DOI: 10.3745/KIPSTD.2011.18D.1.067

제한적 문맥 인식과 다중 스트림을 기반으로 한 숫자 정정 OCR 모델의 설계 및 구현

신 현 경[†]

요 약

재무 데이터 관리를 위한 자동화된 비지니스 서류 영상 처리 시스템에서 숫자 정보 검색 중 발생한 오류는 심각하여 그 시스템의 가용성 및 성능을 결정한다. 그 동안 자동 맞춤법 교정에 관한 방법론들이 개발되어 정보 검색 시스템 개발에 중요한 역할을 해왔으나 이러한 맞춤법 교정은 알파벳 등 기계학습이 가능하고 사전 형태로 보관이 가능한 기호에 한정되어왔다. 반면에 순수한 마코프 수열에 불과한 숫자들의 순열들은 맞춤법 교정을 위하여 사전적 형태로 보관하여 활용하는 것이 불가능 하다. 본 논문에서는 확률론적 정보 검색 알고리즘의 토대위에 제한적 문맥 인식과 복수의 스트림을 적용한 새로운 형태의 숫자 정정 OCR 모델을 제안하였다.

본 논문에서 제안된 숫자 정정 모델은 기존의 송장 문서 처리 시스템에 구현하였으며 제안된 숫자 정정 모델의 효과를 확인하기 위해 비교 테스트를 실행하였고 테스트 결과 상당한 성능이 개선되었음을 보여 주었다.

키워드 : 정보 검색, 스펠링 정정 모델, 제한적 문맥 인식 기반 교정, 자동 문서 처리 시스템, 문자 인식, 자연어 프로세싱

Design and Implementation of OCR Correction Model for Numeric Digits based on a Context Sensitive and Multiple Streams

Shin Hyun Kyung[†]

ABSTRACT

On an automated business document processing system maintaining financial data, errors on query based retrieval of numbers are critical to overall performance and usability of the system. Automatic spelling correction methods have been emerged and have played important role in development of information retrieval system. However scope of the methods was limited to the symbols, for example alphabetic letter strings, which can be reserved in the form of trainable templates or custom dictionary. On the other hand, numbers, a sequence of digits, are not the objects that can be reserved into a dictionary but a pure markov sequence. In this paper we proposed a new OCR model for spelling correction for numbers using the multiple streams and the context based correction on top of probabilistic information retrieval framework.

We implemented the proposed error correction model as a sub-module and integrated into an existing automated invoice document processing system. We also presented the comparative test results that indicated significant enhancement of overall precision of the system by our model.

Keywords : Information Retrieval, Spelling Correction Model, Context Sensitive Correction, Automated Document Processing System, OCR, Natural Language Processing

1. Introduction

An automated business document processing system, founded on variety of information retrieval techniques, performs query and correction procedures on the alphabetic texts and the numeric digits read from input document. In case that the inputs are semi-structured documents having internal XML-type format, the typical query functions employed in a desktop or a web

※ 본 연구는 2010년 경원대학교 교내 연구비 지원을 받아 시행된 연구임.
† 종신회원 : 경원대학교 수학정보학과 조교수
논문접수 : 2010년 9월 10일
수 정 일 : 1차 2010년 10월 25일
심사완료 : 2010년 10월 25일

application can acquire texts and numbers by exploring the pre-defined relational data structure [1], which implies no acquisition errors and low parse errors. However, in case that the inputs are the un-structured documents such as a collection of the texts obtained from a scanned image by an OCR engine, acquisition errors are so significant that data acquirement module should include an auto-validation function to correct input errors.

The conventional information retrieval techniques are designed on the structured text data [2]. The same techniques cannot be delineated effectively to the unstructured data obtained from the scanned document image by OCR. Some examples of the automated document system taking scanned documents as its input are invoice parsing [3], query on legal or medical documents [4-6], digital video surveillance system [7] for reading vehicle license plate, and etc.

In order for the conventional information retrieval techniques to be extended to domain of unstructured documents, two major pre-processings are pre-requisite. 1) Layout analysis that converts to a structured document. Appropriate conversion of the raw OCR texts into a structured data format provides the unit documents with well defined relation so that the standard query techniques can be applied effectively. 2) Correction of errors that occurred from input texts. As a formatted data in an archive for query system, a structured document has no text errors, while an un-structured document usually suffers from errors in texts due to OCR from reading the scanned image. This involves that, prior to transferring an unstructured document into an archive, text error correction is indispensable.

OCR text errors can be classified into the two classes: errors to the alphabetic and the numeric texts. The error in alphabetic word can be resolved by a spelling correction typically using a dictionary matching, on the other hand the error in numeric texts has no standard solution. In this paper we addressed the problems of error correction occurring in the query for numbers. We proposed a correction model based on using the multiple text streams and using the contexts in the form of a relational formula.

This paper organizes as follows: in section 2 related previous research works are presented, in section 3 theoretical and implementation details for our model are introduced, in section 4 test procedure and experimental results are presented, and in section 5 some discussions on our model are described.

2. Related Works

In this section the previous research works related to layout analysis for conversion to semi-structured format from un-structured document and text error corrections are described.

Layout analysis on the documents, one of the hardest problems in the area of document segmentation, has been considered as a useful technique for improvement OCR related performance. For robust recognition, Rasagna et al. used the word clustering method by locality sensitive hashing and presented significant improvement [8]. Li at el. proposed a categorization with electronic abstracts method specialized for the case of biomedical document [9].

Correction of OCR errors has been studied by many researchers using variety of methods. A brief summary is as follows. Xu and Nagy employed a prototype extraction method based on a tolerance correction [10], which was considered to maintain OCR accuracy with decreasing quality of document. AviItzhak et al. adopted the neural networks with centroid-dithering training on the various font types [11]. Drira et al. proposed an an-isotropic diffusion model, PDE based OCR recovery model [12]. Garain et al. suggested a pairwise discrimination principle adopted from artificial immune system. They employed support vector machine framework for implementation [13]. Jain et al., applied independent component analysis method on the un-calibrated camera-based image [14] and showed significant improvement on OCR recognition. On the other hand, Koga et al. proposed discriminant feature extraction followed by dictionary word matching [15], which also showed significant improvement. Shin et al. adopted a super-resolution method as an image restoration from a single image for a biometric technique of iris recognition [16].

An OCR engine converts the scanned documents to text data with hierarchical structure such as character, word, line, and paragraph. Due to the low precision of lines and paragraphs from OCR engine, the hierarchical structure is not useful in practical problems however the accuracy of word level structure is high enough to be used. In development of a query system on scanned document, OCR error is the source of error propagating from the earliest stage. Accurate text error correction methods upgrade the precision of information retrieval process.

An automated invoice parsing system takes scanned invoice documents as input and parses the contents such as purchase order, invoice date, vender name, and details of product items as well. The performance of the system depends on the precision of several factors such as detection of region of interest, field parsing rule, and OCR accuracy. For the study of this paper, let's focus on the case of OCR error. In case OCR errors occurred to alphabetical texts, the system adopts the standard spelling correction rules while in case OCR errors occurred to numeric digits, the system ignores and overrides with the reasonable value. In this paper we proposed a method to correct errors on the numeric digits.

