Cryptogram Decoding for Optical Character Recognition

Gary Huang, Erik Learned-Miller, Andrew McCallum
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{ghuang, elm, mccallum@cs.umass.edu}

Abstract

Optical character recognition (OCR) systems for machine-printed documents typically require large numbers of font styles and character models to work well. When given a document printed in an unseen font, the performance of those systems degrade even in the absence of noise. In this paper, we perform OCR in an unsupervised fashion without using any character models by using a cryptogram decoding algorithm. We present results on real and artificial OCR data.

1 Introduction

Optical character recognition is the task of converting images of text into their editable textual representations. Most OCR systems for machine print text need large collections of font styles and canonical character representations, whereby the recognition process involves template matching for the input character images. Such systems are font dependent and suffer in accuracy when given documents printed in novel font styles. An alternative approach we examine here groups together similar characters in the document and solves a cryptogram to assign labels to clusters of characters. This method does not require any character models, so it is able to handle arbitrary font styles. This approach subsumes the idea of adaptivity [31] in that it can take advantage of patterns such as regularities in image distortions that are particular to each document. In addition, the cryptogram decoding procedure is well-suited for performing OCR on images compressed using token-based methods such as Djvu, Silx, and DigiPaper.

2 Related Work

Treating OCR as a cryptogram decoding problem dates back at least to papers by Nagy [30] and Casey [6] in 1986. There continues to be research done to improve the performance of approaches that use no character models, which are discussed here along with other related work.

In [15], Ho and Nagy develop an unsupervised OCR system that performs character clustering followed by lexicon-based decoding. Their decoding procedure iteratively applies a set of modules to progressively build up assignments based on comparing the "v/p" ratio against manually set thresholds. One major difference between this work and [15] is our use of probabilistic reasoning instead of a predefined ratio. In [23], Lee presents a more unified approach to decode substitution ciphers by using Hidden Markov Models and the expectation maximization algorithm. That work uses n-gram statistics as model priors, whereas ours uses entire word patterns. Breuel [2] introduced a supervised OCR system that is font independent, but it does not take advantage of token-based image compression.

Edwards and Forsyth [12] used a semi-Markov model and a small initial set of character templates to perform OCR on handwritten Latin text. Their idea of inferring the identity of unseen characters using a lexicon is similar to the initial stage of our decoding algorithm.

Lin and Knight [8] decipher the writing order of an ancient hieroglyphic script in a stepwise fashion using n-gram statistics and the EM algorithm [11]. They evaluate the accuracy of their system using known languages and obtained results that are drastically better than random ordering. Our attempt at incorporating n-gram statistics for decoding is encouraged by their result.

Fang and Hull's decoding system [13] focus on improving the recognition of ligatures and other touching characters by using a combination of lexicon, font database, and n-gram statistics. The system is shown to be robust to clustering mistakes and able to correctly decode most ligatures on the two images used for testing.

3 The Model

In our approach, binary images of machine printed text are taken as inputs. Within the image, each ink blot (i.e., connected component) is identified and an effort is made to identify characters composed of multiple ink blots, such as those with accent symbols and the letters `i` and `j`. An object defined in this manner can correspond to (1) exactly one character or punctuation mark, (2) part of a character that is broken into several pieces due to noise, or (3) multiple characters such as the ligatures `fi` and `ffl`. These objects are next clustered using greedy agglomerative clustering, so that the input document is represented by a string of cluster assignments in place of the actual characters. By examining the patterns of repetitions of cluster IDs and comparing them to the patterns of dictionary words, we can decode the the mapping between cluster IDs and characters in the output alphabet. In the rest of this section, we describe each step in detail.

3.1 Character Clustering

A crucial step in clustering is picking an appropriate similarity or distance measure. Two measures that intuitively capture the distance between two binary images A and B are the Hamming distance and the Hausdorff distance. The *Hamming distance* is simply the number of pixels on which A and B differ. It is fast and easy to calculate, but it is not robust to noise. For example, two images of the letter I may have larger Hamming distance than one of them to the letter T because of the difference in the thickness of pen strokes. An alternative distance measure is the *Hausdorff distance* [25, 34] defined as

$$h(A, B) = \max_{a \in A} \min_{b \in B} d(a, b),$$

where d is any metric, such as the Euclidean distance. In words, if the Hausdorff distance from A to B is δ, then for every point $a \in A$, there is a point in B within distance δ as measured by d.

To reduce the effects of noisy pixels on the distance, we "soften" the Hausdorff distance such that $h_p(A, B) = \delta$ means that for at least p percent of the points $a \in A$, there is a point in B within distance δ. Note that h_p is asymmetric and so is not a proper metric, but we can define a symmetric measure by using the mean of $h_p(A, B)$ and $h_p(B, A)$. In our experiments, we use this average with $p = 95$ as the distance measure.

