8-1-2013

Utilizing Big Data in Identification and Correction of OCR Errors

Shivam Agarwal
University of Nevada, Las Vegas, AGARWAL3@unlv.nevada.edu

UTILIZING BIG DATA IN IDENTIFICATION AND CORRECTION OF OCR ERRORS

.

by

Shivam Agarwal

A Thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
August 2013

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Shivam Agarwal

entitled

Utilizing Big Data in Identification ad Correction of OCR Errors

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Kazem Taghva, Ph.D., Committee Chair

Laxmi P. Gewali, Ph.D., Committee Member

Ajoy K. Datta, Ph.D., Committee Member

Emma Regentova, Ph.D., Graduate College Representative

Kathryn Hausbeck Korgan, Ph.D., Interim Dean of the Graduate College

August 2013

ABSTRACT

by

Shivam Agarwal

Dr. Kazem Taghva, Examination Committee Chair

Professor of Computer Science

University of Nevada, Las Vegas

In this thesis, we report on our experiments for detection and correction of OCR errors with web data. More specifically, we utilize Google search to access the big data resources available to identify possible candidates for correction. We then use a combination of the Longest Common Subsequences (LCS) and Bayesian estimates to automatically pick the proper candidate.

Our experimental results on a small set of historical newspaper data show a recall and precision of 51% and 100%, respectively. The work in this thesis further provides a detailed classification and analysis of all errors. In particular, we point out the shortcomings of our approach in its ability to suggest proper candidates to correct the remaining errors.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express the appreciation to my committee chair, Dr. Kazem Taghva for all his support and guidance through every stage of this thesis research. Without his guidance and persistent help, completion of this thesis would not have been possible.

I am very thankful to my graduate coordinator, Dr. Ajoy K Datta for his help and invaluable support during my masters program. I extend my gratitude to Dr. Laxmi P. Gewali, and Dr. Emma Regentova for accepting to be a part of my committee. A special thanks to Edward Jorgensen for his help during my TA work. I would also like to take this opportunity to extend my gratitude to the staff of computer science department for all their help.

I would also like to extend my appreciation towards my parents and my friends for always being there for me through all phases of my work, for their encouragement and giving me their invaluable support without which I would never be where I am today.

TABLE OF CONTENTS

ABSTRACT…… iii

ACKNOWLEDGEMENTS……..…….…….. iv

LIST OF TABLES……vii

LIST OF FIGURES ……viii

LIST OF ALGORITHMS…….……ix

CHAPTER 1 INTRODUCTION……1
1.1 Related Work…… 2
1.1.1 Isolated Word Error Correction Techniques…….……2
1.1.2 Context Based Error Correction……..…..3

CHAPTER 2 BACKGROUND……..…..……...6
2.1 Working of OCR ……..…..….6
2.2 Classification of OCR Errors…….…..…7
2.2.1 Word Error and Non Word Error……....8
2.2.2 *Stopwords*……9
2.3 Used Methods in Detail……9
2.3.1 Longest Common Subsequence Algorithm……..…9
2.3.2 Levenstein Edit Distance……11
2.3.3 Character Confusion Matrix……...13
2.3.3.1 Using Confusion Matrix……...…..15
2.3.3.2 Laplace Smoothing…….....16

CHAPTER 3 PROPOSED APPROACH AND IMPLEMENTATION……....17
3.1 Proposed Approach……....17
3.2 Difference Between Related Work and Proposed Approach……...19
3.3 Methodology Details..... ……23

CHAPTER 4 EXPERIMENTS AND RESULTS…….31
4.1 Evaluation Criteria……...31
4.2 Data Collection……..... 32
4.2.1 Training Data ……...32
4.2.2 Testing Data 32
4.3 Results on Data Set 1…….. 33
4.3.1 Observations for Data Set 1……..34
4.4 Results on Data Set 2 ……37
4.4.1 Observations for Data Set 2……..38
4.5 Conclusion and Future Work…….……..…….39

Appendix..... ……...40

BIBLIOGRAPHY 50

VITA 53

LIST OF TABLES

Table 2.1 OCR Error Example ……7
Table 2.2 LCS matrix for the strings “ABCBDAB” and “BDCABA”...……...11
Table 2.3 Edit Distance Matrix for the strings “paces” and “pieces” ……...12
Table 2.4 Sample Frequency calculation Table ..……..……15
Table 2.5 Sample Structure of Confusion Matrix …….……15
Table 4.1 Precision-Recall values for Data Set 1 ……..34
Table 4.2 F-measure values for Data set 1 ……..……..34
Table 4.3 Precision-Recall values for Data Set 2 ..……..……38
Table 4.4 F-measure values for Data Set 2 ……..…..……38

LIST OF FIGURES

Figure 2.1 Standard OCR Procedure……...7
Figure 3.1 OCR Error Correction Procedure……. 22
Figure 3.2 Sample Error.txt and Original.txt……..24
Figure 3.3 Firing Query to *Google*.... ……26
Figure 3.4 Retrieval of *Google* Next Page Links ……...……....26
Figure 3.5 Keyword Extraction from Web Data ……27
Figure 3.6 Extraction of *Google* Suggestion ……..……28
Figure 3.7 Sample Candidate.txt file ……..29

LIST OF ALGORITHMS

2.1 Longest Common Subsequence Algorithm…….....10
2.2 Levenshtein Edit Distance Algorithm ……..13
3.1 Formal Description of Algorithm……..…….....…….……...…20

CHAPTER 1

INTRODUCTION

The trend to digitize paper based documents such as books and newspapers has emerged greatly in the past years. The aim is to preserve old manuscripts which were written before invention of word processor. Moreover, digitization helps in making non-digitized printed media widely available, distributable, and searchable online. For instance the Library of Congress (http://www.loc.gov/index.html) has huge historical digital collection, all of which has been digitized from paper based books so that they can be preserved well. According to estimation more than 200 million books are being published every year [1]. All these need to be digitized since it is impossible to store and manage all these on a computer. Many institutions have been engaged in large-scale digitization projects. For instance, Google have digitized over 20 million books [2] as a part of their Google Books service until March 2012. The next step is to apply the OCR (Optical Character Recognition) process, which will translate scanned image of each document into machine processable text [3]. OCR errors can occur due to the print quality of the documents, bad physical condition and the error-prone pattern matching techniques of the OCR process. In a report on the accuracy of OCR devices by ISRI [4], it has been observed that the accuracy of character recognition varied from 95.64 to 99.33, depending on the type of OCR devices used. The variation was highest for the poor quality pages. It has already been proven in a research connecting OCR with information extraction, including [5] and [6] that the quality of information extraction is reduced in the presence of OCR errors. There is a great need to do post processing of OCR text in order to correct errors. One way to process OCR text can be to manually

review the OCR output text by hand. But this process can be time consuming, error prone, and costly. Researchers have also proposed dictionary based error correction approach in which, a lexicon or a lookup dictionary is used to spell check OCR recognized words and correct them if they are misspelled [7]. But Dictionaries do not support proper and personal names, names of countries, regions, geographical locations, technical keywords and domain specific terms. One major drawback is that the content of a standard dictionary is static as it is not constantly updated with new emerging words. In order to overcome these issues Context-based error correction techniques were explored which perform error detection and correction on the basis of semantic context. In this thesis we have proposed an approach which performs context sensitive OCR error correction with the help of Big Data of Web.

1.1 Related work

There has been much effort in the field of correcting OCR errors. Post-processing is the last stage of an OCR system whose goal is to detect and correct spelling errors in the OCR output text.

