Efficient Automatic OCR Word Validation Using Word Partial Format Derivation and Language Model

Siyuan Chen, Dharitri Misra, and George R. Thoma
U.S. National Library of Medicine
Bethesda, MD 20894, USA

ABSTRACT

In this paper we present an OCR validation module, implemented for the System for Preservation of Electronic Resources (SPER) developed at the U.S. National Library of Medicine.[1] The module detects and corrects suspicious words in the OCR output of scanned textual documents through a procedure of deriving partial formats for each suspicious word, retrieving candidate words by partial-match search from lexicons, and comparing the joint probabilities of N-gram and OCR edit transformation corresponding to the candidates. The partial format derivation, based on OCR error analysis, efficiently and accurately generates candidate words from lexicons represented by ternary search trees. In our test case comprising a historic medico-legal document collection, this OCR validation module yielded the correct words with 87% accuracy and reduced the overall OCR word errors by around 60%.

1 Introduction

Optical Character Recognition (OCR) plays an important role in digital preservation systems to transform printed material into digital format. However, for historical documents, due to somewhat poor quality of the original pages, the accuracy of the OCR output is not always reliable. Errors in OCR words lead to subsequent performance degradation in indexing, archiving and retrieving the scanned documents in a digital preservation system.[2] In the System for Preservation of Electronic Resources (SPER), which has been developed at the U. S. National Library of Medicine for the long term preservation of biomedical resources,[1,3] errors in OCR output poses problems, since one of the main features of SPER is to automatically identify and extract descriptive metadata from the textual content of the documents to be preserved. In SPER, an OCR console module, based on the FineReader OCR Engine, is used to generate features for each segmented character, which are then used for document layout recognition and descriptive metadata extraction. In contrast to the highly successful layout recognition based on Support Vector Machine and Hidden Markov Model, metadata extraction using string matching techniques was less successful - over 50% of errors being caused by the erroneous OCR output. This highlighted the need for an automated OCR word validation and correction tool as an alternative to costly manual inspection and editing.

Over the past 20 years many OCR post-processing systems have been developed;[4] however, they were not found to be satisfactory for our purpose. Most of these systems either combine a character confusion matrix with language models under a probabilistic model,[5–7] or use approximate string matching techniques to look up the words from lexicons,[8] and are quite successful in choosing the optimal replacement word from a candidate set. However, to our knowledge none of these systems have used the confidence features or any OCR error analysis to generate the candidates efficiently. Consequently, these systems exhibit either slow speed due to a large candidate set, or low accuracy by using small vocabularies. In earlier work by ISRI,[9] the authors proposed an interactive OCR correction model that generates the candidate words directly from a confusion list and so is less dependent on the lexicons. But their system is semi-automatic in nature, requiring the user to select the segmentation positions and choose the replacement from the candidate list.

Most commercial OCR software systems provide a set of features for users to tune the OCR engine for their specific applications; but in our experience the underlying logic and use of these features are unclear. For example, it is very difficult to use the confidence features of a recognized character in the FineReader OCR engine output since the statistics of these values are unpredictable due to different qualities of documents. Similarly, the use of dictionary option under which user is allowed to install domain specific lexicons to improve the OCR quality simply gives a list of possible correct words chosen from these lexicons for each suspicious word, as determined by some internal metric. There is no automatic word validation.

Document Recognition and Retrieval XVII, edited by Laurence Likforman-Sulem, Gady Agam,
Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 7534, 75340O · © 2010 SPIE-IS&T
CCC code: 0277-786X/10/$18 · doi: 10.1117/12.838853

To alleviate the above difficulties, we have developed an OCR word validation module to simultaneously verify and correct the words marked by the OCR engine as suspect. First, it generates a group of suspicious words according to the binary confidence feature at the OCR character level. Next, for each such suspicious word, it derives the partial formats that contain wild cards, and then matches them approximately to the lexicon words organized by a Ternary Search Tree. The set of matched words constitute the candidates for replacement. To choose an optimal replacement from the candidates, we built an N-gram language model to compute the probability of the OCR word string of length 5 that is centered with the suspicious word. In our implementation, we do not consider general segmentation errors but cover the situations when one token is erroneously converted into another, or two tokens are collapsed into one, which are the most frequent errors in our application.

