Unsupervised Analysis for Decipherment Problems

Kevin Knight, Anish Nair, Nishit Rathod
Information Sciences Institute
and Computer Science Department
University of Southern California
knight@isi.edu, {anair,nrathod}@usc.edu

Kenji Yamada
Language Weaver, Inc.
4640 Admiralty Way, Suite 1210
Marina del Rey, CA 90292
kyamada@languageweaver.com

Abstract

We study a number of natural language decipherment problems using unsupervised learning. These include letter substitution ciphers, character code conversion, phonetic decipherment, and word-based ciphers with relevance to machine translation. Straightforward unsupervised learning techniques most often fail on the first try, so we describe techniques for understanding errors and significantly increasing performance.

1 Introduction

Unsupervised learning holds great promise for breakthroughs in natural language processing. In cases like (Yarowsky, 1995), unsupervised methods offer accuracy results than rival supervised methods (Yarowsky, 1994) while requiring only a fraction of the data preparation effort. Such methods have also been a key driver of progress in statistical machine translation, which depends heavily on unsupervised word alignments (Brown et al., 1993).

There are also interesting problems for which supervised learning is not an option. These include deciphering unknown writing systems, such as the Easter Island *rongorongo* script and the 20,000-word Voynich manuscript. Deciphering animal language is another case. Machine translation of human languages is another, when we consider language pairs where little or no parallel text is available. Ultimately, unsupervised learning also holds promise for scientific discovery in linguistics. At some point, our programs will begin finding novel, publishable regularities in vast amounts of linguistic data.

2 Decipherment

In this paper, we look at a particular type of unsupervised analysis problem in which we face a ciphertext stream and try to uncover the plaintext that lies behind it. We will investigate several applications that can be profitably analyzed this way. We will also apply the same technical solution these different problems.

The method follows the well-known noisy-channel framework. At the top level, we want to find the plaintext that maximizes the probability P(plaintext | ciphertext). We first build a probabilistic model P(p) of the plaintext source. We then build probabilistic channel model P(c | p) that explains how plaintext sequences (like p) become ciphertext sequences (like c). Some of the parameters in these models can be estimated with supervised training, but most cannot.

When we face a new ciphertext sequence c, we first use expectation-maximization (EM) (Dempster, Laird, and Rubin, 1977) to set all free parameters to maximize P(c), which is the same (by Bayes Rule) as maximizing the sum over all p of P(p) · P(c | p). We then use the Viterbi algorithm to choose the p maximizing P(p) · P(c | p), which is the same (by Bayes Rule) as our original goal of maximizing P(p | c), or plaintext given ciphertext.

Figures 1 and 2 show standard EM algorithms (Knight, 1999) for the case in which we have a bigram P(p) model (driven by a two-dimensional b table of bigram probabilities) and a one-for-one P(c | p) model (driven by a two-dimensional s table of substitution probabilities). This case covers Section 3, while more complex models are employed in later sections.

3 English Letter Substitution

An *informal substitution cipher* (Smith, 1943) disguises a text by substituting code letters for normal letters. This system is usually *exclusive*, meaning that each plaintext letter maps to only one ciphertext letter, and vice versa. There is surprisingly little published on this problem, e.g., (Peleg and Rosenfeld, 1979), because fast computers led to public-key cryptography before much computer analysis was done on such old-style ciphers. We study this problem first because it resembles many of the other problems we are interested in, and we can generate arbitrary amounts of test data.

We estimate unsmoothed parameter values for an English letter-bigram P(p) from news data. This is a 27x27 table that includes the space character. We then set up a uniform P(c | p), which also happens to be a

Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions, pages 499–506,
Sydney, July 2006.

```
(a) ingcmpnqsnwf cv fpn owoktvcv hu ihgzsnwfv rqcffnw cw owgcnwf kowazoanv...
(b) wecitherkent is the analysis of wocoments pritten in ancient buncquges...
(c) decipherment is the analysis of documents written in ancient languages...
```

Figure 3: Letter substitution decipherment. (a) is the ciphertext, (b) is an automatic decipherment, and (c) is an improved decipherment.