1.1.1 Isolated Word Error Correction Techniques

These techniques do not take into consideration the surrounding context for error correction. The simplest technique is dictionary lookup, but lookup time can be large if dictionary size is huge. However hash tables can be used to gain fast access. The advantage is that it reduces large number of comparisons for sequential search in a dictionary. The disadvantage is the need to devise clever hash function that avoids collisions without requiring huge hash tables. To generate candidates for error correction minimum edit distance techniques, similarity key techniques, rule based techniques, n-

gram based techniques, and neural networks based techniques have been developed [8]. In one of the works [9], each word is classified and multi-indexed according to combinations of a constant number of characters in the word. Candidate words are selected fast and accurately, regardless of error types, as long as the number of errors is below a threshold. Levenstein [10] developed a method of choosing a substitution for error, based on minimum number of insertions, deletions or substitution. In the similarity key based technique, the idea is to map similarly spelled strings into similar keys. When a key is computed for a misspelled string, it provides a pointer to all similarly spelled words in the lexicon which may be accepted as candidates [11]. Yannakoudakis and Fawthrop [12] conducted a study to create a set of rules based on common misspelling pattern and used them to correct errors. Letter n-grams, including trigrams, bigrams, and unigrams have been used in OCR correctors to capture the lexical syntax of a dictionary and to suggest legal corrections [8]. A related work [13] provides a general overview of error correction techniques based on transition and confusion probabilities. In a work related with use of neural network, Cherkassky and Vassilas [14] use backpropagation algorithms for correction.

1.1.2 Context Based Error Correction

Still there is a class of errors that is beyond the reach of isolated-word error correction. This class consists of real word errors, i.e, errors in which one correctly error is substituted for another. These error type require information from the surrounding context for correction. One such approach is proposed by Xiang Tong and David A. Evans [15], based on statistical language modeling (SLM). It uses information from various sources such as letter n-grams, character confusion probabilities, and word

bigram probabilities. It achieves around 60% error reduction rate. There is a current research on a new post-processing method and algorithm for OCR error correction, based on huge database of *Google*'s online web search engine. One of the previous work [16] proposes a Post- Processing and context based algorithm for correcting non-word as well as the real- word OCR errors. The idea centers on using *Google*'s online spelling suggestion which retrieves a large number of tokens from all over the web and suggests the best possible candidate as a correction for errors occurred during OCR process. *Google*'s algorithm automatically examines every single word in the search query for any possible misspelling. It first tries to match the query, composed of ordered association of words, with any occurrence alike in *Google*'s index database. If the query is not found, *Google* tries to infer the next possible correct word in the query based on its *n*-gram statistics deduced from its database of indexed webpages. Then an entire suggestion for the whole misspelled query is generated and displayed to the user in the form of "*did you mean: spelling-suggestion*". This procedure has shown a tremendous improvement in OCR correction rate. Another approach [17] makes use of *Google* Web IT 5-gram dataset which is colossal volume of data statistics represented as word n-gram sequences with their respective frequencies, all extracted from online public web pages. This dataset is used as a dictionary to spell check OCR words by using their context. The query consists of OCR error in combination with four preceding words in OCR text. It is fed to *Google*DataSet, which then generates a list of potential candidates for error correction, along with their frequencies. The candidate with highest frequency is then chosen as the correction. This approach also showed improvements in OCR error corrections. In another approach [18] "dynamic" dictionaries were used via analysis of web pages that fit

the given thematic area. Twenty five non function word were extracted from OCR-corpus and searched as a disjunctive query in the web; a dictionary is then built from retrieved tokens. Candidate ranking is done based on frequency, edit distance, and ground truth data. This improved the quality of converted text. In a research work [19] it has been shown that correction accuracy is improved when integrating word bigram frequency values from the crawls as a new score into a baseline correction strategy based on word similarity and word frequency. A related research shows that dynamic dictionaries can improve the coverage for the given thematic area in a significant way [20].

Still these techniques can be improved by dynamic use of the most recent *Google* data set instead of stored data. Additionally advanced candidate selection algorithms and more efficient query formation techniques may improve results.

CHAPTER 2

BACKGROUND

2.1 Working of OCR

It involves the following basic steps:

1) Scanning the paper documents to produce an electronic image. Problems can arise if the quality of the original document is poor, or scanning equipment is poor. It can lead to errors in later stage.
2) Zoning [21] which automatically orders the various regions of text in the documents. Improper zoning can greatly affect the word order of the scanned material and produce an incoherent document.
3) The segmentation process breaks the various zones into their respective components (zones are decomposed into words and words are decomposed into characters). Errors can occur if text has broken characters, overlapping characters, and nonstandard fonts.
4) The characters are classified into their respective ASCII characters. Improper classification can also lead to erroneous substitution of characters. For instance character ‘e’ is often misrecognized as ‘c’ due to similar shapes. These errors differ from spelling mistakes which humans make. The figure 2.1 shows the typical OCR process:

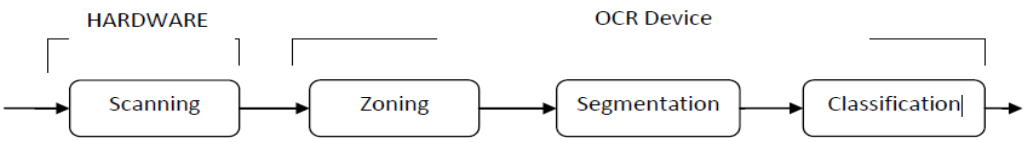

Figure 2.1: Standard OCR Procedure

2.2 Classification of OCR Errors

Before errors can be corrected they have to be identified and classified. A proper classification is important in order to know which kind of errors occur. In related work there is one main classification scheme which divides errors into two classes: *non-word* and *real-word* errors [15]. This classification is not sufficient, so a better classification introduced by Esakov, Lopresti and Sandberg [22] is considered, which divides OCR errors into six classes. Table 2.1 shows some typical example for each type of the errors:

1. Insertion of a character
2. Deletion of a character
3. Substitution of one character for another (1:1 Substitution)
4. Substitution of two characters for one (1:2 Substitution)
5. Substitution of one character for two (2:1 Substitution)
6. Substitution of two characters for two others (2:2 Substitution)

Error type	Example
Insertion	bat → ba t
Deletion	brought→ brough
1:1Substitution	j→ i, v→y , i→r
1:2 Substitution	n→ii , m→ rn
2:1 Substitution	cl→d , tl →k
2:2 Substitution	rw→ nr , rm →nn

Table 2.1: OCR Error Example

2.2.1 Word Error and Non Word Error

Essentially, there are two types of word errors: *non-word* errors and *real-word* errors[15]. A *non-word* error occurs when a word in the OCR text is interpreted as a string that does not correspond to any valid word in a given word list or dictionary. A *real-word* error occurs when a source-text word is interpreted as a string that actually does occur in the dictionary, but is different from the source-text word. For example, if the source text "how was the show" is rendered as "how was he shaw" by an OCR device, then "shaw" is a *non-word* error and "he" is a *real-word* error. Generally, non-word errors will never be found in any dictionary entry. While *non-word* errors might be corrected without considering the context in which the error occurs, a *real-word* error can only be corrected by taking context into account. Most traditional techniques for word-correction deal with *non-word* error correction and do not consider the context in which the error appears. But

for correcting OCR error efficiently, the context can be used as another source of information.

2.2.2 *Stopword*s

*Stopword*s can be defined as those words in the text that do not add to a document substance or meaning [23]. Most Information Retrieval techniques ignore the most commonly occurring *Stopword*s. The list might include words such as "the", "and", " a" , "that" , "but", " to" , "through" etc. For our work the list is taken from *Brown Corpus*.

2.3 Used Methods in Detail

2.3.1 Longest Common Subsequence Algorithm

The longest Common Subsequence (LCS) algorithm is string matching algorithm which finds the longest subsequence that two sequences have in common. It is based on dynamic programming where the problem is solved in terms of smaller subproblems.

Formally LCS problem is defined as follows: Given a sequence $X = (x_1, x_2 \ldots, x_n)$ and sequence $Y = (y_1, y_2 \ldots, y_m)$, find a sequence Z such that it is longest sequence and a subsequence to both X and Y .The subsequence is defined as a sequence $Z = (z_1 z_2 \ldots z_k)$, where there exists a strictly increasing sequence $(i_1, i_2, \ldots i_k)$ of indices of X such that for all $j=1 \ldots k$, $x_{ij} = z_j$ [24] . Basically the best of the three possible cases is taken:

1. The longest common subsequence of the strings $(x_1, x_2 \ldots, x_{n-1})$ and $(y_1 y_2 \ldots y_m)$,
2. The longest common subsequence of the strings $(x_1, x_2 \ldots, x_n)$ and $(y_1 y_2 \ldots y_{m-1})$,
3. If x_n is the same as y_m, the longest common subsequence of the strings $(x_1, x_2 \ldots, x_{n-1})$ and $(y_1 y_2 \ldots y_{m-1})$, followed by the common last character.

