
EM for Cryptograms

Max Welling
UC Irvine

Abstract

We formulate an EM algorithm for solving cryptograms.

1 The Cryptogram Problem

The problem is to find a translation of a sentence with letters from a permutation of the alphabet into English. In fact, we also allow more general mappings of the alphabet to itself.

2 The Cryptogram Model

There is a prior $P(\mathbf{w})$ over English words, given by the empirical frequencies of English words.

The likelihood term is the probability of a code-word given its English translation. This probability will be decomposed as a product of probabilities over the letters of the word:

$$P(\mathbf{c}|\mathbf{w}) = \prod_{\ell} P(c_{\ell}|w_{\ell}) \quad (1)$$

Initially, these conditional probabilities are uniform, but we will learn them from the data.

In this framework, the prior is fixed, the 26×26 dimensional conditional probability table $P(c_{\ell} = i|w_{\ell} = j) = T_{ij}$ is considered a parameter to be updated, the code-words are observed and the words are hidden variables.

Data arrives as a sentence of code-words, $\{\mathbf{c}_1, \dots, \mathbf{c}_N\}$, where each word can have a different length. The log-likelihood of this data is given by,

$$\sum_n \log \left[\sum_{\mathbf{w}_n} \left(\prod_{\ell_n} P(c_{\ell_n}|w_{\ell_n}) \right) P(\mathbf{w}_n) \right] \quad (2)$$

where ℓ_n indexes the letters/symbols in the n 'th word or encrypted word.

This log-likelihood can be iteratively optimized by optimizing the following auxiliary objective,

$$Q(P^t|P^{t-1}) = \sum_n \sum_{\mathbf{w}_n} P^{t-1}(\mathbf{w}_n|\mathbf{c}_n) \left[\sum_{\ell_n} \log P^t(c_{\ell_n}|w_{\ell_n}) + \log P^t(\mathbf{w}_n) \right] \quad (3)$$

where

$$P(\mathbf{w}|\mathbf{c}) = \frac{\prod_{\ell} P(c_{\ell}|w_{\ell})P(\mathbf{w})}{\sum_{\mathbf{w}} \prod_{\ell} P(c_{\ell}|w_{\ell})P(\mathbf{w})} \quad (4)$$

The above quantity is the posterior probability of the decoded word given the encrypted word. In other words, it is the object that helps you decrypt the sentence. Note that there may be words that are not in your dictionary. Therefore certain codewords may give very low probability to all words in your dictionary. You may need to introduce joker-words that can basically be anything that can absorb this possibility.

This calculation is in fact the E-step of the EM algorithm.

The M-step is given by optimizing the objective over the matrix $P(c_\ell = i | w_\ell = j) = T_{ij}$. Since it is a probability, we need to enforce the following normalization:

$$\sum_i T_{ij} = 1 \forall j \quad (5)$$

This can be enforced by adding a Lagrange multiplier term:

$$- \sum_j \theta_j \left(\sum_i T_{ij} - 1 \right) \quad (6)$$

This leads to the following “solution”:

$$T_{ij} \propto \sum_n \sum_{\mathbf{w}_n} P^{t-1}(\mathbf{w}_n | \mathbf{c}_n) \sum_{\ell_n} I[c_{\ell_n} = i, w_{\ell_n} = j] \quad (7)$$

How do you update T_{ij} ? You run over all words in the encrypted text string. For each word in that string you loop over all words in the dictionary (smart tricks may be necessary to speed this step up, e.g. ignoring really low scoring words). For each letter in the you check whether the code-letter equals symbol i and the letter of the possible translation equals j . If so, you score one unit of $P^{t-1}(\mathbf{w}_n | \mathbf{c}_n)$. If the match is made multiple times within a word, you will score multiple times, say $n \times P^{t-1}(\mathbf{w}_n | \mathbf{c}_n)$ with the number of times a match was found in that word. You then add all these scores across all dictionary words to get your total score. Finally, you normalize

$$T_{ij} \leftarrow T_{ij} / \sum_i T_{ij} \quad (8)$$

You then go back to compute

$$P(\mathbf{w} | \mathbf{c}) = \frac{\prod_\ell P(c_\ell | w_\ell) P(\mathbf{w})}{\sum_{\mathbf{w}} \prod_\ell P(c_\ell | w_\ell) P(\mathbf{w})} \quad (9)$$

using you new estimates for $P(c_\ell = i | w_\ell = j) = T_{ij}$ and repeat.

This algorithm is guaranteed to converge because it is an instance of an EM algorithm.

I am not aware that this algorithm exists in the literature (but it might), so search the literature first. If not, and you decide to implement this you may compare against other algorithms.