The Hausdorff measure is more robust than the Hamming measure, but is expensive to compute for the $O(n^2)$ pairwise distances, where n is the number of images. One solution we use that takes advantage of the speed of the Hamming distance and the robustness of Hausdorff distance is the canopy method devised by McCallum et al [29]. First, the Hamming distance is computed for all pairs of images, and two distance thresholds T_1 and T_2 are specified, where $T_1 > T_2$. Next, we go through the list of images in any order and remove one image from the list to serve as the seed of a new canopy. All images in the list within distance T_1 of the seed image are placed into the new canopy, and all images within distance T_2 are removed from the list. This process is repeated until the list is empty. The more expensive Hausdorff measure is then used for pairwise distances within each canopy.

After all pairwise distances have been computed, the images are partitioned using hierarchical agglomerative clustering, a greedy bottom-up algorithm that gives a dendrogram: it starts

by assigning each image to its own cluster, then repeatedly combines the two clusters that are most similar until all data points belong to a single cluster. We compute the similarity between two clusters by the group average. I.e., the distance between clusters G_1 and G_2 is given by $d(G_1, G_2) = \frac{1}{|G_1| \cdot |G_2|} \sum_{A \in G_1} \sum_{B \in G_2} h(A, B)$. To choose the final number of clusters, we use the elbow criterion described in the experiments section.

3.2 Character Decoding

We start this section with some examples to demonstrate our idea. Consider the following string of characters:

$$\alpha\ \beta\ \gamma\ \gamma\ \beta\ \gamma\ \gamma\ \beta\ \delta\ \delta\ \beta,$$

where each Greek letter corresponds to an English alphabet letter. Given that the string stands for an English word, which word is it? After some thought, it should be clear that it is the word "Mississippi," since no other English word has that particular pattern of letters.

For each word represented as a string of cluster assignments, we compute its *numerization string* by going from left to right, assigning 1 to the first cluster ID, 2 to the second distinct cluster ID, 3 to the third distinct cluster ID, etc. For the above string, suppose the cluster assignments are

$$7\ 3\ 20\ 20\ 3\ 20\ 20\ 3\ 17\ 17\ 3,$$

then its corresponding numerization string is

$$1\ 2\ 3\ 3\ 2\ 3\ 3\ 2\ 4\ 4\ 2.$$

By computing the numerization strings for every word in the document and dictionary, we identify code words in the document that map to a unique dictionary word or is shared by a small number dictionary words. In this way, an initial mapping between cluster IDs and output characters can be made.

Formally, let $E = (e_1, e_2, ..., e_n)$ be the sequence of words encoded by cluster assignments, $C = \{c_i\}$ be the set of cluster IDs, and $\Sigma = \{\alpha_j\}$ be the alphabet of the target language. Our goal is to compute the set of assignments that maximizes $P(\{c_i = \alpha_j\}|E)$. By considering one mapping at a time, we write

$$\begin{aligned}
P(c_i = \alpha_j|E) &= \frac{P(E|c_i = \alpha_j)P(c_i = \alpha_j)}{P(E)} \\
&\propto P(E|c_i = \alpha_j)P(c_i = \alpha_j) \\
&\propto P(e_1, e_2, ..., e_n|c_i = \alpha_j) \\
&\approx \prod_{k=1}^{n} P(e_k|c_i = \alpha_j) \\
&= \prod_{k=1}^{n} \frac{P(c_i = \alpha_j|e_k)P(e_k)}{P(c_i = \alpha_j)} \\
&\propto \prod_{k=1}^{n} P(c_i = \alpha_j|e_k),
\end{aligned}$$

where we have applied the naive Bayes assumption, used Bayes' rule, and assumed a uniform prior for $P(c_i = \alpha_j)$.

The quantity $P(c_i = \alpha_j|e_k)$ is calculated by normalizing the count of the number of times cluster ID c_i maps to output letter α_j among the dictionary words that have the same numerization string as e_k. We used Laplace smoothing with $\lambda = 0.001$ to avoid zero probabilities.

Once $P(c_i = \alpha_j|E)$ has been calculated for every c_i and α_j, each cluster c_i is mapped to character $argmax_{\alpha_j} P(c_i = \alpha_j|E)$. Not all assignments will be correct at this point, because of words whose numerization strings don't have much discriminating power. We solve this problem by using the set of mappings of which we are confident to infer the less confident ones.