Let LCS (X_i, Y_j) represent the set of longest common subsequence of prefixes X_i and Y_j. This set of sequences is given by the following:

$$LCS(X_i,Y_j) = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ LCS(x_{i-1}, y_{i-1}) + 1 & \text{if } x_i=y_j \\ Longest(LCS(x_i, y_{j-1}), LCS(x_{i-1},y_j)) & \text{if } x_i \neq y_j \end{cases}$$

The complete algorithm is stated as follows:

Algorithm 2.1 Longest Common Subsequence Algorithm

FUNCTION LCSLength ($X_{[1..m]}$, $Y_{[1..n]}$)

```
1:  C = ARRAY(0..m, 0..n)
2:      For i := 0..m
3:          C[i,0] = 0
4:      For j := 0..n
5:          C[0,j] = 0
6:      For i := 1..m
7:          For j := 1..n
8:              IF(X[i] = Y[j])
9:                  C[i,j] := C[i-1,j-1] + 1
10:             Else:
11:                 C[i,j] := max(C[i,j-1], C[i-1,j])
12:   RETURN C[m,n]
```

Illustration by example

Let X be "ABCBDAB" and Y be "BDCABA". The longest common subsequence between X and Y is "BCBA" of length 4. An array C of dimensions m+1,n+1 is created and is initialized to 0. The table 2.2 shown below, which is generated by the function *LCSLength*, shows the lengths of the longest common subsequences between prefixes of X and Y. The $(i+1)^{th}$ row and $(j+1)^{th}$ column shows the length of the LCS between $X_{1...i}$ and $Y_{1...j.}$ The trace of longest common subsequence between strings X and Y at each iteration is shown in yellow:

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	**4**	4

Table 2.2: LCS matrix for the strings "ABCBDAB" and "BDCABA"

2.3.2 Levenshtein Edit Distance

Levenshtein-Distance is a concept from Information Retrieval [1]. It gives the minimum number of insertions, deletions and substitutions of single characters that are necessary in order to transform a string $x = x_1 \ldots x_n$ into another string $y = y_1 \ldots y_m$. It computes dissimilarity between two strings. It uses dynamic programming, a method of solving a

large problem by regarding the problem as the sum of the solutions to its recursively solved subproblems.

To compute edit distance *ed (x,y)* a matrix $M_{1\ldots n+1,1\ldots m+1}$ is constructed where $M_{i,j}$ is the minimum number of edit operations needed to match $x_{i\ldots i}$ *to* $y_{1\ldots j.}$ Mathematically, each matrix element is calculated as per equation below, where cost (a,b) =0 if $a=b$ and 1 otherwise. The matrix element $M_{0,0}$ is the edit distance between two empty strings.

$M_{0,0} = 0$

$M_{i,j}$ = Min { $M_{i-1,j} + 1$, $M_{i,j-1} + 1$, $M_{i-1,j-1} +$ cost(x_i , y_j) }

Table 2.3 below is an example of matrix produced to calculate the edit distance between the strings "paces" and "pieces". The minimum edit distance between the two strings is given by the matrix entry at position $M_{m+1,n+1}$ which is 2. The trace of the minimum distance path is shown in yellow.

		p	a	c	e	s
	0	1	2	3	4	5
p	1	0	1	2	3	4
i	2	1	1	2	3	4
e	3	2	2	2	2	3
c	4	3	3	2	3	3
e	5	4	4	3	2	3
s	6	5	5	4	3	2

Table 2.3: Edit Distance Matrix for the strings "paces" and "pieces"

Some more instances of edit distance between words are :

1 *ed* (bitten, bittem) =1 (substitution of 'n' with 'm')

2 *ed* (hittin hitting**)** **=**1 (insert 'g' at the end)

Algorithm 2.2 Levenshtein Edit Distance Algorithm

```
1: int FUNCTION count (string s1, string s2)
2:  m = s1.length()
3:  n = s2.length()
4:  for  i = 0 to m do
5:       v[i][0] = i
6:  end for
7:  for  j = 0 to n do
8:       v[0][j] = j
9:  end for
10: for  i = 1 to m
11:      for j = 1 to n
12:              if (s1[i-1] == s2[j-1]) then
13:                      v[i][j] = v[i-1][j-1]
14:              else
15:                      v[i][j] = 1 + min( min ( v[i] [j-1],v[i-1] [j] ), v[i-1] [j-1] )
16:              end if
17:      end for
18: end for
19: RETURN v[m][n]
```

2.3.3 Character Confusion Matrix

The Confusion matrix is designed to handle the interchange errors which occur most frequently during OCR process. The confusion matrix contains original characters A_i and their associated corrupted non original characters B_j. This is a probabilistic model

which can be used to enhance the process of best candidate selection among the possible original words, as a replacement for the OCR error. The probability that OCR produced B_j but B_j was actually A_i in the original text, is given by Bayes theorem:

$$p(A_i \mid B_j) = \frac{p(A_i) * p(B_j \mid A_i)}{\sum_{k=1}^{n} p(A_k) * p(B_j \mid A_k)}$$

The simple way is to compare both the clean text and OCR text character by character to compute the number of times character remains correct and number of times it is corrupted to some other character. Thereafter, using the formula used in [15] to compute the Character Confusion Probability we get:

$$pr(i \mid j) = \frac{num(sub(j,i))}{num(j)}$$

where-

- *num* (*sub*(*j*, *i*)) is the number of times the character *i* was corrupted to character *j* in the corresponding OCR text
- *num*(*j*) is number of times the character *j* occurred in the OCR text

Let us suppose there are 3 characters *i*, *j* and *l* with total occurrence of 1800 in the training data. Since we have both the OCR data and the clean data we can compute the Table 2.4. The Table 2.4 below shows a sample where character *i* occurs 1000 times in clean text. However in OCR text it is correctly recognized as *i* only 950 times, it is corrupted to *j* 30 times and corrupted to *l* 20 times. Based on this we can compute the 4following probabilities:

Probability that OCR read character *i* correctly is given by $P(i|i)$ = (950 | 1000)

Probability that OCR misread character *i* to *j* is given by $P(i|j)$ = (30 | 510)

Probability that OCR misread character *i* to *l* is given by *P(i|l)* = (20 | 290)

Char	# in clean	#i in OCR	*j*	*l*
i	1000	950	30	20
j	500	30	450	20
l	300	20	30	250
Total	1800	1000	510	290

Table 2.4: Sample Frequency calculation Table

	i	*j*	*l*
i	*P(i | i)*	*P(i |j)*	*P(i|l)*
j	*P(j | i)*	*P(j | j)*	*P(j | l)*
l	*P(l | i)*	*P(l | j)*	*P(l | l)*

Table 2.5: Sample Structure of Confusion Matrix

2.3.3.1 Using Confusion Matrix

Let $B = B_0B_1\ldots\ldots\ldots B_n$ be the OCR produced error string and $A = A_0A_1\ldots\ldots\ldots\ldots A_n$ be one of the candidates for correction. Then probability that OCR corrupted string *A* to *B* is given by

$(A_0A_1.....A_n \mid B_0B_1.....B_n)$ which can be computed as:

$P(A_0|B_0) * P(A_1 \mid B_1)\ldots\ldots * P(A_n \mid B_n)$

Where $P(A_n \mid B_n)$ denotes the probability that the n^{th} character in original string A was A_n and it was misrecognized by OCR as B_n.

Example

Let error string *B=sment*

original string *A=spent*

*P (spent| sment) = P(s|s) * P (p|m) * P(e|e) * P(n|n) * P (t|t)*

To get the values of *P(s|s), P(p|m)* etc Confusion Matrix is used.

2.3.3.2 Laplace Smoothing

It is used to ensure that none of the probabilities in the confusion matrix is zero. It normalizes all the zero probability to very small non zero numbers by introducing Smoothing constant. The modified probability is given by:

$$P(a \mid b) = \left(\frac{K + N(a \rightarrow b)}{K * (NOS) + N(b)} \right)$$

Where $p = (a \mid b)$ is the probability of character *a* being misrecognized by OCR as *b*

K is the Smoothing parameter

N(a→b) is number of times *a* was misrecognized as *b* in the OCR text

NOS denotes the total number of alphabets in the OCR text

N(b) = total number of times character *b* occurs in the OCR text

So this way even if *N (a→b)* is zero even then *P(a|b)* will have a very small non zero probability.

