Decipherment for Adversarial Offensive Language Detection

Zhelun Wu, Nishant Kambhatla, Anoop Sarkar
School of Computing Science
Simon Fraser University
Burnaby, BC , Canada
{zhelunw,nkambhat,anoop}@sfu.ca

Abstract

Automated filters are commonly used by online services to stop users from sending age-inappropriate, bullying messages, or asking others to expose personal information. Previous work has focused on rules or classifiers to detect and filter offensive messages, but these are vulnerable to cleverly disguised plaintext and unseen expressions especially in an adversarial setting where the users can repeatedly try to bypass the filter. In this paper, we model the disguised messages as if they are produced by encrypting the original message using an invented cipher. We apply automatic decipherment techniques to decode the disguised malicious text, which can be then filtered using rules or classifiers. We provide experimental results on three different datasets and show that decipherment is an effective tool for this task.

1 Introduction

Under-aged social media users and users of chat rooms associated with software like video games are routinely exposed to offensive language including sexting, profanities, age-inappropriate languages, cyber-bullying, and requests for personal identifying information. A common approach is to have a filter to block such messages. Filters are either rule-based (Razavi et al., 2010) or machine learning classifiers (Yin et al., 2009; Warner and Hirschberg, 2012; Williams and Burnap, 2015). However, users wishing to bypass such filters can subtly transform messages in novel ways which can be hard to detect.

Since malicious users and spammers can change their attacks to avoid being filtered, an approach to offensive text detection that takes into account this adversarial relationship is what can deal with real-world abusive language detection better. Techniques like spelling correction have been used to correct misspelled words in user generated content (Kobus et al., 2008), but in an adversarial setting it is easy to defeat a spelling correction system trained on predictable errors. Context based normalization methods have been proposed to convert erroneous user generated text from social media to standard text (Choudhury et al., 2007; Schulz et al., 2016). We, however, treat this problem as follows: we model malicious users trying to bypass a filtering system as having invented a new cipher, and our goal is to decipher their encrypted messages back to plaintext. We use a large space of possible ciphers that could be invented by those seeking to bypass a filter.

This paper addresses the problem of finding and filtering offensive messages as a special case of decipherment. In decipherment, a message in *plaintext* (original text) is converted into *ciphertext* (encrypted text). Encryption disguises the content of the original *plaintext* to a *ciphertext* so that it cannot be filtered out or blocked. Decryption or decipherment is the process of recovering the *plaintext* from the *ciphertext*. We treat disguised inappropriate content as ciphertext, and the actual intended content as the plaintext. Then we apply decipherment to recover more recognizable plaintext from the ciphertext and apply filters on the plaintext. Conceivably these users may create very complex unbreakable ciphers which cannot be deciphered by our system, but in such cases the ciphers are likely to also be unreadable by other humans who are the intended audience. We do not see many examples of this in our real-world chat messages.

We use Expectation Maximization (Dempster et al., 1977) and Hidden Markov Models (HMM, Rabiner (1989)) for our unsupervised decipherment method (Knight et al., 2006). We use an efficient beam search decoding algorithm to decipher ciphertext into the most likely plaintext. We also compare against supervised noisy channel models

Proceedings of the Second Workshop on Abusive Language Online (ALW2), pages 149–159
Brussels, Belgium, October 31, 2018.

and a state-of-art spelling correction system (Aspell). We discover that our method is immune to many previously unobserved types of changes made by users to disguise the original message.

2 Decipherment for Offensive Language Detection

The goal of decipherment is to uncover the hidden plaintext sequence $p_1 ... p_n$, given a disguised ciphertext sequence $c_1 ... c_n$. We assume that all corrupted forms of offensive language are originally explicit plaintext encrypted using letter substitution, insertion, or deletion.

2.1 NULL Insertion

A user can hide plaintext by inserting additional letters, or making substitutions and deletions in the plaintext:

Offensive text :`you are a B*n@n@ee`
Intended Plaintext :`you are a bunny`

we lowercase the texts and substitute the special symbols and punctuation marks inside the words with `NULL`, and then map `NULL` to any letter or `NULL`. Thus, the corrupted word `B*n@n@ee` is changed to

`b<NULL>n<NULL>n<NULL>ee`

When dealing with insertion of symbols, we treat it as an injective decipherment problem and solve it by adding `NULL` symbols in the ciphertext so that the decipherment model could map candidate letters. We compare two methods: 1) Inserting the `NULL` in a random position of the corrupted offensive key words; 2) Using a character-level n-gram model to segment the ciphertext adapting the technique for unsupervised word segmentation Ando and Lee (2003) in order to find where to insert `NULL` symbols.