3.3 Confidence Estimation

An intuitive way to measure the confidence of an assignment for c_i is to look at the shape of the distribution $P(c_i = \cdot|E)$. If it sharply peaks at α_j and is near zero elsewhere, it indicates a confident assignment; if it is uniformly distributed, then the assignment is equivocal. The measure of *entropy* from information theory [33] quantifies this intuition. That is, for every cluster ID c_i, we calculate the entropy of its assignment by

$$H(c_i) = -\sum_{\alpha_j \in \Sigma} P(c_i = \alpha_j|E) \log(P(c_i = \alpha_j|E)).$$

Sorting the entropies in ascending order gives a list of c_i's whose assignments are in decreasing confidence. Recall that each code word e_k is associated with a list of dictionary words D_k that have the same numerization string. In general, some dictionary words in D_k are incompatible with the mode of $P(c_i = \cdot|E)$. Our refinement strategy is to iterate the c_i's as sorted by entropy, assume the mapping of $c_i = argmax_{\alpha_j} P(c_i = \alpha_j|E)$ to be true, and for each code word that contains c_i, remove from its list of dictionary words those words that are incompatible with the assumed assignment. After each iteration, the assignment probabilities and entropies of unprocessed c_i's are recomputed using the reduced lists of words.

To illustrate this algorithm, let's continue with the example given above, and suppose later on in the same document there is a code word 20 6 1 3 7 17. A priori, because there are many of English words with the same numerization string (1 2 3 4 5 6), there is little hope of decoding the this word. But since the word "Mississippi" has been decoded with high confidence, we can narrow down the list of compatible words to "shrimp", "scrimp", and perhaps just a few others.

3.4 Identifying Ligatures and Partial Mappings

The decoding procedure described above assumes each cluster ID maps to one output character. However, because some clusters contain ligatures and partial characters, additional steps are needed to correctly identify those mappings. To (partially) deal with over-segmentation, prior to the decoding steps described above, we count the number of of times each subsequence of cluster IDs appears in the document. Next, the subsequences that contain only c_i's that appear in no other subsequences are replaced by a single new cluster ID. To correct mapping errors that persist after the decoding step, we considered two refinement strategies, one based on an n-gram character model, and another on string edit distance. In both methods, the output alphabet is conceptually modified to $\Sigma' = \Sigma^*$, the set of strings made of zero or more letters from Σ.

3.4.1 N-gram Based Refinement

The first solution is based on a trigram character language model. That is, we compute the likelihood of a string of characters $w_1 w_2 ... w_m$ by

$$P(w_1 w_2 ... w_m|\theta) = \prod_{i=3}^{m} P_\theta(w_i|w_{i-2} w_{i-1}),$$

where θ is a table containing the trigram probabilities precomputed from some corpus in the same language. In our experiments, θ is calculated from the Reuters news corpus [1].

Our use of an n-gram model is motivated by the intuition that the correct mapping between clusters and output characters optimizes some measure of compatibility κ between the decoding and θ, and that the search for the correct mapping can be guided stepwise by improving values of κ. As widely practiced, we define κ to be the perplexity [27] of of the decoding $W = w_1 w_2 ... w_m$ under the trigram language model given by θ:

$$\kappa(W, \theta) = 2^{-\frac{1}{|C|} \sum_{i=3}^{m} \log_2(P_\theta(w_i|w_{i-2} w_{i-1}))},$$

[1] http://www.daviddlewis.com/resources/testcollections/reuters21578/

where $|C|$ is the number of clusters. The perplexity can be thought of as the average size of the set of characters from which each w_i is chosen given $w_{i-2}w_{i-1}$, so a lower perplexity indicates a better fit to the model.

Note that the factor $1/|C|$ is used instead of the usual $1/m$ because the length m of the decoding changes along with the mapping. If the quantity $1/m$ were used, κ is minimized by mapping every cluster ID to ϵ, the empty string.

Starting with the assignments $\mathcal{A} = (c_i = \alpha_{j_i})_{i=1}^{|C|}$ given by the numerization string algorithm, we hope to obtain a lower value for κ by perturbing $\mathcal{A}$ one c_i at a time. Stochastic algorithms such as simulated annealing and Gibbs sampling [5, 18] can be used to search for better assignments, and it is straightforward to efficiently update the posteriors as assignments change.

Unfortunately, our unsupervised approach to the problem is not amenable to such such iterative refinements, because the ground truth text typically has higher perplexity than the decoding given by the numerization string algorithm. This is due to the fact that ground truth text uses more character types (upper case letters, digits, and punctuations) than the decoding given by $\mathcal{A}$. Even when the n-gram statistics are collected from a corpus containing only lowercase letters and the ground truth text also contains only lowercase letters, we found this remains true. This may be caused by the relatively short document lengths and domain specificity, where general n-gram statistics are insufficient for decoding the cryptograms [14]. In our experiments section, we report only results given by an alternative refinement strategy based on string edit distances.