CHAPTER 3

PROPOSED APPROACH AND IMPLEMENTATION

3.1 Proposed Approach

The proposed OCR error correction starts by first cleaning OCR corpus *T* to remove all characters other than 'a' to 'z' as well as all the *stopword*s like "is", "that" etc. Then the cleaned text T_c is screened through spell checker *Jspell* which gives the set of all probable errors *E*. The original document is then manually read to find actual words corresponding to each error *e* in the Error list *E*. Then each OCR error is concatenated with words immediately preceding or following it to generate queries of variable length. Formally it can be denoted as: $Q=$"w_{-n} ...,w_{-2} ,w_{-1} ,e, w_1,w_2 ,... , w_n" where *Q* represents a sentence made out of $2n+1$ words, where w_{-i} represents the i^{th} error that precedes *e*, and w_{+i} represent i^{th} word following the *E* respectively. The number of words 2n+1 can be theoretically as large as one wishes but in our experiments ranges over 1,3,5,7 and 9.

Afterwards query *Q* is searched in the huge *Google* database and data consisting of top ranked pages P_i where P_i is the i^{th} page returned by *Google* and *i* ranges from *1,2….10*, are saved to a HTML (HyperText Markup Language**)** file. Then the text is parsed to extract all the possible list of corrections called the *Correction Candidates*, denoted as $C=\{c_1,c_2,c_3,\ldots,c_k\}$, where c_k denotes the k^{th} candidate spelling. The parsed data is also searched for *Google*'s "*did you mean*" or "*Showing results for*" token T_i. If any of these token is found then their contents are appended to the list of *Correction Candidates* List *C*. Now Levenstein edit distance method is applied to find candidate c_j having lowest edit distance with respect to error *e*. Additionally Longest Common Subsequence (LCS) algorithm is also applied to find candidate c_k having longest

Common Subsequence with error *e*. Moreover if Google does not give "*did you mean*" or "*Showing results for*" suggestions for an error, then the probability of error *e* being correctly spelled is high. So, while choosing the best candidate an Edit Distance of 0 is also allowed and in LCS method the candidate string identical to error string is allowed as correction. In case there are more than one best candidate c_j or c_k then the Confusion matrix *M* is used. *M* contains the probability *P* of a particular character being misrecognized (by OCR) as one of 26 English alphabets. The matrix *M* is computed by using errors from the ground truth training data. If the error is $e = B_0B_1\ldots\ldots B_n$ and the candidate is $c = A_0\ A_1..A_n$ then probability of *c* being the correct candidate is given by expression:

$$P\left(A_0A_1.....A_n \mid B_0B_1.....B_n\right)$$

Which can be computed as:

$$P\left(A_0|B_0\right) * P\left(A_1 \mid B_1\right)\ldots\ldots * P\left(A_n \mid B_n\right)$$

where $P\left(A_0 \mid B_0\right)$denotes the probability that the character A_0 is misrecognized as B_0 by OCR. The candidate with highest probability is then chosen as a replacement of wrong OCR word. The file containing the best candidates is compared with the original words to compute precision, recall and F-measure.

The proposed algorithm is context-sensitive as it depends on real-world statistics from *Google* data set, primarily extracted from the World Wide Web. Since we know that *Google* search is based on the keyword. So if the input query contains an error, then *Google* search will be based on context of the error and those tokens from the web will be retrieved which are most likely to match the query string.

3.2 Differences between Related Work and Proposed Approach

The proposed approach is a context sensitive OCR error correction approach, it differs totally from all dictionary based approaches since those use static vocabulary for error correction. Moreover in related work [18] query is formed from 25 non function words from OCR corpus and uses frequency as one of the candidate selection criteria but in our work the variable length queries upto length 9 are formed from the context immediately surrounding the error word in OCR corpus. In addition our approach uses Longest Common Subsequence as one of the selection criteria. An approach proposed by Bassil and Alwani [16] uses just the *Google*'s online spelling suggestion as a sole source of spelling candidate generation and uses queries on length 5 only, whereas in our approach candidates are also extracted from the top ten web pages retrieved from *Google* search and experiments are performed on queries of variable length. Another work [17] uses offline *Google* Web IT 5-gram dataset, uses four preceding words to form the query and consider frequency as a sole criteria for candidate selection. On the contrary our approach uses *Google* search to retrieve the latest web data dynamically, gives equal priority to both preceding and succeeding context to form query and applies more sophisticated candidate selection techniques like Levenstein Edit Distance, LCS and Bayesian Character Confusion matrix. Another research [19] deals with crawling of domain centered corpora using the Yahoo web search engine, chooses context and forms query on words frequency basis and collects top 30 documents retrieved from web. However, our work performs domain independent *Google* web search, do not consider frequency while forming context and considers top hundred results for candidate generation.

Algorithm 3.1 Formal Description of Algorithm

Function ErrorCorrection (Errors *E*, OCR text *T*, Ground Truth Training Data *TD*)

{

//removes all Stopwords from OCR text

1: Parsed_OCR = *Cleaning* (*T*)

*// Computes a 26*26 Computes Confusion Matrix for each characters a through z*

2: ConfusionMatrix *M* = *ComputeConfusionMatrix*(OCR Training Data)

3: for i = 1 to *E*

// *puts together the* i^{th} *error with the two preceding and two succeeding words*

4: Query =Concatenate (w_{-n} ...,w_{-2} ,w_{-1} ,*e*, w_1,w_2 ,... , w_n)

//finds the Query Q in huge Google Database

5: Data *D* =Query*Google* (Query *Q*)

// *the HTML data is parsed to retrieve the keywords or correction candidates*

6: Candidate list C_l = *parsedata* (Data)

7: Links *L*[ ] = *LinkExtractor* (Data) //L[ ] *contains link to next Google pages*

8: for j =1 to *L*

9: Data *D'*= *QueryGoogle* (link *L(j))*

10: retrieve *K* from *D'* and append to C_l // *K is the list of keywords*

11: if "*Did you Mean* or "*showing results for*" token present in Data

12: Retrieve the token and append to C_l

// Apply Levenstein edit distance or Longest Common Subsequence to choose best candidate C_b

13: BestCandidate C_b = Edit (e, C_l) or LCS (e, C_l)

14: If (count (BestCandidate C_b) > 1){

// appends best candidate with highest transition probability to list of correct candidates

15: C = ComputeHighestProbability (error e, BestCandidates, M)

16: Else C=C_b // *appends best candidate to list of correct candidates*

17: Return C // *C now contains list of all corrected OCR errors*

Diagrammatic Representation of approach is shown below:

