Spam Deobfuscation using a Hidden Markov Model

Honglak Lee
Computer Science Department &
Department of Applied Physics
Stanford University
Stanford, CA 94305

Andrew Y. Ng
Computer Science Department
Stanford University
Stanford, CA 94305

Abstract

To circumvent spam filters, many spammers attempt to obfuscate their emails by deliberately misspelling words or introducing other errors into the text. For example *viagra* may be written *vigra*, or *mortgage* written *m0rt gage*. Even though humans have little difficulty reading obfuscated emails, most content-based filters are unable to recognize these obfuscated spam words. In this paper, we present a hidden Markov model for deobfuscating spam emails. We empirically demonstrate that our model is robust to many types of obfuscation including misspellings, incorrect segmentations (adding/removing spaces), and substitutions/insertions of non-alphabetic characters.

1 Introduction

Text-based spam filters are workhorses in the battle against spam emails, and work by examining the content of an email to determine if it is spam. For example, the naive Bayes algorithm [Sahami *et al.*, 1998] estimates the probability of an email being spam based on the words contained in the message. Thus, an email containing words like *viagra* and *mortgage* will typically be classified as spam.

To circumvent such spam filters, spam emails frequently contain obfuscated words in their subject lines or message bodies. Humans can still recognize the original words regardless of extra, missing, or replaced characters, but these words are no longer recognized by most text-based filters. For example, some obfuscated words seen in actual spam emails include:

Original word	Obfuscated words
refinance	r.efina.nce, r-efin-ance, re xe finance
mortgage	mort gage, mo>rtglage, mor;tg2age
viagra	v*1agra, v-i-a-g-r-a, v1@gra, vjaggra
unsubscribe	u.n sabcjbe, un susc ribe

Note that some of these obfuscation examples include adding extra spaces to a word; for emails containing this type of obfuscation, a naive tokenizer (for an algorithm such as naive Bayes) will not even correctly identify word boundaries.

Some anti-spam systems use carefully hand-crafted regular expressions to detect obfuscated words. However, these are expensive to maintain, and are not particularly robust to the wide range of possible obfuscation. For example, regular expressions generated by CMOScript[1] fail to correctly detect 6 of the 12 examples above. Also, CMOScript is not particularly robust to misspellings, and thus fails on most examples involving insertions or deletions of alphabetic characters. There has also been work on using string edit distance [Ristad and Yianilos, 1997]; for example, [Oliver, 2005] uses a hand-crafted lexicographical distance to detect specific spam words.

In principle, a learning algorithm for anti-spam may, given sufficient data, eventually learn to classify obfuscated emails as spam. For example, given a few examples of spam with the phrase *mort gage*, naive Bayes (or any other reasonable learning algorithm) will learn that *mort* and *gage* are tokens that are indicative of spam, and thus learn to classify future emails with *mort gage* as spam. However, given the essentially unlimited number of ways to obfuscate any word (see the table above), in practice it seems unlikely that any standard learning algorithm will be able to see, and thereby learn to detect, all of them in any reasonable amount of time.

In this paper, we propose a deobfuscation method based on a hidden Markov model (HMM). An HMM is a statistical model, and is derived by assuming that the system under consideration is a Markov process with hidden states. [Rabiner, 1989] HMMs have

[1]CMOScript is a regular expression rule generator, and is used by SpamAssassin to detect obfuscation. For more information, see the SpamAssassin (http://spamassassin.apache.org/) and the CMOScript (http://sandgnat.com/cmos/) websites.

found significant successes in automatic speech recognition [Jurafsky and Martin, 2000], natural language processing [Manning and Schutze, 1999], and computational biology [Durbin *et al.*, 1998]. This paper develops an HMM model for converting obfuscated text back into the original text intended by the sender. The HMM is given an obfuscated word (or sentence) as input (observation), and produces the deobfuscated word (or sentence) as output. Thus, the HMM has to correct deliberate misspellings, incorrect segmentations (adding/removing spaces), and other word modifications such as substitutions and insertions of non-alphabetic characters.

2 Preliminaries

A hidden Markov model [Rabiner, 1989] is characterized by N hidden states, M emission (observation) symbols, a state transition probability table $P(X_{t+1}|X_t)$, an emission probability table $P(O_{t+1}|X_t \rightarrow X_{t+1})$ specifying the probability of an emission under different state transitions,[2] and an initial state X_0.[3] The HMM model defines a generative random process that proceeds as follows: First, the system is at initial state X_0 at time $t = 0$. Then, the state X_t ($t = 0, 1, 2, \ldots$) evolves according to a Markov process with transition probabilities given by $P(X_{t+1}|X_t)$. On each state transition, an observation is emitted with probability governed by the emission probabilities $P(O_{t+1}|X_t \rightarrow X_{t+1})$. There is also a distinguished observation corresponding to the empty string (in which nothing is emitted). Such a state transition on which nothing is emitted is conventionally called an epsilon (null) transition [Jelinek, 1999].