Given a ciphertext c of length m, a plaintext vocabulary of v tokens, and a plaintext bigram model b:

1. set a s($c|p$) substitution table initially to be uniform
2. for several iterations do:
   a. set up a count table count(c, p) with zero entries
   b. P(c) = 0
   c. for all possible plaintexts $p_1 \cdots p_m$
      (each p_i drawn from plaintext vocabulary)
      compute P(p) = b(p_1 | boundary) · b(boundary | p_m) $\cdot \prod_{i=2}^{m}$ b($p_i|p_{i-1}$)
      compute P(c|p) = $\prod_{j=1}^{m}$ s($c_j|p_j$)
      P(c) += P(p) · P(c|p)
   d. for all plaintexts p of length m
      compute P(p|c) = $\frac{\text{P(p)} \cdot \text{P(c|p)}}{\text{P(c)}}$
      for j = 1 to m
         count(c_j, p_j) += P(p|c)
   e. normalize count(c, p) table to create a revised s($c|p$)

Figure 1: A naive application of the EM algorithm to break a substitution cipher. It runs in O(mv^m) time.

Given a ciphertext c of length m, a plaintext vocabulary of v tokens, and a plaintext bigram model b:

1. set the s($c|p$) substitution table initially to be uniform
2. for several iterations do:
   a. set up a count(c, p) table with zero entries
   b. for i = 1 to v
      Q[i,1] = b(p_i | boundary)
   c. for j = 2 to m
      for i = 1 to v
         Q[i,j] = 0
         for k = 1 to v
            Q[i,j] += Q[k,$j-1$] · b($p_i|p_k$) · s($c_{j-1}|p_k$)
   d. for i = 1 to v
      R[i,m] = b(boundary | p_i)
   e. for j = $m-1$ to 1
      for i = 1 to v
         R[i,j] = 0
         for k = 1 to v
            R[i,j] += R[k,$j+1$] · b($p_k|p_i$) · s($c_{j+1}|p_k$)
   f. for j = 1 to m
      for i = 1 to v
         count(c_j, p_i) += Q[i,j] · R[i,j] · P($c_j|p_i$)
   g. normalize count(c, p) table to create a revised s($c|p$)

Figure 2: An efficient O(mv^2) algorithm that accomplishes the same thing as Figure 1.

27x27 table. We set P(space | SPACE) = 1.0, and all other values to 1/26. We create our ciphertext by encrypting an out-of-domain encyclopedia article. This article contains 417 letters, some of which are shown in Figure 3(a).

The decipherment yielded by EM/Viterbi contains 68 errors—see Figure 3(b).

Can we do better? First, we are not taking advantage of the fact that the cipher system is exclusive. But, as we observe in the rest of this paper, most natural decipherment problems do not have this feature, so we do not take advantage of it in this case (and it is hard to model!).

We can certainly acquire vastly more data for estimating P(p). Using a 1.5-million character data set instead of a 70,000-character data set reduces the number of errors from 68 to 64. Next, we apply fixed-lambda interpolation smoothing to P(p). This reduces errors further to 62.

Next, we adjust our Viterbi search to maximize P(p) · P(c | p)^3 rather than P(p) · P(c | p). This cubing concept was introduced in another context by (Knight and Yamada, 1999). It serves to stretch out the P(c | p) probabilities, which tend to be too bunched up. This bunching is caused by incompatibilities between the n-gram frequencies used to train P(p) and the n-gram frequencies found in the correct decipherment of c. We find this technique extremely useful across decipherment applications. Here it reduces errors from 62 down to 42.

We also gain by using letter trigrams instead of bigrams. This reduces error from the original 68 to 57 (small source data) or 32 (large source data). Combining trigrams with cubing the channel probabilities reduces error to 15, which source-model smoothing further reduces to 10 (or 2.4%), as in Figure 3(c).

So far we have glossed over the number of EM iterations used. From the EM's point of view, the more iterations, the better, as these improve P(c). However, the decipherment error rate may jump around as iterations proceed. Figure 4 shows the effect of EM iterations on error rate. With the worse source models, it is better to stop the EM early. EM initially locks onto the correct theory, but task performance degrades as it tries to make the ciphertext decoding fit the expected bigram frequencies. Better source models do not suffer much.

If we give the system more knowledge about English vocabulary and grammar, it will further improve. We have also been able to get perfect performance by using the best-so-far decipherment in Figure 3 to pull down related English texts from the web, and using these to retrain P(p) to fuel a second decipherment. However, we only present the simple substitution cipher as a prototype of the kinds of applications we are really interested in, which we present in the following sections.