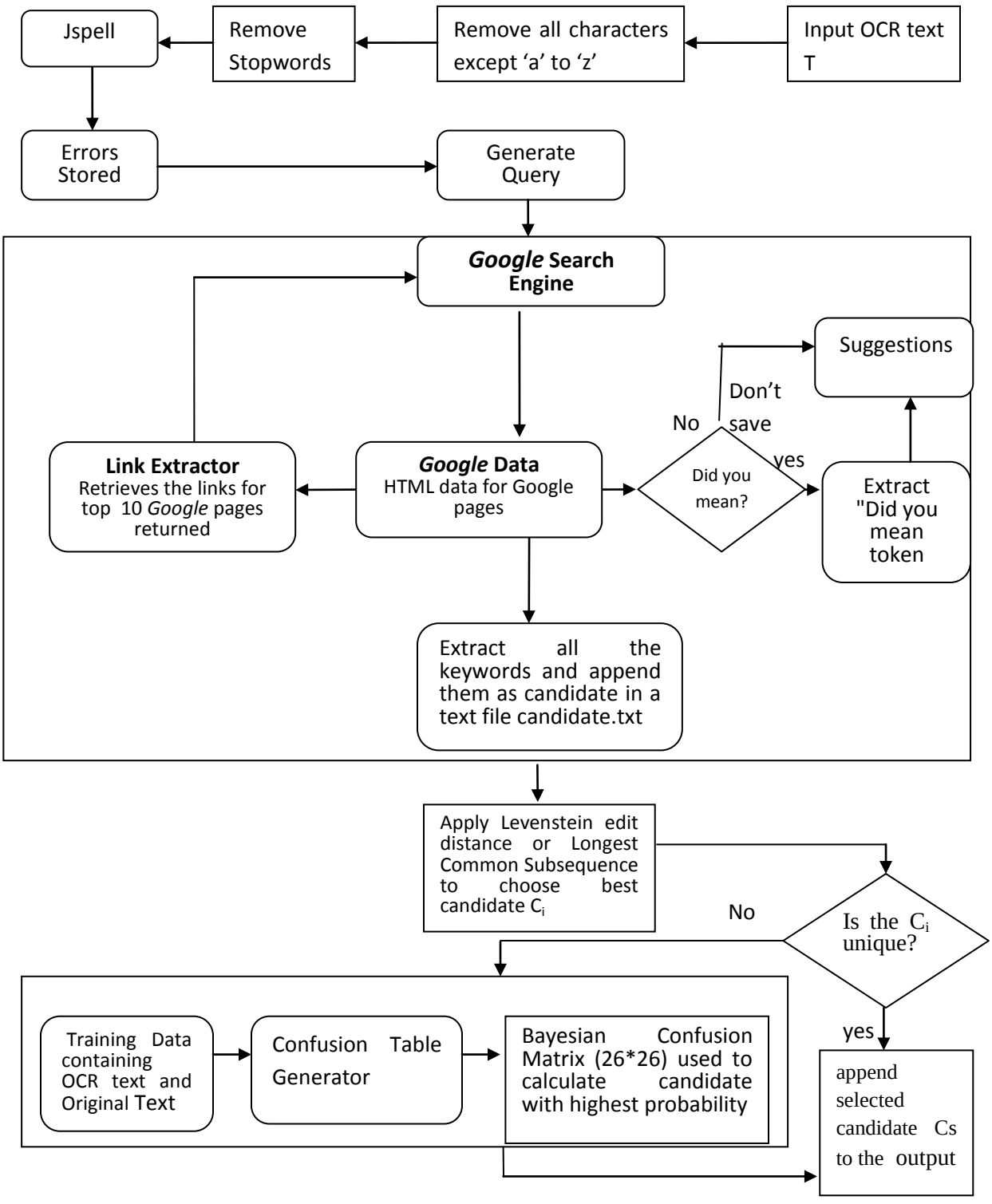

Figure 3.1: OCR Error Correction Procedure

3.3 Methodology Details

Step#1 Training data is used to build the Character Confusion matrix of dimension 26*26. The various modules used for this process are as follows:

- For this the module *CharacterCounter* is created which counts the frequency of occurrence of each character from 'a' to 'z' in the cleaned text as well as in the corrupted OCR text.

- Then module *ComputeConfusionMatrix* is then used to compute the Character confusion matrix containing probability of misrecognition of each of 26 characters as one of the other 26 character.
- The method *LaplaceFilter* is used to assign small non zero probability to the entries of Confusion Table which have a zero value in order to make calculation feasible. A very small value of .0001 was chosen for smoothing constant.

Step#2 Preprocessing the testing Data- The data for testing consists of images and its corresponding OCR text.

- The first module consists of a function *CleanText*. It reads each character of OCR corpus and filters all characters other than those having ASCII value between 97 to 122 (ASCII values for characters a to z) or 65 to 90 (ASCII values for characters A to Z). The output is saved to a text file named *CleanText.txt.*
- The second module consisted of function *RemoveStopwords*. It reads each word in the *CleanText* file and removes all the *stopword*s like "is", "at", "that" etc. The output is saved to a text file *StopwordCleaned*.

Step#3 The evaluation of *Error.txt* and *Original.txt* files

- The cleaned OCR corpus is fed to *Jspell* spell checker to generate the list of possible misspellings or errors E. These misspellings are saved to the text file *Errors.txt*. Then the original images are read manually to find the corresponding correct words for those misspellings and saved to a text file named *Original.txt*. The errors which are originally Proper Nouns, Acronyms or Non-English words would be discarded. It is observed that some of the errors found by *Jspell* are actually correctly spelled but even then these are kept in the error list in order to test precision, i.e, number of correct word which get corrupted by applying procedure. A sample of *Error.txt* and *Original.txt* is given below:

nn	an
mnde	made
tfhese	these
ajwnys	always
uity	unity
rcgardinp	regarding
teveral	several
hohpital	hospital
realise	**realise**
greut	great

Figure 3.2: Sample Error.txt file and Original.txt

Step#4 Query Generation- The module *QueryGenerator* takes input the list of errors and cleaned OCR corpus namely *CleanedStopword.txt*. It generates query strings of varying lengths namely1,3,5,7 and 9. The query is composed of errors the context surrounding it, in the OCR text. For instance for the OCR text :

- " the magic show was a grcat success and fame !"

The cleaned text would be:

- "magic show grcat success fame"

The precise 5 word query sent to *Google* by procedure would be:

- *Q*= magic + show + grcat + success + fame

All the generated queries are stored to text file *query.txt*.

Step 5 # Crawling Web for extraction of data- A module called *QueryGoogle* has been created which takes the list of queries recursively as input and retrieves results from the *Google* Web Search. It parses *Google*'s standard (browser) search HTML results. The HTML source code of top ten pages returned by the *Google* are stored in a text file. We include a short delay after each page retrieval because *Google* block IPs (Internet Protocol) with too many requests in a short time. Figure 3.3 below shows the sample *Google* response on firing the above query Q_5.

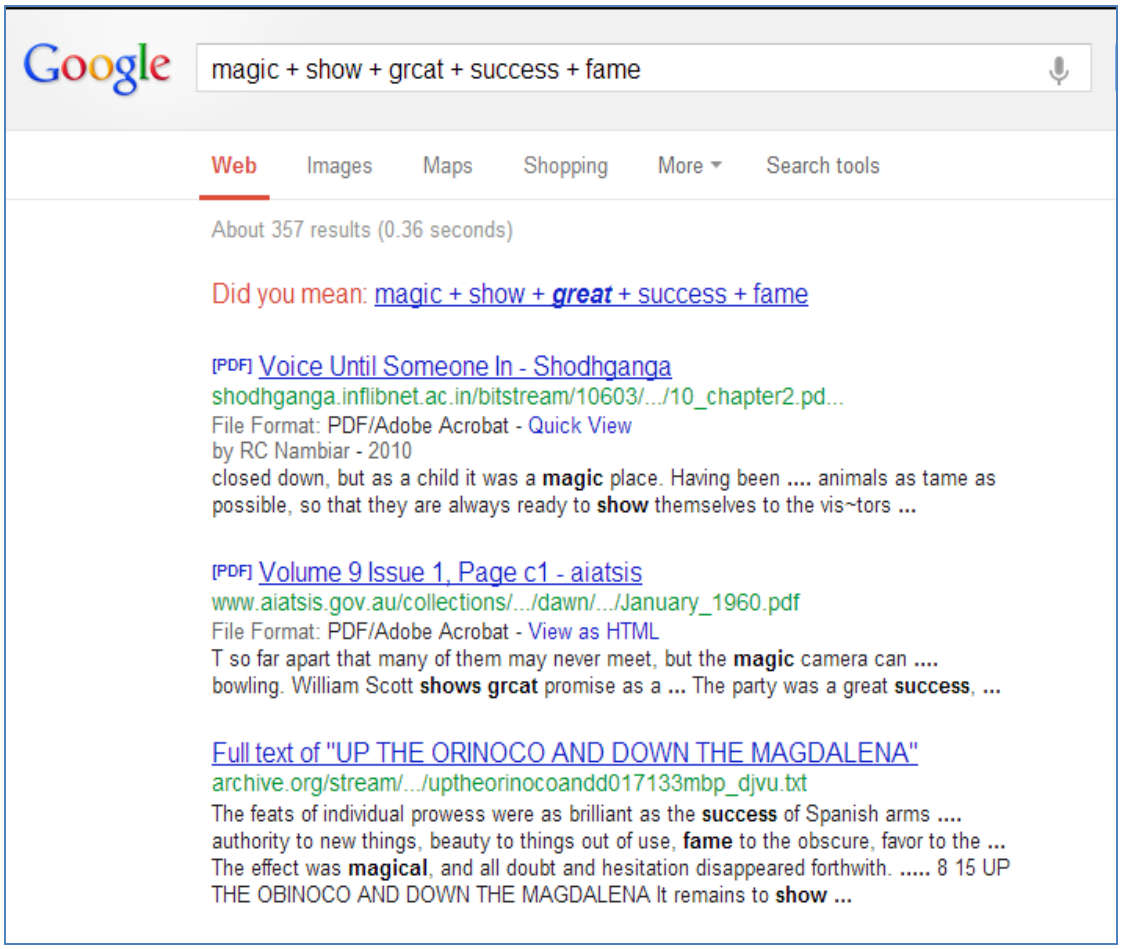

Figure 3.3: Firing Query to *Google*

Step # 6 Link extractor- The module *LinkExtractor* extracts the web links of all the next result pages of *Google*, if present on first search page. All the link are then stored in *Link.txt* text file. For the above query, web links shown below in figure 3.4 (1 thorough 7) will be extracted.

