Matryoshka: Hiding Secret Communication in Plain Sight

Iris Safaka
EPFL

Christina Fragouli
UCLA

Katerina Argyraki
EPFL

Abstract

We want to enable a pair of communicating users to exchange secret messages while hiding the fact that secret communication is taking place. We propose a linguistic steganography approach, where each human message is hidden in another human-like message. A hard open question is how to keep the steganographic message small – existing related tools tend to blow up its size, thereby revealing the use of steganography. We encrypt by compressing each message, mapping it to a plausible sequence of words (using a language model), and letting the human user edit the outcome to produce a human-like message; we decrypt with a Viterbi-like state decoder. Our approach aims in producing text that a human can edit and fix with minimal effort. As a first step, we build a prototype of our system that helps users encrypt English messages (into English messages), and we report on first experiments on Mechanical Turk.

1 Introduction

When we use free communication systems (e.g., Google, Hotmail, Facebook etc.) there is an implicit agreement that these systems will send advertisements to the users. The economic underpinnings of such systems have been routine now for a few years and users have mostly become aware of this tradeoff. Hundreds of millions of users use these systems daily.

From a privacy preservation point of view it should be noted that most messages exchanged between users do not necessarily raise privacy concerns. Additionally, the notion of privacy varies from user to user even though there is general acceptance that messages relating to health, finance, relationships, religious beliefs etc. may be viewed by many as potentially private. Out of tens of e-mails sent daily, there may be just one or two that we might want to keep private from the communication system.

This paper proposes a linguistic approach to confidentiality, reclaiming our right to keep some information private. Our approach takes as input a short human sentence and maps it to other human sentences, which can be mapped back to the original one by a correspondent privy to an agreed code. It thus hides the message in plain sight: it not only keeps the message secret, but also the fact that we have something to hide.

We argue that such an approach is more desirable than simply using cryptographic encryption. A messaging service provider can detect cryptographically encrypted messages, has no incentive to encourage encryption – if widespread, it could harm its revenue – and could, in retaliation, treat our traffic preferentially. And service providers are not the only potential evil entities. They may themselves want plausible deniability in enabling users to exchange secret messages; the communication system may *want* to be able to tell the government "I cannot possibly know whether these users are exchanging secret messages". A linguistic encryption approach that creates encrypted messages indistinguishable from our regular correspondence would directly address such concerns.

Unfortunately, existing linguistic steganography approaches would not be a good fit for our needs. Several of them are easy to detect once we know how the encoder works (see Section 7). Most of them encrypt sentences at a very low rate, leading to encrypted text that is much longer than the secret, which can again give away the presence of steganography. Apart from achieving a high rate of encryption, we need low encoding and decoding complexity that still leads to undetectable encryption.

Which brings up the question: what is non-detectable linguistic steganography? Clearly, a message either looks like human speech or it does not, if a human so asserts. But open communication systems cannot possibly afford human operators look through the billions of e-mails exchanged every day, and randomized sampling would not catch the small percentage of encrypted messages we anticipate. Thus we pose the reverse of Turing's famous test: can a machine tell if a sentence has been written by a human or not? and can it do so, if it needs to process billions of e-mails – thus in a computationally efficient manner?

We present in this paper the design and evaluation of an approach that performs well on the best machine-based test we could devise, by combining automated text generation with human-in-the-loop help. Our approach builds dictionaries that efficiently map natural language text to natural language text. The encoder uses the dictionary to produce candidate steganographic text. We then ask the human user to polish the resulting word sequence to human-like sentences. To decode, we use a completely automated approach based on Viterbi decoding.

We think that using a bit of human help is not bad. We optimize our system to require minimal intervention from the human user and evaluate through Mechanical Turk experiments the time of human effort it requires (less than a few minutes in most cases). We think this is an acceptable amount of effort for the one message per day we may want to protect; and having a human polishing the sentences makes it very hard for a machine to detect it is steganographic text. At the same time, the need for some human effort, even minimal, makes our approach non-threatening to communication systems: we expect that most people would not be willing to put in the extra minutes for every e-mail they send.

The rest of the paper is organized as follows. Section 2 summarizes our setup and high level approach; Sections 3 and 4 describe our encoder and decoder respectively; Section 5 presents experimental results and Section 7 reviews related work.