The experiments we have presented so far should not be viewed as tuning parameters for performance—

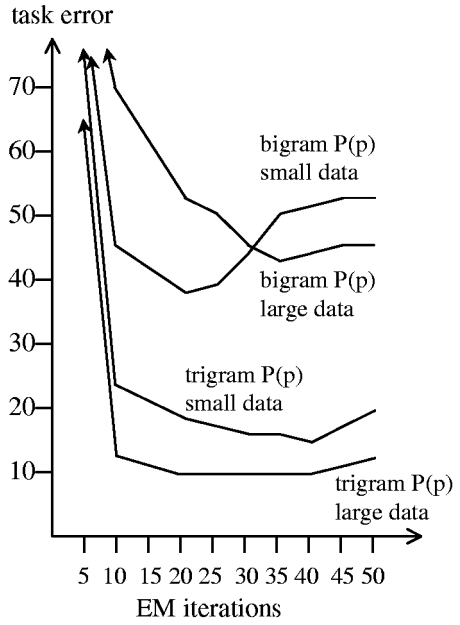

Figure 4: Decipherment error on letter substitution.

indeed, it is not correct to measure accuracy on a tuning/development data set. Rather, we have demonstrated some general strategies and observations (more data, larger n-grams, stability of good language models) that we can apply to other real decipherment situations. In many such situations, there is only a test set, and tuning is impossible even in principle—fortunately, we observe that the general strategies work robustly across a number of decipherment domains.

4 Character Code Conversion

Many human languages are straightforwardly represented at the character level by some widely-adopted standard (e.g., ASCII). In dealing with other languages (like Arabic), we must be equally prepared to process a few different standards. Documents in yet other languages (like Hindi) are found spread across the web in dozens if not hundreds of specialized encodings. These come with downloadable fonts for viewing. However, they are difficult to handle by computer, for example, to build a full-coverage Hindi web-search engine, or to pool Hindi corpora for training machine translation or speech recognition.

Character conversion tools exist for many pairs of major encoding systems, but it has been the experience of many researchers that these tools are flawed, despite the amount of work that goes into them. 100% accuracy is not to be found. Furthermore, nothing exists for most pairs. We believe that mild annotation techniques allow people to generate conversion tables quite quickly (and we show some results on this), but we follow here an unsupervised approach, as would be required to automatically generate a consistently-encoded Hindi web.

Our ciphertext c is a stream of bytes in an unknown encoding, with space separators; we use integers to represent these bytes, as in Figure 5(a). Our plaintext is a large collection of UTF8 standard Hindi. UTF8 builds complex Hindi character "chunks" out of up to 3 simple and combining characters. A Hindi word is a sequence of chunks, and words are separated by spaces.

We know that c is Hindi—we imagine that it was once UTF8, but that it somehow got enciphered.

Modeling is more complex than in the previous section. First, we have to decide what our plaintext tokens will be. Our first approach was to use chunks. Chunk boundaries are essentially those where we could draw a vertical line in written Hindi without disturbing any characters. We could then set up a model of how UTF8 is "encoded" to the mystery sequence in the putative channel—namely, we let each source chunk map to a particular target byte sequence. (By analogy, we would divide up English text into mostly letters, but would chunk ligatures like "fi" together. In fact, in extracting English text from pdf, we often find "fi" encoded by a single byte). This model is quite general and holds up across the encodings we have dealt with. However, there are over 900 chunks to contend with, and vast numbers of target byte sequences, so that the P(c | p) table is nearly unmanageable.

Therefore, we use a simpler model. We divide p into individual characters, and we set up a channel in which plaintext characters can map into either one or two ciphertext bytes. Instead of a table like P(c c | p), we set up two tables: P(f | p) for character fertility, and P(c | p) for character-to-byte substitution. This is similar to Model 3 of (Brown et al., 1993), but without null-generated elements or re-ordering.

Our actual ciphertext is an out-of-domain web page with 11,917 words of song lyrics in Hindi, in an idiosyncratic encoding. There is no known tool to convert from this encoding. In order to report error rates, we had to manually annotate a portion of this web page with correct UTF8. This was quite difficult. We were completely unable to do this manually by relying only on the ciphertext byte sequence—even though this is what we are asking our machine to do! But as Hindi readers, we also have access to the web-site rendering in Hindi glyphs, which helps us identify which byte sequences correspond to which Hindi glyphs, and then to UTF8. The labeled portion of our ciphertext consists of 59 running words (281 ciphertext bytes and 201 UTF8 characters).

Because the machine decipherment rarely consists of exactly 201 UTF8 characters, we report edit distance instead of error rate. An edit distance of 0 is perfect, while the edit distance for long incorrect decipherments may be greater than 201. With a source character bigram model, and the above channel, we obtain an edit distance of 161. With a trigram model, we get 127.