The character-level n-gram model is trained on an unsegmented corpus (spaces removed between words). At each position k, we determine whether to insert a `NULL` symbol or not by calculating an n-gram score using Eqns (1) and (2).

$$v_n(k) = \frac{1}{2(n-1)} \sum_{d \in \{L,R\}} \sum_{j=1}^{n-1} I_{>}(c(s_d^n), c(t_d^n)) \quad (1)$$

$$v_N(k) = \frac{1}{N} \sum_{n \in N} v_n(k) \quad (2)$$

where n is the n-gram order, and s are n-grams that straddle the potential `NULL` position k to the left L and right R and t are n-grams that are on either side of position k also on the left L and right R and $c()$ is the n-gram frequency. For a sequence `A B C D W X Y Z`, for $n = 4$ and position $k = 4$ between `D` and `W`, Eqn (1) for a particular n compares the frequency of the s type n-grams that straddle position $k = 4$ (so, in this case frequency of `B C D W`, `C D W X`, `D W X Y` against the frequency of the t type n-grams on either side: `A B C D` and `W X Y Z`. Eqn (2) then takes the average for each n-gram order (for n=1,2,3,4). The position k that has a score from Eqn (2) higher than a threshold value is chosen as the position for `NULL` insertion.

2.2 Decipherment Model

A decipherment model maximizes the probability of a substitution map that converts the ciphertext sequence c to plaintext sequence p (Eqn 3).

$$P(c) = \sum_{e} P(p) \cdot P(c \mid p) \quad (3)$$

where $P(.)$ is a character-level language model of the plaintext source trained using a monolingual corpus. We model $P(c \mid p)$ as the emission probability of a Hidden Markov Model (Knight et al., 2006) with cipher characters c as observations and plaintext characters p as the hidden states of the HMM. We train this using unsupervised learning using the Forward-Backward algorithm (Rabiner, 1989).

We propose our own initialization algorithm (Algorithm 1), based on the assumption that the previous trained table can help us to reach better local optima with fewer iterations compared to uniform initialization with random perturbations.

With the learned posterior distribution and language model score we use beam search to obtain the best plaintext (Forney Jr, 1973; Nuhn et al., 2013). Beam search combines breadth-first search and child pruning reducing the search space for each partial hypothesis of the decipherment.

Algorithm 1 Initialization with previous trained table

Given a set of cipher text sentences with size of d, a plain text with vocabulary size v and plain text trigram model b
Randomly initialize the $s(c \mid e)$ substitution table and normalize
for $i = 1 \ldots d$ **do**
    preprocess the i-th cipher text sequence by removing repeated characters and lower case the text
    insert `NULL` based on techniques in Sec. 2.1
    if $i \neq 1$ **then**
        initialize $s(c \mid e)$ using the $(i-1)$th trained table $s(c \mid e)$
    apply Forward-Backward algorithm to learn parameters $s(c \mid e)$

3 Data

3.1 Wiktionary

We created an offensive language dataset from English Wiktionary data. We use Wiktionary labels for vulgar, derogatory, etc. and selected example sentences for words tagged with such labels. From the entire English Wiktionary data , using this method, we extracted 2298 offensive sentences and 152,770 non-offensive sentences. This data is organized as {`key word : example sentence`} as in a dictionary. With the duplicates removed, for each `key word`, the corresponding sentences were split into a 3:1 ratio as training and testing data.

We used 1,532 sentences offensive training set data and sampled 1,532 non-offensive sentences as a balanced dataset for training an offensive sentence classifier. We split 716 offensive testing sentences into 4 parts in sequence. The first three parts we set as test sets A, B and C and the latter part as development set. Every set has 179 sentences. The three test sets are taken from the same larger corpus to measure the variance in performance of the decipherment powered classification technique.

The Wiktionary dataset is used to train our language model. The offensive sentences had fewer instances than the non-offensive counterparts. To offset any loss in information, the offensive sentences were duplicated until they were as many in number as the non-offensive sentences, resulting in a balanced training set of 155,251 tokens.

3.2 Other data

Language Model: For the character models we used a combination of Wiktionary and the European Parliament Proceedings Parallel Corpus 1996-2011 (Koehn, 2005). 100K English sentences were sampled from the German-English EuroParl Corpus to give us a 2.7M token English plaintext corpus.

Spelling Correction: We use the English Gigaword corpus (Graff et al., 2003) to train the language model for the noisy channel spelling correction module. The preprocessed corpus has 7.4M lowercase English word tokens. For spelling correction, the Linux system dictionary with 479,829 English words in lower case is used. We used 3,393,741 pairs of human disguised words and original plain text words from the rule-based filtering system to train the error model.