2 Setup and Block Diagram

Problem statement. We want to enable two communication parties, Alice and Bob, to exchange short secret messages, masked as innocuous messages, over their messaging provider, such that an adversary, Eve, is unable to distinguish between normal messages and those carrying hidden information. We assume that Eve has access to all the messages crossing the internal infrastructure of the service provider and that the load of messages is large enough to prevent Eve from visually inspecting each one of them; she instead runs detection algorithms to identify the existence of steganography. Note that this does not imply that our approach is de-facto vulnerable to human inspection attacks – the "human-in-the-loop" is a significant step toward successfully mitigating these attacks. It rather implies that the load is such that Eve does not have any better strategy than running a detection algorithm.

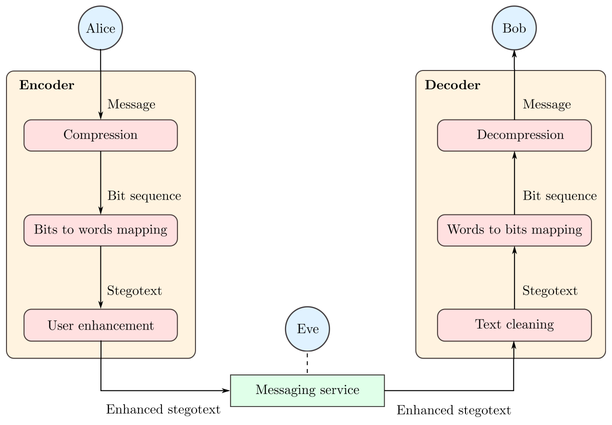

Figure 1: Encoder and decoder block diagrams.

Block Diagram. Our approach consists of using an encoder and decoder that run before and after we use the messaging service, as shown in Fig. 1. At the encoder, we use the following blocks: Compression compresses the secret message we want to send into a bit sequence; Bits-to-Words Mapping breaks each sequence into subsequences of a fixed length, and maps each one to a word among a set of possible choices (at this point we may also output additional words to better approximate human language); User enhancement is the stage where the human user is presented with a candidate sequence of words, and is asked to add words so that the result is as close as possible to human natural language. At the decoder, the block Text Cleaning removes the words inserted by the human user; the block Words-to-Bits maps the remaining words back to bit sequences, the concatenation of which gets decompressed to the hidden message from the Decompression component.

Optionally, a symmetric-key encryption block may be added in our system so as to ensure confidentiality of the secret message even in the case where Eve has both identified the existence of steganography and retrieved the hidden bit sequence. For example, an AES-GCM scheme [9] in 128-bit block cipher mode for authenticated encryption could be applied on the output of the Compression block. In that case the resulting bit sequence would be of the same or slightly larger length (up to 128 bits, depending on the length of the produced authentication tag that gets appended at the end of the ciphertext). In this paper, we do not assert the cryptographic strength of our approach and in fact we have not tested our scheme for it. Our contribution is the idea of open communication that preserves privacy at low cost while cohabitating with the advertisement-driven means of communication.

We next describe the design choices for each of the blocks in Fig.1 and their implementation.

3 Encoder

3.1 Compression

We use a *mixed* Huffman codebook $\mathcal{C}$ to compress a message into a bit sequence, in a lossless way. The symbol alphabet includes all printable ASCII characters (lower and capital case letters, numbers, space, digits, punctuation) and the set of the 300 most *frequently used* words in written English (that make up about 65% of all written material [6]). We denote with C and C^{-1} the compression and decompression functions respectively. We use the character Huffman to enable to compress names and other unusual words; and the word Huffman to efficiently compress the most common words.

To built the codebook, we derive the frequencies of the characters and the words from a large training text (totaling approx. 78M characters, 13M words). The resulting codebook yields a 52% compression ratio on average[1], which is 7% better than if we considered a symbol alphabet of only printable ASCII. Alice and Bob use the same pre-computed codebook to compress/decompress messages, that can be included in the resources of their encoder/decoder. In a smarter version of our system, we could compress using a word Huffman that includes words that are most common in messages considered private, such as love, hate, pregnancy, etc.