Now we introduce another idea that has worked across several decipherment problems. We use a fixed, uniform fertility model and allow EM only to manip-

```
(a) ... 13 5 14 . 16 2 25 26 2 25 . 17 2 13 . 15 2 8 . 7 2 4 2 9 2 2 ...
(b) ... 6 35    . 12 28 49 10 28  . 3 4 6   . 1 10 3 . 29 4 8 20 4   ...
(c) ... 6 35 24 . 12 28 21 4      . 11 6    . 12 25  . 29 8 22 4     ...
(d) ... 6/35/24 . 12/28 21/28     . 3/4 6   . 1/25   . 29 8 20/4     ... *
```

Figure 5: Hindi character code decipherment. (a) is the Hindi ciphertext byte sequence, (b) is an EM decipherment using a UTF8 trigram source model, (c) is a decipherment using a UTF8 word frequency model, and (d) is correct UTF8 (chunks joined with slash). Periods denote spaces between words; * denotes the correct answer.

```
P(13 | 6) = 0.66 *    P( 8|24) = 0.48
P(32 | 6) = 0.19      P(14|24) = 0.33 *
P( 2 | 6) = 0.13      P(17|24) = 0.14
P(16 | 6) = 0.02      P(25|24) = 0.04

P( 5 | 35) = 0.61 *   P(16|12) = 0.58 *
P(14 | 35) = 0.25     P( 2|12) = 0.32 *
P( 2 | 35) = 0.15     P(31|12) = 0.03
```

Figure 6: A portion of the learned P(c | p) substitution probabilities for Hindi decipherment. Correct mappings are marked with *.

ulate substitution probabilities. This prevents the algorithm from locking onto bad solutions. This gives an improved solution edit distance of 93, as in Figure 5(b), which can be compared to the correct decipherment in 5(d). Figure 6 shows a portion of the learned P(c | p) substitution table, with * indicating correct mappings.

15 out of 59 test words are deciphered exactly correctly. Another 16 out of 59 are perfect except for the addition of one extra UTF8 character (always "4" or "25"). Ours are the first results we know of with unsupervised techniques.

We also experimented with using a word-based source model in place of the character n-gram model. We built a word-unigram P(p) model out of only the top 5000 UTF8 words in our source corpus—it assigns probability zero to any word not in this list. This is a harsh model, considering that 16 out of 59 words in our UTF8-annotated test corpus do not even occur in the list, and are thus unreachable. On the plus side, EM considers only decipherments consisting of sequences of real Hindi words, and the Viterbi decoder only generates genuine Hindi words. The resulting decipherment edit distance is encouraging at 92, with the result shown in Figure 5(c). This model correctly deciphers 25 out of 59 words, with only some overlap to the previous 15 correct out of 59—one or other of the models is able to perfectly decipher 31 out of 59 words already, making a combination promising.

Our machine is also able to learn in a semi-supervised manner by aligning a cipher corpus with a manually-done translation into UTF8. EM searches for the parameter settings that maximize P(c | p), and a Viterbi alignment is a by-product. For the intuition, see Figure 5(a and d), in which plaintext character "6" occurs twice and may be guessed to correspond with ciphertext byte "13". EM does this perfectly, except for some regions where re-ordering indeed happens. We are able to move back to our chunk-based model in semi-supervised mode, which avoids the re-ordering problem, and we obtain near-perfect decipherment tables when we asked a human to re-type a few hundred words of mystery-encoded text in a UTF8 editor.

5 Phonetic Decipherment

This section expands previous work on phonetic decipherment (Knight and Yamada, 1999). Archaeologists are often faced with an unknown writing system that is believed to represent a known spoken language. That is, the written characters encode phonetic sequences (sometimes individual phonemes, and sometimes whole words), and the relationship between text and sound is to be discovered, followed by the meaning. Viewing text as a code for speech was radical some years ago. It is now the standard view of writing systems, and many even view written Chinese as a straightforward syllabary, albeit one that is much larger and complex than, say, Japanese kana. Both Linear B and Mayan writing were deciphered by viewing the observed text as a code/cipher for an approximately-known spoken language (Chadwick, 1958; Coe, 1993).

We follow (Knight and Yamada, 1999) in using Spanish as an example. The ciphertext is a 6980-character passage from Don Quixote, as in Figure 7(a). The plaintext is a very large out-of-domain Spanish phoneme sequence from which we compute only phoneme n-gram probabilities. We try deciphering without detailed knowledge of spoken Spanish words and grammar. The goal is for the decipherment to be understandable by modern Spanish speakers.

