Cipher Breaking Project

Maria Shugrina
MIT 6.437 Spring 2013

Intro

We are given an observed ciphertext $\mathbf{y} = \tilde{f}(\tilde{\mathbf{x}})$, where $\tilde{\mathbf{x}}$ is the unknown original text and $\tilde{f} : \mathbb{X} \to \mathbb{X}$ is an unknown cipher function from the set of all substitution cipher functions $\mathbb{F}$ that had been used to scramble $\tilde{\mathbf{x}}$. Here the alphabet $\mathbb{X} = \{a, b...z, [\ ], [.]\}$. The goal is to recover an estimate $\hat{\mathbf{x}}$ of the original text.

Vanilla Algorithm

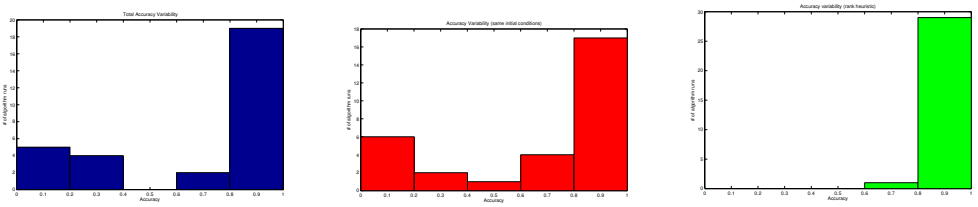
Here is the Monte Carlo algorithm based on Metropolis-Hastings sampling algorithm that had been implemented for Part I of the project:

1. $f \leftarrow rand\_choose1(\mathbb{F})$
2. $i, j \leftarrow rand\_choose2(\mathbb{X})$; $f' \leftarrow swap\_substitutions(f, i, j)$
3. $if\ p_{\mathbf{y}|f}(\mathbf{y}|f') > \hat{p}:\ \ \hat{p} = p_{\mathbf{y}|f}(\mathbf{y}|f')$; $\hat{\mathbf{x}} = f'^{-1}(\mathbf{y})$
4. $if\ Bernoulli(p = min(1, \frac{p_{\mathbf{y}|f}(\mathbf{y}|f')}{p_{\mathbf{y}|f}(\mathbf{y}|f)})) = 1:\ \ f = f'$
5. *Repeat 2-4 until max_iterations reached; output* $\hat{\mathbf{x}}$

Intuition and Enhancements

0.1 Controlling for Randomness

Although the vanilla algorithm above tends to converge to a solution with accuracy over 0.90, it was experimentally observed that some runs produce inferior results (when the total number of iterations is capped). In fact, for a particular example and 30 algorithm runs, the error variability is very high (Fig. 1 left). Setting initial f to the same random permutation for every run has little effect on the observed error variability (Fig. 1 center).



(a) Totally random (b) Same initial conditions (c) Divergence-based initial guess

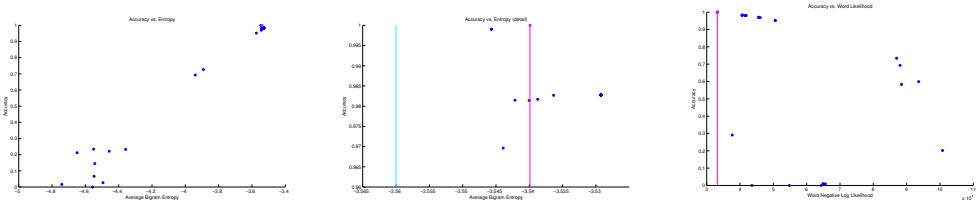
Figure 1: Variability in accuracy due to randomness. a) the initial permutation is chosen randomly for every run, b) initial permutation is the same for every run, c) initial permutation is chosen using letter frequency probabilities. (Number of iterations was capped at 5K for a) and b) and at 2K for c) due to faster convergence).

Metropolis-Hastings, though guaranteed to produce representative samples as number of iterations approaches infinity, takes time to converge. We conjecture that the random exploration of the space of $\mathbb{F}$ is sensitive to randomization and can result in inferior results when the number of iterations is capped at a low number. As the inititialization of f has no theoretical significance, we pick:

$$f \leftarrow div\_guess(\mathbb{F}, \mathbf{y}) \approx \underset{f_i \in \mathbb{F}}{argmin}\, D(\tilde{p}_c(.;\, f_i(\mathbf{y}))||p_c(.))$$

where $\tilde{p}_c(.; f(\mathbf{y}))$ is the empirical distribution for the characters in $\mathbf{y}$ decoded by f, and $p_c(.)$ is the empirical letter probability distribution trained on a large corpus and provided for Part I. In practice, we simply pick f_i that assigns to each character in $\mathbf{y}$ a character that has same probability rank in p_c. Though this may not guarantee minimizing divergence, we omit theoretical analysis here. This yields dramatically lower error variance and faster convergence rate (Fig. 1 right).

0.2 Modeling Text



(a) Entropy vs. accuracy (b) Entropy for high accuracy decodings (c) Negative word log likelihood

Figure 2: Entropy and word likelihood as predictors of accuracy for 30 random runs on a single example (simply for intuition). a) Entropy vs. accuracy with magenta marking ground truth and cyan marking binary entropy of English as reported by Shannon. b) zoom of a for higher accuracy results. c) negative word log likelihood of the decoded text as predictor of accuracy with magenta representing ground truth decoding.