3.2 Bits to Words Mapping

To map bits to words, we first parse the bit sequence into sub-sequences of length b bits[2]. Each sub-sequence uniquely identifies a set of words that we call a *bin* and denote as $\mathcal{B}$. Inside each of the 2^b bins we have placed p bin words. We call the unique mapping between b-bit sequences and bins a *dictionary* $\mathcal{D}$ and we denote with D the bits to bin conversion and with D^{-1} the reverse operation. $\mathcal{W}_B$ is the set of all bin words and $\mathcal{W}_S$ the set of stop words, i.e., words not included in $\mathcal{D}$.

Example: Assume a dictionary of size 4 entries ($b = 2$) with bins $\mathcal{B}_0 = \{$to, the, an$\}$, $\mathcal{B}_1 = \{$I, we, you$\}$, $\mathcal{B}_2 = \{$music, piano, weather$\}$, $\mathcal{B}_3 = \{$like, hate, hear$\}$, and a message that has been compressed into the bit sequence "01110010". This gets mapped into a bin sequence as follows: "01" corresponds to $\mathcal{B}_1$, "11" to $\mathcal{B}_3$, "00" to $\mathcal{B}_0$, "10" to $\mathcal{B}_2$. Any selection of words from the bin sequence $\mathcal{B}_1$, $\mathcal{B}_3$, $\mathcal{B}_0$, $\mathcal{B}_2$ embeds the hidden message, e.g., the words "I like the weather".

Alice and Bob share a pre-computed dictionary $\mathcal{D}$, or independently compute it by using the same text corpus R plus a secret key $\mathcal{K}$, as described next. Note that we can easily customize our approach to produce sentences on a topic of interest of the user, by using bin words that come from a specialized text; for instance, in one of our experiments, we used words from a text on dreams, and in another from texts on animals.

[1]We compress 4,825 sentences of length between 4 and 15 words.

[2]If needed, the compressed bit sequences is padded with 0's at the end. This operation is reverted while retrieving the hidden message.

Dictionary Building. To be able to create natural language sentences, we need to be able to find words between consecutive bins that would also appear close to each other in human speech. We try to do so in two ways: first, we populate each bin with p words drawn according to the frequency distribution of the words in R, so that each bin can have more frequent words and less infrequent words; second, we allow a bin word to appear in more than one bin. E.g., assume that the word "sunny" is placed only in bin $\mathcal{B}_1$, and the word "weather" in bin $\mathcal{B}_2$; unless the bins occur in that order there is no chance that these words could get selected together. Instead, if both words appear in more than one bin, there exist more combinations of bin sequences that allow these words to get selected. Repeating words in bins allows in addition for the dictionary size to be more than $\frac{|\mathcal{W}_B|}{p}$ entries, which allows for larger b and greater covert rate. Clearly, repeating words in bins can be viewed as a form of "noise" in the Alice-Bob channel.

We build our dictionary as follows. Let $\mathcal{W}_R$ denote the set of unique words derived from a text corpus R and $\mathcal{E}$ a set of frequently used English words:

1. Assign $\mathcal{W}_B = \mathcal{W}_R - \mathcal{E}$ and $\mathcal{W}_S = \mathcal{W}_R \cap \mathcal{E}$.

2. For every word $w_i \in \mathcal{W}_B$, its number of occurrences o_i in the corpus R is counted, and its frequency of occurrence f_i is calculated as $f_i = \frac{o_i}{\sum_{j=0}^{|\mathcal{W}_B|} o_j}$.

3. A vector $\mathbf{v}$, with $|\mathbf{v}| = p \cdot 2^b$, is constructed be repeating $\lceil f_i \cdot p \cdot 2^b \rceil$ times each word $w_i \in \mathcal{W}_B$. The vector $\mathbf{v}$ is randomly shuffled, using $\mathcal{K}$ as seed.

4. All the elements between $\mathbf{v}[i]$ and $\mathbf{v}[i+p]$, $i \in \{0, p, 2p \ldots, (p-1)2^b\}$, are placed in the bin B_j, $j \in \{0, 1, \ldots, 2^b - 1\}$. The sets $\mathcal{W}_B$ and $\mathcal{W}_S$ are accordingly updated[3].