First, it is necessary to settle on the basic inventory of sounds and characters. Characters are easy; we simply tabulate the distinct ones observed in ciphertext. For sounds, we use a Spanish-relevant subset of the International Phonetic Alphabet (IPA), which seeks to capture all sounds in all languages; the implementation is SAMPA (Speech Assessment Methods Phonetic Alphabet). Here we show the sound and character inventories:

Sounds:

```
B, D, G, J (ny as in canyon), L (y as
in yarn), T (th as in thin), a, b, d,
e, f, g, i, k, l, m, n, o, p, r,
rr (trilled), s, t, tS (ch as in chin),
u, x (h as in hat)
```

(a) primera parte del ingenioso hidalgo don quijote de la mancha

```
(b) primera parte des intenioso liDasto don fuiLote de la manTia
(c) primera parte del inGenioso biDalGo don fuiLote de la manTia
(d) primera parte del inxenioso iDalGo  don kixote  de la manSa     *
```

Figure 7: Phonetic decipherment. (a) is written Spanish ciphertext, (b) is an initial decipherment, (c) is an improved decipherment, and (d) is the correct phonetic transcription.

Characters: ñ, á, é, í, ó, ú, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

The correct decipherment (Figure 7(d)) is a sequence of 6759 phonemes (here in SAMPA IPA).

We use a P(c | p) model that substitutes a single letter for each phoneme throughout the sequence. This considerably violates the rules of written Spanish (e.g., the K sound is often written with two letters q u, and the two K S sounds are often written x), so we do not expect a perfect decipherment. We do not enforce exclusivity; for example, the S sound may be written as c or s.

An unsmoothed phonetic bigram model gives an edit distance (error) of 805, as in Figure 7(b). Here we study smoothing techniques. A fixed-lambda interpolation smoothing yields 684 errors, while giving each phoneme its own trainable lambda yields a further reduction to 621. The corresponding edit distances for a trigram source model are 595, 703, and 492, the latter shown in Figure 7(c), an error of 7%. (This result is equivalent to Knight & Yamada [1999]'s 4% error, which did not count extra incorrect phonemes produced by decipherment, such as pronunciations of silent letters). Quality smoothing yields the best results. While even the best decipherment is flawed, it is perfectly understandable when synthesized, and it is very good with respect to the structure of the channel model.

6 Universal Phonetic Decipherment

What if the language behind the script is unknown? The next two sections address this question in two different ways.

One idea is to look for universal constraints on phoneme sequences. If we somehow know that P(K AE N UW L IY) is high, while P(R T M K T K) is low, that we may be able to exploit such knowledge in deciphering an alphabetic writing system. In fact, many universal constraints have been proposed by linguists. Two major camps include syllable theorists (who say that words are composed of syllables, and syllables have internal regular structure (Blevins, 1995)) and anti-syllable theorists (who say that words are composed of phonemes that often constrain each other even across putative syllable boundaries (Steriade, 1998)).

We use the same Don Quixote ciphertext as in the previous section. While the ultimate goal is to label each letter with a phoneme, we first attack a more tractable problem, that of labeling each letter as C (consonant) or V (vowel). Once we know which letters stand for consonant sounds, we can break them down further.

Our first approach is knowledge-free. We put together a fully-connected, uniform trigram source model P(p) over the tokens C, V, and SPACE. Our channel model P(c | p) is also fully-connected and uniform. We allow source as well as channel probabilities to float during training. This almost works, as shown in Figure 8(b). It correctly clusters letters into vowels and consonants, but assigns exactly the wrong labels! A complex cluster analysis (Finch and Chater, 1991) yields similar results.

Our second approach uses syllable theory. Our source model generates each source word in three phases. First, we probabilistically select the number of syllables to generate. Second, we probabilistically fill each slot with a syllable *type*. Every human language has a clear inventory of allowed syllable types, and many languages share the same inventory. Some examplars are (1995):

	V	CV	CVC	VC	CCV	CCVC	CVCC	VCC	CCVCC
Hua		✓							
Cayuvava	✓	✓							
Cairene		✓	✓						
Mazateco	✓	✓			✓				
Mokilese	✓	✓	✓	✓					
Sedang		✓	✓		✓	✓			
Klamath		✓	✓				✓		
Spanish	✓	✓	✓	✓	✓	✓			
Finnish	✓	✓	✓	✓			✓	✓	
Totonac		✓	✓		✓	✓	✓		✓
English	✓	✓	✓	✓	✓	✓	✓	✓	✓

For our purposes, we allow generation of V, VC, VCC, CV, CVC, CCV, CVCC, CCVC, or CCVCC. Elements of the syllable type sequence are chosen independently of each other, except that we disallow vowel-initial syllables following consonant-final syllables, following the phonetic universal tendency to "maximize the onset" (the initial consonant cluster of a syllable). Third, we spell out the chosen syllable types, so that the whole source model yields sequences over the tokens C, V, and SPACE, as before. This spelling-out is deterministic, except that we may turn a V into either one or two Vs, to account for dipthongs. The channel model again maps {C, V} onto {a, b, c, ...}, and we again run EM to learn both source and channel probabilities.