3.4.2 String Edit Distance Based Refinement

We begin this section with a motivational example. Suppose we are given the partially decoded words

```
?ost
fri?tens
enou?
```

where `?` denotes the same cluster ID that needs to be deciphered. Recall that each cluster maps to an element of Σ', not necessarily to a single character. The first word alone does not give much information, since it can be `cost`, `post`, and `almost`, among others. From the second and third words, it becomes clear that the question mark stands for the letters `gh`. Essentially, this puzzle is solved by a knowledge of the English lexicon and a mental search for words that are most similar to those partial decodings.

The notion of similarity between two strings $s = s_1 s_2 ... s_n$ and $t = t_1 t_2 ... t_m$ can be measured by the Levenshtein distance, which is the number of character insertions, deletions, and substitutions required to make s and t identical [24]. Let $ed(i, j)$ be the edit distance between $s_1 ... s_i$ and $t_1 ... t_j$, which can be computed recursively using dynamic programming:

$$ed(i, 0) = i$$

$$ed(0, j) = j$$

$$ed(i, j) = min \begin{cases} ed(i, j-1) + 1 \\ ed(i-1, j) + 1 \\ ed(i-1, j-1) + I(i \neq j) \end{cases}$$

where $I(\cdot)$ is an indicator function.

The first step in this strategy is to identify the set $\tilde{C} \subset C$ of clusters that are candidates for correction. Our initial definition of $\tilde{C}$ is the set of cluster IDs appearing *only* in non-dictionary words, but this criterion misses those clusters appearing in decoded words that happen to be in the dictionary by accident. Instead, we define $\tilde{C}$ to be the set of clusters that occur more frequently in non-dictionary words than in dictionary words, where frequency is measured by the normalized character count.

aegean	aluvic
bernoulli	dlr
exxon	fluoroscan
multilaterally	zinn

Table 1: Some correctly deciphered non-dictionary words from the ASCII code data.

For every decoded word w_i that contains an element of $\tilde{C}$, we find the dictionary word that is closest to it in edit distance and tally the edit operations that involve elements of $\tilde{C}$. If w_i happens to be in the dictionary, we count the identity mappings that involve elements of $\tilde{C}$. To avoid having to calculate the edit distance of w_i to every dictionary word, we prune the list of dictionary words by computing the ratio $r(w_i, d_j) = \frac{comm(w_i,d_j)}{max(|w_i|,|d_j|)}$ for every dictionary word d_j, where $comm(w_i, d_j)$ is the number of (non-unique) character trigrams w_i and d_j have in common [21]. Let $d(w_i) = argmax_{d_j \in D} r(w_i, d_j)$, which can be found efficiently by using an inverted index of character trigrams. Next, only the string edit operations between w_i and $d(w_i)$ need to be tallied. In the case that multiple dictionary words share the same maximum ratio with w_i, the edit operations of w_i are ignored, because in our experience, using such words skews the edit counts toward commonly occurring letters such as `e`. After the edit counts have been tabulated, each cluster ID in $\tilde{C}$ is re-mapped to the string it most frequently edits to.

4 Experiments

In this section, we describe the data sets and results from our experiments. The lexicon used comes from the Spell Checker Oriented Word Lists [2] and contains 10683 words.

4.1 Artificial Data

Artificially generated data provides a sanity check for the performance of the decoding algorithm under optimal input conditions and allows us to examine the robustness of the algorithm by varying the amount of noise present. We use two types of artificial data in our experiments, one to simulate perfect character segmentation and clustering, and another that more closely resembles conditions for real-world image data.

4.1.1 ASCII Codes

The best-case scenario for the decoding algorithm is when (1) there is a bijective mapping between clusters and the output alphabet Σ, and (2) the alphabet of the lexicon used by the decoder equals Σ. To simulate this condition, we clean data from the Reuters corpus by removing all numerals and punctuation marks, and lowercasing all remaining letters. The three hundred files with the most words after preprocessing selected, and the ASCII codes of the text is given to the decoder. The number of words in these files range from 452 to 1046. Table 3 shows the performance of the algorithm, and Table 1 lists some correctly decoded words that are not in the dictionary. Most errors involve mislabeling the letters `j` and `z`, which make up 0.18% and 0.07% of the characters, respectively. In comparison, the letter `e`, which comprises 9.6% of the characters, was recalled 100% of the time.