3.3 Real World Chat Messages

We obtained 4,713,970 unfiltered, unlabelled chat messages from a provider of filtering services for chat rooms aimed at under-aged users. The data was provided by *Two Hat Security* which provides human powered chat room monitoring and filtering. Two Hat combines artificial intelligence with human expertise to classify chat room discussions and images on a scale of risk, taking user profiles and context into account. We pre-processed the chat messages to produce a cleaner version of the original message using tokenization rules.

We used 4,700,000 chat messages provided to us by Two Hat Security as a training set to train a letter based language model. To build a more comprehensive language model, this language model was interpolated with the ones trained on Wiktionary and Europarl datasets.

We also collect a separate set of 500 plaintext chat messages flagged as offensive by the filtering system to use as a development set. For the test set, we sampled 500 chat messages from 265,626 chat messages which were identified as offensive using a rule-based filtering system that uses lists of offensive words. These messages were marked inoffensive and cleared for posting in an older version of the same rule-based filtering system (two years older). We use this data to evaluate our decipherment system: can we match the performance improvement of two years of rule development?

Over two years new rules were added to account for user behaviour of trying to bypass the rule-based filtering system. Can decipherment discover these patterns automatically?

4 Experimental Setup

4.1 Language Model

Character Language Model: We used the SRILM toolkit (Stolcke et al., 2002) to train a character language model (LM) from Wiktionary and Europarl data. We trained two LMs and interpolated them using a mixture model. The interpolation weights were tuned on the development set. We used `py-srilm-interpolator` (Kiso, 2012) for the mixture model.

Word Language Model: We trained a word trigram language model on English Gigaword data.

4.2 Encipherment

The sentences in the test set are encrypted using different techniques: the Caesar 1:1 substitution cipher, the Leet simple substitution cipher[1] and replacing the offensive keywords with real human-disguised versions of curse words obtained from the rule-based filtering system. This was done to mimic the way people disguise their messages online.

4.3 Decipherment

HMM: For the HMM based decipherment, we run 100 random restarts (Berg-Kirkpatrick and Klein, 2013), running the EM algorithm to convergence 100 times, to find the initialization that leads to the local optima with highest likelihood.

Spelling Correction: We use settings for the noisy channel model based spelling correction as stated in Norvig (2009): $p_{spell\_error} = 0.05$, and $\lambda = 1$. The maximum edit distance limit is set to 3, and the error model is trained on the real chat messages we collected. The error model trained on pairs of misspelled words and the correctly spelled English words from the Linux system dictionary.

4.4 Evaluation

We evaluate our approach in terms of the classifier accuracy and the risk level from the rule-based filtering system for the real chat messages. We are using a simple logistic regression classifer from the LibShortText toolkit (Yu et al., 2013) to train a classifier to classifies offensive and normal sentences. After training the classifier, we classify the original test sentences without any encryption.

We do not use a very sophisticated classifier because the goal of classification here is not to correctly classify offensive messages, but rather, to measure how well the decipherment method can recover the original messages containing offensive text back from the encrypted messages. We compare the classification accuracy between the original and deciphered messages. If the classification accuracy gap between the original and deciphered messages is small, the decipherment approach can recover the original user-intended messages from encrypted messages.

Tuning: We have a set of 179 sentences as the development set which is used for tuning the classifier and other hyper-parameters. After tuning, we were able to achieve the highest classification accuracy of 86%.

5 Experiments and Results

On the test sets A, B and C (see Sec. 3.1), in both our experiments on Caesar ciphers and Leet code, we first measure the number of offensive instances that were correctly classified by our logistic regression classifier. Then, the same classifier is run on the encrypted versions of these messages. The first two columns in the Table 1 and Table **??** show the classification accuracy obtained in those settings.

In our experiments, we apply Spelling Correction and our Decipherment techniques on the encrypted messages separately and finally use our classifier to determine the number of offensive instances that were correctly classified in decrypted text. The objective here is to measure how well either of the techniques is able to recover the originally intended message from the encrypted version of the message, simulating a real online user's behaviour of disguising an offensive text to beat a rule-based filter. In other words, we aim at decreasing the gap between the classification accuracies on a test set and its encrypted form.

5.1 Decipherment of Caesar Ciphers

Caesar cipher is a simple substitution cipher in which each plaintext token is 'shifted' a certain number of places down the alphabet. We encrypted three test sets A, B, C with Caesar cipher encryption by shifting 3 letters to the right.

