A System for Cursive Handwritten Address Recognition

Jianchang Mao
IBM Almaden Research Ctr
650 Harry Road
San Jose, CA 95120, USA
mao@almaden.ibm.com

Prasun Sinha
Dept. of Computer Science
Univ. of Illinois at UC
Urbana, IL 61801, USA
prasun@crhc.uiuc.edu

K. Mohiuddin
IBM Almaden Research Ctr
650 Harry Road
San Jose, CA 95120, USA
moidin@almaden.ibm.com

Abstract

This paper presents a cursive handwritten address recognition system, which consists of four main modules: (i) over-segmentor, (ii) dynamic zip locator, (iii) zip candidates generator, and (iv) city-state-zip verifier. The dynamic zip locator and city-state-zip verifier are based on a flexible matcher for matching a sequence of graphemes with a list of generalized strings. The dynamic zip locator is able to locate zip without knowing exactly where the zip starts. The zip candidates generator uses a hidden Markov model (HMM) with position-dependent state transition probabilities. A scheme for utilizing prefixes is designed to reduce computation and memory requirement. Finally, the system employs a mechanism for rejection based on rank features extracted from the matching. The overall system achieves an acceptance rate of 83.5% with 3.6% error for 5-digit encoding on 805 USPS cursive address images.

1. Introduction

Automatic mail sorting requires encoding the destination address on a mail piece into a unique mail delivery point code. Given an image of a mail piece, we first need to locate where the destination address block is. In this paper, we assume that the destination address block has been located. Zip code location deals with locating and extracting the zip code, if present in the address block. Zip code location and recognition is very crucial in an automatic address recognition system, because the rest of the recognition depends heavily on it.

Existing zip code location and extraction methods [4, 3] locate zip code by using some heuristics, such as aspect ratios of connected components, spaces between connected components and location of a continuous digit string based on preliminary OCR results. These methods may work well on machine-print address blocks, but are not very reliable on handwritten address blocks, especially those with cursive handwriting. In handwritten addresses, detecting space that separates words has been a difficult task because of irregular gaps (sometimes no gaps) between characters and words. The initial OCR results of the zip code do not necessarily produce 5 or 9 digits because of touching digits and broken digits. Errors in the initial OCR results (e.g., wrong splitting and merging) are often not recoverable.

This paper presents a system which employs a method for dynamically locating and recognizing zip codes in the address blocks on postal mail pieces. As opposed to existing methods for zip code recognition, the proposed method works directly on a sequence of graphemes without knowing the exact location of zip code. The syntax constraint is integrated in zip code location, digit extraction, and recognition under a single framework to avoid making premature decisions whose errors are often not recoverable. Other features of our system are also discussed in the paper.

2. System Overview

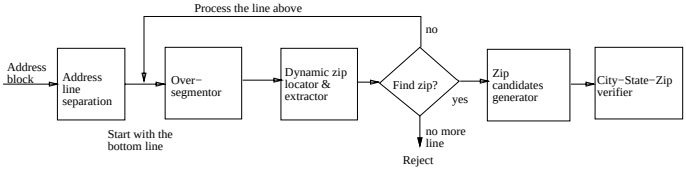

Figure 1. System diagram.

Figure 1 shows the diagram of our address recognition system. The input to the system is an address block. Address lines in the address block are first separated using a projection based approach which can handle a skew angle from -10 to $+10$ degree. Since the zip code is typically present in the bottom-most line of an address-block, our system processes the address-block line by line, starting from the bottom line until the zip code is located with a high confidence or the zip code is not found in the two bottom lines of the address-block. The processing of an address line involves two or four modules in the system, depending

on whether a zip is located in the line. After the digits of the zip code are extracted, a sorted list of possible zip codes are generated by the zip code candidate generator which uses a Hidden Markov Model with position-dependent transition probabilities and a numeric expert classifier. This list of possible zip codes are then verified by the city-state-zip verifier. A flexible matcher is proposed for matching a sequence of graphemes with a generalized string. The same matcher is used for both zip location and city-state verification.