Word Choice. The next step is to select one word to use from each bin, so that the sequence of words we select form natural or nearly natural sentences. To this end, we use an n-gram language model: n-grams are sequences of n words derived from a corpus; a corresponding n-gram model captures the probability of any n words appearing sequentially.

[3]Because of the rounding operation at step 3, some words with very low frequency might not be eventually placed in a bin. At the end, the sets $\mathcal{W}_B$ and $\mathcal{W}_S$ are accordingly updated to contain all words within the bins and words from the corpus not placed in a bin, respectively.

We use the OpenGrm library [10] for generating an n-gram language model L_R, encoded as cyclic weighted finite-state automaton (FSA), based on a corpus R. The generated n-gram model is represented in the form of an *acceptor*, i.e., an automaton with the input and output labels of a transition being equal. Finite-state acceptors are used to represent sets of strings; in the case of the n-gram model, the set of n-grams observed in a corpus R.

Every sentence in corpus R corresponds to a valid path in the model L_R. Every transition in the automaton is associated with a weight that represents the probability of the transition. For every valid path in the automaton, an associated probability can be computed as the accumulation of the transition weights along the path. We smooth the model using the Witten-Bell algorithm [4] so that there are valid paths in the model that do *not* correspond to any sentence in corpus R. These paths model unseen word sequences based on the statistics of the n-grams observed in R.

Word Choice Algorithm. Given a bin sequence $\mathcal{B}_1, \ldots, \mathcal{B}_m$, a language model L_R and a parameter k:

1. $\mathcal{B}_1, \ldots, \mathcal{B}_m$ is parsed into $\lceil \frac{m}{k} \rceil$ sub-sequences, each of k elements long.
2. For every sub-sequence, a corresponding FSA is generated; we refer to this FSA as the *bin model* associated with the given sub-sequence of bins:
   (a) For every bin $\mathcal{B}_i$, a state S_i is created. A state S_{k+1} is also created.
   (b) For every word $w_j \in \mathcal{B}_i$, a transition is added from state S_i to destination state S_{i+1}. The transition is marked with input label w_j and output label w_j.
   (c) For every state S_i and for every word $w_r \in \mathcal{W}_S$, a transition is added from state S_i to destination state S_i (self-loop). The transition is marked with input label w_r and output label w_r.
3. For every bin model generated, a sequence of $q \geq k$ words is produced:
   (a) The bin model is *intersected* with L_R; the resulting FSA is the *combined model*.
   (b) The shortest path in the combined model is computed, i.e., the path with the highest probability. The output labels of the shortest path is the sequence $w_{i1}, \ldots, w_{iq}$

The final stegotext $w_1, \ldots, w_{q'}$, with $q' \geq m$, is formed as the concatenation of the sequences $w_{i1}, \ldots, w_{iq}$, in the order they were produced.

Note that the *bin model* is an acceptor that represents a set A of all strings of words that can be derived by performing the cartesian product between the bins of a bin sub-sequence, namely all the possible combinations of the bin words of the sub-sequence. This acceptor also represents a set B of strings of words, which is a proper superset of set A. Set B includes all the strings of A augmented by an arbitrary number of stop words (words from set $\mathcal{W}_S$), placed in-between the bin words and at arbitrary places. The *combined model* model is an acceptor that represents strings of words accepted by both the language model and the bin model. These strings are essentially the ones appearing in the language model and (a) contain words from the bins in the given sub-sequence (in the order of the sub-sequence), (b) may or may not include stop words. The probabilities of L_R are retained in the combined model; by selecting, thus, the most probable path we select the most probable string of the combined model that satisfies (a) and (b), and also is the most likely to appear in natural language (according to language model L_R).

3.3 User Enhancement

The user is presented with a sequence of words and is asked to polish this to human-understandable sentences *without* changing the order of the given words – in our user-enhancement interface the bin words are presented in non-editable fields separated by editable fields where the user may insert new words and reuse/delete potentially generated stop words, at their will. In Fig. 2 we demonstrate examples of user-enhanced outputs of our encoder. Note that we have purposely excluded from the bin words set the 100 most frequently used English words (e.g., the words "a", "and", "to", etc.) in order to give flexibility to the user while enhancing the stegotext. We do not allow the users to add words that are included in the bins (our user-enhancement interface automatically checks and enforces this) since it would introduce bits in the original compressed bit sequence and could create ambiguity at the decoder. In our experiments users rarely attempted to enhance using bin words.