Figure 8(c) shows that this almost works. To make it work, 8(d), we force the number of syllables per word in the model to be fixed and uniform, rather than learned. This prevents the system from making analyses that are too short. We also execute several EM runs with randomly initialized P(c | p), and choose the run with the highest resulting P(c).

(a)	primera	parte	del	ingenioso	hidalgo	don	quijote	de	la	mancha
(b)	VVCVCVC	VCVVC	VCV	CVVCVVCVC	VCVCVVC	VCV	VCVVCVC	VC	VC	VCVVVC
(c)	CCV.CV.CV	CVC.CV	CVC	VC.CVC.CV.CV	CV.CVC.CV	CVC	CVC.CV.CV	CV	CV	CVC.CCV
(d)	CCV.CV.CV	CVC.CV	CVC	VC.CV.CV.V.CV	CV.CVC.CV	CVC	CV.V.CV.CV	CV	CV	CVC.CCV
(e)	NSV.NV.NV	NVS.NV	NVS	VS.NV.SV.V.NV	NV.NVS.NV	NVS	NV.V.NV.NV	NV	NV	NVS.NSV

Figure 8: Universal phonetic decipherment. The ciphertext (a) is the same as in the previous figure. (b) is an unsupervised consonant-vowel decipherment, (c) is a decipherment informed by syllable structure, (d) is an improved decipherment, and (e) is a decipherment that also attempts to distinguish sonorous (S) and non-sonorous (N) consonants.

We see that the Spanish letters are accurately divided into consonants and vowels, and it is also straightforward to ask about the learned syllable generation probabilities—they are CV (0.50), CVC (0.20), V (0.16), VC (0.11), CCV (0.02), CCVC (0.0002).

As a sanity check, we manually remove all P(c | p) parameters that match C with Spanish vowel-letters (a, e, i, o, u, y, and accented versions) and V with Spanish consonant-letters (b, c, d, etc), then re-run the same EM learning. We obtain the same P(c).

Exactly the same method works for Latin. Interestingly, the fully-connected P(c | p) model leads to a higher P(c) than the "correctly" constrained channel. We find that in the former, the letter i is sometimes treated as a vowel and other times as a consonant. The word "omnium" is analyzed by EM as VC.CV.VC, while "iurium" is analyzed as CVC.CVC.

We went a step further to see if EM could identify which letters encode *sonorous* versus *non-sonorous* consonants. Sonorous consonants are taken to be perceptually louder, and include n, m, l, and r. Additionally, vowels are more sonorous than consonants. A universal tendency (the *sonority hierarchy*) is that syllables have a sonority peak in the middle, which falls off to the left and right. This captures why the syllable G R A R G sounds more typical than R G A G R. There are exceptions, but the tendency is strong.

We modify our source model to generate S (sonorous consonant), N (non-sonorous consonant), V, and SPACE. We do this by changing the spell-out to probabilistically transform CCVC, for example, into either N S V S or N S V N, both of which respect the sonority hierarchy. The result is imperfect, with the EM hijacking the extra symbols. However, if we first run our C, V, SPACE model and feed the learned model to the S, N, V, SPACE model, then it works fairly well, as shown in Figure 8(e). Learned vowels include (in order of generation probability): e, a, o, u, i, y. Learned sonorous consonants include: n, s, r, l, m. Learned non-sonorous consonants include: d, c, t, l, b, m, p, q. The model bootstrapping is good for dealing with too many parameters; we see a similar approach in Brown et al's (1993) march from Model 1 to Model 5.

There are many other constraints to explore. For example, physiological constraints make some phonetic combinations more unlikely. AE N T and AE M P work because the second sound leaves the mouth well-prepared to make the third sound, while AE N P does not. These and other constraints complement the model by also working across syllable boundaries. There are also constraints on phoneme inventory (no voiced consonant like B without its unvoiced partner like P) and syllable inventory (no CCV without CV).