4.1.2 Leetspeak

Leetspeak (or Leet) is a form of slang used in Internet chat rooms and forums that involves the substitution of letters with similar looking numerals (e.g., `3` for `e`), punctuation marks (e.g., `|-|` for `h`), or similar sounding letters (e.g., `ph` for `f`). In addition, letter substitutions may vary from one

[2] http://wordlist.sourceforge.net/

a	@, 4	h	}{, #	o	0	v	\/
b	8, b	i	!, i	p	9, |o	w	w
c	c, [	j	j	q	q, 0_	x	><, x
d	d	k	k	r	r, 2	y	y
e	3, e	l	1, l	s	$, z	z	%
f	ph	m	(V), m	t	7, t		
g	g, 6	n	n	u	u, v		

Table 2: Character substitutions for our Leetspeak data. When a character has multiple mappings, the substitution is picked uniformly at random.

```
gold is expected to continue its rise this year due to renewed
inflationary pressures especially in the us

g01d !$ ex|oect3d t0 [0n7!nve i7z ri$e 7#!$ y3@2 due t0 r3new3d
!nphl@t!0n@2y |orezzur3z ez9eci4l1y in t#e uz
```

Figure 1: A snippet of a Reuters news story translated into Leet.

word to the next, so that the letter `s` may be written as `$` in one word and `5` in the next. As an example, the word `Leetspeak` itself may be written as `!337$p34k`.

Understanding Leet requires resolving some of the same issues as the character recognition task. More than one character in Leet can be used to represent the same alphabet letter, which mirrors the problem of split clusters. Multiple Leet characters can be used to represent the same alphabet letter, and this mirrors the problem of over-segmentation of character images.

Table 2 shows the substitutions we made to each letter, and Figure 1 gives a partial Reuters news story translated into Leet. Note that the substitutions are defined such that no two original letters share any characters in their mappings. This is done only as a simplification of the problem, since Leetspeak can be much more involved than what is presented here. We ran the decoding algorithm on the same 300 Reuters stories encoded in Leet, and Table 3 gives the character and word accuracies. The decoding performance on Leet is just as good as on the ASCII data with similar types of errors, so our algorithm seems to be robust to multiple representations of the same character and split characters.

4.2 Document Image Data

We evaluated our program on two sets of document images. The first one consists of 201 Reuters news stories preprocessed in the manner described above and then rendered in unusual font styles (see Figure 2). These images are clean but do contain ligatures. The second set of images comes from the OCR data set of the Information Science Research Institute at UNLV [32], which includes manually-keyed ground truths and segmentations of pages into text zones. From a collection of Department of Energy reports in the UNLV data set that were scanned as bi-tonal images at 300 dpi, we selected 314 text zones that are primarily text (excluding zones that contain tables or math formulas) for recognition.

4.2.1 Image Rectification

Many of the scanned images are slanted, where lines of text are not parallel to the top and bottom edges of the image. Although the clustering step is able to deal with slanted character images, rectification is performed to facilitate identifying the reading order and inter-word spacing for decryption.

Our rectification algorithm is based on an entropy measure of ink distributions. For each horizontal line of pixels in the image, we count the number of pixels occupied by ink, so that a projection profile of the image obtained as in [22] and [17]. The idea is that the ideal rotation of the image gives a projection profile with the highest entropy. We search for the best rotation angle from the

	ASCII	Leetspeak
character accuracy	99.80	99.65
word accuracy	98.84	98.06

Table 3: Decoding performance on 300 Reuters news stories encoded in ASCII code and Leetspeak.

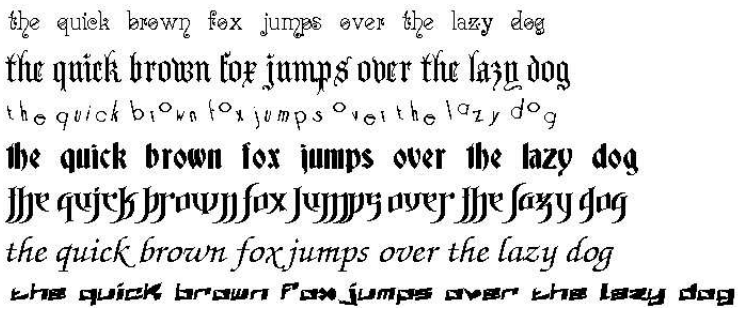


Figure 2: Samples of the unusual fonts used to create document images of Reuters stories.

range of $[-10°, 10°]$ in $1°$ increments. To speed up this process, the image is initially rotated $1°$, then $-1°$, and only the direction that increases the entropy is taken. The search stops as soon as the entropy decreases, as it indicates over-rotation.

After rectification, the image is despeckled by removing isolated single-pixel ink blots. Each connected component is extracted and resized to fit within a 60 x 60 pixel image centered at its centroid. To cluster the images, pairwise distances are computed by shifting one of the images around a 3 x 3 window and taking the smallest Hausdorff distance.

4.2.2 Determining Word Demarcations

Our decoding algorithm relies on accurate segmentation of the sequences of cluster IDs into word units, so a principled method is needed to identify word demarcations. Figure 3 shows a typical histogram for horizontal spacing between adjacent connected components on an image, where the left hump corresponds to spaces within a word, and the right hump spaces between two words. We model such histograms as mixtures of two Poisson distributions, one for intra-word spaces and another for inter-word spaces. The model is optimized by gradient ascent to find a threshold c above which a horizontal spacing constitutes a word break.