We note here that our scheme is not restricted to solely operate on the English language. The input of the encoder, i.e., the secret message, may be written in any language, e.g., in French. Such a message would get compressed using the *French* mixed Huffman codebook – generated in the same fashion as the English codebook (Section 3.1). The language of the output of our encoder may also be chosen by the user and, in addition, independently of the input language: for example, Alice wishes to hide a message written in French and send to Bob a stegotext written in German. To do so, she simply needs a corpus R in German, out of which the dictionary D_R

and the language model L_R are created (exactly as described in Section 3.2 with the only difference that set $\mathcal{E}$ should contain the most popular words of the targeted output language); these are used in the Bits-to-Words block to map the compressed bit sequence (regardless of which codebook was used during compression) into German sentences which Alice finally enhances. This is possible since no additional information about the targeted output language is needed to build D_R and L_R other than the one gained through the corpus R (frequency of words, n-grams etc.).

4 Decoder

In this section we mainly describe the Decompression block. Text Cleaning simply removes all words that are not in the word bins, and have been inserted during the User Enhancement. Words-to-Bits Mapping takes each word w_i and retrieves all bins into which this word appears. At this point we have a set of possible binary sequences. Decompression decides which of these actually occurred, by attempting to decompress each using our Huffman code and decide which of the resulting sentences is a natural sentence. For this last task it uses a state-based probability estimator.

Notation. Given a bin word w_i and a dictionary $\mathcal{D}$ with bins $\mathcal{B}_0, \ldots, \mathcal{B}_{2^b-1}$, we define a set $\mathcal{A}_i$ as the set of bins in which word w_i appears. Given a sequence of bin words $w_1, \ldots, w_m$ there exists a unique sequence $\mathcal{A}_1, \ldots, \mathcal{A}_m$. We define a set of states $\mathcal{S}_t$, $t \in \{1, \ldots, \lceil \frac{m}{r} \rceil\}$ and $r \leq m$:

$$\mathcal{S}_t = \mathcal{A}_{r(t-1)+1} \times \mathcal{A}_{r(t-1)+2} \ldots \times \mathcal{A}_{r(t-1)+r},$$

where $\times$ denotes the cartesian product over sets. We refer to r as the *grouping factor*.

By construction, every state in a set $\mathcal{S}_t$ is a distinct sequence of r bins. We refer to state $i \in \mathcal{S}_t$, as state i at *step* t. Each state in every step is associated with a *state probability* ε_i. We define a *transition probability* a_{ij}, between two states $i \in \mathcal{S}_t$ and $j \in \mathcal{S}_{t-1}$, such that:

$$a_{ij} = \mathbb{P}(s_t = i \mid s_{t-1} = j)$$

where s_t denotes the state at the current step and s_{t-1} the state at the previous step.

Decoding Algorithm. Given a dictionary $\mathcal{D}$ and a sequence of bin words $w_1, \ldots, w_m$, the decoder performs the following steps:

1. For every word w_i, a set $\mathcal{A}_i$ is constructed.

2. From the sequence $\mathcal{A}_1, \ldots, \mathcal{A}_m$ the sequence of sets $\mathcal{S}_1, \ldots, \mathcal{S}_T$, is constructed, where $T = \lceil \frac{m}{r} \rceil$, for some grouping factor $r \leq m$.

3. For every state $i \in \mathcal{S}_1$, its *initial* state probability is computed: $\varepsilon_i = \mathbb{P}(s_1 = i)$

4. For every state $i \in \mathcal{S}_t$, $1 < t \leq T$, its state probability ε_i is computed: $\varepsilon_i = \max_{j \in \mathcal{S}_{t-1}} \varepsilon_j \, a_{ij}$

Once all the state probabilities have been computed, the most probable state sequence $s_1^*, \ldots, s_T^*$ is derived as:

$$s_T^* = \arg\max_{i \in \mathcal{S}_T} \varepsilon_i$$

$$s_{t-1}^* = \arg\max_{j \in \mathcal{S}_{t-1}} \varepsilon_j \, a_{j s_t}$$

Finally, the estimation of the most probable message sent is computed as $m^* = C^{-1}(D^{-1}(s_1^*, \ldots, s_T^*))$.