In a typical HMM application, only the emissions O_t can be observed directly, and the states X_t are hidden. Thus, the sequence of hidden states $x_{1\ldots T} = (x_1 x_2 \ldots x_T)$ must typically be inferred using the observations $o_{1\ldots T} = (o_1 o_2 \ldots o_T)$. More formally, one of the basic tasks in HMM inference is finding the *maximum a posteriori* (MAP) sequence of states:

$$x^{MAP}_{1\ldots T} \quad = \arg\max_{x_{1\ldots T}} P(x_{1\ldots T}|o_{1\ldots T}),$$

The Viterbi algorithm [Viterbi, 1967] is a dynamic programming algorithm for efficiently finding the MAP sequence.

Another basic task in HMM applications is to estimate the parameters, such as the state transition probabilities and the emission probabilities, from data. If we have a training set comprising sequences of both states $x_{1\ldots,T}$ and observations $o_{1\ldots,T}$, we can straightforwardly compute the maximum likelihood estimates (MLE) of the parameters; if only the observations are available during training, the EM (Baum-Welch) algorithm [Rabiner, 1989] can be applied.

3 Lexicon Tree Hidden Markov Model

In this section, we describe our first model, which we call the lexicon tree hidden Markov model. In order to deobfuscate *mortg3ge* into *mortgage*, clearly the system must know that *mortgage* is an English word, so that it can distinguish it from other, seemingly equally likely strings, such as *mortgege* and *mortgige*.

Thus, our model integrates lexicon (dictionary) and context information into the HMM. More formally, the state space of our HMM consists of one start state S_0, one final state S_f, and additional states corresponding to all the characters of each word in the dictionary. For computational efficiency, this HMM is represented as a prefix tree with all terminal nodes[4] connected to S_f, and S_f connected to S_0.[5] We used a standard English dictionary (45475 words), and built a lexicon tree HMM with 110919 states. The transition diagram for this HMM is shown in Figure 1.

The set of emission symbols consists of 70 characters including the 26 letters of the alphabet, the space, and all other standard ASCII characters[6] except the control characters. For simplicity, all letters in the text were converted to lowercase as a preprocessing step. From most states of the HMM, we can make a non-epsilon state transition in which we print out one character and advance to the next character in the word. Self-transitions (in which the state remains the same after printing out a character) are also possible. In detail, assuming for the moment that we only wish to model character substitutions and insertions within words, we define the state transition probabilities as follows:

$$P_0(X_{t+1}|X_t) = \begin{cases} 1-\eta & \text{if } X_{t+1} = X_t \\ \eta\, f_{X_{t+1}}/f_{X_t} & \text{if } X_{t+1} \text{ is } X_t\text{'s child} \\ \eta\, h_{X_t}/f_{X_t} & \text{if } X_{t+1} = S_f, X_t \text{ is a terminal node} \\ 0 & \text{otherwise.} \end{cases}$$

Here, η is a parameter that controls the self-transition (single character insertion) probability, f_X is defined as the total frequency of all words sharing the common

[2] In this paper, we assume the emission depends on the state transition (Mealy machine). HMMs in which the emission depends only on the current state $P(O_t|X_t)$ can also be used (Moore machine).

[3] In the fully general case, the initial state is sampled from some initial-state distribution π. In this paper, for simplicity we assume $\pi(X_0) = 1$.

[4] Every terminal node corresponds to the last character of some word in the dictionary. Actually, in our model, we allow a small but non-zero probability of the state-sequence ending at a "non-terminal node" (for example, if a text is truncated), and thus technically all nodes are potentially terminal nodes. But for ease of exposition, we will abuse terminology and refer only to these distinguished end-of-word nodes as "terminal nodes."

[5] In our model, $P(S_0|S_f) = 1$, with no emissions.

[6] I.e., the first 128 ASCII characters

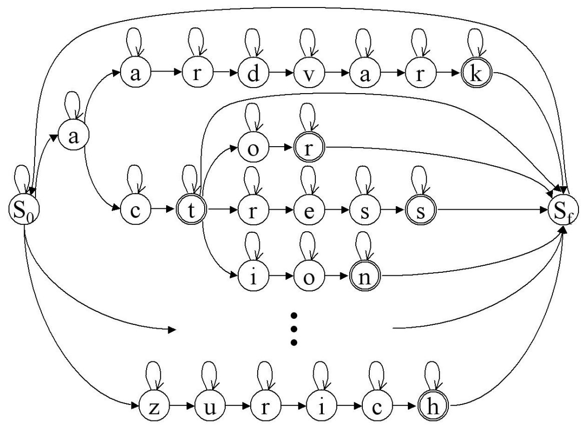

Figure 1: Transition diagram of lexicon tree HMM. For simplicity, only six words (aardvark, act, actor, actress, action, zurich) are shown. The arrows in this figure indicate the set of possible successor states X_{t+1} that we can make a transition to from X_t. Thus, a valid sequence of states corresponds to a sequence of words in the dictionary. If $X_t = S_f$, then the state makes an epsilon-transition back to S_0.

prefix $(S_0 \ldots X)$[7], and h_X[8] is defined as the frequency of the word $(S_0 \ldots X)$. For S_f, the probability of transitioning back to S_0 is 1. Note that if η is 1, then the probability of taking a path corresponding to a certain word is proportional to the empirical frequency of that word in normal English usage.

