A POST-PROCESSING ERROR-CORRECTION SCHEME USING A DICTIONARY FOR ON-LINE BOXED AND RUN-ON HANDWRITING RECOGNITION

Homayoon S.M. Beigi and **T. Fujisaki**
T.J. Watson Research Center
International Business Machines
P.O. Box 704
Yorktown Heights, New York 10598

W. Modlin and **K. Wenstrup**
Entry System Technology
International Business Machines
1000 N.W. 51st Street
Boca-Raton, Florida 33432

Abstract

Because of similar shape letters such as "v" and "u"; "k" and "h"; "1", "l", and "I"; and so on, plus many noisy channels such as digitizer tablets that intended hand-inputs should go through, any on-line recognition of handwriting letters cannot avoid producing errors. The use of a large vocabulary dictionary is necessary to make the system practically usable. However, imposing a limitation on the vocabulary handled by a recognition system is not very desirable for practical purposes. This paper presents a novel structure of a dictionary-driven error-correction post-processor for an on-line handwriting recognition environment which does not impose any coarse restriction on the vocabulary, however, it makes use of a large dictionary to reduce the errors made by the recognizer. This error- correction scheme reduces the error of a run-on and boxed handwriting recognizer by more than one half.

1 Introduction

The notion of using a handwriting interface to computers has been of great interest to many researchers for a number of years. With today's technology, a low-priced LCD digitizer tablet for capturing handwriting is conceivable with a size no bigger than a notebook. These devices capture the $< x, y >$ coordinates of the writer's pen on the surface of the digitizer tablet. Also, with the introduction of new pen-based operating systems such as the Penpoint^{TM} and Pen-Windows^{TM} operating systems handwriting recognition applications have become very practical. For a survey of different handwriting recognition methods see [1].

However, because of similar shape letters such as "v" and "u"; "k" and "h"; "1", "l", and "I"; and so on, plus many noisy channels such as digitizer tablets that intended hand-inputs should go through, any on-line recognition of handwriting letters cannot avoid producing errors. The use of a large vocabulary dictionary is necessary to make the system practically usable. Imposing a limitation on the size of the vocabulary handled by a recognition system is, however, not very desirable for practical purposes. This paper presents a novel structure of a dictionary-driven error-correction post-processor for an on-line handwriting recognition environment which does not impose any coarse restriction on the vocabulary, however, it makes use of a large dictionary to reduce the errors made by the recognizer. This error- correction scheme reduces the error of a run-on and boxed handwriting recognizer by more than one half. Figure 1 shows a flow-chart of the proposed error-correction scheme.

2 Description

This error-correction scheme was used with the IBM discrete and run-on [2] handwriting recognition systems. After the system has made its based candidates available for the inputted handwritten word, the error- correction routine takes over. This routine starts with the recognizer's top answer in ascii and a cache of memory containing the identity and the recognition scores of those hypotheses which along the search process of the recognizer were somehow not allowed to mature into the top recognition result. These hypotheses, however, have had good enough scores from the stroke matcher to have been extended in the search process. For a detailed description of the search process, a definition of strokes and other relevant information see [2].

As the first stage, a punctuation filter separates the punctuation present at the end of the recognized word and keeps it in a separate buffer. Then, verify the main body of the word by matching the sequence of characters against a word-processor dictionary to see if any word exists with that spelling. This match is case insensitive. The dictionary which was used here was an IBM word-processing dictionary with 100,000 words, compressed into an almost 100,000 byte data base. If the word has a very good overall score from the recognizer (better than a predefined threshold), or if it verifies with some existing word in the dictionary, the original top recognition is passed through a Case-Correction utility which operates in a majority-rule fashion.

A simple case-correction utility which was used here, checks if the first character of the word is recognized as a capital letter, and retains that letter. Then it counts the number of lower case and upper case characters and based on a majority ruling decision, converts the case of characters to get a uniform, gramatically acceptable case sequence. A more advanced version of this case-correction utility has also been used which carries information about special cases in addition to some matrix of confusions for that particular writer.

If the word does not verify with the dictionary, a character-match score is associated with it. This character match score is simply the sum of scores of the individual characters in the recognized word, given by the recognizer. Then, the main body of the recognized sequence of characters is sent to the stroke-match and spell-aid blocks to obtain a list of candidates for the intended word. These two blocks access the word-processor dictionary to obtain their candidates.

