Decoding Running Key Ciphers

Victor Redko
260659220

Marie Payne
260686859

Abstract

Artificial intelligence and natural language processing techniques can be used to decode substitution ciphers given the plaintext and keytext follow conventions of the English language. Groupings of n letters for n as large as 3 can be used in decoding methods comprising the Viterbi algorithm and in classifiers such as Multinomial Naive Bayes, logistic regression and support vector machines. We evaluate these methods with a baseline comparison model.

1 Introduction

In cryptography, a letter substitution cipher is constructed by inputting a plaintext and a key into a substitution function to produce a ciphertext. The original plaintext can be recovered by inputting the ciphertext and the key into the inverse of the function. A common choice for the substitution scheme is the *tabula recta,* where c = (p + r) mod 26 for same-case letters in the English alphabet, where c is the ciphertext, p is the plaintext and r is the keystream. The function is iterated over every letter in the plaintext, presuming the key is truncated if longer than the plaintext or wrapped around if shorter. If the plaintext or key contain patterns that can be deduced, then they can be decoded through the ciphertext by the means of statistical attacks. In the case that the key is perfectly random, never reused, and kept secret, this results in an unbreakable one-time pad. A variation of the substitution cipher, the running key cipher, uses a stream of characters from, for example, a book to create a long non-repeating key. The plaintext is typically preprocessed to remove occurences of whitespace and punctuation. An example of a generated ciphertext using this method and the *tabula recta* substitution function is given in Table 1.

Since we know the plaintext and keystream are both obtained from the English language, our goal is to determine the most likely plaintext and keystream belonging to a subset of English that produced the given ciphertext.

2 Related Work

N-grams were initially proposed as a method of statistical attack for substitution ciphers and further explored by Bauer and Tate, (Craig Bauer, 2002). Comparisons were made between different values of N up to 6, and tested on ciphertexts of length 1000 produced from the Project Gutenberg corpus with an average accuracy of 30%. This provided a foundation with which to base other methods on, but doesn't produce viable plaintexts as a single concept.

Griffing (Griffing, 2006) employed the Viterbi algorithm, a dynamic programming method for finding the most likely sequence of hidden states or the Viterbi path, in tandem with Hidden Markov models. This method achieves a median 87% accuracy rate on ciphertexts of length 1000. A shortcoming with this work is that it is computationally intensive, with results searching through 26^5 states at every position in the ciphertext. It also doesn't perform any comparisons between sources or baselines.

Corlett and Penn (Eric Corlett, 2010) produced a method that employs the A* search and Viterbi algo-

Table 1: To encode a message using a running key cipher, letters at corresponding positions in the message and key are added together modulo 26 after being converted to numbers, with 'a' at 0. The ciphertext is composed after these numbers are turned back into letters. This scheme assumes that only the 26 English letters of the alphabet are used.

Plaintext	NLPISCOOL	13 11 15 8 18 2 14 14 11
Key	FUNFUNFUN	5 20 13 5 20 13 5 20 13
Ciphertext	SFCNMPTIY	18 5 2 13 12 15 19 8 24

rithms to decode various ciphertexts sourced from websites. Since this method is exact, it will determine the plaintext and keystream with a 100% accuracy, however it is computationally involved and can lead to exponential time consumption.

Reddy and Knight (Sravana Reddy, 2012) used a method with Gibbs sampling of n-grams up to 6 to produce iterations of probable plaintext/keystream pairs. They tested their methods on the Wall Street Journal and Project Gutenberg corpuses with varying results depending on which source the keytext and plintext came from. On average, their method consistently produced accuracies of 93% on ciphertexts of length 1000. A shortcoming of this work is that it was hard to reproduce given the small level of detail in their paper, unfortunately.

3 Methods

In this paper, we implement an n-gram solution and perform tests with Multinomial Naive Bayes, logistic regression and support vector machine classifiers, as well as implement the Viterbi algorithm and compare its performance based on different corpora and ciphertext length used.

3.1 Data Preprocessing

Our implementation takes the corpus as a txt input and removes all punctuation and whitespace characters through regex matching before converting to lowercase. Each sentence gets stored as its preprocessed form linked to its raw, human readable form. The spaces need to be rejected in order to properly determine the n-grams of the training corpora, with the methods we used. All characters that aren't part of the English 26-letter alphabet are removed.