3. Query and Correction Model

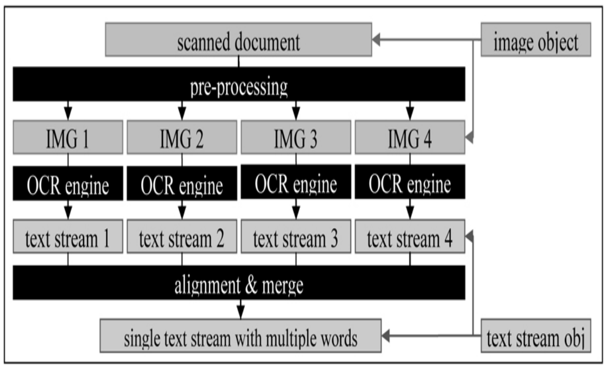
The proposed correction method consisted of the following three approaches:

1. use of multiple text streams by generation of the multiple pre-processed images,
2. use of baysian voting scheme,
3. use of context sensitive error correction.

Each of these approaches is explained throughout this section.

3.1 Use of Multiple Text Streams

The idea of using multiple streams was stemmed from an effort to recover from OCR errors by means of yielding the multiple text data by different processes. Refer to [17], in which multiple images taken from the different angles were used, for the case of video streams. For example, the texts with dotted font type tend to cause OCR error that can be easily fixed by gaussian smoothing and the texts within half-toning region can be recognized more accurately by low resolution conversion.



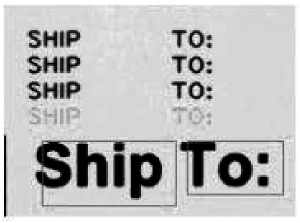
(Fig. 1) Schematic diagram for conversion from a document image to a text stream

A schematic diagram of the procedure is shown in (Fig. 1). This procedure is in two stages: create the multiple text streams; merge the multiple text streams into a single stream with multiple texts.

3.2 Construction of Multiple Text Streams

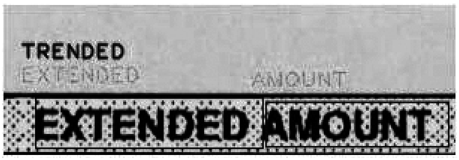
In order for a given scanned document to create multiple image streams we applied the edge conversion [IMG2], the gaussian smoothing [IMG3], and the low resolution conversion [IMG4]. Four text streams were obtained as the outputs from an OCR engine fed by the three of pre-processed images and the original image [IMG1]. For the study of this paper, we set the configuration of OCR engine to produce the texts with unit of word not with unit of character. As the result, the text stream 1 contains the list of text words from IMG1, the text stream 2 from IMG2, the text stream 3 from IMG3, and the text stream4 from IMG4. The word instances in the separate streams were to be grouped in terms of the bounding boxes, which is explained as below.

3.3 Conversion of Multiple Text Streams into Single Stream with Multiple Text Instances

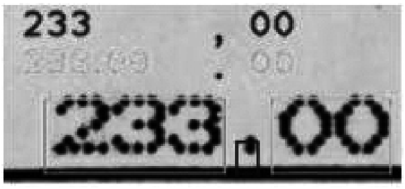


(Fig. 2) Bounding box with multiple OCR words

(Fig. 2) shows a cropped region of a scanned document with the embedded OCR texts. The text words "Ship" and "To:" were recognized by OCR engine successfully for all four IMG1, 2, 3, and 4. For example, on top of the green bounding box surrounding "Ship", the four texts were embedded: each of different colored texts indicates IMG1, 2, 3, and 4 from top to bottom, respectively. In general bounding box of a text in a stream supposedly has a matched box in the other stream. An exception can happen when a text cannot be recognized in a stream while it can be recognized in other streams, e.g. texts in heavily textured background can be read only for the pre-processed image with low-resolution conversion.

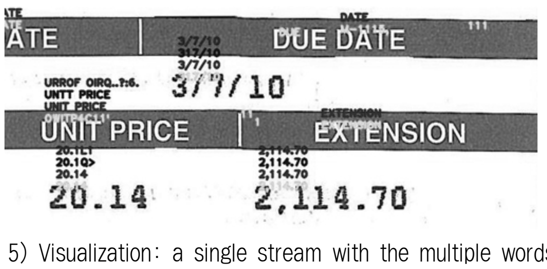


(Fig. 3) OCR texts in heavily textured background



(Fig. 4) Fragmentation of the bounding boxes

For example as seen in (Fig. 3) the texts "EXTENDED" and "AMOUNT" were recognized from two (IMG3 and IMG4) and one image stream (IMG4), respectively. Another problem is that a single box can be matched with the multiple subsets of the box, e.g., as can be seen in ((Fig. 4) a text '233.00' in the image can be modified to '233 00' (the dot '.' was removed by a pre-processing) which results in the two bounding boxes covering '233' and '00'. In order to resolve this problem we used a rule based on the union of bounding boxes: pick a bounding box of the stream 1, search the intersecting bounding boxes in stream 2, 3, 4. Take the union of those collected bounding boxes. This bounding box replaces the original one. Pick a next bounding box of the stream 1 and continue the same process for searching boxes in other streams. Once the matching process completed, the connected components of the new bounding boxes in stream 1 are the bounding boxes for texts. This resulting bounding box was used to align the texts in the streams into the single stream in which the bounding box contains the four (or less) instances of texts. Our assumption is that the probability of having a true (correct) text among the four candidate texts is higher than that of having correct text from a single stream.