Now, to take into account character deletions, we augment our model by allowing epsilon transitions,[9] and define an augmented set of transition probabilities as follows:

$$
\begin{aligned}
Q(X_{t+1}|X_t) &= \epsilon P_0(X_{t+1}|X_t) \\
P(X_{t+1}|X_t) &= (1-\epsilon) P_0(X_{t+1}|X_t).
\end{aligned}
$$

Here, ϵ is a parameter that controls the chance of an epsilon transition; $Q(X_{t+1}|X_t)$ is the chance of making an epsilon transition from X_t to X_{t+1} (without emitting an observation); $P(X_{t+1}|X_t)$ is the non-epsilon state transition probability.[10] The term $P_0(X_{t+1}|X_t)$ is as defined in the previous paragraph.

We parameterize the emission probabilities as follows:

$$
P(O_{t+1}|X_t \to X_{t+1}) = \begin{cases} (1-\tau) P_{\text{emit}}(O_{t+1}|X_{t+1}) \\ \qquad + \tau P_{\text{random}}(O_{t+1}) & \text{if } X_{t+1} = X_t \neq S_0 \\ P_{\text{random}}(O_{t+1}) & \text{if } X_{t+1} = X_t = S_0 \\ P_{\text{emit}}(O_{t+1}|X_{t+1}) & \text{otherwise,} \end{cases}
$$

where τ is a parameter of the model. Informally, $P_{\text{emit}}(O_{t+1}|X_{t+1})$ will be a (learned) distribution of characters O_{t+1} that are commonly used to represent X_{t+1}. For example, $P(O_{t+1} = \text{'1'} | X_{t+1} = \text{'l'})$ may be large, since '1' (one) is often used to obfuscate 'l' (the letter *ell*). $P_{\text{random}}(O_{t+1})$ is a distribution over non-meaningful characters that may be inserted into the middle of a word to obfuscate it (as in *mort***gage*). Note that the parameterization for emission probability under a self-transition (when $X_t \neq S_0$) includes both the terms P_{random} and P_{emit}. This parameterization reflects an observation that self-transition emissions are usually either non-meaningful obfuscation characters (such as "***") or repetitions of the current state character (as in *viaaagra*).

3.1 Feature-based Model of Emission Probability

We now describe how the probabilities P_{emit} and P_{random} were estimated from data.

We collected a training set corpus of obfuscated spam emails (160 lines of text), and hand-annotated them with the true (unobfuscated) text. The unobfuscated text was manually aligned to the obfuscated text, so that we know the true state of the HMM during each character emission. For example, *m0rt*g*age* (obfuscated) is annotated with *mort-g-age* (unobfuscated) where '-' refers to a self-transition.

Given this training data, one straightforward approach to estimating $P_{\text{emit}}(O|X)$ would have been to use a maximum likelihood-like estimate, in which we estimate $P_{\text{emit}}(O|X)$ as the fraction of times in the training set that O was observed during a transition into state X. However, the size of the training set is too small to accurately obtain estimates this way. Further, spammers frequently use new characters O to obfuscate a character X. For example, spammers are more likely to use the character '<' than the character '>' to obfuscate the letter 'c' (because '<' looks more visually similar to 'c'). But unless we observe this particular example in our training set, this would not be captured by a straightforward maximum likelihood estimate of the parameters using a 40x70 table to represent $P_{\text{emit}}(O|X)$. More specifically, this algorithm would easily be defeated by spammers finding

[7] $(S_0 \ldots X)$ denotes a string corresponding to the path in the graph from S_0 to X. All frequency statistics such as f_X and h_X were measured from a large email corpus.

[8] Note that h_X is nonzero only for terminal nodes (the last character of any word).

[9] In principle, a character deletion can occur at any point in a word. Thus, there can be an epsilon transition at any state. For computational efficiency, we restrict our HMM so that epsilon transitions cannot form a loop. [Jelinek, 1999] We do so by disallowing epsilon transitions from the initial state (S_0) to the first character of each word. This roughly corresponds to assuming that the first character of the original word (or an obfuscated version of this character) appears in the text. By considering the rest of the word, the resulting model is typically still able to correctly identify obfuscated words in which the first character is missing.