[1] `https://en.wikipedia.org/wiki/Leet`

Test Set	Original text	Caesar cipher encrypted text	Noisy Channel Spelling Correction	Deciphering per line	Deciphering whole set
A	86% (154/179)	4% (8/179)	28% (51/179)	55% (99/179)	86% (154/179)
B	86% (154/179)	3% (7/179)	22% (41/179)	51% (92/179)	86% (154/179)
C	84% (152/179)	5% (10/179)	22% (41/179)	58% (104/179)	84% (152/179)

Table 1: Classification Accuracy of Spelling Correction and Decipherment Results in Caesar Cipher Encrypted Text

After being encrypted with a Caesar cipher, all the letters in the original messages are replaced, making the text unreadable. Applying spelling correction on such text sequences is almost futile as the text does not contain lexically correct words. HMM based decipherment approach, however, is capable of handling such obfuscated text sequences by design. We apply both the methods on three separate test sets A,B and C that built from the Wiktionary dataset (see Sec. 3.1).

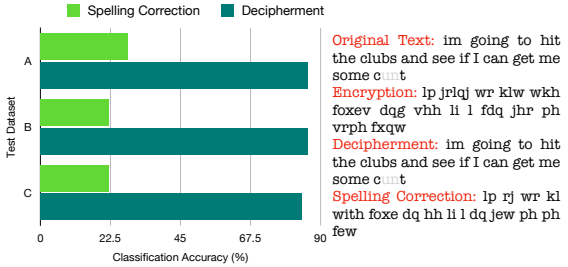

Figure 1: Decipherment vs. Spelling Correction approaches on Caesar cipher encrypted offensive messages. (Example from Test set *A*.)

The classification accuracies in Table 1 show that the HMM decipherment approach outperforms our other baselines.

With 100 random restarts on EM during the decipherment of the Caesar cipher encrypted dataset A, our system reports a mean loglikelihood of -27923 and a standard deviation of 23. The Figure 2 shows every loglikelihood value in 100 random restarts. As the Caesar cipher is relatively easy to decipher, the loglikelihood does not vary greatly. The highest loglikelihood yields a classification accuracy of 86% (154/179), which is the same as the original plain text classification accuracy. The decipherment process seems to recover the whole message that was encrypted by the Caesar cipher.

5.2 Decipherment of Leet Substitution

Leet is a quasi-encryption method which makes use of modified spellings and verbiage primarily used by the Internet community for many pho-

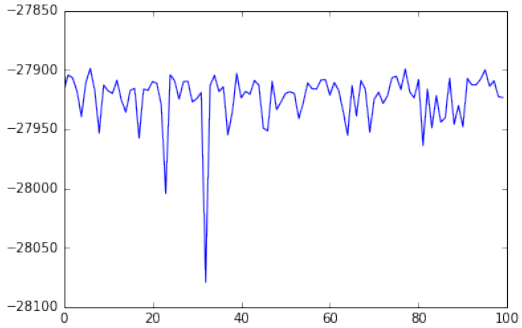

Figure 2: 100 Random Restarts Loglikelihood in Caesar Cipher Decipherment

netic languages. For the Leet substitution cipher, we referred to the KoreLogic's Leet rules (Kore Logic Security, 2012) which tagged as "#KoreLogicRulesL33t" . We used John the Ripper password cracker (Solar Designer and Community, 2013) to apply the KoreLogic Leet rules encrypting our test sets.

Encrypting a text with a Leet substitution does not change all the letters in the original messages. A spelling correction, therefore, performs better on such text sequences compared to the ones enciphered with a Caesar cipher. So the noisy channel spelling correction is able to recover the original form of some of the corrupted messages. HMM based decipherment approach, however, still manages to outperform spelling correction. For decipherment of Leet ciphers, we employ beam search to decode the final results having obtained the posterior probabilities from EM training. We apply both the methods on the test sets A,B and C.

Table 1 shows that the noisy channel spelling correction method is able to obtain an average classification accuracy of 65 out of the 179 encrypted messages. For HMM based decipherment techniques, however, we record higher accuracy, as before. With a beam width of 5, when applied to the whole set instead of a per-sentence basis, the resulting messages are flagged at an average of 82% across the three test sets. The *deciphered*

Test Set	Original Text	Leet Encryption Set	Noisy Channel Spelling Correction	Deciphering per line	Deciphering whole set
A	86% (154/179)	59% (107/179)	68% (122/179)	56% (102/179)	82% (147/179)
B	86% (154/179)	64% (115/179)	60% (108/179)	62% (112/179)	82% (148/179)
C	84% (152/179)	62% (112/179)	65% (117/179)	67% (121/179)	81% (146/179)