(Fig. 5) Visualization: a single stream with the multiple words

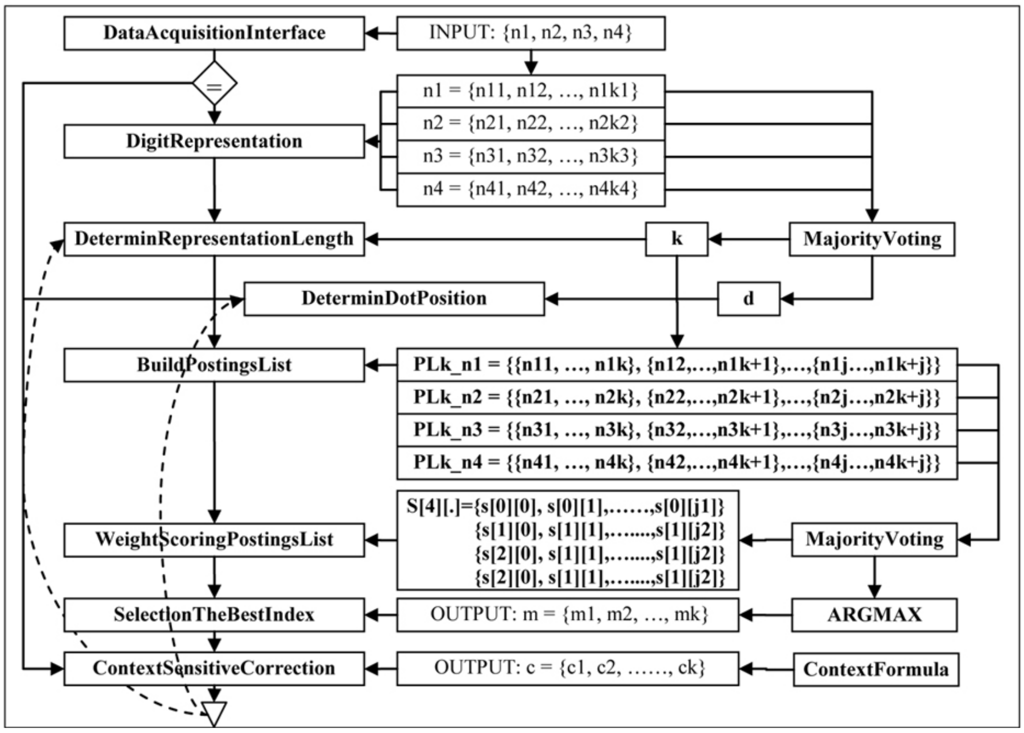
An example of 'single text stream with multiple words' is visualized in (Fig. 5) As seen in the (Fig. 5) a numeric digit word '20.14' was recognized as '10.1L1', '20.1Q>', '20.14', and '20.14' according to processing method, respectively.

3.4 Method of Classification of Text Words into the Two Groups

The text words in a document were grouped into the two classes: 1) alphabetics [class-A] and 2) numerical digits [class-D]. For the classification we defined a predicator function [pred-F] which counts the number of digits in a word. If the number of numerical digits in a word was greater than the number of non-digits then the word was moved into the class-D otherwise into class-A.

For the case of class-A, one of the standard methods of error correction called "dictionary matching" was used. For a given bounding box, if one of the words was included in a look-up dictionary then the word was selected as the corrected word. On the other hand, for the case of class-D, a dictionary instance is not even feasible since any sequence of digits can be meaningful. For resolution of this problem, as the main subject of this paper, a new recovery model designed for correction of the OCR errors occurred to the numeric digits is proposed. (Fig. 6) is an overview of the digit correction model.

A general and brief description of the diagram is as follows: suppose a set of numbers, {n1, n2, n3, n4} was an input to the DataAcquisitionInterface of this model. DigitRepresentation function made the input numbers split into the ASCII digits. DeterminRepresentationLength function set a number, k, by applying MajorityVoting module. Once the value k is set, for each digit representation of the input numbers, say n1, BuildPostingsList function created a list of postings-list of digits of size k. 'PLk_n1' indicates the positings list of n1 of size k. In order to utilize weight scoring method, a weight matrix S[4][.] was allocated. The value S[i][j] corresponds to the score of an element in PLk_ni[j]. WeightScoringPostingsList function evaluated the score values by using MajorityVoting method to build the matrixS[4][.]. From the weight matrix S[4][.], the selection of the best index (or an element in postings list) is straightforward by ArgMax. The output of SelectionTheBestIndex function is the corrected text of the input texts. In this paper, we applied


DataAcquisitionInterface
INPUT: {n1, n2, n3, n4}
=
n1 = {n11, n12, …, n1k1}
n2 = {n21, n22, …, n2k2}
n3 = {n31, n32, …, n3k3}
n4 = {n41, n42, …, n4k4}
DigitRepresentation
DeterminRepresentationLength
k
MajorityVoting
DeterminDotPosition
d
BuildPostingsList
PLk_n1 = {{n11, …, n1k}, {n12,…,n1k+1},…,{n1j…,n1k+j}}
PLk_n2 = {{n21, …, n2k}, {n22,…,n2k+1},…,{n2j…,n2k+j}}
PLk_n3 = {{n31, …, n3k}, {n32,…,n3k+1},…,{n3j…,n3k+j}}
PLk_n4 = {{n41, …, n4k}, {n42,…,n4k+1},…,{n4j…,n4k+j}}
WeightScoringPostingsList
S[4][.]={s[0][0], s[0][1],……,s[0][j1]}
{s[1][0], s[1][1],…….,s[1][j2]}
{s[2][0], s[1][1],…….,s[1][j2]}
{s[2][0], s[1][1],…….,s[1][j2]}
MajorityVoting
SelectionTheBestIndex
OUTPUT: m = {m1, m2, …, mk}
ARGMAX
ContextSensitiveCorrection
OUTPUT: c = {c1, c2, ……, ck}
ContextFormula

(Fig. 6) The proposed error correction model for the numeric digit texts

ContextSensitiveCorrection if a formula can be considered. The more detailed explanations with the implementation codes of the methods are described as below in this section.

For the rest of this section we presented the implementation details of each module consisting of our error correction model: a string acquisition (Data AcquisitionInterface), a tokenization of number (Digit Representation), an algorithm for determination of the output string length (DeterminRepresentationLength), an algorithm for determination of floating point position (DetermineDotPosition), an algorithm to build a postings list of the input numbers (BuildPositinsList), an algorithm to build weight scoring matrix (WeightScoringPostings List), an algorithm to deal with the multiple selection (SelectionTheBestIndex), and an algorithm to be used for context based correction (ContextSensity Correction).

We started with describing MajorityVoting (Plurality Voting) since it served as the basic module to the others. Followingly, we presented the modules in sequential order.

3.5 Plurality Voting

As seen in (Fig. 6) the digit correction model was based on a voting scheme, denoted as MajorityVoting of which implementation was described in <Algorithm 1>. In the algorithm, for the convenience of presentation, we used a fixed sized vector (size of 4) as an input. But the algorithm can be applied to the vectors with any size.

Implementation details of the MajorityVoting is as follows: suppose the four ascii strings were input arguments, s[4] = {s[1], s[2], s[3], s[4]}.In order to utilize a voting scheme, a weight vector, w[4] = {w[1], w[2], w[3], w[4]}, was defined and initialized as zero vector, as seen in LINE1. From LINE2 to LINE7 the weight vector was estimated as follows: The value of w[1] was increased by 1 when s[1] was matched with s[2], LINE2, and repeated the same process for s[3] and s[4], LINE3 and LINE4. The value of w[2] was increased by 1 if s[2] was matched with s[3], LINE5, and repeat the same process for s[4], LINE6. The value of w[3] is increased by 1 if s[3] is matched with s[4], LINE7. At LINE8 the index of the maximum weight was selected by a simple arg-max method. The arg max method had problem when w[4] has the multiple maximums – there are three cases:

case 1) when each s[k] is unique;
case 2) when s[1]=s[2] and s[3]=s[4];
case 3) s[1]=s[3] and s[2]=s[4].