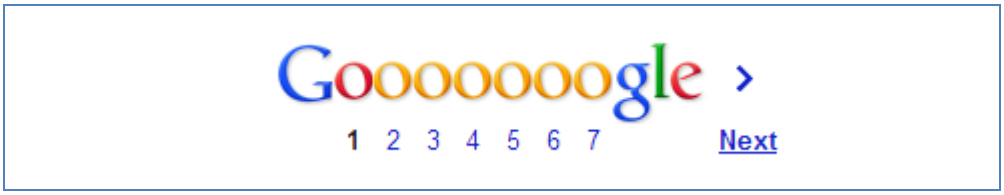

Figure 3.4: Retrieval of *Google* Next Page Links

Step # 7 Fetching the HTML data from all the Web Links and parsing it to generate list of possible correction candidates

- For fetching Data from the Links method *QueryGoogle* is called recursively and retrieved data is stored in different HTML files.
- The module *ExtractFirstPage* is called which parses content of top Google HTML page and extracts all the keywords, saves them in a text file named *FirstPagekeywords.txt*. Further the module *ExtractNextPageKeywords* retrieves the keywords from all the next web pages returned by Google, saves them in a file named *NextPagekeywords.txt*. Figure 3.5 shown below is a sample web snippet; all the keywords in bold i.e “shows”, “fame”, “grcatesr” will be extracted .
- The module *MergeKeywords* facilitates in combining the contents of *FirstPagekeywords.txt and NextPagekeywords.txt* to a text file named *Merged_keywords.txt*. In order to remove redundant words all the unique keywords present in *Merged_keywords.txt* are extracted and written to another file *Unique_keywords.txt* .

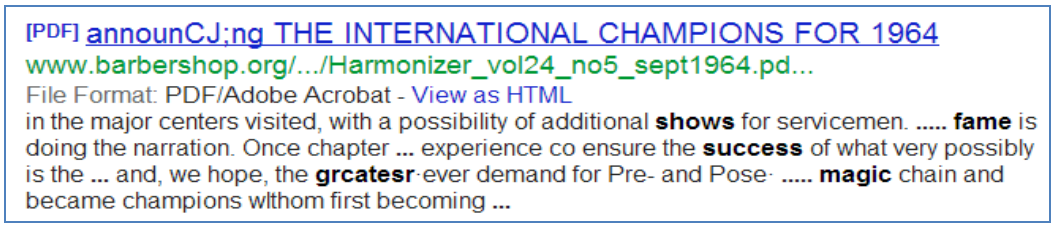
[PDF] announCJ;ng THE INTERNATIONAL CHAMPIONS FOR 1964
www.barbershop.org/.../Harmonizer_vol24_no5_sept1964.pd...
File Format: PDF/Adobe Acrobat - View as HTML
in the major centers visited, with a possibility of additional **shows** for servicemen. ..... **fame** is doing the narration. Once chapter ... experience co ensure the **success** of what very possibly is the ... and, we hope, the **grcatesr** ever demand for Pre- and Pose· ..... **magic** chain and became champions wlthom first becoming ...

Figure 3.5: Keyword Extraction from Web Data

- Then the top page returned by the *Google* is parsed by the module *SuggestionExtractor* to extract the contents of "*Did you mean*" or "*Showing results for*", if present on the top page. For the snippet shown in figure 3.6, the contents "magic show great success fame" will be retrieved and stored in the text file named *Googlesuggestion.txt*. The contents are also appended to the text file *Unique_keywords.txt* to generate the file *Candidate.txt,* containing an exhaustive final list of all possible candidates for error correction.

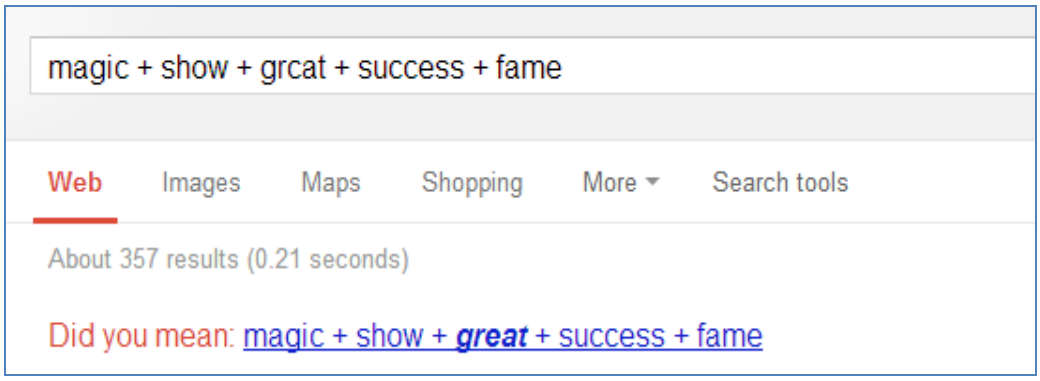

Figure 3.6: Extraction of *Google* Suggestion

Step# 8 Choosing the best correction from *Candidate.txt* file .

- To implement Levenstein Edit Distance Algorithm, the module *ComputeEditDistance* is created which takes as input the error *e* and *Candidates.txt* file and gives the best candidates C_b as output. It computes the number of insertions, deletions or substitutions required to transform candidate to the error word. Also if the file *Googlesuggestion.txt* has some content corresponding to an error e, then candidate with an edit distance of zero (with error) is not considered for correction. A sample of *Candidate.txt* shown below in figure 3.7:

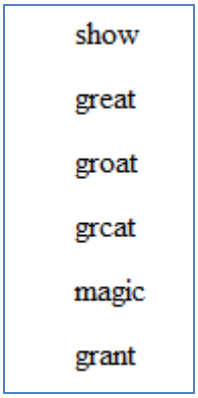

Figure 3.7: Sample Candidate.txt file

Computations made by module for some of these candidates, will be:

- ComputeEditDistance (grcat, great) =1 (Substitution of c with e)
- ComputeEditDistance (grcat, groat) =1 (Substitution of c with o)
- ComputeEditDistance (grcat, grant)=2 (substitution of 'c' with 'a', 'a' with 'n')
- ComputeEditDistance (grcat, grcat) =0

Now, the Candidate "grcat", has the lowest Edit Distance with an error "grcat". But our algorithm does not consider this string for correction since *Google* generates suggestion content "*Did you mean*" for the error "grcat". The candidates then considered for correction are strings "great" and "graot", having an edit distance of one from the error string. Since both candidates have same edit distance from the error, then module *ComputeProbability* is used to break the tie by generate the following conditional probabilities:

- $P(great \mid grcat) = P(g|g) * P(r|r) * P(e|c) * P(a|a) * P(t|t)$

- *P (groat |grcat)= P(g|g) * P(r|r) * P(o|c) * P(a|a) * P(t|t)*

The Confusion Matrix is used to compare character confusion values *P(e|c)* and *P(o|c)* i.e the probability of character 'e' being misrecognized as 'c' and probability of character 'o' being misrecognized 'c', by the OCR. The candidate with the highest conditional probability with respect to the error , is then chosen as the correction.

