rd IEEE/IFIP International Conference in Central Asia on Internet*, pages 1–5. IEEE.

[Simmons, 1984] Simmons, G. J. (1984). The prisoners problem and the subliminal channel. In *Advances in Cryptology*, pages 51–67. Springer.

[Spinbot, 2014] Spinbot (2014). Paraphrasing tool. `http://paraphrasing-tool.com/`. Accessed: 2016-08-16.

[Stamatatos, 2009] Stamatatos, E. (2009). Intrinsic plagiarism detection using character n-gram profiles. In *Proceedings of the 3rd PAN Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse*.

[Stein et al., 2011] Stein, B., Lipka, N., and Prettenhofer, P. (2011). Intrinsic plagiarism analysis. *Language Resources and Evaluation*, 45:63–82.

[Stolcke, 2002] Stolcke, A. (2002). SRILM - an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, pages 901–904.

[Taskiran et al., 2006] Taskiran, C. M., Topkara, U., Topkara, M., and Delp, E. J. (2006). Attacks on lexical natural language steganography systems. In *Proceedings of SPIE volume 6072*. International Society for Optics and Photonics.

[Times Higher Education, 2014] Times Higher Education (2014). Sinister buttocks? Roget would blush at the craft cheek. `https://www.timeshighereducation.com/news/sinister-buttocks-roget-would-blush-at-the-crafty-cheek/2015027.article`. Accessed: 2016-08-16.

[Topkara et al., 2006a] Topkara, M., Topkara, U., and Atallah, M. J. (2006a). Words are not enough: Sentence level natural language watermarking. In *Proceedings of the ACM Workshop on Content Protection and Security*. ACM.

[Topkara et al., 2007] Topkara, M., Topkara, U., and Atallah, M. J. (2007). Information hiding through errors: a confusing approach. In *Proceedings of SPIE volume 6505*.

[Topkara et al., 2006b] Topkara, U., Topkara, M., and Atallah, M. J. (2006b). The hiding virtues of ambiguity: quantifiably resilient watermarking of natural language text through synonym substitutions. In *Proceedings of the 8th workshop on Multimedia and security*, pages 164–174. ACM.

[van Dijk and Willems, 2001] van Dijk, M. and Willems, F. (2001). Embedding information in grayscale images. In *Proceedings of the 22nd Symposium on Information and Communication Theory*, pages 147–154.

[Vybornova and Macq, 2007] Vybornova, O. and Macq, B. (2007). Natural language watermarking and robust hashing based on presuppositional analysis. In *Proceedings of the 2007 IEEE International Conference on Information Reuse and Integration*, pages 177–182. IEEE.

[Wang et al., 2009] Wang, Z.-H., Chang, C.-C., Kieu, T. D., and Li, M.-C. (2009). Emoticon-based text steganography in chat. In *Proceedings of the IEEE Asia-Pacific Conference on Computational Intelligence and Industrial Applications*, pages 457–460. IEEE.

[Westfeld, 2001] Westfeld, A. (2001). F5–a steganographic algorithm. In *Proceedings of the 4th International Workshop on Information Hiding*, pages 289–302.

[Westfeld and Pfitzmann, 1999] Westfeld, A. and Pfitzmann, A. (1999). Attacks on steganographic systems. In *Proceedings of the 3rd International Workshop on Information Hiding*, pages 61–76. Springer.

[Wilson et al., 2014] Wilson, A., Blunsom, P., and Ker, A. D. (2014). Linguistic steganography on twitter: hierarchical language modeling with manual interaction. In *Proceedings of SPIE volume 9028*. International Society for Optics and Photonics.

[Wilson et al., 2015] Wilson, A., Blunsom, P., and Ker, A. D. (2015). Detection of steganographic techniques on twitter. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2564–2569. SIGDAT.

[Wilson and Ker, 2016] Wilson, A. and Ker, A. D. (2016). Avoiding detection on twitter: embedding strategies for linguistic steganography. In *Proceedings of Electronic Imaging 2016: Media Watermarking, Security, and Forensics*. Society for Imaging Science and Technology.

[Winstein, 1999] Winstein, K. (1999). Tyrannosaurus lex. *Unpublished*.

[Wyseur et al., 2008] Wyseur, B., Wouters, K., and Preneel, B. (2008). Lexical natural language steganography systems with human interaction. In *Proceedings of the 6th European Conference on Information Warfare and Security*, pages 313–320.

[Xiang et al., 2014] Xiang, L., Sun, X., Luo, G., and Xia, B. (2014). Linguistic steganalysis using the features derived from synonym frequency. *Multimedia Tools and Applications*, 71:1893–1911.

[Xin-guang et al., 2006] Xin-guang, S., Hui, L., and Zhong-liang, Z. (2006). A steganalysis method based on the distribution of characters. In *Proceedings of the 8th International Conference on Signal Processing*, pages 54–56. IEEE.

[Yu et al.,] Yu, Z., Huang, L., Chen, Z., Li, L., Zhao, X., and Zhu, Y. Steganalysis of synonym-substitution based natural language watermarking. *International Journal of Multimedia and Ubiquitous Engineering*, 4:21–33.

[Yuling et al., 2007] Yuling, L., Xingming, S., Can, G., and Hong, W. (2007). An efficient linguistic steganography for chinese text. In *Proceedings of the 2007 IEEE International Conference on Multimedia and Expo*, pages 2094–2097. IEEE.

[Zechner et al., 2009] Zechner, M., Muhr, M., Kern, R., and Granitzer, M. (2009). External and intrinsic plagiarism detection using vector space models. In *Proceedings of the 3rd PAN Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse*.

[Zhang et al., 2005] Zhang, Y., Liu, T., Chen, Y.-h., Zhao, S.-q., and Li, S. (2005). Natural language watermarking. *Journal of Chinese Information Processing*, 2005.

[Zhu et al., 2013] Zhu, T., Phipps, D., Pridgen, A., Crandall, J. R., and Wallach, D. S. (2013). The velocity of censorship: High-fidelity detection of microblog post deletions. In *Proceedings of the 22nd USENIX Security Symposium*, pages 227–240.