〈Algorithm 1〉 Majority voting algorithm for the fixed size input vector

```
MajorityVoting(s[4], conf)                          // AGR: four numeric digits and confidence level
1    float w[4] = {0}                               // define and initialize weights
2    if (s[1] == s[2]) w[1] += 1                    // estimates the weights LINE2-LINE7
3    if (s[1] == s[3]) w[1] += 1
4    if (s[1] == s[4]) w[1] += 1
5    if (s[2] == s[3]) w[2] += 1
6    if (s[2] == s[4]) w[2] += 1
7    if (s[3] == s[4]) w[3] += 1
8    int index = arg max{w[1], w[2], w[3], w[4]  // arg max to select the best index
9    conf = w[index] / 4                            // evaluate the confidence level
10   return s[index]
```

In this paper we selected s[1] if the multiple maximum case occurs. LINE9 shows the way of evaluation of confidence level for the selection.

The MajorityVoting algorithm introduced above is data-type independent, e.g., it is applicable to the numbers, the ascii characters, the sequence of ascii characters, the strings, and the abstract data types.

3.6 Data Acquisition Interface

One of the functionalities of DataAcquisitionInterface is to filter out the non-digit inputs. For the non-numeric texts, we employed the dictionary matching method separately. The other functionality is to check whether all the inputs are same. If the inputs are all the same, there is no need to invoke the correction process.

3.7 Digit Representation

For the study of numeric digit correction we explored digit-by-digit representation of a number and applied the correction method on each digit in the representation. Algorithm 2 explained how a number was converted to its representation form. A number is a sequence consisted of the digits, the dot (‘.’), the comma (‘,’), and some pre-fixes ‘$’, ‘+’, and ‘–‘, e.g., n = n1 n2 n3 ⋯ nk where nj is one of {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} or ‘,’, or ‘.’. For a special case, n1 can be ‘$’, ‘+’, ‘–‘. In this paper we assumed that we could differentiate the digits from the comma, the dot, and the other pre-fix signs so that a representation of number could be a collection of the pure digits. It should be mentioned that in some pathological cases, ‘$’ can be confused with ‘5’ or ‘8’.

3.8 Determination of Representation Length

Suppose we have a set of the four numbers, {m1, m2, m3, m4}, with digit representations as follow:
m1 := {m11, m12, ⋯ , m1k1}
m2 := {m21, m22, ⋯ , m2k2}
m3 := {m31, m32, ⋯ , m3k3}
m4 := {m41, m42, ⋯ , m4k4}

Notice that the lengths of representations can be various. We used the majority voting algorithm described above to determine the length of the resulting number as seen in <Algorithm 2>.

〈Algorithm 2〉 determination of length of representation

```
DeterminRepresentationLength(m1, m2, m3, m4)
1  int k1 = length of the representation of number m1
2  int k2 = length of the representation of number m2
3  int k3 = length of the representation of number m3
4  int k4 = length of the representation of number m4
5  return MajorityVoting(k1, k2, k3, k4)
```

3.9 Determination of Dot Position

If inputs were floating point numbers, e.g., {123.45, 128.45, 123.95, 12345}, as explained in the section [DigitRepresentation], they were extracted into the sequence of pure digits. In our method, the positions of dot were treated independently. Dot position was the count from the right end of text. For example, in ‘123.45’ the dot position is defined as 2. The method of determining the dot position is similar to Determin RespresentationLength using the MajorityVoting.

<Algorithm 3> build postings list of digit representation

```
BuildPostingsList(n, PL[1], …, PL[k])
1  PoLi_n1 := {n1}-{n1, n2}-{n1, n2, n3}- … -{n1, n2, …, nk}. // build postings list
2  PoLi_n2 := {n2}-{n2, n3}-{n2, n3, n4}- … -{n2, n3, …, nk}. // build postings list
3  …
4  PoLi_nk := {nk}.                                            // build postings list
5  PL[1] := set of sequence of k-gram digit with length 1: {{n1} {n2 }{n3}… }
6  PL[2] := set of sequence of k-gram digit with length 2: {{n1, n2}{n2, n3}…}
7  …
8  PL[k] := set of sequence of k-gram digit with length k {{n1, n2, …, nk}}
```

3.10 Build Postings List

Suppose we determined the length of resulting number from [DeterminRepresentationLength], say ‘n’. Following the original definition of postings list, any order-preserving subsequence of digit up to a length ‘n’ should be included in the postings list. For example, ‘123.45’ with n =5, the postings list would be constructed from a collection of {1}, {2}, {3}, {4}, {5}, {12}, {23}, …., {12345}. We categorized by the starting digit: PoLi1:= {1}➔{12}➔{123}➔{1234}➔{12345}, PoLi2 := {2}➔{23}➔{234}➔{2345}, …, PoLi5 := {5}. Using the PoLi set, we further defined PL set which is k-gram of PoLi’s. As seen in <Algorithm 3> below, PL[j] is the set of sequence of k-gram with length ‘j’. This set of PL’s will be used as the argument of PostingsListWeightScoring function to determine the error corrected output.

3.11 Postings List Weight Scoring Scheme

Suppose the length of representation was determined, say k. Among all the postings lists of a number generated by BuildPostingsList, we elected the posting lists of size k, e.g., PL[k]. For each of the four input numbers, {m1, m2, m3, m4}, we picked the corresponding postings list. For the convenience of notation, let us denote the selected postings lists as follows:

m1PL := PL[k] obtained from BuildPostingsList (m1, PL[1], …., PL[k1])
m2PL := PL[k] obtained from BuildPostingsList (m2, PL[1], …., PL[k2])
m3PL := PL[k] obtained from BuildPostingsList (m3, PL[1], …., PL[k3])
m4PL := PL[k] obtained from BuildPostingsList (m4, PL[1], …., PL[k4])

Our assumption was that the digit representation of true solution (the error corrected digit text) occurred frequently among the set m1PL, m2PL, m3PL, and m4PL. As an example, suppose ‘123.45’ is the true value and an OCR engine returned ‘128.45”, “123.45”, “$123.5”, “812345” from each stream, respectively. We can observe high frequency of the digits ‘1’ at the first or second position in the representation, ‘2’ at the second or third position, and so on. In the example, m1PL = {12845}, m2PL = {12345}, m3PL = {}, m4PL = {{81234}, {12345}}.

PostingsListWeightScoring function assigns a weight value in the form of a scoring matrix. The scoring matrix is an array of scoring vector for each stream. For example, a scoring vector S1 is for a postings list m1PL (S2 for m2PL, S3 for m3PL, and S4 for m4PL). The size of scoring vector S1 is the same with that of postings list. As an efficient way of constructing the scoring matrix, we used the index tree instances. An index tree of postings list is in form of {i1, i2, i3, i4}, where ‘i1’ indicates ‘i1’-th element in m1PL, ‘i2’-th element in m2PL, ‘i3’-th element in m3PL, and ‘i4’-th element in m4PL. Once a set of the index tree is constructed, the score is evaluated by using MajorityVoting as the following way: for an element in the postings list of the first stream, m1PL[i1], the score will be 1 if mPL[i1] = MajorityVoting(m1PL[i1], m2PL[i2], m3PL[i3], m4PL[i4]), otherwise 0. In other words, the weight score is the probability of being the k-gram in majority population.