Approximations. We assume that the state and transition probabilities can be approximated by the probabilities given by an m-order Markov model of the English characters $x \in \mathcal{X}$, where $\mathcal{X}$ denotes here all the printable ASCII characters. That is, we use probabilities of sequences of English characters to approximate the probabilities of bin sequences, i.e., of the states at each step t. A bin sequence can be converted back to a bit sequence using a dictionary $\mathcal{D}$, and consecutively to a message in English using a Huffman codebook $\mathcal{C}$.

The initial state probabilities are computed as follows:

$$\begin{aligned} \varepsilon_i &\approx \mathbb{P}(\mathcal{B}_1, \ldots, \mathcal{B}_r) = \mathbb{P}(D^{-1}(\mathcal{B}_1, \ldots, \mathcal{B}_r)) = \\ &= \mathbb{P}(C^{-1}(D^{-1}(\mathcal{B}_1, \ldots, \mathcal{B}_r))) = \mathbb{P}(x_1, \ldots, x_n) = \\ &= \prod_{j=1}^{n} \mathbb{P}(x_j | x_{j-(m-1)}, \ldots, x_{j-1}) \end{aligned}$$

The transition probabilities are computed as follows:

$$\begin{aligned} a_{ij} &\approx \mathbb{P}(\mathcal{B}_{r+1}, \ldots, \mathcal{B}_{r'} \mid \mathcal{B}_1, \ldots, \mathcal{B}_r) \\ &= \mathbb{P}(D^{-1}(\mathcal{B}_{r+1}, \ldots, \mathcal{B}_{r'}) \mid D^{-1}(\mathcal{B}_1, \ldots, \mathcal{B}_r)) \\ &= \mathbb{P}(C^{-1}(D^{-1}(\mathcal{B}_{r+1}, \ldots, \mathcal{B}_{r'})) \mid C^{-1}(D^{-1}(\mathcal{B}_1, \ldots, \mathcal{B}_r))) \\ &= \mathbb{P}(x_{n+1}, \ldots, x_{n'} \mid x_1, \ldots, x_n) \\ &= \prod_{j=n+1}^{n'} \mathbb{P}(x_j | x_{j-(m-1)}, \ldots, x_{j-1}) \end{aligned}$$

Key Points. We make two assumptions: (a) The message sent by Alice is valid English language, (b) Bob has at his disposal an m-order Markov model that accurately models the English language. The first assumption implies that messages with typos, very rare words, loans from spoken language, etc. may be very difficult or impossible to retrieve. The second assumption implies that the Markov model of the English characters has been trained over a large English text corpus and also for sufficiently large values of m, i.e., character sequences. The same assumptions hold in the case where Alice uses a different input language than English.

Corpus						Parameters		
Name	Description	$\lvert\mathcal{W}_R\rvert$	$\lvert\mathcal{W}_B\rvert$	$\lvert\mathcal{W}_S\rvert$	b	p	k
dreams	A Guardian article about dreams and academic anxiety.	425	277	148	6	5	4
animals	A collection of stories for kids about animals.	1,600	823	777	6	15	3
facebook	A longform article from BuzzFeed about Facebook.	1,423	461	962	5	15	3

Table 1: Input parameters for the MTurk experiments.

"I have ***become*** *tired of* ***facebook's*** *many* ***years*** *of existence. The* ***change*** *over the* ***years*** *by the* ***engineers*** *sucks. It seems* ***facebook's*** *wacky* ***algorithm*** *will* ***never*** *make sense. The* ***posts*** *make the* ***code*** *on* ***facebook*** *obsolete."*

(a) A human understandable stegotext enhancement.