3.2 N-Grams

If the same corpus is used for both testing and training, it is divided as a ratio of 80-20 training-testing for optimal results. Our implementation allows for different corpora to be used for testing and for training through command line arguments, however. All possible n-gram combinations are computed based on the alphabet, then linked to a probability based on the training source text. These probabilities are then fed into the different classifiers for performance comparison. Our baseline model doesn't calculate predictions based off of the n-gram frequencies but rather through random estimation of the letters. The Hidden Markov Model predicts the state, transition and emission probabilities through the frequency distribution of n-grams, which is then fed into our implementation of the Viterbi algorithm to predict plaintext/keystream pairs.

3.3 Viterbi Algorithm

The Viterbi algorithm is based on a Hidden Markov Model (HMM) representation of the text, where the states represent the letters at each position, the transition probabilities are interpreted as the n-gram frequencies, and the emissions are determined based on those frequencies rather than aligned emission. The most probable 'path' of letters is computed by saving the most probable state sequence for the previously given letters. Both smoothing and unsmoothed HMM results were used for comparison. An example of a plaintext/key pair that the Viterbi algorithm predicted appears in Table 2.

The Viterbi algorithm is a dynamic programming algorithm that uses recursion to predict the most probable sequence of hidden states, plaintext/key pairs of letters in this case. Let n be the length of the ciphertext. Then $V_{i-n\ldots i}(k_{i-n}\ldots k_i)$ is the probability of the most likely partial key from positions i to n, represented by V. Let $m_i(k_i)$ be the ciphertext character at position i (plaintext letter at position i computed with the tabula recta with the key letter at position i as input). Each ciphertext character can then be represented by the equation:

$$V_{i-n\ldots i}(k_{i-n}\ldots k_i) = T_1T_2T_3$$

Table 2: A plaintext/key pair predicted with the Viterbi algorithm, with the correct plaintext for comparison.

Correct Plaintext	operatingbudgetforthedayschoolsinthefivecountiesofdallasharrisbexartarrantandelpaso wouldbewhichwouldbeasavingsofyearlyafterthefirstyearscapitaloutlayofwasabsorbedp arkhousetoldthesenate
Predicted Plaintext	attherentstrestiontherethepresthatermenothedthedtheparnofortherethecouramenthedthep erethedinatinattheresatementerandayearestativerecalationthessentthatinattheresea gersofthentionthenati
Predicted Key	ootwenerentangtforpaseaitlleoldonthoredeclepatounsmolsamittrisbeatfeatoredsuneakeis aldtafouinodaseldbeatotiteniealrontetparittinestosalscapemolatofaysesasammiasast isthourrcestscrstthen

$$T_1 = P(k_i|k_{i-n} \dots k_{i-1})$$
$$T_2 = P(m_i(k_i)|m_{i-n}(k_{i-n}) \dots m_{i-1}(k_{i-1}))$$
$$T_3 = max_{k_{i+1}}[V_{i-(n-1)\dots i+1}(k_{i-(n-1)} \dots k_{i+1})]$$

This product can be vulnerable to underflow, therefore the log probabilities and addition instead of multiplication is used in application.

4 Results

Table 3 shows the average decoding accuracy of all the methods tried against different training and testing texts. The baseline method, which tended to guess random letters for both the plaintext and the key, would have on average around a 4% accuracy regardless of the source texts and the ciphertext length. The three classifiers, Naive Bayes, Support Vector Machines and Linear Regression, tended to guess plaintexts and keys that were strings of frequent characters, e.g. 'eeee eee ee'. Occasionally, words composed of frequently occurring characters would be predicted, such as 'the' or 'or', but overall these methods remained mostly consistent with their prediction patterns. By analyzing the predicted strings, the SVM classifier tended to vary a bit more than the Naive Bayes and Linear Regression algorithms, and it would predict a wider range of frequent letters with slightly better results. Ultimately though, classifiers are not ideal for deciphering running keys and their plaintexts, since n-grams are not considered in their models and they only rely on the statistical attack of picking frequently occurring English letters, instead of candidate English words and phrases. The Viterbi variations predicted the most English-like plaintext and keys out of all the methods implemented, as shown in the example in Table 2. Since only lower order n-grams were used, the results could have been more stark, but they stand out regardless. Interestingly, the unsmoothed performed better than the +1 smoothing variation, perhaps since the more frequent n-grams got assigned a higher probability rather than redistributing their weight on n-grams that weren't seen at all (because of the statistical anomalies of the English language, this was probably a vast majority). Out of all the methods used, Viterbi is probably the most suitable for decoding running key ciphertext, as it generates worthy candidate phrases for the keys and message strings. One shortcoming it seems to have is with word boundaries, since spaces have been removed it is disadvantageous for determining proper English words. The Gibbs sampling method (Sravana Reddy, 2012) addresses this issue and takes into account word boundaries in the sample spaces for each iteration. On low ciphertext lengths this difference is more subtle, but on ciphertexts of length 1000 the performance gain is much larger.