This paper is organized as follows: we present the motivation and the framework of our OCR validation module in Section 1; formulate the partial word format derivation and N-gram model in Section 2; describe the training and testing methods in Section 3; discuss the system performance in Section 4; and finally, present the conclusions in Section 5.

2 Method

2.1 Generation of Suspicious Words

The FineReader OCR Engine (Ver. 8.1) used by SPER assigns a confidence value to each recognized character by a binary feature *isSuspicious*. In addition, OCR words that are not found in the built-in dictionaries may also be indicated by a binary feature *isFromDict*. So our target suspicious words are the OCR words that contain characters with positive values of *isSuspicious*, and that cannot be found from the built-in dictionary, indicated by negative values of *isFromDict*.

2.2 Word Partial Format Derivation

A partial format is defined as a word token that contains one or more wild cards ".", where each wild card represents an unknown character. In our application we define *word partial format derivation* as a process to estimate the partial formats of an OCR word by a combination of the positions of its low confidence characters and certain character transformation assumptions given below. Then we use the estimated partial formats to generate the candidate words by approximate string matching.

Errors in a word produced by the OCR may be seen as transformation errors in one or more characters. In our application we consider three kinds of erroneous character transformation: a character is transformed into another character, a character is transformed into two characters, and two characters are transformed into a single character. Considering the different positions of low confidence characters (marked in the OCR) in the second case, we have five kinds of erroneous transformations as demonstrated in the examples in Table 1, where c in the left column is the correct character, e is the error character, and the low confidence character is marked by a "*".

Table 1. OCR Conversion Assumption

	Character Conversions	Examples
(1)	$c \rightarrow e^*$	"CHA**R**GE"→"CHA**B**GE"
(2)	$c \rightarrow e_1^* e_2$	"infor**m**ation"→"infor**rn**ation"
(3)	$c \rightarrow e_1 e_2^*$	"A**d**ulteration"→"A**cl**ulteration"
(4)	$c \rightarrow e_1^* e_2^*$	"S**u**lphate"→"S**ii**lphate"
(5)	$c_1 c_2 \rightarrow e^*$	"libe**ll**ant"→"libe**U**ant"

Based on the above assumptions, the process of partial format derivation is to estimate the partial formats by iteratively replacing the low confidence characters and their neighbors with the appropriate number of wild cards. Different replacements result from different transformation rules shown in Table 1. So a suspicious word with t low confidence characters could generate up to 4^t partial formats, the maximum value occurring when all of the low confidence characters appear inside the word and none of them either is next to another low confidence character or share the same neighbor with another low confidence character, i. e., when all of the five rules except for the rule (4) can be applied to each of the t low confidence characters. For example, the suspicious word "Agr**i**cu**H**ure" with two low confidence characters "i" and "H"will generate the following $4^2 = 16$ partial formats:

"Agr.cu.ure", "Agr.c.ure", "Agr.cu.re", "**Agr.cu..ure**", "Ag.cu.ure", "Ag.c.ure", "Ag.cu.re", "Ag.cu..ure", "Agr.u.ure", "Agr..ure", "Agr.u.re", "Agr.u..ure", "Agr..cu.ure", "Agr..c.ure", "Agr..cu.re", "Agr..cu..ure"

We can see that among these the partial format "Agr.cu..ure" can be matched to the word "Agriculture".

2.3 Approximate String Matching by Ternary Search Tree

We represent our domain-specific lexicons by Ternary Search Trees. The Ternary Search Tree offers a fast algorithm to sort and search strings, which are more efficient than hashing and other commonly used search methods. Each node of the Ternary Search Tree contains a *split character* and three pointers to its *low*, *high* and *equal* child nodes (based on the split character). When sorting or searching a query word, the characters of the word are iteratively compared with the split characters of nodes in the search path to select the next search direction from the three branches; in the case of sorting: new nodes will be inserted at the end of the search path; in the case of searching: a boolean value "true" will be returned when the search process ends in comparing the last character of the word and the split character of a leaf node, and finds them be equal, otherwise "false" will be returned. Appropriate adjustment of the splitting rules can yield more advanced search methods, such as partial-match search (where the query word is allowed to contain"don't care characters"), and nearest search (where the matched words are within a specified Hamming distance from the query word). Details of these search algorithms appear in literature.[10]