"Does ***facebook's*** *CEO* ***feed*** *people* ***feed*** *dogs. Can't* ***yet*** *use* ***data*** *base* ***set*** *book. Two* ***posts*** *are* ***uses*** *people* ***facebook*** *apps. Mary* ***Cox*** *able* ***humans*** *into* ***keeping*** *up"*

(b) A sloppy stegotext enhancement.

Figure 2: Examples of stegotexts enhanced by MTurk users. The bin words are shown in bolt.

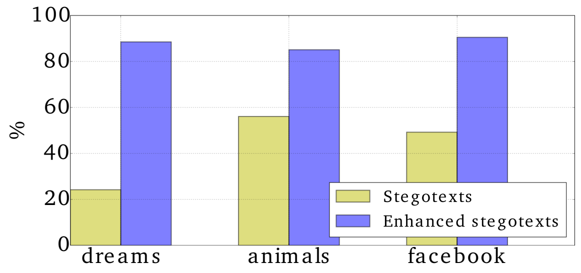

Figure 3: Sentences classified as natural language.

5 Experimental Evaluation

We use Amazon's Mechanical Turk (MTurk) [4] to evaluate our end-to-end system, and in particular the effort required by human users to turn various stegotexts produced by our encoder into meaningful natural language texts. We encode 50 different messages (of 4-15 words) using the parameters in Table 1. For each corpus R we produce a dictionary D_R and a 5-gram language model L_R, and we use the same mixed Huffman codebook. We restrict the location of the MTurk users to be the United States and we allow each user to enhance up to 3 stegotexts per experiment.

First, we measure the *completion time* (in minutes), i.e., the time needed to enhance the output of the encoder, and the *extra words* inserted by the user while doing so. We find that on average the users needed 5.0, 4.0 and 5.8 minutes, in each experiment respectively, which is comparable to the time needed to write a short e-mail. The average number of extra words per sentence was 8 for the *dreams* experiment, 4 for *animals* and 4 for *facebook*; by increasing p and reducing k we see that the produced sentences need less user-effort, i.e., they are closer to being good quality NL sentences, hence the users need to insert less words. To demonstrate the effectiveness of our approach, we compare to a *random* technique, where words are simply uniformly at random chosen from the bin sequence to produce the stegotexts. We observed that ours consistently outperformed the random one: the users needed always more time for enhancing and introduced more extra words. For example, users needed roughly 2 mins more on average to enhance the sentences of the *animals* experiment, and introduced almost double the number of extra words per sentence.

Second, we measure the *covert rate* achieved, that is, the number of hidden bits per word of enhanced stegotext (the sum of bin words and stop words produced by the encoder and the extra words inserted by users). In all our experiments the average covert rate achieved was roughly 3 bits/word; for a hidden message of 5 words, each of 5.5 characters on average, the final enhanced stegotext will be around 73 words.

We next use an One-Class Support Vector Machine (SVM) [11] to classify sentences as natural language (NL) or non-NL (out-of-class). We form a *training* set of 150K NL sentences derived from various texts of Wikipedia [8], the Brown [1] and the Reuters [2] corpora. For each sentence i in the training set we use the lexicalized Stanford Parser [5] to infer its score (the probability of the most likely parsing) and its Part-of-Speech (POS) tags to construct a feature vector with the score, the number of nouns, verbs, adjectives and adverbs (each normalized by the sentence length in words) and the length in words:

$$\mathbf{z}_i = <\text{score, \#nouns, \#verbs, \#adverbs, \#adjectives, length}>,$$

We use the observations $\mathbf{z}_1, \ldots, \mathbf{z}_n$ ($n = 150K$) to train the SVM using a Gaussian Radial Base Function (RBF). To check our classifier we tested 10K NL sentences – 95% were correctly classified – and 10K random sentences [6] out of which 97% were classified as non-NL.

We analyze (Fig. 3) the sentences produced by our encoder before and after enhancement from MTurk users,

[4]`https://www.mturk.com/mturk/`

[5]`http://nlp.stanford.edu/`

[6]Produced using an on-line random sentence generator

and we find that 25%, 58% and 50%, for each experiment respectively, were classified as NL; although the output of the encoder would not always pass as is the test, this result indicates that the sentences we produce are not completely non-NL. After user enhancement these percentages rise to 88% on average – some poorly enhanced sentences were detected (like the ones in Fig. 2(b)) as expected. For comparison, the sentences produced by the *random* technique were almost always classified as non-NL.