[10] Note that $\sum_X P(X|X_t) + Q(X|X_t) = 1$. See [Jelinek, 1999] for more details.

new characters to obfuscate old ones. (For example, 'ç', '(', '[', '{', '<', can all be used to obfuscate 'c'.)

Thus, to improve the generalization of the algorithm, we took an approach in which *features* of O and X are used to estimate $P_{\text{emit}}(O|X)$. More formally, we used a softmax regression/maximum entropy [Della Pietra *et al.*, 1997] model:

$$P_{\text{emit}}(O|X) = \frac{\exp(\sum_k \lambda_k f_k(O,X))}{\sum_O \exp(\sum_k \lambda_k f_k(O,X))}. \quad (1)$$

Here, $f_k(O,X)$'s are features measuring the "similarity" between the characters O and X,[11] and the λ_k's are parameters of the model.

The features indicate quantities such as whether O and X are the same character; their visual similarity (shape context similarity [Belongie *et al.*, 2002]; pixel-level similarity); phonetic similarity (e.g., *Sialis* for *Cialis*); and additional features for certain special characters such as spaces. More precisely, we used the following features:

1. Character Equality (denoted CE).
$$f_1(O|X) = \begin{cases} 1 & \text{if } O \text{ and } X \text{ are the same character} \\ 0 & \text{otherwise} \end{cases}$$
2. Shape Context Similarity (denoted SC). The "shape context distance" algorithm [Belongie *et al.*, 2002] measures the visual similarity between two characters, and currently has the best reported result in the literature for handwritten digit recognition.[12] We used $f_2(O,X) = -D_{\text{SC}}(O,X)$.
3. Pixel-Level Similarity (denoted PL). $f_3(O,X) = -\min(D_{\text{PL}}(O,X), D_{\text{PL}}(X,O))$. Here, D_{PL} measures the pixel-level dissimilarity between the characters O and X.[13]
4. Phonetic Similarity (denoted PH). This measures whether the two characters O and X are phonetically similar. For example, b, p, f and v are phonetically similar; and so are d and t, etc. Based on the Soundex algorithm [Setter, 1997].[14]
5. Space Features (2 binary-valued features, denoted SP).
$f_5(O,X) = 1 \iff O$ is punctuation, X is space
$f_6(O,X) = 1 \iff O$ is not punctuation, X is space

The model for P_{emit} was trained using softmax regression/maximum likelihood, implemented using Newton's method.[15]

Finally, the model for $P_{\text{random}}(O_t)$ was learned also using the same softmax parameterization as in Equation (1) (but with features $f_k(O,X)$ replaced by $f_k(O)$), and using the following three binary-valued features:

Random Character Features (denoted RC).
$f_1(O) = 1 \iff O$ is a letter
$f_2(O) = 1 \iff O$ is a number (0-9)
$f_3(O) = 1 \iff O$ is punctuation

Section 3.4 will also present results from an ablation analysis in which we use only a subset of the features above.

3.2 Decoding Method

Given obfuscated input $\vec{o} = o_{1\ldots T}$, we want to find the most likely sequence of true states $\vec{x}^{MAP} = x^{MAP}_{1\ldots T}$. Thus we want to maximize the posterior probability $P(\vec{x}|\vec{o},\vec{\theta})$. Here, $\vec{\theta}$ are the parameters of the model.

$$\begin{aligned} \vec{x}^{MAP} &= \arg\max_{\vec{x}} P(\vec{x}|\vec{o},\vec{\theta}) \\ &= \arg\max_{\vec{x}} P(\vec{x},\vec{o}|\vec{\theta}) \\ &= \arg\max_{\vec{x}} P(\vec{x}|\vec{\theta})P(\vec{o}|\vec{x},\vec{\theta}) \end{aligned}$$

Above, $P(\vec{x}|\vec{\theta})$ and $P(\vec{o}|\vec{x},\vec{\theta})$ are the total transition and emission probabilities for the sequence.

The Viterbi algorithm [Viterbi, 1967] can be used to perform this maximization, but a straightforward implementation of it has time complexity $O(N^2T)$, where T is length of the input and $N \approx 10^5$ is the number of states. Using a sparse representation of the state transition probabilities, this is improved to $O(NKT)$,

[11] Here, "the character X" really means "the character corresponding to the HMM state X." But in this section we will use X to simply denote its corresponding character, since there is little risk of confusion.

[12] To compute $D_{\text{SC}}(O,X)$, we rendered the characters X and O in Arial font, and measured their image similarity using [Belongie *et al.*, 2002]. We found empirically that this measure of similarity worked best on simple, and sans serif, fonts such as Arial.