Formally, the probability of a particular spacing s_i is defined by

$$\begin{aligned} P(s_i|c,\lambda_1,\lambda_2) &= P(s_i \in P_1|c)P_1(s_i|\lambda_1) + P(s_i \in P_2|c)P_2(s_i|\lambda_2) \\ &= P(s_i \in P_1|c)P_1(s_i|\lambda_1) + (1 - P(s_i \in P_1|c))P_2(s_i|\lambda_2) \\ &= I(s_i > c)P_1(s_i|\lambda_1) + (1 - I(s_i > c))P_2(s_i|\lambda_2), \end{aligned}$$

where I is the indicator function, and P_j $(j = 1, 2)$ are Poisson distributions:

$$P_j(s_i|\lambda_j) = \frac{e^{-\lambda_j}\lambda_j^{s_i}}{s_i!}.$$

Given the list of spaces $(s_1, ..., s_N)$, the objective function is simply defined by the likelihood of the data:

$$\Omega(c,\lambda_1,\lambda_2) = \prod_{i=1}^{N} P(s_i|c,\lambda_1,\lambda_2).$$

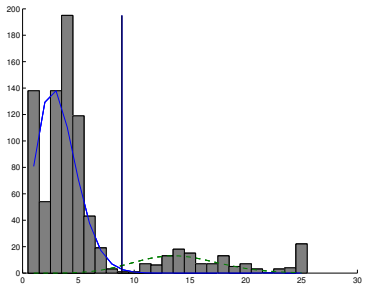


Figure 3: A typical histogram of horizontal spaces in an image. The x-axis is the gap size measured in pixels, and the y-axis is the count. The solid and dashed curves are the two Poisson distributions fitted by gradient ascent, and the vertical line indicates the threshold c.

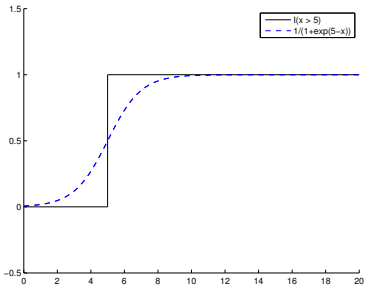


Figure 4: Sigmoid approximation to the indicator function $I(x > 5)$. The indicator function is softened to facilitate the optimization procedure for finding word demarcations.

The goal is to find the parameters $\theta = (c, \lambda_1, \lambda_2)$ that maximize Ω. One technique for doing so is gradient ascent, where θ is initialized to a random point θ_0, and at iteration $t+1$ it is updated by $\theta_{t+1} \leftarrow \theta_t + \rho \nabla_\theta \Omega(\theta_t)$, where ρ is the learning rate and $\nabla_\theta \Omega$ is the gradient of Ω. The learning rate ρ is adapted using the bold driver algorithm, and the search continues until the objective function does not improve much from the previous iteration.

The indicator function is discontinuous so is not everywhere differentiable, thus complicating the optimization routine. We avoid this problem by approximating I by a shifted sigmoid function (see Figure 4): $I(s_i > c) \approx \frac{1}{1+e^{c-s_i}}$.

4.2.3 Choosing the Number of Clusters

A fundamental issue in data clustering is choosing the number of partitions for the data. The heuristic we use is based on the "elbow criterion." In each step of agglomerative clustering, the distance between the two clusters to merge is plotted, giving a curve that resembles the exponential function (see Figure 5). The number of clusters to form is then be derived from a point c where the slope of the curve begins increasing faster than some threshold value τ. In our experiments, τ is set to 0.005.

4.2.4 Document Image Results

Figure 6 shows the histograms of character accuracies on the Reuters and UNLV test images. On the UNLV images, the mean accuracy of word demarcations, averaged over the number of images, is 95.44%. Althogh this figure initially looks promising, images with very low accuracies are caused by unrecoverable errors in word segmentation. Our decoding algorithm also misses all digits, punctuation marks, and uppercase letters.