Along the search process, the top Match hypotheses are kept in a cache along with their scores. Basically, the objective of the stroke-match is to generate all the possible words from the strokes in this cache and to calculate the total matching score of each of these words. Then the N word hypotheses with the best matching scores are inserted into the global word hypothesis list.

The spell-aid module takes the recognition output and tries to find words in the dictionary which resemble this sequence of characters and inserts them in the global word hypothesis list. This module's output is very dependent on the initial characters. Therefore, most of the time, the first character of the input word is retained.

Since the stroke-match and spell-aid modules produce outputs heavily dependent on the first character of the input, for the Boxed mode, if the first recognized character, has the worst score among all the characters recognized in that word, an additional word hypotheses is created by replacing the first character of the originally recognized word with other candidates from the stroke cache. If the newly fromed word is not verified, other characters in the alphabet are tried in the order of their probability of occurance as the first letter of a word. This probability distribution has been obtained from a list of 270,000 words and their frequency in the English language, extracted from a 320,000,000 word corpus.

The shape-matching scores of the input word are then obtained from the inputted cache and normalized between 0 and the highest score in the character sequence. Then the scores are transformed such that the highest score is the best score and 0 is the worst score (Call this the character-match Score.) For the characters in the input string which have not been recognized (wild-card characters), a score of -1 is assigned which makes their matching score worse than any other character's score. At this time, look at the hypothesized words given by Spell-Aid or Stroke-Match. If the characters in those hypotheses are present in the candidate list, assign to them their corresponding score transformed in the above manner. If any character in the hypothesis does not have a stroke-match associated with it, then give that character a score 0 in the transformed space (worst character-match score). For each word hypothesis in the list, count the number of substitutions made from the original recognition to the hypothesis. (Call this the Substition Score.) For every hypothesized word also calculate the difference in its length and the length of the recognized sequence. (Call this the word-length Score).

In the the created list of hypotheses, the best hypothesis is the one which has the following properties in descending order of importance:

1. Smallest word-length score.

2. Smallest substitution score.

3. Smallest sum of absolute value of the differences of the character-match scores between the hypothesis word and the originally recognized word. If there is a character in the original recognition with a character score equal of -1, then that word hypothesis which changes this character has higher priority.

To maintain a robust performance, the post-processor will return the original answer of the recognition, if its total word-match score is better than some threshold. This ensures that carefuly written words which are not in the dictionary are still returned untouched, taking away the limits imposed by a limited vocabulary. Once a word hypothesis is chosen to be the final answer from the recognition, it passes through the case-correction block and is padded with possible punctuation filtered initially. This is the final answer of the recognition system.

3 A Test of the System

Table 1 presents the results from a test conducted over 12 writers with an average of 1250 characters of writing for each writer in Boxed mode, and 180 characters in Run-On mode. The character recognition errors have been shown to decrease by more than 50%.

At the time of search, some possible sequences, although valid words, might get a lower precedence and therefore not be chosen as the best hypothesis. It is desirable to retrieve the best of those words. Also, although character accuracy is an important goal, sometimes, a high character accuracy might be obtained with a very low word accuracy. In the case of natural writing, low word accuracies are not desirable and no matter how high the character accuracy is, sometimes, it is impossible for a human to understand the intended written text. However, with a high word accuracy, even if the character accuracy is not great, the content of the writing is more likely to be understood. The Word accuracies were increased using the dictionary error correction by over 30% in the case of boxed and about 20% in the case of Run-On writing.

In the proposed system, the dictionary error correction is turned off when correcting (overwriting) all or part of a word. This ensures the ability to correct a word once the wrong recognition result is provided.

4 Conclusion

The basic features of the post-processor are summarized as follows:

1. The Dictionary is an external and independent module which could easily be replaced.

2. The Error Correction Logic is independent of the language and the replacement of dictionary module is sufficient for supporting different languages. The system supports very large dictionaries (A dictionary of 100K compressed words is currently being used).

3. Even though the error correction module is designed to be placed downstream from the shape-matcher and segmenter modules and operate as a post-processor, it has an interface through which it receives shape-matching information from the recognizer. Most post-processing methods use some sort of a confusion matrix to try new candidates for the characters. In here, the module will use the information obtained from shape matching to dynamically generate a confusion matrix for the user and to use those confused candidates for correction. This information (list and scores of alternative candidates from shape-matching, etc.) is being utilized to obtain optimal error-correction.