Table 2: Classification Accuracy of Spelling Correction and Decipherment Results in Leet Substitution Cipher with Beam Search Width of 5

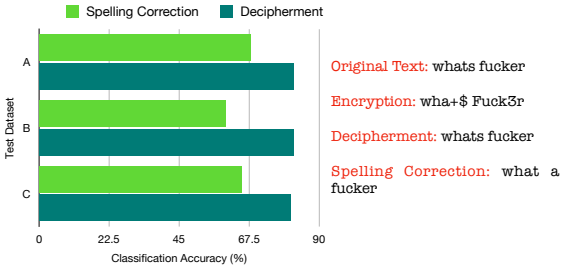

Figure 3: Decipherment vs. Spelling Correction approaches on Leet cipher encrypted offensive messages. (Example from Test set A.)

messages appear quite close to being as accurate as the original raw test set.

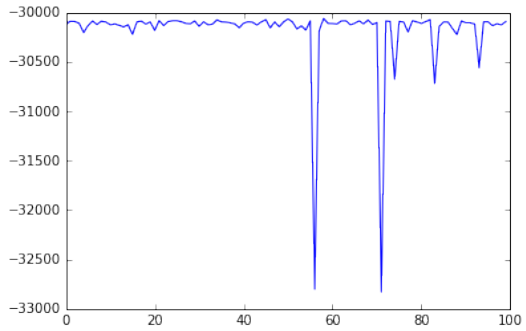

Figure 4: 100 Random Restarts Loglikelihood in Leet Substitution Cipher Decipherment

On performing 100 random restarts on EM during the decipherment of the Caesar cipher encrypted test dataset A, our system records an average loglikelihood of -30183 and a standard deviation of 390. Figure 4 shows every loglikelihood value in 100 random restarts. The high deviation in the loglikelihood shows that a Leet substitution is harder to solve than a Caesar cipher.

5.3 Decipherment of Real Chat Offensive Words Substitution Dataset

In each of the offensive test sets A, B and C, the offensive keywords in the messages are substituted with real human corrupted words obtained from real-word chat messages. These chats are transformed versions of these offensive words that are matched using a hand-written rule-based system. We used enciphered offensive words collected from real chat messages and the corresponding plain text for this task.

Original Text: hes really bitchy in the morning
Encryption: hes really bitchyou in the morning
Decipherment: hes really bitchy in the morning
Spelling Correction: hes really bitchy in the morning

Since this quasi-encryption is based on real chat messages, it closely mimics the inventive ways users employ to disguise their messages to bypass the filter system, which can involve both insertion and deletion.

It is then imperative that we handle insertion, deletion and substitution in the disguised words to recover the original plain text words. In the insertion case, for example, if the original word `hello` is disguised as `helo`, a `NULL` symbol is inserted inside the disguised word `helo` to decipher. The ideal position to insert the `NULL` symbol is `he<NULL>lo` but it is not known during training. To circumvent this, we experiment with two techniques: (1) to insert `NULL` at random before the beginning of the EM training, and (2) to insert the `NULL` using the method in Sec. 2.1.

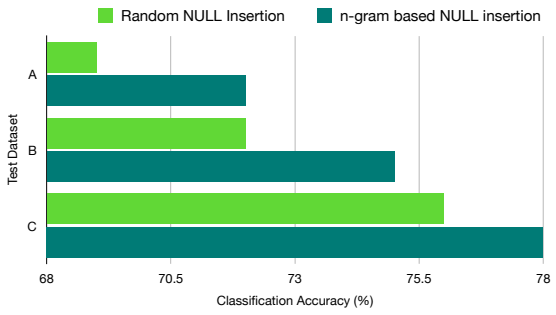

Figure 5: Random vs. n-gram count based `NULL` insertions.

This insertion techniques uses the word-

Test Set	Wiktionary Encrypted Set	Noisy Channel Spelling Correction	Aspell Spelling Correction	NULL Insertion	Decipherment
A	64% (116/179)	72% (130/179)	77% (138/179)	Random	69% (124/179)
				n-gram Count Based	72% (129/179)
B	72% (129/179)	76% (137/179)	73% (131/179)	Random	72% (130/179)
				n-gram Count Based	75% (136/179)
C	75% (136/179)	77% (138/179)	77% (138/179)	Random	76% (137/179)
				n-gram Count Based	78% (140/179)

Table 3: Classification Accuracy of Spelling Correction and Decipherment Results in Real Chat Offensive Words Substitution Wiktionary Dataset

level language model 4.1 to learn the n-gram counts.From Table 3 and Figure 7, the n-gram count based insertion decipherment has a higher classification accuracy than the random insertion NULL decipherment. An advantage of this type of HMM decipherment method is that it tends to not change the words that are already correct since these words have the highest language model score. Rather, it changes the words that are corrupted or misspelled. Irrespective of the type of encryption, the HMM decipherment approach always deciphers the messages which fit with the language model we trained.