3. Over-segmentation

Over-segmentation refers to a process of finding a superset of the perfect split points for a word or text line image. The idea is to detect all the split points at a cost of having false alarms. We will use the term *graphemes* (primitive structures of characters) for over-segments obtained after over-segmentation, and the term *segment* for a combination of graphemes. In our over-segmentor, a set of split points on word strokes is located by following the upper contour and lower contour of each connected component and looking for smooth or sharp valleys, horizontal stretches, and high curvature points. A set of heuristics, such as horizontal and vertical overlaps between neighboring graphemes, is used to screen the split points. After splitting, a character may consist of one, two or three graphemes.

4. A Flexible Matcher for Matching Graphemes with Generalized Strings

A generalized string can contain digits, alphabetic characters, special symbols, or *wild* characters. For example, city and state names can be represented as ordinary strings, e.g. $< SanJose >$. Zip syntax can be represented as $< \delta\delta\delta\delta\delta >$ for 5 digits, and $< \delta\delta\delta\delta\delta - \delta\delta\delta\delta >$ for 9 digits, where δ is a wild character for digits.

Given a sequence of N graphemes and a string of length W, the dynamic programming technique can be used for obtaining the best grouping of the N graphemes into W segments [1]. When a segment is matched to a symbol in the generalized string, we use the posterior probability of the segment being that symbol as the match score. If the symbol is a wild character, the probability is computed by summing up the probabilities over the individual characters represented by the wild character. The matching score of a path in the table is the product of these individual scores on the path.

As we will see in the next section, there is a need for matching a given string with the *best* subsequence of graphemes. Hence, zero-cost (or score=1) deletions at the beginning and end of a grapheme sequence should be allowed if desired. Our zip locator locates the zip by allowing zero-cost deletion at the beginning of the grapheme sequence without priorly knowing where the zip starts.

For city-state verification, about 29 city-state names on the average need to be matched to the sequence of graphemes. Often, many city names have the same prefixes, which means that the upper-left portion of their dynamic programming tables are the same for words with the same prefix.

5. Dynamic Zip Location

It is found that a zip code, if present on an address line can usually be located in the last ten graphemes of that address line (with a US address format). Hence, the last 10 graphemes of an address line are selected to match with the 5-digit zip syntax string. Zero-cost deletion at the beginning is allowed since we do not know exactly where the zip starts. Penalties are given to prevent splitting a segment into horizontally heavily overlapped pieces, or merging two graphemes which are separated by a gap. These graphemes can be part of the city name or state name preceding the zip code. An alpha-numeric neural network classifier with 108 input units, 80 hidden units, and 43 output units is used for generating posterior probabilities in the matching. The 43 output units correspond 43 classes (25 alphabets excluding the letter 'o', 10 digits, 7 special symbols, and 1 for non-character. The letter 'o' is merged into the same class as zero '0'). The inputs to the neural network are 108 features extracted from a segment [2].

We found the first and the second best paths through the dynamic programming table to be sufficient for covering the correct path. The address line is rejected if the matching score of the best path is below a pre-specified threshold, or the number of good (bad) digits on the best path is below (above) a pre-specified threshold. Based on this criterion, about 98% of test images are accepted. A zip code is said to be *correctly* located if and only if all the segments obtained by the zip locator and extractor align with the ones labeled manually.

Since we do not know *a priori* whether a 9-digit zip is present, we always look for 9-digit zip code as well as 5-digit zip code. The same dynamic programming approach is extended for locating the 9-digit zip code. The generalized string for 9-digit zip codes is $< \delta\delta\delta\delta\delta - \delta\delta\delta\delta >$, or $< \delta\delta\delta\delta\delta\delta\delta\delta\delta >$. The last 17 graphemes for an address line are matched to these generalized strings with zero-cost deletion at the beginning. The decision on whether a 5-digit zip or 9-digit zip is biased to 9-digit zip because if the address does contain a 9-digit zip then the 5-digit location usually also returns a good match value. If the address contains a 9-digit zip code then only the 5 segments corresponding to the first 5 digits in the 9-digit zip are used to generate a list of candidate zip codes.

6. Zip Candidate Generation

Given the 5 segments obtained by the zip locator, we use a HMM to generate a list of zip candidates. The HMM has

50 states, $S = \{S_{i,j}\}$, where $i = \{1..10\}$ (indicating the digit class) and $j = \{1..5\}$ (indicating the position of the 5 segments). State transitions are allowed only from $S_{i,j}$ to $S_{i,j+1}$. These state transition probabilities are computed from the database of zip codes. We found that the state transition probabilities are position-dependent (e.g., the probability of '0' followed by '0' at the first place is different from the one at the fourth place in 5-digit zip codes). The initial state probabilities are also pre-computed using the zip code database. Ideally, these probabilities should be computed from the distribution of the real mail stream in a particular mail sorter. A numeric expert neural network is used to approximate the symbol probabilities.