4. Most of the dictionary driven error-correction back-ends will map a given word to *a word* in the dictionary which is closest to it, in some defined distance. However, this is not practically usable in real application environments. The intended text may contain words which are not a part of the particular vocabulary being used (e.g.,proper nouns). The Proposed system will allow users to override the dictionary vocabulary constraint by

writing carefully. Carefully written words will not be changed by the error-correction module even if they are not a part of the vocabulary. This is made possible by using the information from the shape-matcher on the closeness of the matched candidates to the actual writing.

5. In some cases, all the candidates for a character are refused by other modules (such as language models) at the search level. In these cases, no character is returned for corresponding locations in the word. This module makes the revival of these characters higher priority than changing the characters in other positions for obtaining a word which will verify through the dictionary.

6. Most existing dictionary modules, which are used for spell-aid of word-processors, are very dependent on the first character. This module, allows the usage of these existent modules by conducting a separate search for the possible candidates for the first character. The shape-matching information is used to assess the reliability of the first recognized character compared to the rest of the characters in the word. If the first character is not reliable, then it will try replacing it with other candidates from the shape matching. If the newly formed word is not verified, the module will try other characters in the alphabet in the order of their probability of occurance as the first letter of a word. This probability distribution has been obtained from a list of 270,000 words and their frequency in the English language, extracted from a 320,000,000 word corpus.

A system having the above features was implemented and has improved word accuracy by up to 30% while reducing the error by over 1/2 in a large-scale system evaluation test with 12 external handwrinting subjects.

References

[1] C.C. Tappert, C.Y. Suen and T. Wakahara, "The State of the Art in On-Line Handwriting Recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, N0. 8, pp. 787-808, August 1990.

[2] T. Fujisaki, H.S.M. Beigi, C.C. Tappert, M. Ukelson and C.G.Wolf, "Online Recognitoin of Unconstrained Handprinting: A Stroke-based System and Its Evaluation," _______, S. Impedovo, Editor, ELSEVIER: Italy, 1992.

Figure 1: Flow-Chart of Dictionary Post-Processor

	Accuracy (No Dict.)	Accuracy (Dict.)
Boxed WalkUp	79.6%	89.2%
Boxed Trained	92.1%	96.9%
Runon Walk Up	70.1%	73.2%
Runon Trained	87.9%	91.5%

Figure 2: Post-Processing Language Model

5 Dictionary Functions

All the functions which access the compressed dictionary, including the spell-aid and the stroke-match functions, rely on a common search routine to provide the search navigation and to explode the words from their compressed form. This search routine then utilizes caller provided routines to perform the functions necessary for the specific access method, including: initializing the search structures, testing whether to search a specific letter of the alphabet, and comparing a word extracted from the dictionary against the input according to the access criteria. High level pseudocode for the search routine follows:

```
Initialization (access method specific function)
FOR each letter of the alphabet
  IF search this letter (access method specific test)
    FOR each word under this letter
      Extract word
      IF not in proper length range
        continue
      Compare dictionary word with input
         (access method specific function)
```

6 Appendix: Dictionary Functions

The test to determine whether to search a letter of the alphabet is generally simple for both spell-aid and stroke-match. In the spell-aid function, usually only those words beginning with the same letter as the input word are searched. In a few cases where the first letter might be confused (for example, words beginning with 'ph' or 'f'), additional letters are searched if the first letter does not yield a good result. In stroke-match, a letter is searched if it is one of the choices for the first stroke in the stroke-match pattern.

Once the function decides to search a letter, every word under that letter is extracted from the dictionary. Some words are discarded immediately if their lengths do not fall within the acceptable range for the given input word or pattern. The acceptable range of lengths is determined at initialization time. For spell-aid, the difference in length to allow between the input word and a matching dictionary word is supplied as an argument to the spell-aid function. The range of lengths for the stroke matching routine is determined by an algorithm which goes through the pattern stroke by stroke and progressively determines the minimum and maximum lengths a word can be which fits the pattern.