We conducted an additional experiment to test with the Aspell program. Table 3 shows that deciphered messages obtain better accuracies than Aspell on test sets B and C while on test set A, it is about 5% less accurate. Note that the noisy channel model was trained on data obtained from a hand-tuned filtering system that was domain specific and created specifically for these chat messages. What we find in our results is that the decipherment system can match this domain expertise using unsupervised learning without any domain specific knowledge.

Figure 6 shows every loglikelihood value in 100 random restarts on test-set B. The mean of loglikelihood is -41444.7 and the standard deviation was 115.53. The greater diversity among the real chat offensive words renders the decipherment much harder than controlled synthetic-scenarios.

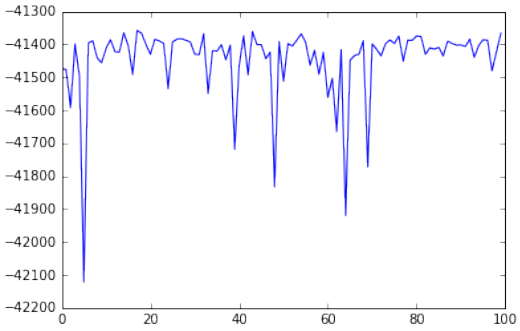

Figure 6: 100 Random Restarts Loglikelihood in Decipherment of Real Chat Offensive Words Substitution on Test Set B

5.4 Decipherment of Real Offensive Chat Messages

We use two versions of a rule-based offensive language filtering system to evaluate our decipherment approach. The first version of the filter system was two years old and the second version of the filter system has benefited from daily updates from humans monitoring the chat room. The question we want to ask in this evaluation is if our decipherment system can replicate the human effort of two years of rule development in this filtering system.

We use a test set of 500 offensive messages sampled from the set of chat room messages which were *not* flagged as offensive by the first (older) version of the filter system but which *were* flagged as offensive by the second version of the filter system. This allows us to evaluate how many of those offensive messages can be identified using decipherment.

User corrupted text: fvk u
User corrupted text: f2ck u
Deciphered text: fuck you

Before deciphering these messages, we preprocessed the text to remove repeated characters such that only two sequential repeated characters re-

main. For example, the text `heeeellllooo` is preprocessed to `heelloo`. Further, all the special symbols were replaced with `<NULL>`.

The preprocessed test-set is then deciphered with the HMM based decipherment technique. The deciphered messages are then passed back into the first version of the filtering system. This lets us check if deciphering the user-disguised message into a simpler plaintext message allows the two year old filter to catch it as offensive. The above example shows that decipherment can handle substitutions and insertions.

The two year old rule-based filter was able to successfully flag **51.6%** of the deciphered messages as offensive text. Thus, the decipherment approach is able to transform about half of the corrupted messages into a readable form so that the filter system without the benefit of two years of development can recognize them as offensive. This shows that decipherment can lead to much faster development of filtering systems for offensive language.

6 Discussion

We observe that HMM decipherment trained on the whole set is particularly able to recover most encrypted letters (cipher text) into their original letters (plain text). When we decipher each line

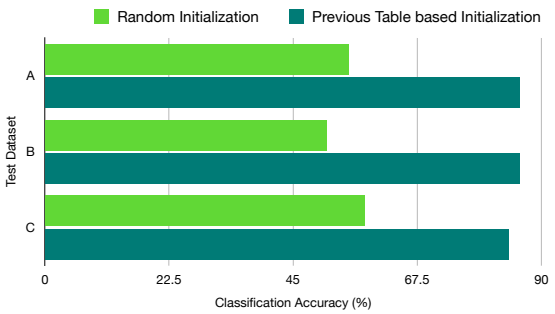

Figure 7: Decipherment per line vs on whole set.

of messages individually, the EM training does not observe enough data to learn the posterior probabilities. In contrast, the substitution table trained by previous messages being passed into next message initialization table will not lose the information learned from the previous messages. Therefore, the results of whole set decipherment are better than per line decipherment.

As Tables (1 2, 3) show, HMM decipherment training can recover most of the words in our test sets compared to the original message classification accuracy of each set. From the results shown in Table 1 and Table 2, no matter what encryption of substitution cipher was used, be it Caesar cipher or Leet substitution cipher, the HMM decipherment with language model could always recover the original messages.