Finally, we investigate the performance of our probabilistic decoder by measuring the *character* error rate achieved. We build a 6-order Markov model of English characters using a large text corpus.[7] We found that in each experiment the decoded messages with *zero* error rate where 96%, 93%, 95% resp. For the few cases where the decoder did some mistakes, we found that the decoded messages where only partially corrupted, namely on average 15% of characters were wrongly decoded.

6 Related Work

Existing approaches to linguistic steganography apply automated text modifications techniques (see Bergmair [3] and references therein), the majority of which require access to sophisticated NLP tools and large linguistic datasets to operate. Moreover, they usually introduce unnaturalness into the text, easily detectable by steganalysis methods [12, 5], and they usually achieve a covert rate of less than 1 bit per word. For example, Spammimic [8] produces steganographic e-mails (with a rate of approx. 0.3 bits per word), that can get easily detected as suspicious due to long length and theme (Alice and Bob do not usually exchange spam). Closer to our work is the approach of Grosvald *et al.* [7]. Similarly to us, the proposed approach consists of mapping the hidden message to a sequence of words which the user modifies to produce text. Differently to us, no significant attention is given in forming the bin words sets and in choosing the words, so that minimal user-effort (indeed no formal evaluation is presented) is required. Finally, a demonstrated example implies a covert rate of 0.5 bits per word. Our approach differs from the existing ones since its design aims in achieving high covert rate, while remaining practical and usable. The "human-in-the-loop" design choice we do may require higher amount of user-effort but it also introduces elements that make robust our stegotexts to sophisticated attacks.

[7]The compressed data of the model amounts to 43MB.

[8]`http://spammimic.com/`

7 Acknowledgments

We are grateful to Balachander Krishnamurthy and David Applegate for their contributions; their employer's policy prevented them from being included as co-authors of this work.

References

[1] The brown corpus. `http://www.essex.ac.uk/linguistics/external/clmt/w3c/corpus_ling/content/corpora/list/private/brown/brown.html`. Accessed April 6, 2016.

[2] Reuters corpora. `http://trec.nist.gov/data/reuters/reuters.html`. Accessed April 6, 2016.

[3] BERGMAIR, R. A comprehensive bibliography of linguistic steganography. In *Electronic Imaging 2007* (2007), International Society for Optics and Photonics, pp. 65050W–65050W.

[4] CARPENTER, B. Scaling high-order character language models to gigabytes. In *Proceedings of the Workshop on Software* (2005), Association for Computational Linguistics, pp. 86–99.

[5] CHEN, Z.-L., HUANG, L.-S., YU, Z.-S., ZHAO, X.-X., AND ZHAO, X.-L. Effective linguistic steganography detection. In *Computer and Information Technology Workshops, IEEE 8th International Conference on* (2008), IEEE, pp. 224–229.

[6] FRY, E. B., AND KRESS, J. E. *The reading teacher's book of lists*, vol. 55. John Wiley & Sons, 2012.

[7] GROSVALD, M., AND ORGUN, C. O. Free from the cover text: a human-generated natural language approach to text-based steganography. *Journal of Information Hiding and Multimedia Signal Processing 2*, 2 (2011).

[8] JONES, E. Extracting text from wikipedia. `http://www.evanjones.ca/software/wikipedia2text.html`. Accessed April 6, 2016.

[9] MCGREW, D., AND VIEGA, J. The galois/counter mode of operation (gcm). *Submission to NIST. http://csrc. nist. gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec. pdf* (2004).

[10] OPENGRM. Ngram library. `http://www.opengrm.org/`.

[11] SCHÖLKOPF, B., WILLIAMSON, R. C., SMOLA, A. J., SHAWE-TAYLOR, J., PLATT, J. C., ET AL. Support vector method for novelty detection. In *NIPS* (1999), vol. 12, Citeseer, pp. 582–588.

[12] TASKIRAN, C. M., TOPKARA, U., TOPKARA, M., AND DELP, E. J. Attacks on lexical natural language steganography systems. In *Electronic Imaging* (2006), pp. 607209–607209.