2.4 Probability of Matching by Edit Transformation

The edit distance between two strings is defined as the minimum number of edit operations to transform a test string to a reference string by replacement, insertion or deletion of characters. In the original algorithm[11] the cost functions of insertion and deletion are set to 1; the cost of replacement is 0 when one character is replaced by itself, and 1 otherwise. The edit distance can be computed by Viterbi algorithm, discussed below.

If we substitute the original cost functions of the three edit operations with their probability functions, we can model the probability that a true word string w is transformed into an OCR word string o by the dynamic programming process:

$$Pr(o_{(1,j)}|w_{(1,i)}) = max \begin{cases} Pr(o_{(1,j-1)}|w_{(1,i)}) * P_I(o_j) \\ Pr(o_{(1,j)}|w_{(1,i-1)}) * P_D(w_i) \\ Pr(o_{(1,j-1)}|w_{(1,i-1)}) * P_R(w_i, o_j) \end{cases} \quad (1)$$

where $P_I(o_j)$ is the probability that the o_j is inserted; $P_D(w_i)$ is the probability that w_i is deleted; and $P_R(w_i, o_j)$ is the probability that w_i is replaced by o_j.

In[7] $P_I(x)$ is simply taken as the ratio of inserting the character x over the number of appearances of x in the OCR word set. This estimate will not be accurate unless we know the OCR word and the true word simultaneously. In[5] this probability is estimated by the ratios of inserting x over the the total number of letters in the training set without considering any context information. In our model, however, we estimate the probability of insertion, deletion and replacement based on the current character in the path of Viterbi decoding, which, in our experience, is more accurate.

$$\begin{aligned} P_I(x) &= P_I(x|y) = \tfrac{num(ins(x)|y)}{num(y)} \\ P_D(x) &= \tfrac{num(del(x))}{num(x)} \\ P_R(x,y) &= \tfrac{num(rep(x,y))}{num(x)} \end{aligned} \quad (2)$$

In Equation 2, $num(x)$ is defined as the number of operations applied in transforming true words to the OCR words in a ground truth set when the current character is x in the path of Viterbi decoding. Note that $num(x)$ is different from the total number of appearances of x in the ground truth set. And $num(ins(x)|y)$ is the number of insertions of x when the current character is y.

2.5 N-gram Model Formulation

In order to use the context information around the suspicious word, we choose the OCR word from the candidates such that the joint probability of true word string W and OCR word string O is maximum, i.e.

$$w_o = \arg\max_{w \in C} Pr(W, O) \quad (3)$$

Assuming that the appearance of a specific word solely depends on its previous bi-gram words, we can model the probability $Pr(W, O)$ as:

$$Pr(W, O) = \prod Pr(w_i|w_{i-2}, w_{i-1}) * Pr(o_i|w_i) \tag{4}$$

Note that in this equation the indexes of w and o refer to their locations in W and O respectively.

Finally, we choose the optimum replacement word by making the assumption that suspicious words occur in isolation; that is, the OCR words around the suspicious word are true words. This is based upon the following observations pertaining to our test cases (discussed in Section 4.1).

(a) For only about 10% suspicious words, the neighbor or next nearest neighbor is also a suspicious word.

(b) In over 70% cases, the neighboring suspicious words do not need to be replaced.

Making the above assumption, and ignoring the items not related with suspicious word w_s, we use the following equation to choose the optimum replacement word w_o:

$$w_o = \arg\max_{w \in C} \tilde{Pr}(W, O|w_s = w) \tag{5}$$

$$\tilde{Pr}(W, O) = \sum_{i=s}^{s+2} log\ Pr(w_i|w_{i-1}, w_{i-2}) + log\ Pr(o_s|w_s) \tag{6}$$

In the above equations we simply use "$w_s = w$" to denote the assumption that the true form of the OCR word w_s is w. Note that here we ignore the general segmentation errors, (which require the construction of more complicated models as in[12]), since the associated cost factor is very high and so treat them them differently as explained in Section 3.3 (a).