The module *ComputeLCS* is also used (independently from Edit Distance) to compute the best candidate, which has the longest common subsequence with the error, as the correction. This method takes the error *e* and *Candidates.txt* file as input. For the error "grcat" and the Candidate file shown above, some of computations made by the module *ComputeLCS* are shown below:

- LCS (grcat, great) = 4 Longest subsequence (grat)
- LCS (grcat, groat) = 4 Longest subsequence (grat)
- LCS (grcat, grant) =4 Longest subsequence (grat)
- LCS (grcat, cat) =3 Longest subsequence (cat)

Again there is more than one candidate having the longest LCS with an error string. So in order to choose the best candidates among these, the character confusion matrix is used. For simplicity the uppercase characters in the candidate strings were converted to lower case for comparison with error word.

Step#9 Finally we compute the precision, recall and F-measure for both the Levenstein Edit Distance and the LCS algorithms.

CHAPTER 4

EXPERIMENTS AND RESULTS

4.1 Evaluation Criteria

To evaluate the performance of the experiments, we need to evaluate and determine the evaluation measures. There are four possible outcomes when we try to apply the procedure to correct the errors:

1. correct → correct: A correct character is still correct at output. This is a true negative (TN).

2. correct → wrong: A correct character is corrected to a wrong character at output. This is a false positive (FP).

3. wrong→correct: A character is corrected by the procedure. This is a true positive (TP).

4. wrong → wrong: A wrong character is still wrong. This is a false negative (FN).

Now, using the TN, FP,TP and FN, the measures Precision and Recall [21] can be derived as :

$$Recall\ \ R = \left(\frac{TP}{TP + FN} \right)$$

$$Precision\ P = \left(\frac{TP}{TP + FP} \right)$$

The Recall measures the ability of a system to correct errors. In order to get higher recall, the number of True corrections (TP) should be more and number of False Corrections (FN) should be least. The precision denotes the accuracy of the system; i.e not corrupting the correctly spelled words. To gain higher precision the number of corrections needs to be more and introduced errors (FP) should be less. Since we consider both recall and

precision as equally important, so the harmonic mean of R and P, the simplified F measure [25] is given by:

$$F = \left(\frac{2 * P * R}{P + R} \right)$$

4.2 Data Collection

4.2.1 Training Data – We need a set of training data for building the character confusion matrix. The data for first experiment has been taken from a book titled "Notes on Witchcraft" with 60 pages, which has been manually corrected with reference to non-OCR version image of the book. After removing all the characters except *a* to *z* the training data contained 13,104 words.

4.2.2 Testing Data- The data for testing the procedure is taken from Library of Congress (http://www.loc.gov/index.html) which is the largest library in the world, with millions of books, recordings, photographs, maps and manuscripts in its collection. The Library has created a website named Chronological America which provides access to digitized historic materials primarily through a Web interface enhanced with dynamic HTML interactivity for magnification and navigation. It contains digitized newspapers from years 1836 to 1922. These newspaper materials were digitized to technical specifications designed by the Library of Congress i.e TIFF 6.0, 8-bit grayscale, 400 dpi, uncompressed, with specified tag values.

The testing data is taken from The Mt. Sterling advocate, a newspaper present in the Library of Congress collection. The pages were chosen based on various criteria such as readability, date of publication and convenience to map with its corresponding OCR text. The total of 7 newspaper images are chosen as testing data. Data contains 8,400

words after removal of *Stopwords* and non-alphabetical characters. The corresponding OCR text is first is screened through, a spell checker software API called *Jspell*. It has suggested 103 possible errors. The original newspaper images are then read manually in order to find the correct words corresponding to these misspelling errors. Finally, the file of errors and original are prepared.

4.3 Results on Data Set 1

Table 4.1 below shows the value of Precision and Recall for the Test Data. The Recall is attains a lowest value for the 1 word query, it increases fairly as query length is increased to 3, it reaches its maximum value 51.5% at a query length 5. Then the value decreases a little for a 7 word query. The LCS method gives the highest values for Recall and Precision. The procedure does not introduce any errors, since the original data does not contain many wrongly spelled words. The procedure is build such that the Precision attains the highest performance, even at the cost of low Recall. For cases when *Google* do not generate suggestion of form of "*Did you mean*", the string matching algorithms LCS and Edit Distance are adapted to allow a candidate identical to the misspelling error as a replacement. Consequently, this improved Precision though Recall dropped a little since some errors are replaced by themselves. Table 4.2 shows the overall accuracy of system in the form of F-measure:

	1word	3word	5word	7word	9 word
Edit	R=32 P=100	R=44 P= 100	R= 48.5 P= 100	R=43 P= 100	R=40 P=100
LCS	R=30 P=100	R=45 P= 100	R=51 P=100	R=44 P= 100	R=41 p=100

Table 4.1: Precision –Recall values for Data set 1

Fmeasure	1word	3word	5word	7word	9 word
Edit	48.48	61.1	65.3	60.1	57.1
LCS	46.15	62.1	68	61.1	58.15

Table 4.2: F-measure values for Data set 1

4.3.1 Observations on Data Set 1

It is observed that our approach corrects more errors than the correction suggested by Google's "Did you mean". If we make candidate selection only on the basis of the Google's "Did you mean" suggestion, then we correct 42 errors out of 95 misspellings but our approach is able to correct 49 misspellings. So there is an improvement of around 16.6% in error correction using our approach. There are many possible reasons for this. Firstly, it is observed that in case the context surrounding the error is also misspelled then there is least chance Google gives correct suggestion. For instance when the query "tne+territory+mnke+advances+tho" containing error "mnke" is fed to Google search engine, the Google loads the suggestion "the+territory+mnke+advances+tho". Hence, Google here focuses on correcting the commonly misspelled word "tne" which is the first

misspelled string in the search query. However, our approach corrects the error "mnke" to "make". In the second category, Google is not able to load any suggestions. For instance for the query **"**day+Oclock+Mrther+tf+jnd" containing error "Mrther" Google does not give any "Did you mean" suggestion. However, our approach corrects the error to "mother". The third reason that our approach is able to correct more errors than Google is due to use of Character Confusion Matrix. For instance for the query "work underway roaa surVs Yucca" containing error "roaa", Google does not give any suggestion but our procedure selects two possible candidates "rosa" and "road" on basis of LCS. Then the conditional probabilities p(road | roaa) and p(rosa | roaa) are computed, after which the word "road" is selected as the best candidate.

We can clearly see that the F-measure is lowest for the 1 word query. For instance the error word "mnde" is not corrected when fed as a single word query to web search. However when it is fed along with its context in OCR text, web generated the correct spelling candidate "made". If context of the query is not available it becomes unlikely for the web to identify error and retrieve relevant webpages. The performance improves a bit for a 3 word query as it provides some context but the Recall or F-measure is best for the 5 word query as it gives the web necessary and sufficient context to generate the possible relevant corrections. For instance the error "bo" is not corrected when the 3 word query is fed but when the query is expanded to 5 words, the error "bo" got corrected to "be". The score does not increase further for 7 or 9 word query as 5 word query provides sufficient and necessary context, expanding query does not affect the performance in terms of the retrieval of candidates. Also too much context sometimes redirects to the webpage which is the actual source of error and also number of retrieved tokens become less since

Google returns pages that match all the search terms. For instance, in case of 9 word query "queryaplte+variety+la+contention+anawer+tiled+aupcrlor+court+today" the web considered the error "anawer" as correct and retrieved the webpage from which the OCR corpus is taken, however for 5 word query web retrieved the correct spelling suggestion "answer".

