Cryptogram Decoding for Optical Character Recognition

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

Abstract

OCR systems for printed documents typically require large numbers of font styles and character models to work well. When given an unseen font, performance degrades 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 and Related Work

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. 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.

Treating OCR as a cryptogram decoding problem dates back at least to papers by Nagy [11] and Casey [2] in 1986. There continues to be research done to improve the performance of approaches that use no character models.

In [3], 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 [3] is our use of probabilistic reasoning instead of predefined ratio thresholds. In [8], 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 [1] introduced a supervised OCR system that is font independent, but it does not take advantage of token-based image compression.

2 The Model

We take binary images of machine printed text as inputs. Within an 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 patterns of dictionary words, we can decode the mapping between cluster IDs and characters in the output alphabet. In the rest of this section, we describe each step in detail.

2.1 Character Clustering

Two straightforward measures of 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 or minor variations in stroke thickness. *Hausdorff distance* [9] 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. 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.

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. To make the Hausdorff measure symmetric, we take the mean of $h_p(A,B)$ and $h_p(B,A)$. In our experiments, we use this average with $p = 95$.

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. We take advantage of the speed of the Hamming distance and the robustness of Hausdorff distance by using the canopy method devised by McCallum et al [10]. 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. Inter-cluster similarity is computed 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.

2.2 Character Decoding

Consider the following word encoding:

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

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 document and dictionary word, we identify code words in the document that map to a unique dictionary word or are 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.

2.3 Confidence Estimation

An intuitive way to measure the confidence of an assignment for c_i is to look at how peaky the distribution $P(c_i = \cdot|E)$ is. *Entropy* quantifies this measure. For every cluster ID c_i, the entropy of its assignment is

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

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.

2.4 Ligatures and Partial Mappings

The decoding procedure described above assumes each cluster ID maps to one output character. However, some clusters contain ligatures and partial characters. 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 use a refinement strategy based on string-edit distances. The output alphabet is conceptually modified to $\Sigma' = \Sigma^*$, the set of strings made of zero or more letters from Σ.

We begin with an 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 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.

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 [6]. 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.

3 Experiments and Analysis

We performed experiments on artificial and real data. We used the Spell Checker Oriented Word Lists (http://wordlist.sourceforge.net/), which contains 10,683 words, as a lexicon.

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 to more closely resemble conditions of real-world image data.

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 2 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.

Leetspeak (or Leet) is a form of slang used in Internet chat rooms and forums that involves the substitution of letters by 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 word to the next, so that the letter `s` may be written as `$`

aegean	aluvic
bernoulli	dlr
exxon	fluoroscan
multilaterally	zinn

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

	ASCII	Leetspeak
character accuracy	99.80	99.65
word accuracy	98.84	98.06

Table 2. Decoding performance on 300 news stories encoded in ASCII and Leetspeak.

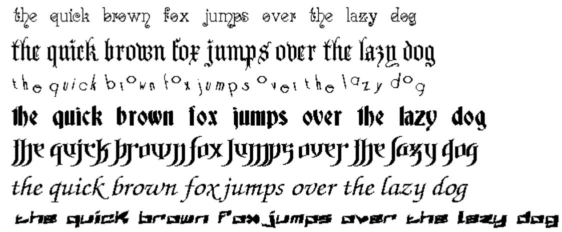


Figure 1. Samples of unusual fonts used to create document images of Reuters stories.

in one word and `5` in the next. As an example, the word `Leetspeak` itself may be written as `!337$p34k`. An example sentence in the Reuters story translated to Leetspeak is

```
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
```

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

Understanding Leetspeak requires resolving some of the same issues as the character recognition task. More than one character in Leetspeak 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.

To generate Leetspeak data to test our decoding algorithm, we defined the substitutions 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 complex than what is presented here. We ran the decoding algorithm on the same 300 Reuters stories encoded in Leet, and Table 2 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.

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 1). 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 [12], 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.

Many of the images are slanted, where lines of text are not parallel to the top and bottom edges of the image. Although clustering can deal with slanted character images, rectification makes it easier to determine the reading order and inter-word spacing needed 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 [7] and [5]. We simply search for the rotation, in 1° increments, that minimizes the projected entropy.

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.

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 2 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:

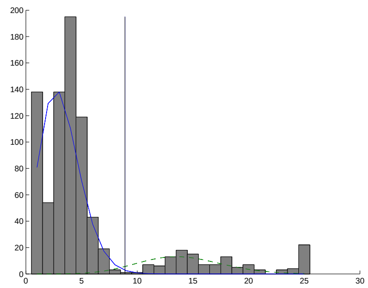
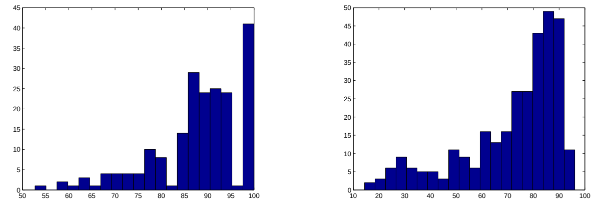


Figure 2. A typical histogram of horizontal spaces in an image. The x-axis is the gap size in pixels, and the y-axis is the count. The solid and dashed curves are the two Poisson distributions fitted by gradient ascent, and the vertical line indicates the threshold c.



(a) (b)

Figure 3. 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 Dept. 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%.

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

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

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

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: $I(s_i > c) \approx \frac{1}{1+e^{c-s_i}}$.

To choose the final number of clusters, we use the "elbow criterion" heurstic: In each step of agglomerative clustering, the distance between the two clusters to merge is plotted, giving a curve that resembles the exponential function. 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 manually set to 0.005.

Figure 3 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%. Although 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.

One shortcoming of our unsupervised approach, similar to the results presented in [3], 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 [4]. To this end, we plan to combine the scores from cryptogram decoding with outputs from a robust maximum-entropy character classifier used by Weinman and Learned-Miller [13].

We have 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 alone, our decoding approach, when augmented with appearance models, can improve recognition performance in a complete OCR system.

References

[1] T. Breuel. Classification by probabilistic clustering, 2001.

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

[3] T. K. Ho and G. Nagy. OCR with no shape training. In *International Conference on Pattern Recognition*, 2000.

[4] T. K. Ho and G. Nagy. Identification of case, digits and special symbols using a context window, 2001.

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

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

[7] K. Laven, S. Leishman, and S. Roweis. A statistical learning approach to document image analysis. In *8th International Conference on Document Analysis and Recognition*, 2005.

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

[9] D. A. Lisin, M. A. Mattar, M. B. Blaschko, M. C. Benfield, and E. 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.

[10] A. McCallum, K. Nigam, and L. 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, 2000.

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

[12] T. A. Nartker, S. V. Rice, and S. 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.

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