Similarly we estimate the N-gram probability when the suspicious word is correct but unknown for the collected lexicons by the following equation:

$$\begin{aligned} \tilde{Pr}(W, O|w_s = w_s) = \tilde{Pr}(W, O|w_s = \text{“unknown”}) \\ + log\ Pr(w_s \text{ is not in the vocabulary}) \end{aligned} \tag{7}$$

In Equation 7 we substitute w_s into the probability matching function and substitute "unknown" in the tri-gram model. The resulting value is corrected by the probability that w_s is not in the vocabulary, which we estimate by Lidstone's Law:

$$Pr(w_s) = \frac{\lambda}{N + B\lambda} \tag{8}$$

where N is the size of the data set and B is the size of the lexicon. We set λ equal to 0.5 in our implementation.

3 Implementation

3.1 Preparing the Data Set and Lexicons

Currently SPER has been applied successfully to extract descriptive metadata from medico-legal historical documents from the Food and Drug Administration, which record notices of judgment (NJ) of court cases for misbranding and adulteration of foods, drugs and devices, and cosmetics.[1] From this collection of over 40,000 NJ pages, we choose 14,150 processed pages to train and test the OCR validation module.

The success of the OCR correction module depends heavily on the use of appropriate lexicons.[8] The lexicons we collected from online resources for processing NJ text include the general English lexicon, personal names, U. S. place names and NLM's Specialist Lexicon. These are listed in Table 2.

The last entry in Table 2 is a domain-specific lexicon, which we created from the FDA documents in a semi-automatic fashion; (a) First we OCR the NJ pages in the data set and generate the OCR word set C; (b) then we look up the words in C in the known lexicons and record their frequencies; (c) we manually validate the unknown words with high frequency (set as 50 in our implementation); (d) finally, we place the remaining unknown words into the categories of"digit", "unknown" and "others" by regular expression checking. Our final FDA lexicon contains about 33,000 known words.

Table 2. Collected Lexicons.

Lexicons	Sources	Constitution
General English	CMU AI Repository[13]	around 10,000 words
Personal Names	CMU AI Repository	13484 family names 8608 given names
Place Names	Census 2000 U.S. Gazetteer[14]	16964 U.S. places
Specialist	NLM/LHC NLP Repository[15]	360688 words in biomedical area
FDA NJ	FDA NJ documents	33009 words

3.2 Training Algorithms

In order to compute the character transformation probability in Equation 2, it is necessary to obtain a ground truth data set containing OCR word and true word pairs. We obtain this data set in two phases. In the first phase, we add pepper salt noise to the original document pages and OCR the noisy pages; then we match the OCR words in the noisy text to the original text by edit operations at the page, text line, and word token levels; we record the word pairs in which the original OCR word can be found in the known lexicons; we validate the resulting ground truth pairs and estimate the initial transformation probabilities based on them. In the second phase we use the initial transformation probabilities to correct the suspicious words and re-estimate the probabilities based on the corrected results.

For the N-gram model, since we cannot observe all the N-grams in the limited data set, we estimate the frequency of bi-grams and tri-grams by the Witten-Bell smoothing algorithm described in.[16] We also compress tri-gram and bi-gram lexicon by Pearson's chi-square test to save space and improve the processing speed. We simply remove all tri-grams and bi-grams with sample size lower than 20, or between 20 and 40 but having expected value less than 5 in the chi-square test table according to Snedecor and Cochran.[17] We obtain 16,377 bi-grams and 14,304 tri-grams from the data set of 6,697,287 words (drawn from the training set of 14,150 pages).