The problem of recognizing the zip code is equivalent to finding the best state sequence for the given observation sequence. A modified Viterbi algorithm is used to find the best n paths. We are currently using $n = 30$ and $n = 20$ for the best and the second best segmentation paths obtained by the zip locator.

To increase our coverage of the true zip code, we used confusion statistics generated by our numeric classifier. For every digit, we have a table of three most confusion digits. The top choice from the Viterbi algorithm was used to generate more candidate zip codes by replacing low confident digits with their confusing digits. The list of possible zip codes obtained using the above methods is then screened using the dictionary of zip codes. Our experiments show that the coverage rates of the correct zip being in the top n choices are between 85.37% and 95.73% for $n = 1, 2, \cdots, 20$.

7. City-State-Zip Verification

As a result of zip code generation, an ordered list of valid zip codes is obtained. A lexicon of city-state-zip names is created by retrieving city-state pairs by zips from the City-State-Zip database.

The city name usually starts on a separate line, which is followed by the state name and then followed by the zip code on the same line or the next line. Since the zip code has already been located and matched, based on the number of remaining graphemes on the zip line, we can reliably determine whether the city-state appears on a different line. If yes, we concatenate two lines into a single list of graphemes. We also append the 5 segments corresponding to the 5-digit zip at the end of the grapheme list. We found that this leads to a more reliable match than just using city and state. The flexible matcher is used for matching the list of graphemes with every string in the lexicon.

At this stage, the system needs to select only one choice. In order to keep the error rate low, the system has to decide when to accept the top choice. The reject criterion is mainly based on the rank features defined as follows. When we match a sequence of graphemes to a string, it is not necessary that each character in the string is on the top among all the possible character classes for the corresponding segment. Let us define the rank of a character in a string as the number of characters which had better matching scores with the segment than this character. A histogram of ranks with 5 unequal bins for all the characters in the city-state-zip string is created. A high contribution to the 1st and 2nd bins and low value in the last bin indicates a good match and is accepted.

The dynamic programming scheme was tested for city-state-zip level recognition using 805 cursive script images containing 5-digit zip codes from the USPS database. The average lexicon size of city-state-zip produced by the generator was 28.7 words. Each lexicon entry is expanded if necessary to cover different writing variations of the same city-state. Out of the 805 images, 672 (83.5%) were accepted by the mechanism for rejection. The error rate is 3.6%. Since these city-state-zip lexicons were obtained after zip code extraction, some of the lexicons may not contain the true city-state-zip.

8. Summary

We have described a cursive handwritten address recognition system. The system employs a flexible matcher for matching a sequence of graphemes with a list of generalized strings. This flexible matcher can be used for finding the best subsequence of graphemes to match a generalized string. The dynamic zip locator used in the system makes use of this ability to locate zip without knowing exactly where the zip starts. The same matcher is also used for city-state-zip verification. We found that the reject criterion based on rank features is very reliable for rejecting errors.

Acknowledgment: We would like to thank Raymond Lorie, SriGanesh Madhvanath, and Tom Truong, IBM Almaden Research Center, for helping in retrieving city-state-zip database, and Marlon Abayan, Soloman's Brother Inc., for his initial work on cursive word recognition.

References

[1] P. D. Gader, J. M. Keller, R. Krishnapuram, J. H. Chiang, and M. A. Mohamed. Neural and fuzzy methods in handwriting recognition. *Computer*, pages 79–85, February 1997.

[2] P. Sinha and J. Mao. Combining multiple OCRs for optimizing word recognition. In *Intl. Conf. on Pattern Recognition. Brisbane, Australia*, Brisbane, Australia, Aug. 1998.

[3] S. N. Srihari. Recognition of handwritten and machine-printed text for postal address interpretation. *Pattern Recognition Letters*, 14:291–302, April 1993.

[4] N. W. Strathy and C. Y. Suen. A new system for reading handwritten zip codes. In *Proceedings of the International Conference on Document Analysis and Recognition*, pages 74–77, Montreal, Canada, August 1995.