After completion of building weighted scoring matrix, process of selection of the best index tree was performed by a simple ArgMax method. If [k, j] is the best selection, then j-th k-gram of k-th stream is the corrected text.

For the presentation of building a scoring matrix with the example given above, denote scoreM to be a score matrix. The construction process is the following:

Set of index tree is {1, 1, -1, 1}, {1, 1, -1, 2}, where -1 indicates the empty case.

For case {1, 1, -1, 1}:

```
    "12845" = MajorityVoting(m1PL[1], m2PL[1],
              m3PL[-1], m4PL[1], conf)
    ScoreM[1][1] += conf, where conf = 0.33.
    ScoreM[2][1] += 0
    ScoreM[4][1] += 0
```

For case {1, 1, -1, 2}:

```
    "12345" = MajorityVoting(m1PL[1], m2PL[1],
              m3PL[-1], m4PL[2], conf)
    ScoreM[1][1] += 0
    ScoreM[2][1] += conf
    ScoreM[4][2] += conf, where conf = 0.66.
```

As the result of this process we can see that the output of PostingsListWeightScoring is "12345". A pseudo code implementation of this process is summarized in <Algorithm 4>.

3.12 Context Sensitive Correction Scheme

We have introduced a modeling method for correction of OCR error occurred to the digits. In this section we further discussed on a context sensitive correction model for the case of available relational data structure. Suppose we have a relational formula among a set of three numbers, {n1, n2, n3}, e.g., n1 = n2 * n3. We created a context sensitive correction model if the three numbers failed to satisfy the formula relation. Our assumption on this model here was that only one of the numbers among the three was allowed to be wrong. Once we found out

<Algorithm 4> scoring algorithm for numerical digit representation

```
PostingsListWeightScoring(m1PL, m2PL, m3PL, m4PL)
int x = max{m1PL.length, m2PL.length, m3PL.length, m4PL.length}
float scoreM[4][x] = {0}
postingsList p1 = m1PL[0]     // pointer to the head of m1PL
postingsList p2 = m2PL[0]     // pointer to the head of m2PL
postingsList p3 = m3PL[0]     // pointer to the head of m3PL
postingsList p4 = m4PL[0]     // pointer to the head of m4PL
while p1 is not null, i1++    // index tree selection process
    p2 = m2PL[0]
    while p2 is not null, i2++
        p3 = m3PL[0]
        while p3 is not null, i3++
            p4 = m4PL[0]
            while p4 is not null, i4++
                // index tree: {i1, i2, i3, i4}
                scoreM[0][i1] += (p1 == MajorityVoting(p1, p2, p3, p4, conf)) ? conf : 0
                scoreM[1][i2] += (p2 == MajorityVoting(p1, p2, p3, p4, conf)) ? conf : 0
                scoreM[2][i3] += (p3 == MajorityVoting(p1, p2, p3, p4, conf)) ? conf : 0
                scoreM[3][i4] += (p4 == MajorityVoting(p1, p2, p3, p4, conf)) ? conf : 0
                p4 = p4.next
            end while
            p3 = p3.next
        end while
        p2 = p2.next
    end while
    p1 = p1.next
end while
[int, int] [k,j] = arg max{scoreM[0][0], …, scoreM[4][x]}     // selection of the best index tree
return mkPL[j]
```

〈Algorithm 5〉 context sensitive correction algorithm

```
ContextSensitiveCorrection(n1, n2, n3)
1    int d1 = DigitDistance(n1, n2 * n3)     // counts of disparity of digits between n1 and n2 * n3
2    int d2 = DigitDistance(n2, n1 / n3)
3    int d3 = DigitDistance(n3, n1 / n2)
4    return arg min {d1, d2, d3}
```

〈Algorithm 6〉 distance measurement between the two numeric digits

```
DigitDistance(n1, n2)
1    char digits1[ ] = DigitRepresentation(n1, digits1)    // split each digit of a number n1
2    char digits2[ ] = DigitRepresentation(n2, digits2)    // split each digit of a number n2
3    int distance = 0
4    for k = 0 to k = digit1.length
5        if digits1[k] != digits2[k]
6            distance++                                    // counting measure
7    return distance
```

which number was wrong then correction was straightforward.

<Algorithm 5> describes how to detect the number damaged with error. The algorithm is based on the idea of OCR errors occur to a subsequence of digit representation. For example, a set of the three number {308, 20, 15} does not satisfy n1 = n2 * n3. The all of the possible scenarios are among the following three cases:

Case 1. 300 = 20 * 15 : one count of digit error '8' in 308

Case 2. 308 = 20.53 * 15 : three count of digit errors '.53' in 20

Case 3. 308 = 20 * 15.4 : two count of digit errors '.4' in 15

We determined to select the case having the minimum counts of digit error, i.e., correct '308' to '300' for the example case.

DigitDistance in Algorithm 5 is a distance measure that is a simple counting measure by digit-by-digit comparison. An implementation detail is presented in <Algorithm 6>.

4. Experiments and Results

4.1 Data Set

We constructed a ground truth data set sampled by random selection of 1,000 scanned invoice documents from an archive. The ground truth data were divided into the four statistically independent subsets: set1 with 240 samples, set2 with 340 samples, set3 with 230 samples, and set4 with 190 samples.

In the automated invoice processing, the contents of an output are the details of transactions: the number of the product items in the document, the count of product items that are shipped, and the unit cost of the item. As an example from (Fig. 7) under the column "PRICE", there are three numbers (33.87, 19.34, and 17.56), under the "SHIPPED" column, there are also three numbers (4, 4, and 5), and under the "AMOUNT" column the three numbers (135.48, 96.70, and 97.40).

As an important property, every invoice document should contain its unique "purchase order number" or "customer order number". For the convenience of notation, we name it as PO number. In (Fig. 7) for an example of price item line detection, the rectangles are drawn to indicate the detected regions. The PO number appeared at the right above 'DESCRIPTION' column. Exploiting this property of uniqueness of PO number, each of the selected documents was labeled with the PO number and the values for the price line items (such as the shipped quantity, the unit cost, and the extension price) were manually acquired.

In this study a ground truth data was defined as an instance of XML configuration type nested structure whose fields are consisted of "PO" number, the number of price lines, the value of unit cost, the value of shipped quantity.

<PO number> PO number </PO number>
<LINE number> number of price lines </LINE number>
<LINE number 1> unit cost, shipped quantity <LINE number 1>
…
…
<LINE number n> unit cost, shipped quantity <LINE number n>

The format of the output from the invoice parsing system is designed to be same with the format of the ground truth data. For example, from the invoice seen in (Fig. 7) the system extracted PO number, the number of lines, the unit cost, the shipped quantity, and the extension amount as below:
PO number: 1374145
The number of the price lines: 3.