3.3 Validation Algorithm

We automatically validate the low confidence suspicious word by following the general steps below:

- *Step 1*: OCR the incoming page, generate the OCR text lines with various font features at the character level.
- *Step 2*: Generate the suspicious words according to the binary confidence feature.
- *Step 3*: Look up each suspicious word w_{ocr} in the collected lexicons. If found, then return w_{ocr}, otherwise go to next step.
- *Step 4*: Derive the partial formats of w_{ocr} and match the partial words to the domain-specific lexicon. If the matched word set is not empty, then go to the next step, otherwise return w_{ocr}.
- *Step 5*: Get the optimal matched word w_{opt} according to Equation 5 and 6.
- *Step 6*: Verify that w_{ocr} is a regular word; return w_{opt} if the verification denied w_{ocr}, otherwise go to next step.
- *Step 7*: Reject or accept w_{opt} by comparing the values of $\tilde{Pr}(W, O|w_{opt})$ and $\tilde{Pr}(W, O|w_{ocr})$ according to Equation 6 and 7. If w_{opt} is rejected, return w_{ocr} instead.

This validation process is also graphically presented in Figure 1.

In our implementation, this general validation procedure is augmented by a number of practical enhancements. For example:

(a) If no replacement is found for a suspicious word after the final step, we check if the word is actually two different words connected by some noisy character. If so, we validate these two words separately. Similarly, if a suspicious word spans two lines, we determine if it is really two different words, by combining heuristic methods and the established formulas; if so, we decompose and validate the individual components.

(b) When the wild cards appear at the beginning of a suspicious word, partial matching of a Ternary Search Tree will be quite slow, as it would require visiting almost all the nodes in the tree (that is: all the words in the lexicon) repeatedly. Instead of filtering out such partial words, we set up a ternary tree for this group of partial words first, and then recursively use the search result of the ascendance nodes over this tree. So we avoid most of the overlapped work.

(c) In Step 6 we use a set of regular expressions to check a suspicious word and determine if we need to replace it. These regular expressions include"if any non-word character except for some specific symbols appears in the word", "if any digit is surrounded by word characters", "if any lower case is followed by upper case except for some last names" and so on.

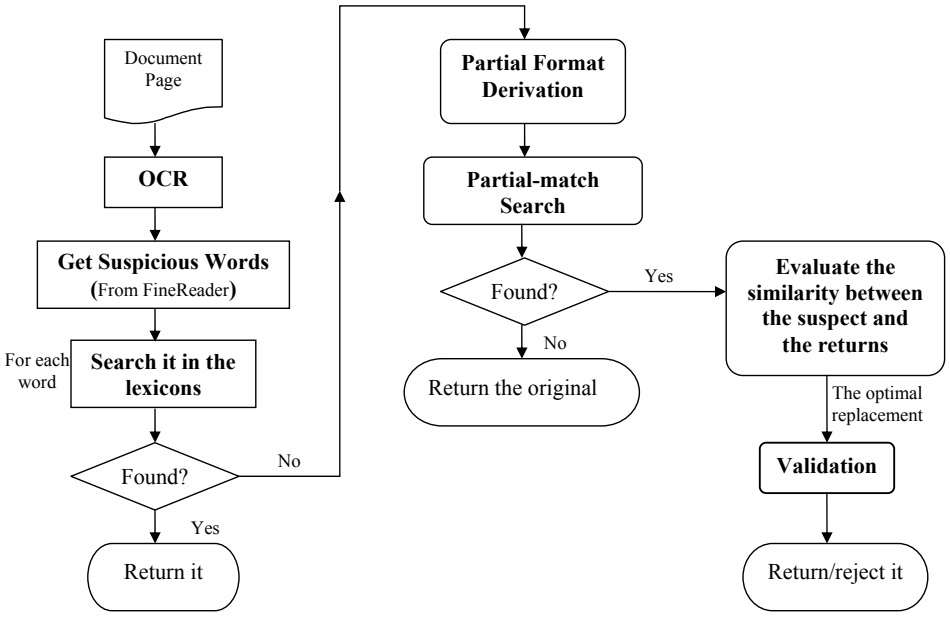

Figure 1. Framework of Validation

4 Experiments

We design two sets of experiments to evaluate the performance of our module. In the first set, we test the accuracy of validation by comparing the ground truth words and the validated words, and also measure the speed of validation; in the second set, we test the overall accuracy of OCR text before and after the validation. These tests are described in Section 4.1 and 4.2 respectively.