Once a dictionary word is determined to fall within the allowable range of lengths, it is compared against the input word or pattern using the appropriate word testing function. This results in a scalar score for each word. The 'n' best results are returned, where 'n' is a caller supplied value.

6.1 Stroke-Match:

For boxed recognition, replace the wild character with the best hypothesis in the cache. Then keep a list of all the scores for the characters in the recognized sequence.

The stroke-match algorithm determines which words from the dictionary fit a supplied stroke pattern. Each element of the input stroke pattern contains the following information: a possible character, the beginning stroke, the number of strokes used by this possibility, and the score for this choice. The output from this algorithm is a list of dictionary words which fit this pattern along with their cumulative scores.

This stroke pattern is used to build a table during initialization. In this two-dimensional table, rows represent strokes and columns represent the character (first column=a, second column=b, etc.) Each entry contains the score for this character starting at this position, and the number of strokes (stored as a 16 bit entry). In this way, when a dictionary word is to be compared to the pattern, it is a simple task to traverse the table. Start at the first row. If the first character can start with the first stroke, start a score counter with this score, and add the number of strokes needed to the row counter. Success is achieved if we finish the dictionary word with the number of strokes in the pattern, and a score has been computed from the stroke scores.

One complicating factor arises from the fact that we can have the same character possibility in the stroke pattern beginning at the same character with different numbers of strokes. For example, we can have a two or three stroke 'a' beginning at stroke one. We want to store both possibilities since they can contribute to forming completely different words. However, each entry in the table has room for only one 16 bit entry. To accomodate this situation, there is a secondary table called the dups table. When two different stroke lengths for the same character in the same position arise, the main entry in the stroke-match table sets its highest order bit which is reserved to indicate this situation. The rest of the main entry is a row index into the dups table. The dups table then contains entries in this row as would be found in the main table. The dups table is limited to four columns so that we can store a maximum of four different stroke lengths for this character at this position.

To use the dups table, we must now introduce a stack in the search process. When we reach a letter in the dictionary word for which there are multiple stroke length possibilities in the dups table, we store all the information needed to return to this point in the search in the stack along with the score for each of the different lengths. If our search fails using one of the lengths for this character, we can then pop off the stack back to this decision and try a different stroke length.

One additional heuristic has been added to the stroke-match algorithm to make it more robust for stroke-match patterns which might be missing the correct answer for one or two of the stroke positions. Upon entry, the best choice for the characters fitting this pattern are supplied (not necessarily a word.) During initialization, the table is searched for this character pattern and the scores and stroke lengths of each of the characters is recorded. Then, during the search for any dictionary word, if we come to a position in which no entry exists for the needed character in this position, we look to the top choice array. If the best choice for this position is greater than a certain threshold, we assume that no really good choices exist for this position. Then, the needed character is assumed to be in this position with the same number of strokes as the top score answer. The inserted character is given a bad score so that we are penalized for having to go out of the stroke-match table to fit the dictionary word.

6.2 Spell-Aid:

The spell-aid algorithm uses a simple one pass comparison between the input word and the dictionary word to determine a score for the match. As the words are scanned, two counters are kept. The variable "match" records how many letters were matched between the two words; the variable "contig" counts the number of pairs of contiguous characters which were successfully matched.

During the match, a pointer into each word is kept until one of the pointers reaches the end of its word. During each step of the algorithm, we first try to match directly the characters pointed to by the two pointers. If these characters match, then the match counter is incremented. The contig counter is incremented if this is the second exact match in a row.

If there is not an exact match, however, some fitting matches are tried. First, we attempt to transpose the following two characters in one of the words and match these two characters in their new order. If this transpose works, then the "match" counter and pointers are all incremented by two, but the "contig" counter is not incremented.

If the transpose match doesn't work, an attempt is made to skip a letter in either of the words. If this succeeds, the "match" counter is incremented one and the word pointers are moved to after the matching characters. Next a "stretch" match is attempted by skipping two letters in the longest word and attempting a match. If none of these fitting matches succeeds, then each of the word pointers is incremented without affecting the "match" counter.

After one of the pointers traverses its entire word, a score is determined for this word. The primary score is determined as:

(2 * match) - (difference in lengths)

This score is then shifted left by eight, and the "contig" counter is added. In this way, the match is given a 16 bit score in which the above value is the primary deciding factor and the "contig" counter acts as a tie breaker.