7 Brute-Force Phonetic Decipherment

Another approach to universal phonetic decipherment is to build phoneme n-gram databases for all human languages, then fully decipher with respect to each in turn. At the end, we need an automatic procedure for evaluating which source language has the best fit.

There do not seem to be sizeable phoneme-sequence corpora for many languages. Therefore, we used source character models as a stand in, decoding as in Section 3. We built 80 different source models from sequences we downloaded from the UN Universal Declaration of Human Rights website.[1]

Suppose our ciphertext starts "cevzren cnegr qry..." as in Figure 9(a). We decipher it against all 80 source language models, and the results are shown in Figure 9(b-f), ordered by post-training P(c). The system believes 9(a) is enciphered Spanish, but if not, then Galician, Portuguese, or Kurdish. Spanish is actually the correct answer, as the ciphertext is again Don Quixote (put through a simple letter substitution to show the problem from the computer's point of view). Similarly, EM detects that "fpn owoktvcv hu ihgzsnwfv rqcffnw cw..." is actually English, and deciphers it as "the analysis of wocuments pritten in..."

Many writing systems do not write vowel sounds. We can also do a brute force decipherment of vowelless writing by extending our channel model: first, we deterministically remove vowel sounds (or letters, in the above case), then we probabilistically substitute letters according to P(c | p). For the ciphertext "ceze ceg qy...", EM still proposes Spanish as the best source language, with decipherment "prmr prt dl..."

8 Word-Based Decoding

Letter-based substitution/transposition schemes are technically called ciphers, while systems that make whole-word substitutions are called *codes*. As an example code, one might write "I will bring the parrot to

[1] www.un.org/Overview/right.html

```
(a) cevzren cnegr qry vatravbfb uvqnytb qba dhvwbgr qr yn znapun

    P(c)      proposed  final
 perplexity   source    edit-dist          best P(p | c) decipherment

(b) 166.28    spanish    434    primera parte del ingenioso hidalgo don quijote de la mancha
(c) 168.75    galician   741    primera palte der ingenioso cidalgo don quixote de da mancca
(d) 169.07    portug.   1487    privera porte dal ingenioso didalgo dom quivote de ho concda
(e) 169.33    kurdish   4041    xwelawe berga mas estaneini hemestu min jieziga ma se lerdhe
  ...
(f) 179.19    english   4116    wizaris asive bec uitedundl pubsctl bly whualve be ks asequs
```

Figure 9: Brute-force phonetic decipherment. (a) is ciphertext in an unknown source language, while (b-f) show the best decipherments obtained for some of the 80 candidate source languages, automatically sorted by P(c).

Canada" instead of "I will bring the money to John"—or, one might encode every word in a message. Machine translation has code-like characteristics, and indeed, the initial models of (Brown et al., 1993) took a word-substitution/transposition approach, trained on a parallel text.

Because parallel text is scarce, it would be very good to extend unsupervised letter-substitution techniques to word-substitution in MT. Success to date has been limited, however. Here we execute a small-scale example, but completely from scratch.

In this experiment, we know the Arabic cipher names of seven countries: m!lyzy!, !lmksyk, knd!, bryT!ny!, frns!, !str!ly!, and !ndwnysy!. We also know a set of English equivalents, here in no particular order: Mexico, Canada, Malaysia, Britain, Australia, France, and Indonesia. Using non-parallel corpora, can we figure out which word is a translation of which? We use neither spelling information nor exclusivity, since these are not exploitable in the general MT problem.

To create a ciphertext, we add phrases X Y and Y X to the ciphertext whenever X and Y co-occur in the same sentence in the Arabic corpus. Sorting by frequency, this ciphertext looks like:

3385	frns!	bryT!ny!
3385	bryT!ny!	frns!
450	knd!	bryT!ny!
450	bryT!ny!	knd!
410	knd!	frns!
410	frns!	knd!
386	knd!	!str!ly!
386	!str!ly!	knd!
331	frns!	!str!ly!
331	!str!ly!	frns!
	etc.	

We create an English training corpus using the same method on English text, from which we build a bigram P(p) model:

511	France/French	Britain/British
511	Britain/British	France/French
362	Canada/Canadian	Britain/British
362	Britain/British	Canada/Canadian
182	France/French	Canada/Canadian
182	Canada/Canadian	France/French
140	Britain/British	Australia/Australian
140	Australia/Australian	Britain/British
133	Canada/Canadian	Australia/Australian
133	Australia/Australian	Canada/Canadian
	etc.	