4.1 Performance of Word Validation

We apply the validation module to a randomly selected suspicious word set, containing 2028 pairs of OCR word and the corresponding truth word. The accuracies of verification (that the suspicious word is indeed an invalid one), and validation (selecting the corresponding valid word) are listed in Table 3. We define the Type I error, or false negative, as correcting the suspicious word when it is correct, and Type II error, or false positive, as retaining the suspicious word when it is incorrect. We define the accuracy of verification as the ratio of sum of true positive and true negative over the total number of suspicious words, and the accuracy of validation as the ratio of correct returns over the total suspicious words. Note that the accuracy of validation is lower than the verification since the replacement is not always correct. The module is conservative about correcting a word since we have 139 false positive decisions and only 15 false negative decisions. The 92.41% verification accuracy and 87.82% validation accuracy are satisfactory for our application.

Table 3. Accuracies.

Total suspicious words	2028
True positive	630
False positive	139
True negative	1244
False negative	15
Correct returns	1781
Accuracy of Verification	92.41%
Accuracy of Validation	87.82%

We evaluate the performance of our approximate string matching method by comparing it to two other commonly used methods: n-gram matching and cross correlation matching.[5,7] In n-gram matching we use the normalized correlation coefficient to measure the similarity between the bi-gram vector of the query and that of the lexicon word. In cross correlation matching we use the normalized peak of the cross correlation vector of the words as the similarity measure. For both of these methods we set the threshold such that the F score is maximum in the validation set. Since we have only one target word for each search we use the following formulation for F:

$$\begin{aligned} F &= \frac{1}{\alpha \times \frac{1}{P} + (1-\alpha) \times \frac{1}{R}} \\ &= \begin{cases} \frac{1}{\alpha \times \|M\| + (1-\alpha)} & \text{if true word is among } M \\ 0 \quad \text{otherwise} \end{cases} \end{aligned} \tag{9}$$

In the above equation P is *Precision* (the number of retrieved true words divided by the number of retrieved candidate words); R is *Recall* (the number of retrieved true words divided by the number of target words in the lexicon). In our case the number of retrieved true words is 0 or 1, and the number of target words is always 1. $\|M\|$ is the size of the returned match set M. We set $\alpha = 0.01$ since we try to keep $\|M\|$ below 100. Then we apply the two matching methods to the same test set, and record the performance as shown in Table 4. According to the experimental result our method outperforms the other two methods by achieving the maximum retrieval rate and the minimum number of average returned matches. The speed of search is lower than cross correlation matching by only 28 ms but with a much higher F-score. We implement the experiments in a PC with Intel(R) Pentium(R) D CPU 3.40GHz and 1.00 GB of RAM.

Table 4. Approximate String Matching Methods (1573 known words)

Methods	Retrieval %	matches #	F-score	Speed
Proposed	93.79%	45.3	0.88	79.8 ms
Bi-gram	89.72%	86.4	0.63	510.0 ms
Cross Corr.	85.32%	80.0	0.57	53.2 ms

4.2 Overall Improvement

In addition to evaluating the accuracy of each validation, we compare the OCR accuracies before and after the validation. We randomly select a test set of 400 pages, each with more than five suspicious words. We generate the ground truth text of these pages and evaluate the accuracy of OCR text before and after the validation. We apply the UNLV's OCR Frontier Toolkit[18] to obtain these accuracies at the character level and word level respectively. We list the test results in Table 5. The detailed algorithm to calculate these accuracies may be found in.[19]

Table 5. OCR Accuracy in char and word levels(400 pages)

	Before	After		Before	After
Total Chars	1451790	1451790	Total words	214909	214909
Errors	8700	5056	Misrecognized	4667	1890
Accuracy	99.4%	99.65%	Accuracy	97.83%	99.12%
Error reduction	41.6%			59.45%	

Although our test pages are originally of good quality (with average 11 suspicious words per page) and target words with regular format only, we still find considerable improvement after the application of our OCR validation model. As

the results show, the word accuracy has been improved by over 1% and the word errors have been reduced by around 60%, which we consider to be impressive.