5 Discussion

A major weakness in our unsupervised approach, similar to the results presented in [15], is its inability to recognize numerals, punctuation marks, and uppercase letters. Using image-to-character classifiers to identify these special characters beforehand proves beneficial, as discussed in [16]. To this end, we have done preliminary experiments that combine the scores from cryptogram decoding with outputs from a robust maximum-entropy character classifier used by Weinman and Learned-Miller [37]. The results are noticeably better, with mean character accuracy in the mid 80% range

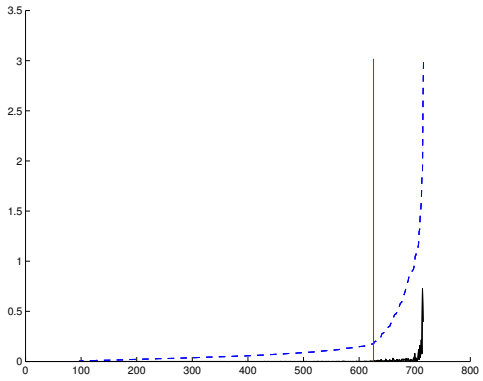
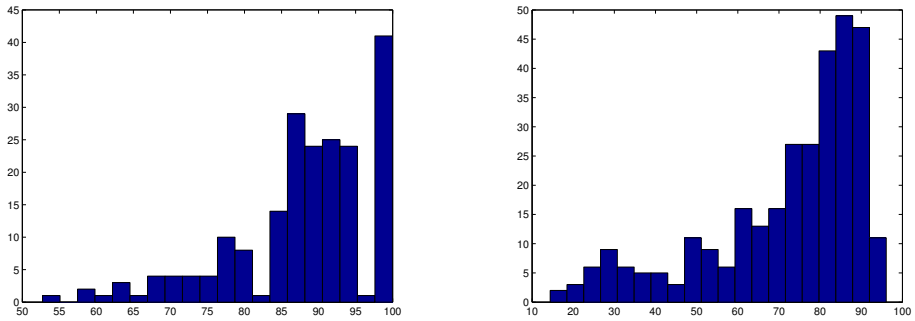


Figure 5: The dashed curve corresponds to the distance between the merged objects in each step of agglomerative clustering. The solid curve is its derivative, and the vertical line indicates the cutoff point c where the derivative first becomes greater than τ. There were 716 objects on the document image, and $c = 626$, so the final number of clusters is $k = 90$.



(a) (b)

Figure 6: Histograms of character accuracies. (a) For 201 Reuters stories rendered in unusual fonts. Averaged over the number of images, mean accuracy is 88.09%. (b) For 314 Department of Energy documents. Averaged over the number of images, mean accuracy is 73.78%. Limiting evaluation to lowercase characters gives a mean accuracy of 78.85%.

for the UNLV data set, although there remain many simple steps that can improve accuracy but are not yet implemented, such as using character heights to distinguish between lowercase and uppercase letters.

In our system that combines scores from the decoder and classifier, the most common error comes from the segmentation stage, as the segmentation scheme used was fairly simple and both sides require proper character segmentation as a preprocessing step. A conservative estimate shows it accounts for at least 7% of the errors. An extensive treatment of alternative segmentation techniques used by OCR systems can be found in [7]. To be robust to ink smears and fractured characters, a segmentation-free approach such as [12] and as done in many handwritten recognition systems [1, 4, 9, 10, 20, 26, 28], can be used, but for our system to remain font-independent we need to find the alphabet of the document automatically, perhaps by using strategies developed for document compression such as the Lempel-Ziv algorithm.

More immediately correctable of the common errors are those that confuse characters with similar appearances, such as upper- and lowercase `s` and `c`, and characters such as `l`, `I`, and `1`. This type of errors can be corrected by imposing language constraints or use a spell checker in a post-processing

step. We estimate that this problem accounts for around 8% of the errors. There have been several lines of work on OCR error correction. Taghva and Stofsky [35] devise an interactive correction system that ranks candidate replacements by n-gram statistics and longest common subsequences. In [36], an automatic correction system trained on word bigram and character n-gram statistics obtained from the same domain as the test data is shown to achieve large error reductions. Kolak and Resnik's OCR error correction [19] is based on a noisy channel model trained on pairs of ground truth and OCR output words, and compares favorably against unsupervised string edit corrections.

6 Conclusion

We presented an unsupervised OCR system using character clustering with canopies and a cryptogram decoding algorithm based on numerization strings. Its performance was evaluated on artificial and real data. Under ideal input conditions, where both character segmentation and clustering are correct, our decoding algorithm can correctly decode almost all words, even those absent from the lexicon. Although not sufficient in and of itself, our decoding approach, when augmented with appearance models, can improve recognition performance in a complete OCR system.

Acknowledgements

We would like to thank Jerod Weinman for his insightful discussions and code for training character models.

References

[1] Flvio Bortolozzi, Alceu de Souza Britto Jr., Luiz S. Oliveira, and Marisa Morita. Recent advances in handwriting recognition. In *Proceedings of the International Workshop on Document Analysis*, pages 1–30, 2005.

[2] Thomas M. Breuel. Classification by probabilistic clustering, 2001.

[3] Eric Brill and Robert C. Moore. An improved error model for noisy channel spelling correction. In *Association for Computational Linguistics*, 2000.

[4] Horst Bunke, M. Roth, and Ernst Gnter Schukat-Talamazzini. Off-line cursive handwriting recognition using hidden markov models. *Pattern Recognition*, 28(9):1399–1413, 1995.