[13] Implementational details: To achieve invariance to the scaling (size) and translation of the character, we used

$$D_{\text{PL}}(I_1,I_2) = \min_{m_x,m_y} \min_{d_x,d_y} \frac{M(I_1, T_{m_x,m_y,d_x,d_y}(I_2))}{N(I_1) + N(T_{m_x,m_y,d_x,d_y}(I_2))},$$

where I_1 and I_2 are rendered images corresponding to the characters; m_x, m_y (=1.0, 1.1, ..., 1.5) and d_x, d_y are scaling and translation factors along the x and y directions; $T_{m_x,m_y,d_x,d_y}(I_2)$ is I_2 scaled and translated according to m_x, m_y, d_x, d_y; $M(I_1,I_2)$ is the number of mismatched pixels between I_1 and I_2; and $N(I)$ is the number of non-white pixels in I.

[14] More formally, $f_4(O,X) = 1 \iff \{O,X\} \subseteq$ one of {b,p,f,v}, {c,s,g,j,k,q,x,z}, {d,t}, {l}, {m,n}, {r}, {b,d,g}. See [Setter, 1997] for details.

[15] Only emissions for non-self transitions were used as the "training set" for learning P_{emit}; and emissions for self-transitions were used to learn P_{random}. The latter introduces a little error into the model, since some self-transitions may have resulted in observations generated from P_{emit}, but in practice this simple procedure appears to work well.

where K is the maximum out-degree (number of possible successor states) of any state in the HMM. ($K \approx 40$ in our HMM.) However, this is still too slow, and even in our carefully optimized implementation, was able to process text only at a rate of about 10 characters per second.

We can further speed up the algorithm by applying beam search [Jelinek, 1999]. In our implementation, we used a beam width of 10,000,000. The resulting algorithm is able to deobfuscate 240 characters/sec—a 20-fold speedup.

3.3 Parameter Learning

Apart from the parameters λ_k used to define P_{random} and P_{emit} (see Section 3.1), our model has five additional parameters that have to be learned.[16] Letting $\vec{\theta}$ denote these parameters, we learn $\vec{\theta}$ using locally greedy hillclimbing search to maximize the log likelihood of the training data:

$$\begin{aligned}&\arg\max_{\vec{\theta}} \textstyle\sum_i \log P(\vec{x}^{(i)}_{true}, \vec{o}^{(i)}|\vec{\theta}) \\ &= \arg\max_{\vec{\theta}} \textstyle\sum_i \log P(\vec{x}^{(i)}_{true}|\vec{\theta}) P(\vec{o}^{(i)}|\vec{x}^{(i)}_{true}, \vec{\theta}).\end{aligned}$$

Here, $\vec{x}^{(i)}_{\text{true}}$ is the sequence of true states in the ith training example. Note that, in the training set, $\vec{x}^{(i)}_{\text{true}}$ is aligned to the corresponding sequence of emissions. Thus, the objective in the optimization above does not require decoding, and can be computed quickly (in the inner loop of greedy hillclimbing search). We learned the parameters $\vec{\theta}$ using the same training set as the one used to estimate the λ_k's for P_{random} and P_{emit}.

3.4 Experimental Results

To informally test the basic functionality of our model, we applied it to 60 obfuscated variants of *viagra*.[17] Our algorithm correctly decoded 59 of the 60 examples.[18]

We also performed more rigorous testing on a held-out test set of 606 lines of actual spam containing 849 obfuscated words. We define the deobfuscation accuracy as the fraction of correctly deobfuscated words. Table 1 shows the accuracy of our algorithm for different types of obfuscation, such as insertion, substitution, misspelling, etc.[19] Examples of these types of deobfuscation are given in Table 2; and some examples of deobfuscated text are shown in Table 3. Words in the output that were modified from the original (i.e., ones that were deobfuscated) are written in italics, and the algorithm's errors are underlined. From Table 1, we see that our model appears to do well on misspelling, segmentation, insertion and substitution obfuscations.

Table 1: Results of lexicon tree HMM decoding

Type	Correct	Total	Accuracy
Insertion	408	425	0.96
Substitution	160	176	0.909
Misspelling	268	293	0.915
Segmentation	65	68	0.956
All obfuscations	796	849	0.938
All words	4542	4805	0.945

Table 4: Accuracy on all obfuscated words, using different feature subsets.

Features	Deobfuscation Accuracy
CE only	0.867
CE+RC	0.931
CE+RC+PL	0.934
CE+RC+PL+PH	0.936
CE+RC+PL+PH+SC	0.938
All features	0.938

We also perform an experiment to analyze the utility of the different features in our feature-based emission probability model. Table 4 shows the result of running an ablation analysis, in which the experiment is repeated using different subsets of the features. We see that the most important features are the "random character features" (RC), used to model the emissions of non-meaningful characters during self-transitions. Also, the "pixel-level similarity feature" (PL) and the "phonetic similarity feature" (PH) significantly improve accuracy. However, the "shape context feature" (SC) and the "space features" (SP) appeared to play a minor role in the algorithm's performance.[20]

[16] The parameters η, τ, ϵ were described in the text. Two additional parameters δ and ζ were used to adjust the probability of transition from S_0 to its children, and of transition into S_f (i.e., controlling the probability of continuing vs. terminating the sequence at the end of a word). Details omitted due to space constraints.