5 Conclusions

We have presented an automatic OCR validation module that tries to solve the following problem: "What is the appropriate replacement for a word marked as low confidence by the OCR engine?". We develop a word partial format derivation process to search the possible replacements in lexicons organized by Ternary Search Trees. The proposed approximate string matching method is better than other commonly used methods in retrieval accuracy, and it is much faster than the bi-gram matching method. Then we combine the string character transformation probability and a tri-gram model to measure the probability that a query word is the truth word. We choose the word from the candidate set such that this probability is maximized. When considering the original word as a possible candidate we simultaneously realize the functions of verification and correction. In our experiment we obtain 92% verification accuracy and 87% validation accuracy for the test set; the overall word errors have been reduced by around 60%.

Acknowledgment
This research was supported by the Intramural Research Program of the National Library of Medicine, National Institutes of Health. We thank Dr. Jie Zou for his useful comments.

REFERENCES

[1] G. Thoma, S. Mao, D. Misra, J. Rees. Design of a Digital Library for Early 20th Century Medico-legal Documents. In *Proc. ECDL 2006*. Eds: Gonzalo J et al. Berlin: Springer-Verlag; LNCS. September 2006;4172:147-57.
[2] K. Taghva, J. Borsack, A. Condit. Evaluation of model based retrieval effectiveness with OCR text. In *ACM Transactions on Information Systems*, vol. 14 , no. 1, pages 64-93, 1996.
[3] D. Misra, S. Mao, J. Rees, G. Thoma. Archiving a Historic Medico-legal Collection: Automation and Workflow Customization. In *Proceedings of IS&T Archiving 2007 Conference*, pages 157-161, Arlington, Virginia, May 2007.
[4] K. Kukich. Techniques for automatically correcting words in text. In *ACM Computing Surveys*, vol. 24, no. 4, 1992.
[5] X. Tong, D. Evans. A Statistical Approach to Automatic OCR Error Correction in Context. In *Proceedings of the Fourth Workshop on Very Large Corpora*, pages 88-100,Copenhagen, Denmark, August 1996.
[6] O. Kolak, P. Resnik. OCR error correction using a noisy channel model. In *Proceedings of the Second International Conference on Human Language Technology Research*, pages 257-262, San Diego, California, March 2002.
[7] T. Lasko, S. Hauser. Approximate string matching algorithms for limited-vocabulary OCR output correction. In *Proceedings of SPIE Conference on Document Recognition and Retrieval VIII*, pages 232-240, San Jose, California, January 2001.
[8] G. Ford , S. Hauser, D. Le, G. Thoma. Pattern Matching Techniques for Correcting Low Confidence OCR Words in a Known Context. In *Proceedings of SPIE Conference on Document Recognition and Retrieval VIII*, pages 241-249, San Jose, California, January 2001.
[9] K. Taghva, E. Stofsky. OCRSpell: an interactive spelling correction system for OCR errors in text. In *International Journal on Document Analysis and Recognition(IJDAR)*, vol. 3, pages 125-137, 2001.
[10] J. Bentley and R. Sedgewick. Fast Algorithms for Sorting and Searching Strings. In *Proceedings of The Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 360-369, New Orleans, Louisiana, January 1997.
[11] R. O. Duda, P. E. Hart, D. G. Stork. *Pattern Classification* (Second Edition), pages 418-419. John Wiley & Sons, Inc., 2001.
[12] O. Kolak, W. Byrne, P. Resnik. A Generative Probabilistic OCR Model for NLP Applications. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, vol. 1, pages 55-62, Edmonton, Canada, 2003
[13] http://www.cs.cmu.edu/afs/cs/project/airepository/ai/areas/nlp/0.html
[14] http://www.census.gov/geo/www/gazetteer/places2k.html
[15] http://lexsrv3.nlm.nih.gov/SPECIALIST/index.html
[16] C. Manning, H. Schutze. *Foundations of Statistical Natural Language Processing* (Sixth printing with corrections in 2003). MIT Press, Cambridge, Massachusetts, and London, England.
[17] G. Snedecor, W. Cochran. *Statistical Methods* (Eighth Edition). Ames: Iowa State University Press, 1989.
[18] http://www.isri.unlv.edu/ISRI/OCRtk
[19] S. V. Rice. *Measuring the Accuracy of Page-Reading System*. Ph.D. dissertation.