Each corpus induces a kind of world map, with high frequency indicating closeness. The task is to figure out how elements of the two world maps correspond.

We train a source English bigram model P(p) on the plaintext, then set up a uniform P(c | p) channel with 7x7=49 parameters. Our initial result is not good: EM locks up after two iterations, and every English word learns the same distribution. When we choose a random initialization for P(c | p), we get a better result, as 4 out of 7 English words correctly map to their Arabic equivalents. With 5 random restarts, we achieve 5 correct, and with 40 or more random restarts, all 7 assignments are always correct. (From among the restarts, we select the one with the best post-EM P(c), not the best accuracy on the task.) The learned P(c | p) dictionary is shown here (correct mappings are marked with *).

```
P(!str!ly!   | Australia/Australian) = 0.93 *
P(!ndwnysy! | Australia/Australian) = 0.03
P(m!lyzy!   | Australia/Australian) = 0.02
P(!mksyk    | Australia/Australian) = 0.01

P(bryT!ny!  | Britain/British) =      0.98 *
P(!ndwnysy! | Britain/British) =      0.01
P(!str!ly!  | Britain/British) =      0.01

P(knd!      | Canada/Canadian) =      0.57 *
P(frns!     | Canada/Canadian) =      0.33
P(m!lyzy!   | Canada/Canadian) =      0.06
P(!ndwnysy! | Canada/Canadian) =      0.04

P(frns!     | France/French) =        1.00 *

P(!ndwnysy! | Indonesia/Indonesian) = 1.00 *

P(m!lyzy!   | Malaysia/Malaysian) =   0.93 *
P(!lmksyk   | Malaysia/Malaysian) =   0.07

P(!lmksyk   | Mexico/Mexican) =       0.91 *
P(m!lyzy!   | Mexico/Mexican) =       0.07
```

9 Conclusion

We have discussed several decipherment problems and shown that they can all be attacked by the same basic

method. Our primary contribution is a collection of first empirical results on a number of new problems. We also studied the following techniques in action:

- executing random restarts
- cubing learned channel probabilities before decoding
- using uniform probabilities for parameters of less interest
- checking learned P(c) against the P(c) of a "correct" model
- using a well-smoothed source model P(p)
- bootstrapping larger-parameter models with smaller ones
- appealing to linguistic universals to constrain models

Results on all of our applications were substantially improved using these techniques, and a secondary contribution is to show that they lead to robust improvements across a range of decipherment problems.

All of the experiments in this paper were carried out with the Carmel finite-state toolkit, (Graehl, 1997), which supports forward-backward EM with epsilon transitions and loops, parameter tying, and random restarts. It also composes two or more transducers while keeping their transitions separate (and separately trainable) in the composed model. Work described in this paper strongly influenced the toolkit's design.

Acknowledgements

We would like to thank Kie Zuraw and Cynthia Hagstrom for conversations about phonetic universals, and Jonathan Graehl for work on Carmel. This work was funded in part by NSF Grant 759635.

References

Blevins, J. 1995. The syllable in phonological theory. In J. Goldsmith, editor, *Handbook of Phonological Theory*. Basil Blackwell, London.

Brown, P., S. Della Pietra, V. Della Pietra, and R. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2).

Chadwick, J. 1958. *The Decipherment of Linear B*. Cambridge University Press, Cambridge.

Coe, M. 1993. *Breaking the Maya Code*. Thames and Hudson, New York.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(B).

Finch, S. and N. Chater. 1991. A hybrid approach to the automatic learning of linguistic categories. *Artificial Intelligence and Simulated Behaviour Quarterly*, 78.

Graehl, Jonathan. 1997. Carmel finite-state toolkit. http://www.isi.edu/licensed-sw/carmel/.

Knight, K. 1999. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4).

Knight, K. and K. Yamada. 1999. A computational approach to deciphering unknown scripts. In *ACL Workshop on Unsupervised Learning in Natural Language Processing*.

Peleg, S. and A. Rosenfeld. 1979. Breaking substitution ciphers using a relaxation algorithm. *Communications of the ACM*, 22(11).

Smith, L. 1943. *Cryptography*. Dover Publications, NY.

Steriade, D. 1998. Alternatives to syllable-based accounts of consonantal phonotactics. In *Proc. of Conf. on Linguistic and Phonetics (LP'98)*.

Yarowsky, D. 1994. Decision lists for lexical ambiguity resolution: Application to accent restoration in Spanish and French. In *Proc. ACL*.

Yarowsky, D. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proc. ACL*.