[17] These obfuscated variants of *viagra* were obtained from http://www.cockeyed.com/lessons/viagra/viagra.html.

[18] For example, *Viagorea*, *ViagDrHa*, *VyAGRA*, *via---gra*, *viagrga*, etc. The only error was *viaverga*, which was decoded as "*via verge*."

[19] Because the different types of obfuscation are not mutually exclusive, the first four rows in the table do not sum up to equal the "all obfuscations" line. The "all words" line shows accuracy on all words, including ones not originally obfuscated. Note that in all cases, individual lines (not words) of text are given as input to the algorithm, and thus it must choose its own segmentation of the text into words.

[20] Although the shape context method works very well for handwritten digit recognition and for general shape comparisons [Belongie *et al.*, 2002], there appear to be many instances in which shape context penalizes for slight variations in a character's shape, while pixel-level similarity does not (for example, in the slight differences between '1', 'I', 'l', and '|'). Empirically, the shape context distance is large when one image cannot be easily transformed to the other by an affine transformation. For example, '|' cannot be easily changed to '1' or 'I' by an affine transformation, and thus the corresponding shape context distances are large, even though the characters are visually very similar.

Table 2: Examples of different types of deobfuscation

Type	**Obfuscated Input**	**Deobfuscated Output**
Insertion+Misspelling	u-n-s-u-s-c-r-i-b-e link	*unsubscribe* link
Insertion+Segmentation	Re xe finance	*refinance*
Substitution	M/\ cromei)i/\	*macromedia*
Substitution	gre4t pr1ces	*great prices*
Misspelling	heyllooo it's me Chelsea	*hello* it's me chelsea
Misspelling	veerrryyy cheeaapp	*very cheap*
Misspelling	u.n sabscjbe	*unsubscribe*
Segmentation+Insertion	con. tains forwa. rdlook. ing sta. tements	*contains forward looking statements*
Segmentation+Insertion	pa, rty for sen, ding this re.port.	*party for sending this report*
Segmentation+Substitution	get your u n iversi t y d i pl0 m a	get your *university diploma*
Segmentation+Misspelling	ree movee below:	*remove* below

Table 3: Additional examples of obfuscated text and of HMM output

Obfuscated Input	**Deobfuscated Output**
hello dear home owpyner,	hello dear *homeowner*
we h;avve been n3otifie^d that you^r mo>rtglage rate	we *have* been *notified* that *your mortgage* rate
is fi>xegvd avqt a very hivgh in_tevrest rate.	is *fixed* data very *high interest* rate
therefor%e you arge current overpapying,	*therefore* you *are current overpaying*
which s:ums-up to t'housxarn ds oabf dolltyrars annuallly .	which *sums* up to *thousands of dollars* annually
l:uckily fafor you wtce cuqan gqjufuarantee	*luckily for* you the cuban *guarantee*
th'e lowest rates in thkhe u.s. (3.50%).	*the* lowest rates in *the* us
so huzmrry beqcause the rate f:orecast is npot looking goo_d!	so *hurry because* the rate *forecast* is *not* looking *good*
there iws no obligations, ahsnd iwt free	there *is* no obligations *and it* free
lock ostn the 3.50*%, even with baiwd cr'ezsdit!!	lockout the even with baird *credit*
cx6lick h'ere nepow for details	*click here* new for details

4 Out-of-dictionary HMM

Although our model has a large dictionary with 45475 words, real input text usually contains many words that do not appear in the dictionary. In this section, we reduce the decoding error of our HMM in such situations by using a bigram model for out-of-dictionary words.

More formally, this augments our previous model by adding to it 26 additional states (corresponding to the letters of the alphabet). Each of the 26 states may transition to any other state (including itself), in addition to a final state S_f indicating the end of a word. The transition diagram of the resulting HMM is shown in Figure 2.

The state transition probabilities for the 26 new states simply indicate the distribution over the next character in a word given the previous character (or the probability of a word terminating after that character). These probabilities are estimated from a large text corpus. (Thus, for example, $P(X_{t+1} = \text{`u'} | X_t = \text{`q'})$ will be close to 1, since 'u' usually follows 'q'.)

The emission probability $P(O_t|X_t)$ for the out-of-dictionary portion of the HMM is identical to that used in the lexicon tree HMM.

Finally, the state transition distribution out of the initial state ($P(X_{t+1}|X_t = S_0)$) was also modified to assign a positive probability to transitioning to each of the 26 new states (proportional to the probability that an English word starts with that letter).