The vanilla algorithm uses bigram character probabilities to model $p_x(\mathbf{x})$. If $\mathbf{x}$ is comprised of the character sequence $c_1, c_2 ... c_N$, then $p_x(\mathbf{x}) = p_c(c_1) \prod_{i=2}^{N} p_{c|c}(c_i|c_{i-1})$. A normalized version of this likelihood is the bigram entropy of $\mathbf{x}$, as formulated by [Shannon 1951]:

$$F_2 = -\frac{1}{N-1} \Sigma_{i=1}^{N} log_2(p_{c|c}(c_i|c_{i-1})) \approx 3.56$$

One natural way to evaluate the "goodness" of our decoding and to terminate the algorithm is to match the entropy of $\hat{x}_n$ against the known binary entropy of English (3.56). To get some intuition for the predictive power of F_2 for the accuracy of $\hat{x}$, we run the algorithm 30 times for an particular example (Fig. 2 a, b). Of course, no conclusions are possible from one example alone, but the plots confirm our natural intuitions:

- F_2 in general correlates with accuracy
- For $\hat{x}$'s with high accuracy, F_2 is a poor predictor of fine tuned accuracy

The second conclusion implies that maximizing p_x will not yield results of highest accuracy, suggesting that a richer model of text is necessary to improve performance. We propose a mixture model:

$$p'_{\mathbf{y}|f}(\mathbf{y}|f) = p'_{\mathbf{x}}(f^{-1}(\mathbf{y})) = q_m(C)p_{x|m}(\mathbf{x}|m = C) + q_m(W)p_{x|m}(\mathbf{x}|m = W)$$

where C is the character bigram model, and so $p_{x|m}(\mathbf{x}|m = C) = p_x(x)$ from the vanilla implementation, and W is a word-based language model. Making a gross and unjustifiable assumption of independence, we model the probability of a word sequence $\mathbf{x} = w_1 w_2 ... w_m$ as:

$$p_{x|m}(\mathbf{x}|m = W) = \prod_{i=1}^{m} p_w(w_i)$$

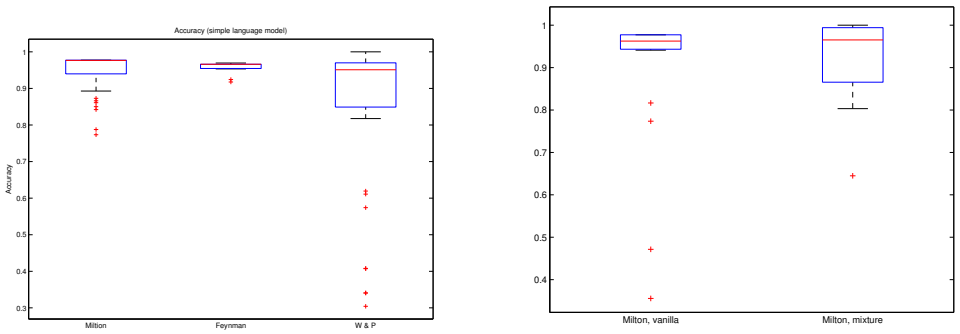
where word probabilities are taken from the unigram counts made available by norvig.com. Anecdotal evidence supports the intuition that even this word likelihood is a better predictor of accuracy for decodings that already have high accuracy (Fig. 2 c). We pick maximally ignorant uniform prior for $q_m(.)$ for simplicity.

In practice, $p_{x|m}(\mathbf{x}|m = W)$ is smoothed to accomodate probabilities of unknown words with probability $\approx 1/n\_words$. This means that for large word sequences, this probability estimate is very sensitive to the number of words in sequence. Thus we switch to the mixture approach only when F_2 approaches 3.56 and disallow f' that affect the word count of the decoded text, justifying this by the observation that even with the original text modeling our approach has high accuracy and is therefore very unlikely to not recover the space substitution.

Enhanced Algorithm

1. $f \leftarrow div\_guess(\mathbb{F}, \mathbf{y})$
2. $\bar{p}_{\mathbf{y}|f}(.|.) \leftarrow p_{\mathbf{y}|f}(.|.)$
3. $i, j \leftarrow rand\_choose2(\mathbb{X})$; $f' \leftarrow swap\_substitutions(f, i, j)$
4. $if\ \bar{p}_{\mathbf{y}|f}(\mathbf{y}|f') > \hat{p}:\ \ \hat{p} = \bar{p}_{\mathbf{y}|f}(\mathbf{y}|f')$; $\hat{\mathbf{x}} = f'^{-1}(\mathbf{y})$
5. $if\ Bernoulli(p = min(1, \frac{\bar{p}_{\mathbf{y}|f}(\mathbf{y}|f')}{\bar{p}_{\mathbf{y}|f}(\mathbf{y}|f)})) = 1:\ \ f = f'$
6. $if\ F_2(f^{-1}(\mathbf{y}))\ near\ 3.56:\ \ \bar{p}_{\mathbf{y}|f}(.|.) \leftarrow p'_{\mathbf{y}|f}(.|.)$; *update* $\hat{p}$
7. *Repeat 3-6 until max_iterations reached; output* $\hat{\mathbf{x}}$

Performance



(a) Vanilla p_x (b) Mixture p_x

Figure 3: a) Performance of our algorithm on 3 supplied texts averaged across 50 runs with random cipher function in each run when no word likelihood is used. b) Performance of our algorithm on one particular text on 20 runs of the algorithm with and without using the mixture model.

We ran evaluation on the 3 supplied test texts and acknowledge that more rigorous testing is desirable. For each input text we randomly generated 50 cipher functions and ran decoding on each, producing bar graphs of average error (Fig. 3 a).

Anecdotally, improvement in the rate of unknown words and accuracy was observed when the mixture model was used. Unfortunately, computing word likelihood is not fully optimized (takes about 4 minutes per run), so we have only been able to generate one comparison plot (Fig. 3 b). The error variability actually increased, but fewer outliers with lower accuracy were observed in this example. (Note: I will submit a full box plot comparison after the deadline as a separate file.)

Efficient implementation was achieved by not performing redundant computation and relying on matrix operations as much as possible.