[5] George Casella and Edward I. George. Explaining the gibbs sampler. *The American Statistician*, 46:167–174, 1992.

[6] Richard .G. Casey. Text OCR by solving a cryptogram. In *International Conference on Pattern Recognition*, volume 86, pages 349–351, 1986.

[7] Richard G. Casey and Eric Lecolinet. A survey of methods and strategies in character segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):690–706, 1996.

[8] Shou de Lin and Kevin Knight. Discovering the linear writing order of a two-dimensional ancient hieroglyphic script. *Artificial Intelligence*, 170:to appear, 2006.

[9] Alceu de S. Britto Jr. A two-stage hmm-based system for recognizing handwritten numeral strings. In *ICDAR '01: Proceedings of the Sixth International Conference on Document Analysis and Recognition*, page 396, Washington, DC, USA, 2001. IEEE Computer Society.

[10] Michael Decerbo, Ehry MacRostie, and Premkumar Natarajan. The BBN Byblos Pashto OCR system. In *HDP '04: Proceedings of the 1st ACM workshop on Hardcopy document processing*, pages 29–32, New York, NY, USA, 2004. ACM Press.

[11] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

[12] Jaety Edwards and David A. Forsyth. Searching for character models. In *NIPS*, 2005.

[13] Chi Fang and Jonathan J. Hull. A modified character-level deciphering algorithm for OCR in degraded documents. In *SPIE Conference on Document Recognition II*, 1995.

[14] George W. Hart. To decode short cryptograms. *Communications of the ACM*, 37:102–108, 1994.

[15] Tin Kam Ho and George Nagy. OCR with no shape training. In *International Conference on Pattern Recognition*, 2000.

[16] Tin Kam Ho and George Nagy. Identification of case, digits and special symbols using a context window, 2001.

[17] Rajiv Kapoor, Deepak Bagai, and T. S. Kamal. A new algorithm for skew detection and correction. *Pattern Recogn. Lett.*, 25(11):1215–1229, 2004.

[18] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[19] Okan Kolak and Philip Resnik. OCR error correction using a noisy channel model. In *Human Language Technology Conference*, 2002.

[20] Andras Kornai. Experimental HMM-based postal OCR system. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1997.

[21] Karen Kukich. Technique for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[22] Kevin Laven, Scott Leishman, and Sam Roweis. A statistical learning approach to document image analysis. In *8th International Conference on Document Analysis and Recognition*, 2005.

[23] Dar-Shyang Lee. Substitution deciphering based on hmms with applications to compressed document processing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(12):1661–1666, 2002.

[24] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, 163:845–848, 1965.

[25] Dimitri A. Lisin, Marwan A. Mattar, Matthew B. Blaschko, Mark C. Benfield, and Erik G. Learned-Miller. Combining local and global image features for object class recognition. In *Proceedings of the IEEE Workshop on Learning in Computer Vision and Pattern Recognition*, June 2005.

[26] Zhidong Lu, Issam Bazzi, Andrs Kornai, John Makhoul, Premkumar Natarajan, and Richard Schwartz. A robust, language-independent OCR system. In *Proceedings of the 27th AIPR Workshop: Advances in Computer-Assisted Recognition*, 1999.

[27] Christopher D. Manning and Hinrich Schtze. *Statistical Natural Language Processing*. MIT Press, 1999.

[28] Urs-Viktor Marti and Horst Bunke. Handwritten sentence recognition. In *15 th International Conference on Pattern Recognition*, 2000.

[29] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178, New York, NY, USA, 2000. ACM Press.

[30] George Nagy. Efficient algorithms to decode substitution ciphers with applications to OCR. In *Proceedings of International Conference on Pattern Recognition, ICPR 8*, 1986.

[31] George Nagy. At the frontiers of OCR. In *Proceedings of the IEEE*, 1992.

[32] Thomas A. Nartker, Stephen V. Rice, and Steven E. Lumos. Software tools and test data for research and testing of page-reading OCR systems. In *International Symposium on Electronic Imaging Science and Technology*, 2005.

[33] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:623–656, 1948.

[34] Michael D. Shapiro and Matthew B. Blaschko. Stability of hausdorff-based distance measures. In *Visualization, Imaging, and Image Processing*, 2004.

[35] Kazem Taghva and Eric Stofsky. OCRSpell: an interactive spelling correction system for ocr errors in text. *IJDAR*, 3:125–137, 2001.

[36] Xian Tong and David A. Evans. A statistical approach to automatic OCR error correction in context. In *Proceedings of the Fourth Workshop on Very Large Corpora*, 1996.

[37] Jerod Weinman and Erik Learned-Miller. Improving recognition of novel input with similarity. In *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.