7 Related Work

For detecting offensive languages, rule-based filtering systems and machine learning techniques are widely used. Razavi et al. (2010) leveraged a variety of statistical models and rule-based patterns and applied multi-level classification for flame detection. Chen et al. (2012) introduced a Lexical Syntactic Feature (LSF) architecture to detect offensive content and identify potential offensive users in social media. Kansara and Shekokar (2015) proposed a framework that detects bullying texts and images in using feature extraction and classifiers. Djuric et al. (2015) leveraged word embedding representations (Mikolov et al., 2013) to improve machine learning based classifiers. Nobata et al. (2016) unified predefined linguistic elements and word embeddings to train a regression model. Su et al. (2017) presented a system to detect and rephrase profane words written in Chinese. samghabadi2017detecting

Recently, deep learning methods have been employed for abusive language detection. Zhang et al. (2018) presented an algorithm for detecting hate speech using a combination of Convolutional Neural Networks (CNN) and Gated Recurrent Unit (GRU). Gambäck and Sikdar (2017) used four CNNs to classify tweets to one of four predefined categories. Park and Fung (2017) adopted a two-step approach with two classifiers. The first step performs classification on abusive language and the second step classifies a text into types of sexist and racist abusive language given that the language is abusive. Three CNN-based models have been used for classification: CharCNN, WordCNN and HybridCNN. Badjatiya et al. (2017) used an LSTM model with features extracted by character n-grams for hate speech detection. As malicious users can change their ways of transforming text to avoid being filtered, an approach such as ours which takes into account the adversarial relationship between the chat room user and the offensive language filter is likely to perform better at filtering.

Previous work on decipherment are often based on noisy-channel frameworks. Knight and Yamada (1999) proposed to use EM algorithm to estimate the mapping distribution over the sound to characters, then generated the plaintext using Viterbi algorithm (Forney Jr, 1973). The learning objective is to maximize the probability of the mapping ciphertext phoneme-tokens to plaintext characters. Further, Knight et al. (2006) studied using EM for unsupervised learning of the substitution maps for 1:1 letter substitution ciphers. Ravi and Knight (2011) proposed to regard foreign language as ciphertext and English as plaintext, converting a translation problem into one of word substitution decipherment. They employed iterative EM approach and a Bayesian learning approach to build the translation mechanism using monolingual data.

We use the EM based decipherment technique to convert user-disguised offensive text into a filter-recognizable plaintext.

8 Conclusion

The HMM decipherment can decipher disguised text based on the language model regardless of the encryption type. The decipherment approach we proposed can cover more disguised cases than spelling correction methods. However, due to the limitation of edit distance and lack of traning data, the noisy channel spelling correction has its limitations and cannot handle high edit distance case. Large edit distances are common in real chat messages, and thus the decipherment approach has its advantages. The difference between the decipherment with traditional spelling correction methods like Aspell is that decipherment method only needs a language model to decipher cipher text, and does not need a dictionary to refer to. The language model has the advantage that we can train a domain specific language model to decipher specific topic messages as real chat messages usually have some sort of topics or domain, such as sports, news and so on. The future work is that we can try different language messages to decipher, as long as we can have the corresponding language model.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful remarks. The research was also partially supported by the Natural Sciences and Engineering Research Council of Canada grants NSERC RGPIN-2018-06437 and RGPAS-2018-522574 and a Department of National Defence (DND) and NSERC grant DGDND-2018-00025 to the third author. We would also like to thank Ken Dwyer and Michael Harris from the Two Hat Security Company who collected and organized the real-world chat data for us. Thanks also to the Two Hat CEO, Chris Priebe, who supported this research and also provided an internship to the first author where he worked on spelling correction for chats.

References

Rie Kubota Ando and Lillian Lee. 2003. Mostly-unsupervised statistical segmentation of japanese kanji sequences. *Natural Language Engineering*, 9(2):127–149.

Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. 2017. Deep learning for hate speech detection in tweets. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 759–760. International World Wide Web Conferences Steering Committee.

Taylor Berg-Kirkpatrick and Dan Klein. 2013. Decipherment with a million random restarts. In *EMNLP*, pages 874–878.

Ying Chen, Yilu Zhou, Sencun Zhu, and Heng Xu. 2012. Detecting offensive language in social media to protect adolescent online safety. In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*, pages 71–80. IEEE.