At Line1. 33.87 (unit cost), 4 (shipped quantities), 135.48 (extension amount)
At Line2. 19.34 (unit cost), 5 (shipped quantities), 96.70 (extension amount)
At Line3. 17.56 (unit cost), 5 (shipped quantities), 87.80 (extension amount)

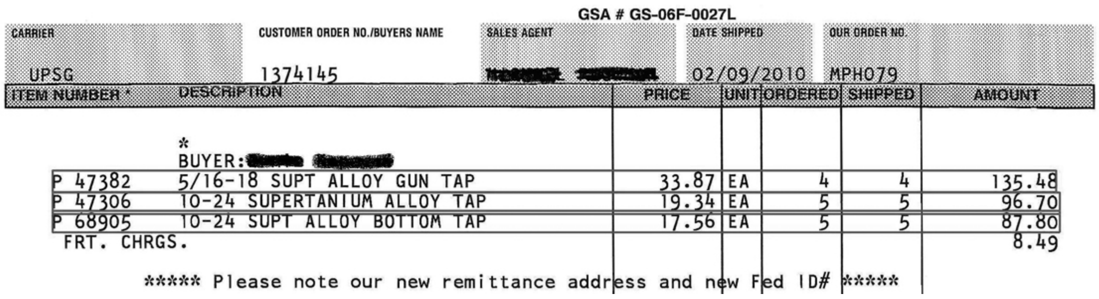
Using this information the auto validation module created the output data as
<PO number> 1374145 </PO number>
<LINE number> 3 </LINE number>
<LINE number> 33.87, 4 <LINE number>
<LINE number> 19.34, 5 <LINE number>
<LINE number> 17.56, 5 <LINE number>

4.2 Experiment Procedure

There exists an automated invoice parsing system (AIPS) which takes a scanned document as an input and extracted details of product items as an output. The AIPS is a two-stage process: at the first stage it detects the region of interest; at the second stage, it parses the information from texts within the region. For illustration purpose, the three red rectangles in (Fig. 7) containing the product items as an output of the first stage. At the second stage, a parser extracts invoice information from the texts within the region. For example, consider the top rectangles from the first stage. The collection of texts are {"P", "47382", "5/16-18", "SUPT", "ALLOY", "GUN", "TAP", "33.87", "EA", "4", "4", "135.48"} from which a parser in the AIPS extracts a triplet {"4", "33.87", "135.48"} at the second stage. Of the two stages in AIPS, the proposed correction model involves only on the second stage. For the purpose of estimating performance effect, we implemented an API module that redirected the interface of the existing AIPS in order to run simulation the following two scenarios in parallel:

Scenario-1: Invoice extraction in the original way without integration with our correction module.
Scenario-2: Invoice extraction in the new way integrated with our correction module.

A schematic diagram for this test procedure is demonstrated in (Fig. 8). Using the data set prepared as described in 4.1, a test simulator invokes AIPS. Using the intermediate output, the tester checks PO numbers (read from the input invoice) with the one in the ground truth set. If the query on PO number fails, then the process is recorded as failure and start over with the next document in the data set. If the PO number found is matched, then the tester checks the number of lines detected with the ground truth data. If the number of lines is not matched

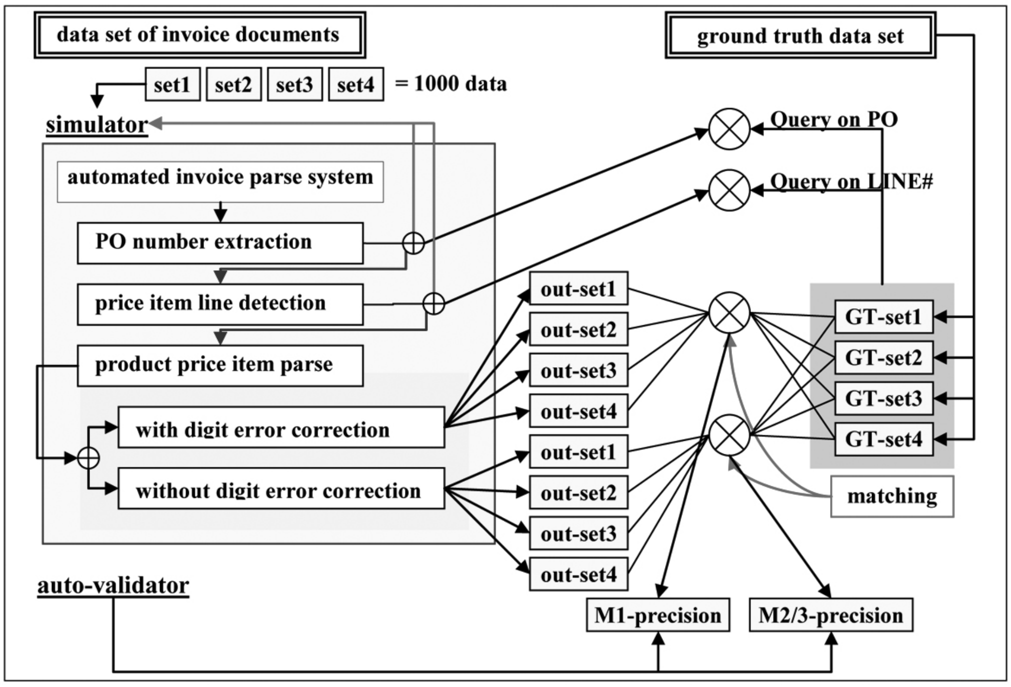

GSA # GS-06F-0027L

CARRIER	CUSTOMER ORDER NO./BUYERS NAME	SALES AGENT	DATE SHIPPED	OUR ORDER NO.
UPSG	1374145	[redacted]	02/09/2010	MPH079

ITEM NUMBER *	DESCRIPTION	PRICE	UNIT	ORDERED	SHIPPED	AMOUNT
	*					
	BUYER: [redacted]					
P 47382	5/16-18 SUPT ALLOY GUN TAP	33.87	EA	4	4	135.48
P 47306	10-24 SUPERTANIUM ALLOY TAP	19.34	EA	5	5	96.70
P 68905	10-24 SUPT ALLOY BOTTOM TAP	17.56	EA	5	5	87.80
FRT. CHRGS.						8.49

***** Please note our new remittance address and new Fed ID# *****

(Fig. 7) An example of price item line detection. The red rectangles indicate the detected regions



(Fig. 8) A test procedure for evaluation of performance effect by the digit error correction

then the process is recorded as failure and start over with new data. If the number of lines is matched with the value in the ground truth set, the test simulator forks the process into the two sub-processes: the one runs with correction model to parse product items (shipped quantity, unit cost, and amount), while the other runs without correction model. The outputs from two sub-processes are compared by auto-validation process.

For estimation of the statistics, we used the ground truth data constructed at section 4.1. The auto-validator performed a matching process to determine "correct" (matched) or "wrong" (mismatched). The matching was quite a stringent as the following:

· Search the matching PO number from the ground truth data, if it cannot match then the auto-validator registered as "wrong";
· Match with the number of price lines, if the match fails then the auto-validator registered as "wrong";
· Match with the field values of unit cost and shipped quantity, if the match fails then the auto-validator registered as "wrong";

4.3 Evaluation Measure

The auto-validator is a binary classifier with the categories of "correct" and "wrong". We employed the two typical measures for the binary classifier, "precision" and "recall rates". The precision is a value of the number of correct match divided by the number of sample data, and the recall is a value of the number of actual match divided by the number of correct match.