There doesn't appear to be a significant difference between the two source texts at low ciphertext lengths, other than the Project Gutenberg sources slightly outperforming the Brown corpus documents.

As the minimum ciphertext length increased so did the accuracy of correct plaintext and keystream characters. This is unsurprising, since the level of message uncertainty is high for shorter plaintexts because the possible English combinations of plaintexts and keys is higher. The message uncertainty approaches some limit close to 0 the longer the length of the ciphertext.

Table 3: The decoding accuracies for extracted plaintext and keystreams with methods used, where S and U in the table denote smoothed and unsmoothed respectively.

Ciphertext Length Min	Source Texts	Baseline Accuracy	Naive Bayes Accuracy	SVM Accuracy	LR Accuracy	Viterbi (S) Accuracy	Viterbi (U) Accuracy
10	Gutenberg	3.7%	8.0%	9.4%	7.9%	9.6%	17.4%
	Brown	3.9%	9.9%	7.8%	7.8%	7.9%	14.9%
50	Gutenberg	3.8%	11.2%	12.4%	13.1%	21.4%	**29.4**%
	Brown	3.7%	10.5%	12.3%	13.9%	25.0%	23.9%
100	Gutenberg	3.9%	13.6%	15.7%	15.6%	26.3%	**36.7**%
	Brown	3.9%	12.6%	17.6%	11.8%	23.7%	23.4%

5 Discussion

The results, though inaccurate and difficult to draw reliable conclusions from, were promising and trending upwards. Provided the hardware/optimization techniques for larger n-gram counts, ciphertext lengths and corpus filesizes, the results from Reddy (Sravana Reddy, 2012) and Griffing's (Griffing, 2006) research could've been reproduced.

6 Future Work

Though the results shown were promising, further comparisons could be done through implementing Gibbs sampling. Initially we were going to implement Gibbs sampling and extend it by introducing iterations that filtered for valid English words, but Reddy's Decoding Running Key Ciphers (Sravana Reddy, 2012) was lacking in sufficient details and emails sent to the authors for clarity went unanswered.

Another simple approach using supervised learning can be proposed, where the relations between the plaintext and keystream are learned based on the ciphertexts. Since the sample space for ciphertext generation is small, this could be in scope. Because Viterbi is proven to be hugely computationally intensive, it was difficult to properly parallelize and produce results with higher orders of n for the n-grams, which would have been incredibly interesting to see. Future effort with this venture will be directed towards optimization.

7 Conclusion

We compared the performance of a sampling of different methods that decode running key ciphers against a baseline random guesser, classifers, and with different source texts for training and testing sections.

8 Statement of Contribution

Victor contributed the code for the data preprocessing, n-gram generation, cache, random guesser, classifier tests, HMM and Viterbi algorithms.

Marie contributed the initial proposal, final report, the results from different corpora, and the code for the argument parsing, data preprocessing, and adjustments for large datasets.

References

Christian N.S. Tate Craig Bauer. 2002. A statistical attack on the running key cipher. *Cryptologia*, 26(4):274.

Gerald Penn Eric Corlett. 2010. An exact a* method for deciphering letter-substitution ciphers. *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 48(1):1040.

Alexander Griffing. 2006. Solving the running key cipher with the viterbi algorithm. *Cryptologia*, 30(4):361.

Kevin Knight Sravana Reddy. 2012. Decoding running key ciphers. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 50(1):80.