The HMM formalism allows us to combine the out-of-dictionary HMM with the lexicon tree HMM (as shown in Figure 2). Using this construction, Viterbi decoding automatically selects a path through one of the two models, thereby identifying both within-lexicon and out-of-dictionary words in the obfuscated text.[21]

Table 5 shows experimental results using the out-of-dictionary HMM. We see that its accuracy in decoding obfuscated words is slightly lower than the model using only the lexicon tree HMM. This is because almost all obfuscated words are in our dictionary. The decoding accuracy of misspelled words in emails is also slightly reduced, since the model is now more likely to attribute a misspelling to an out-of-dictionary word.

[21] Implementational details: We calibrated the relative probabilities of transitioning to the lexicon tree HMM vs. the out-of-dictionary HMM using the method given in [Raina *et al.*, 2003] (essentially logistic regression).

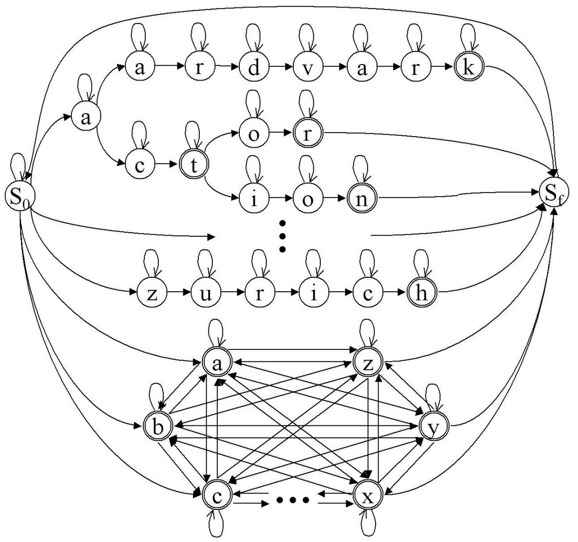

Figure 2: Lexicon tree HMM combined with out-of-dictionary HMM

However, the out-of-dictionary HMM causes the decoding accuracy for out-of-dictionary words to be significantly improved. As a result, overall decoding accuracy is improved compared to the original lexicon tree HMM.

These results suggest that there is a trade-off between deobfuscation accuracy and out-of-dictionary word decoding accuracy. Assuming that a content-based filter is applied to the output of our algorithm to identify spam, its classification accuracy will not be significantly affected by the decoding errors made on out-of-dictionary words.[22] This is because most spam words will be already included in (or can easily be added to) the dictionary. Thus, for the particular application of spam filtering, it may not be necessary to use an out-of-dictionary HMM during deobfuscation.

Table 5: Accuracy of out-of-dictionary (OOD) HMM

Type	Success	Total	Accuracy
Insertion	403	425	0.948
Substitution	160	176	0.909
Misspelling	253	293	0.863
Segmentation	64	68	0.941
All obfuscations	781	849	0.92
OOD words	80	162	0.494
All words	4592	4805	0.956

5 Identifying Obfuscated Emails

So far, our discussion has focused on the problem of deobfuscating spam email, so that a separate text-based filter can correctly detect the spam words. However, if it were possible to detect *whether an email was deliberately obfuscated*—which one might hope is an easier problem than actually recovering the original, deobfuscated content—then this would have been equally useful for identifying spam. This is because the fact that an email was deliberately obfuscated is a strong indicator that it is spam. However, we show in our experiments that the problem of deciding if an email was deliberately obfuscated is surprisingly difficult. In particular, a straightforward application of supervised learning (without attempting to deobfuscate the email) performs worse than our HMM (which does attempt to deobfuscate the email) on the task of deciding if an email was deliberately obfuscated.

Table 6: Examples of out-of-dictionary (OOD) words, decoded using the lexicon tree HMM with and without the OOD HMM.

Original	With OOD	Without OOD
caliphate	caliphate	calibrate
corvette	corvette	corbett
dieting	dieting	editing
fuki sushi	fuki sushi	fujitsu hi
inexpiable	inexpiable	inexplicable
kournikova	kournikova	our nikolai
lodleveler	lodleveler	leveler
palomino	palomino	palming
panasonic	panasonic	pan sonic
scathing	scathing	scratching
seraphim	seraphim	serafin
teetotalers	teetotalers	the totallers

To rigorously explore this idea, we consider a simple feature-based model for identifying obfuscation. We used logistic regression to learn to predict if an email was deliberately obfuscated. Our model used the following features:

1. Number of non-alphabetic characters inside words
2. Average length of words
3. Number of out-of-dictionary words
4. Number of tokens.