The test simulator, described in section 4.2, provides the statistics, {{p1, r1}, {p2, r2}, {p3, r3}}, from the three methods (M1, M2, and M3). "{pk, rk}" indicates the precision and the recall of Mk , for k=1, 2, 3, respectively.

4.4 Performance Analysis

M#, Pr, and

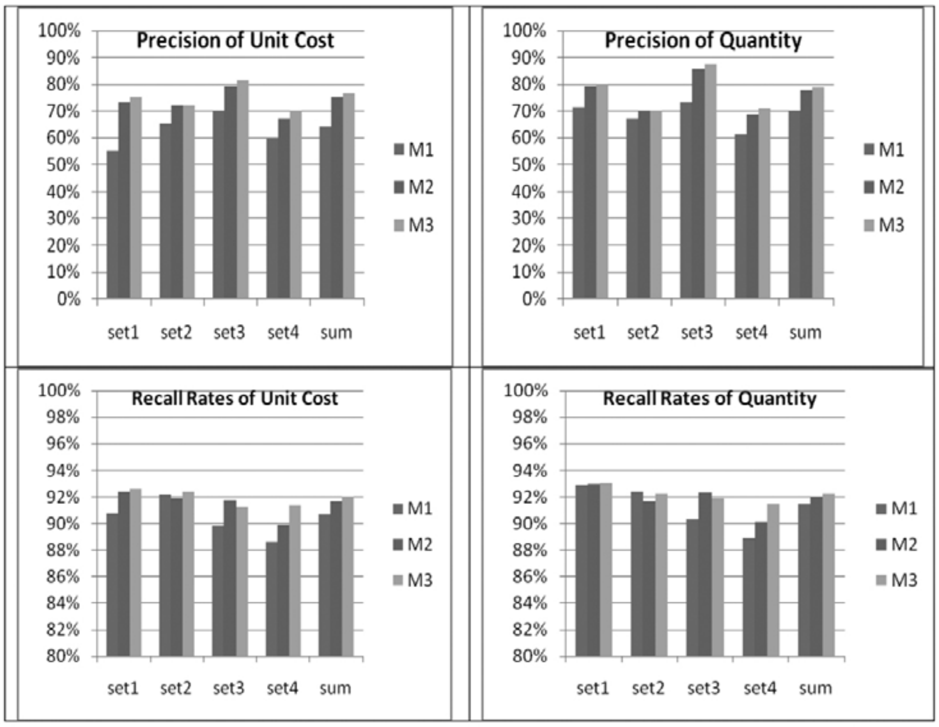
Rc denotes the number of data matched, the precision, and the recall rates, respectively.

<Table 1> presents the results of the experiments. In the table the method 1 (M1) indicates that the tests procedure without including our module, the method 2 (M2) indicates that the tests procedure with digit correction using multiple stream (without context sensitive correction), and the method 3 (M3) indicates that the test procedure with full version of our model. M#, Pr, and Rc denotes the number of data matched, the precision, and the recall rates, respectively.

〈Table 1〉 experimental results of preformance effect by the digit correction algorithm

Data ID	Method	Total Sample	PO Match	Line # Match	Unit Cost			Shipped Quantity		
					M#	Pr	Rc	M#	Pr	Rc
Set1	M1	240	239	216	119	55.0%	90.8%	154	71.3%	92.9%
	M2				158	73.1%	92.4%	171	79.2%	93.0%
	M3				162	75.0%	92.6%	173	80.1%	93.1%
Set2	M1	340	334	274	179	65.3%	92.2%	184	67.2%	92.4%
	M2				198	72.3%	92.0%	192	70.1%	91.7%
	M3				198	72.3%	92.4%	193	70.4%	92.2%
Set3	M1	230	223	183	128	70.0%	89.4%	134	73.2%	90.3%
	M2				145	79.2%	91.7%	157	85.8%	92.4%
	M3				149	81.4%	91.3%	160	87.4%	91.9%
Set4	M1	190	188	132	79	60.0%	88.6%	81	61.4%	89.0%
	M2				89	67.4%	89.9%	91	68.9%	90.1%
	M3				93	70.5%	91.4%	94	71.2%	91.5%
Sum	M1	1000	984	805	505	64.3%	90.7%	553	70.5%	91.5%
	M2				590	75.2%	91.7%	611	77.8%	92.0%
	M3				602	79.7%	92.0%	620	79.0%	92.3%

The results show that M3 gained over M1 about 15.4% and 8.5% for the unit cost and the shipped quantities, respectively, and M2 gained over M1 about 10.9% and 7.3% for the unit cost and the shipped quantities, respectively. The performance enhancement of the context sensitive correction is about 4.5% and 1.2% for the unit cost and the shipped quantities, respectively. For the purpose of visualization of the precision and recall rates, (Fig. 9) is presented. The graphs show consistent improvement in the precision and recall rates.



(Fig. 9) comparison graphs of the precision and the recall rates in terms of correction methods (M1, M2, and M3)

As seen in the <Table 1> 805 number of data (out of 1000) have passed the line number matching stage. The rest of 195 invoices already had problem of parsing the correct information regardless of digit correction. We considered the 805 invoices as the validatable set.

As summarized in <Table 2> for the querying the unit cost field, 85 additional invoice documents were automatically validated by using the multiple streams (M2) and 97 additional invoice documents were validated by using context sensitive correction (M3), for the querying the shipped quantity field, 58 additional invoice documents were automatically validated by using the multiple streams (M2) and 69 additional invoice documents were validated by using the context sensitive correction (M3).

〈Table 2〉 number of invoices recovered by digit error correction

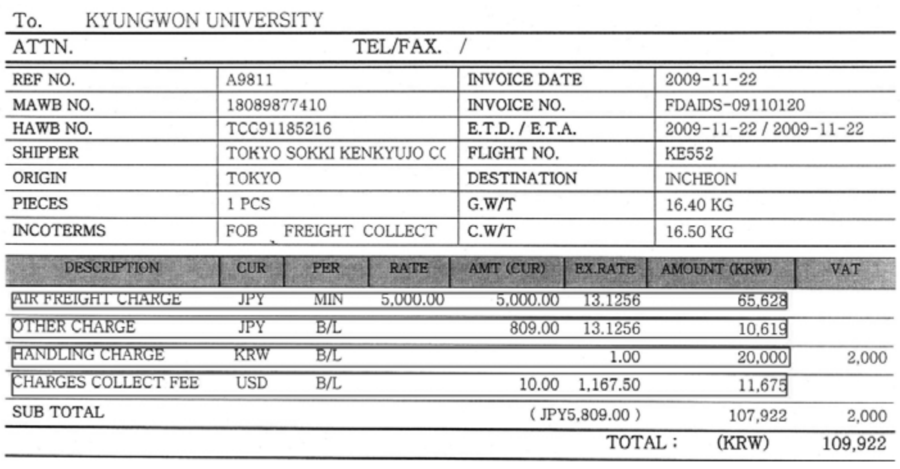
Data ID	Method	Validatable Data	Unit Cost		Shipped Quantities	
			Recovered #	%	Recovered #	%
Set 1	M2	216	39	18.1%	17	7.9%
	M3		43	19.9%	19	8.8%
Set 2	M2	274	19	6.9%	8	2.9%
	M3		19	6.9%	9	3.3%
Set 3	M2	183	17	9.3%	23	12.6%
	M3		21	11.5%	27	14.8%
Set 4	M2	132	10	7.6%	10	7.6%
	M3		14	10.6%	13	9.8%
Sum	M2	805	85	10.6%	58	7.2%
	M3		97	12.0%	68	8.4%