Monojit Choudhury, Rahul Saraf, Vijit Jain, Animesh Mukherjee, Sudeshna Sarkar, and Anupam Basu. 2007. Investigation and modeling of the structure of texting language. *International Journal of Document Analysis and Recognition (IJDAR)*, 10(3-4):157–174.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. *Maximum likelihood from incomplete data via the EM algorithm*. Journal of the royal statistical society. Series B (methodological), page 1-38.

Nemanja Djuric, Jing Zhou, Robin Morris, Mihajlo Grbovic, Vladan Radosavljevic, and Narayan Bhamidipati. 2015. Hate speech detection with comment embeddings. In *Proceedings of the 24th international conference on world wide web*, pages 29–30. ACM.

G David Forney Jr. 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Björn Gambäck and Utpal Kumar Sikdar. 2017. Using convolutional neural networks to classify hate-speech. In *Proceedings of the First Workshop on Abusive Language Online*, pages 85–90.

David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2003. English gigaword. *Linguistic Data Consortium, Philadelphia*.

Krishna B Kansara and Narendra M Shekokar. 2015. A framework for cyberbullying detection in social network. *International Journal of Current Engineering and Technology*, 5.

Tetsuo Kiso. 2012. A python wrapper for determining interpolation weights with srilm. https://github.com/tetsuok/py-srilm-interpolator.

Kevin Knight, Anish Nair, Nishit Rathod, and Kenji Yamada. 2006. Unsupervised analysis for decipherment problems. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 499–506. Association for Computational Linguistics.

Kevin Knight and Kenji Yamada. 1999. A computational approach to deciphering unknown scripts. In *ACL Workshop on Unsupervised Learning in Natural Language Processing*, pages 37–44. 1.

Catherine Kobus, François Yvon, and Géraldine Damnati. 2008. Normalizing sms: are two metaphors better than one? In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 441–448. Association for Computational Linguistics.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86.

Kore Logic Security. 2012. Kore logic custom rules. http://contest-2010.korelogic.com/rules.txt.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. 2016. Abusive language detection in online user content. In *Proceedings of the 25th international conference on world wide web*, pages 145–153. International World Wide Web Conferences Steering Committee.

Peter Norvig. 2009. Natural language corpus data. *Beautiful Data*, pages 219–242.

Malte Nuhn, Julian Schamper, and Hermann Ney. 2013. Beam search for solving substitution ciphers. In *Citeseer*.

Ji Ho Park and Pascale Fung. 2017. One-step and two-step classification for abusive language detection on twitter. In *Proceedings of the First Workshop on Abusive Language Online*, pages 41–45.

Lawrence R Rabiner. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Sujith Ravi and Kevin Knight. 2011. Deciphering foreign language. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 12–21. Association for Computational Linguistics.

Amir H Razavi, Diana Inkpen, Sasha Uritsky, and Stan Matwin. 2010. Offensive language detection using multi-level classification. In *Canadian Conference on Artificial Intelligence*, pages 16–27. Springer.

Sarah Schulz, Guy De Pauw, Orphée De Clercq, Bart Desmet, Veronique Hoste, Walter Daelemans, and Lieve Macken. 2016. Multimodular text normalization of dutch user-generated content. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 7(4):61.

Solar Designer and Community. 2013. John the ripper password cracker. http://www.openwall.com/john.

Andreas Stolcke et al. 2002. Srilm-an extensible language modeling toolkit. In *Interspeech*.

Hui-Po Su, Zhen-Jie Huang, Hao-Tsung Chang, and Chuan-Jie Lin. 2017. Rephrasing profanity in chinese text. In *Proceedings of the First Workshop on Abusive Language Online*, pages 18–24.

William Warner and Julia Hirschberg. 2012. Detecting hate speech on the world wide web. In *Proceedings of the Second Workshop on Language in Social Media*, pages 19–26. Association for Computational Linguistics.

Matthew L Williams and Pete Burnap. 2015. Cyberhate on social media in the aftermath of woolwich: A case study in computational criminology and big data. *British Journal of Criminology*, 56(2):211–238.

Dawei Yin, Zhenzhen Xue, Liangjie Hong, Brian D Davison, April Kontostathis, and Lynne Edwards. 2009. Detection of harassment on web 2.0. In *Proceedings of the Content Analysis in the WEBb*, pages 1–7.

H Yu, C Ho, Y Juan, and C Lin. 2013. Libshorttext: A library for short-text classification and analysis. Technical report, University of Texas at Austin and National Taiwan University.

Ziqi Zhang, David Robinson, and Jonathan Tepper. 2018. Detecting hate speech on twitter using a convolution-gru based deep neural network. In *European Semantic Web Conference*, pages 745–760. Springer.