Trained on 100 examples (lines of text) and tested on a balanced held-out test set, this model gave 70% classification accuracy (8% false positives, and 22% false negatives). This high error rate suggests that it may be difficult to achieve high anti-spam accuracy using an approach that does not attempt to decode the text, but instead relies only on detecting obfuscation.[23]

[22] One exception that is theoretically possible is if a non-spam out-of-dictionary word is incorrectly decoded as a spam word. However, we have not observed a single instance of this in any of our experiments.

[23] Note that this algorithm does incorporate lexical information (the third feature). If this feature is removed, the error increases to 36%. Also, giving the algorithm an entire email rather than a single line would reduce these error rates, but we still view our results as strongly indica-

We also experimented with a variation of the HMM model for determining if an email is obfuscated. Details are omitted due to space, but briefly, we built two separate HMM models, one for $P(O, X \mid \text{obfuscated})$ and one for $P(O, X \mid \text{non-obfuscated})$. Thus, for example, under the non-obfuscated HMM model, the chance of replacing 'i' with '1' (as in *v1agra*) is much lower. Given a new email, we then predict that it was obfuscated if $P(O \mid \text{obfuscated}) > P(O \mid \text{non-obfuscated})$, and that it was non-obfuscated otherwise.[24] This algorithm attained 84% accuracy on the test-set. The reason appears to be that even normal, non-obfuscated, email contains many out-of-dictionary words, misspelled words, and non-alphabetic characters (for example, in emoticons, unusual URLs, signature files, etc.), and thus can appear to the algorithm to be obfuscated.

While there is clearly still room for improvement in detecting obfuscation—and given sufficient engineering effort, a better obfuscation classifier can certainly be built—these experiments seem to indicate that it is important to actually decode/deobfuscate an email (so that a content-based classifier can determine if it is spam), rather than blindly try to detect the obfuscation attempt without also trying to decode the email.

6 Discussion and Conclusions

In this paper, we proposed an HMM-based model for spam deobfuscation. This model can be used to deobfuscate an email prior to applying a content-based filter. The lexicon tree HMM version of our model exhibited good performance for different kinds of obfuscation patterns including insertions, substitutions, misspellings, and segmentations. The out-of-dictionary HMM version, in contrast, has lower overall decoding error, but slightly higher error on obfuscated spam words. We believe that these methods hold rich promise in applications to spam filtering.

Acknowledgments

We give warm thanks to Rajat Raina, Mike Brzozowski, and Daphne Koller for helpful discussions. We also thank Christian Shelton and Rion Snow for help with spam and email corpora, and Serge Belongie for providing code for his shape context algorithm. H. Lee was supported by a Stanford Graduate Fellowship (SGF). This work was also supported by the Department of the Interior/DARPA under contract number NBCHD030010.

tive of the difficulty of detecting obfuscation attempts. In a separate experiment in which we trained and tested the same algorithm on entire emails' subject+message bodies, the error rate was only slightly lower—26% (using balanced training and test sets of 50 emails each).

[24] Assuming a uniform prior on whether an email is obfuscated, this corresponds to picking the most likely classification by computing $P(\text{obfuscated} \mid O)$ using Bayes rule.

References

[Belongie *et al.*, 2002] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. In *IEEE Transactions on PAMI, Vol. 24, No. 24*, 2002.

[Della Pietra *et al.*, 1997] Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. In *IEEE Transactions on PAMI, Vol. 4, No. 19*, 1997.

[Durbin *et al.*, 1998] R. Durbin, S.R. Eddy, A. Krogh, and G.J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, Cambridge UK, 1998.

[Jelinek, 1999] F. Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, 1999.

[Jurafsky and Martin, 2000] D. Jurafsky and J. H. Martin. *Speech and Language Processing.* Prentice Hall, 2000.

[Manning and Schutze, 1999] C. D. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing.* MIT Press, 1999.

[Oliver, 2005] J. Oliver. Using lexigraphical distancing to block spam. In *Presentation at MIT Spam Conference. Cambridge, MA.*, 2005.

[Rabiner, 1989] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE 77(2):257-286*, 1989.

[Raina *et al.*, 2003] Rajat Raina, Yirong Shen, Andrew Y. Ng, and Andrew McCallum. Classification with hybrid generative/discriminative models. In *NIPS 16*, 2003.

[Ristad and Yianilos, 1997] E. S. Ristad and P. N. Yianilos. Learning string edit distance. In *IEEE Transactions on PAMI*, 1997.

[Sahami *et al.*, 1998] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk e-mail. In *AAAI'98 Workshop on Learning for Text Categorization*, 1998.

[Setter, 1997] Kevin Setter. Soundex algorithm. http://physics.nist.gov/cuu/Reference/soundex.html, 1997.

[Viterbi, 1967] Andrew J. Viterbi. Error bounds for convolutional codes and an asymtotically optimum decoding algorithm. In *IEEE Transactions on Information Theory 13(2):260-267*, 1967.