5. Conclusion

The automated document system (or document understanding system) has drawn a lot of attention in the industry. The core technology for the system is a well behaving query and correction model. There have been intense efforts and evolutions for the dictionary based text query and recovery model, but not much for the non-alphabetic texts.

To the best of our knowledge, the proposed model of query and correction for the digits using multiple OCR text streams followed by the k-gram of the postings list is a new attempt. At the previous section we showed that the use of multiple streams improved the performance of an automated document system more than 7%, which implies, considering the strict matching criterion on the outputs from the invoice parsing system, the correction of OCR errors has been significantly (at least 8%) enhanced.

The context sensitive correction module adopted in this model required a problem domain specific formula: for an invoice document, three numbers in a line should satisfy a relational formula $z = x * y$ (unit cost * quantities = extension amount). There should be a generic scheme for query reformulation, e.g. apply all the possible operations like addition, subtraction, multiplication, division, and etc and automatically find out the relational formula through some form of training processes. In our study, the context sensitive correction improved the overall accuracy of the system at least about 1.2%. In (Fig. 10) we


To. KYUNGWON UNIVERSITY

ATTN. TEL/FAX. /

REF NO.	A9811	INVOICE DATE	2009-11-22
MAWB NO.	18089877410	INVOICE NO.	FDAIDS-09110120
HAWB NO.	TCC91185216	E.T.D. / E.T.A.	2009-11-22 / 2009-11-22
SHIPPER	TOKYO SOKKI KENKYUJO C(	FLIGHT NO.	KE552
ORIGIN	TOKYO	DESTINATION	INCHEON
PIECES	1 PCS	G.W/T	16.40 KG
INCOTERMS	FOB FREIGHT COLLECT	C.W/T	16.50 KG

DESCRIPTION	CUR	PER	RATE	AMT (CUR)	EX.RATE	AMOUNT (KRW)	VAT
AIR FREIGHT CHARGE	JPY	MIN	5,000.00	5,000.00	13.1256	65,628	
OTHER CHARGE	JPY	B/L		809.00	13.1256	10,619	
HANDLING CHARGE	KRW	B/L			1.00	20,000	2,000
CHARGES COLLECT FEE	USD	B/L		10.00	1,167.50	11,675	
SUB TOTAL				(JPY5,809.00)		107,922	2,000
					TOTAL :	(KRW)	109,922

(Fig. 10) An example requires general context correction demonstrate the necessity of generic context correction. This invoice does not contain a formula $z = x * y$ with the variables of the quantity, unit cost, and extension but it contains $z = x * y + a$ with the variables of 'AMT', 'EX. RATE', 'AMOUNT', and 'VAT'.

References

[1] C. D. Manning, P. Raghavan, and H. Schultze, "An Introduction to Information Retrieval", Cambridge University Press, 2008.

[2] R. Kosala and H. Blockeel, Web Mining Research: A Survey ACM SIGKDD Explorations Newsletter, vol. 2, no. 1, pp. 1-15, 2000.

[3] C. Mascolo, "Specification, analysis and prototyping of mobile code systems", PhD thesis, Universita di Bologna, 2001.

[4] A. Perez, F. Rodriguez, and B. Terrazas, "Ontology based legal information retrieval to improve the information access in e-government", IWWW conf. Proc. 15th ICWWW, 2006.

[5] E. L. Rissland, J.J. Daniels, "A hybrid CBR-IR approach to legal information retrieval", ICAIL, Proc. 5th ICAIL, pp52-61, 1995.

[6] T. Honkela, S. Kaski, K. Lagus, T. Kohonen, "WEBSOM - self organizing maps of document collections", Proceedings of WSOM, pp.310-315, 1997.

[7] H. Li, D. Doermann, O. Kia, "Automatic Text Detection and Tracking in Digital Video," IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL 9, PART 1, pages 147-155, 2000.

[8] Rasagna, V., Kumar, A., Jawahar, C. V., and Manmatha, R. "Robust Recognition of Documents by Fusing Results of Word Clusters," ICDAR. IEEE, 566-570. 2009.

[9] Li, L. and Tan, C. L., "Improving OCR Text Categorization Accuracy with Electronic Abstracts," DIAL. IEEE, 82-87, 2006.

[10] Xu, Y. and Nagy, G. "Prototype Extraction and Adaptive OCR," IEEE Trans. Pattern Anal. Mach. Intell. 21, 1280-1296, 1999.

[11] Avi-Itzhak, Hadar I. and Diep, Thanh A. and Garland, Harry, "High Accuracy Optical Character Recognition Using Neural Networks with Centroid Dithering," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 17, 1995.

[12] F. Drira, F. LeBourgeois, H. Emptoz, "Document Images Restoration by a New Tensor Based Diffusion Process: Application to the Recognition of Old Printed Documents," ICDAR, pp. 321-325, 2009.

[13] U. Garain, M. P. Chakraborty, D. Majumder, "Improvement of OCR Accuracy by Similar Character Pair Discrimination: an Approach based on Artificial Immune System," The 18th ICPR'06, 2006.

[14] Garain, U., Jain, A., Maity, A., Chanda, B., "Machine reading of camera held low quality text images: An ICA based image enhancement approach for improving OCR accuracy,", ICPR08(1-4)., 2008.

[15] Koga, M., Mine, R., Kameyama, T., Takahashi, T., Yamazaki, M., and Yamaguchi, T., "Camera-based Kanji OCR for Mobile-phones: Practical Issues," ICDAR. IEEE, 635-639, 2005.

[16] K. Shin, B. Kang, and K. Park, "Super-resolution Iris Image Restoration using Single Image for Iris Recognition", KSII Trans. Internet and Information System, v. 4, no. 2, 2010.

[17] F. Daniyal, M. Taj, and A. Cavallaro, "Content and task-based view selection from multiple video streams," Multimedia Tools Appl., v. 46, no. 2-3, pp. 235-258, 2010.

신 현 경
e-mail : hyunkyung@kyungwon.ac.kr
2002년 State University of New York at Stony Brook. 대학원 응용 수학과 (공학박사)
2008년~현 재 경원대학교 수학정보학과 조교수
관심분야 : Image Processing. Neural Network. Machine Learning.