Also our approach is not able to correct all the errors even for a five word query. There are many reasons for it. To start with, it is observed that nearly 30% of the errors that procedure is not able to correct, are originally *Stopword*s such as "the", "at", "an", "of" etc. This is due to reason that *Stopword*s do not generally add meaning to search. But misspelled *Stopword*s in OCR corpus do not affect the retrieval performance either. Many a times (nearly 10%) spelling correction is a variant of errors and string matching algorithms are not able to relate these. For instance the error string "spld" is wrongly corrected to its plural "splds" instead of "sold". An s-stemmer can help to recognize such pair of words. In few cases, an error word becomes a valid word in other language, hence web gives tokens related to those web pages. For instance the word state is misrecognized by OCR as "stato" which is a valid word in Italian thus *Google* treated it as an authentic word and do not generate spelling suggestions. A more language restrictive search technique might help deal this issue. In some cases erroneous words are mistaken for acronyms, proper noun, hence the irrelevant webpages are fetched by search engine, which in turn generates irrelevant correction candidates. For instance the error string "aro" (originally "are") is wrongly corrected to "aro", one of correction candidates generated by *Google* search. More advanced error detection technique may allow the procedure to judge the cases where replacement of error by itself is not allowed. In rare

cases the web found the original web source from where the error is taken, hence error goes undetected, so no relevant candidates for spelling correction are suggested. As an instance for the context string "established+uniform+errades+for+burley" containing error "errades", the top webpage retrieved is from Chronological America webpage. Hence, better Information Retrieval models may prove handy. It is also observed that sometimes web generated wrong candidates due to the ambigious context. For the context "Central+Kentucky+Wo+started+this" the error string "wo" is misrecognized as "who" instead of "we". In some rare cases algorithms chose the wrong candidate as correction. The error "ajwnys" is wrongly corrected to "ajwny" instead of "always", by the LCS. Weighted string matching technique can be used, where candidate string with valid entry in a dictionary would be given more weight. In some cases the content surrounding the misspelling is itself corrupted or misspelled so web could not identify the correct context. Moreover, some errors are too much distorted, difficult to get even a valid spelling candidate generation. This mostly occurred for the contents of headings in the Bold font, as an example the text "Distinctive spring Papering" is corrupted to "gLtsfttttitot Iptttffi lajttrtttg".

4.4 Results on Data Set 2

In another experiment the above mentioned data from "Notes on Witchcraft" book is used both for training and testing. The data after cleaning contains13,104 words . Half of the data is used for training or building character confusion matrix and the other half is used for testing the procedure. Table 4.3 below shows precision-recall values obtained for the queries of various lengths using each of the methods. Table 4.4 shows the F-Measure:

	1word	3word	5word	7word	9 word
Edit Method	R = 44 P = 77	R= 48 P= 67	R = 50 P = 66	R=47.5 P= 61	R = 47 P = 50
LCS Method	R=43 P=78	R= 47 P= 66.5	R=49 P=67	R=48 P= 59.8	R=45 p=50

Table 4.3: Precision –Recall values for Data Set 2

Fmeasure	1word	3word	5word	7word	9 word
Edit 1	56	55.9	56.9	53.4	48.4
LCS1	55.4	55.1	56.6	53.3	47.5

Table 4.4: F-measure values for Data Set 2

4.4.1 Observation on Data Set 2-

For this dataset the recall is best for the five word query. But here unexpectedly the precision is best for the single word query. The reason is that the number of errors that remain misspelled are maximum for single word query, so even the words which are misspellings in original text remain misspelled, increasing the value of Precision. For instance, the original text and OCR both contained word “restauration” which remains wrongly corrected as “restauration”, when fed as a single word query. However when fed with context it got corrected to “restoration” (which is ideally the correct spelling), but this would be considered as corrupting the word and hence Precision drops. Even then the combined measure of Precision and Recall , the F-measure attains its maximum value for the five word query.

However Precision drops substantially compared to DataSet1. This is due to the fact that the original image itself has many old English or misspelled words such as “praestigious”,“Magitians”,“maner”, “heros” etc. So the procedure in addition to correcting the spelling mistakes caused by OCR, corrected those misspellings too. But since these misspellings are identical in OCR text and original image, so it actually becomes a corruption instead of correction. Also the text contains words from German vocabulary such as “satisfie” which is corrupted to English word “satisfied” by the procedure. It is observed that there are several misspellings that remained uncorrected. To start with some German words still remained in the OCR corpus even after manually removing them. Due to this, these words became part of the context of the query and web retrieved results from irrelevant webpages including pages in German. A non English Language word detector may help in efficient query formation.

4.5 Conclusion and Future Work

We designed an OCR post processing system based on Big Data. Our experimental results on a small set of historical newspaper data show a recall and precision of 51% and 100%, respectively. In future dynamic use of Confusion Matrix may help. Also stemming techniques can be incorporated. Moreover there is a great need for advanced and restrictive web search. Advanced error detection techniques can also be used for improving results.

Appendix

Chapter A

Table A.1 below show the list of errors corrected by my procedure. For most of the errors (numbered 1 to 47) the LCS algorithm solely choose the correct candidate. For the errors 48 and 49 the Character Confusion Matrix helped in choosing the best candidate. Also the errors numbered 42 to 49 were corrected by our approach but not by the Google "Did you mean" suggestion.

	Error	Original
1.	joung	young
2.	gieat	great
3.	wesks	weeks
4.	geting	getting
5.	suiveying	surveying
6.	poweH	power
7.	Ppge	Page
8.	defandants	defendants
9.	ffom	from
10.	Wtednesday	Wednesday
11.	wjre	wire
12.	suppuit	support
13.	nver	never
14.	tneir	their
15.	bo	be

16.	highrly	highly
17.	oompany	company
18.	dorived	derived
19.	maehine	machine
20.	atteation	attention
21.	nbout	about
22.	ovef	over
23.	aproximateiy	approximately
24.	invesjor	investor
25.	farmors	farmers
26.	unitl	until
27.	wBat	what
28.	receivin	receiving
29.	Fhiance	finance
30.	tfhese	these
31.	ajwnys	always
32.	Rcgardinp	Regarding
33.	teveral	several
34.	jmistmas	christmas
35.	Cnited	United
36.	Hohpital	Hospital
37.	amprovements	improvements
38.	anawer	answer

39.	yestrday	yesterday
40.	aibilty	ability
41.	tho	the
42.	Fnited	united
43.	mnke	make
44.	brough	brought
45.	matler	matter
46.	ordets	orders
47.	Mrther	mother
48.	roaa	road
49.	mation	nation

Table A.1: List of errors corrected by our procedure

Table 4.2 gives list of errors not corrected by our approach. These can be divided into various categories. The main reason for not being able to correct these errors is lack of presence of correct spelling correction candidate in the webpages retrieved. Majority of these errors (numbered 1 to 15) were originally *Stopwords*. For the errors numbered 15 to 36 the web either misrecognized them as some proper noun or acronym, or the context surrounding these words is also corrupted. Some of the errors specially 37 to 43 are too distorted from original spelling. For some of the errors namely 45 to 46 correct candidate could not be selected by procedure. They can be picked by using an stemmer

	Error	Original
1.	anJ	and
2.	vho	who
3.	tne	the
4.	bv	by
5.	tho	the
6.	js	is
7.	nn	an
8.	id	is
9.	tharf	than
10.	Thp	Than
11.	Wo	We
12.	jthey	they
13.	aro	are
14.	nbout	about
15.	ot	of
16.	damsile	damsite
17.	rewardeu	rewarded
18.	riehts	rights
19.	eiiort	effort
20.	Stntc	state
21.	okl	ok
22.	Vears	years

23.	phono	phone
24.	Stntc	state
25.	okl	ok
26.	royival	revival
27.	Engago	Engage
28.	baiiks	Banks
29.	gjad	glad
30.	stato	state
31.	mnde	made
32.	uity	unity
33.	greut	great
34.	dele	date
35.	crroom	groom
36.	wnr	war
37.	rriembers	members
38.	zation	caption
39.	rebuiS	rebuilt
40.	gLtsfttttitot	Distinctive
41.	Ipttffi	spring
42.	lajttrtttg	Papering
43.	niuke	make
44.	engilneer	engineers
45.	ajwnys	always

46.	greut	great

Table A.2: List of errors not corrected by our procedure

a	5858
b	1512
c	4078
d	3470
e	10705
f	1570
g	1755
h	2545
i	6844
j	295
k	675
I	3611
m	2452
n	5865
o	5355
p	2563
q	166
r	6194
s	6518
t	6658

u	2656
v	912
w	1383
x	348
y	1213
z	62

Table A.3 Count of Characters in Training data

b->e	1
r->s	2
g->s	3
f->t	4
a->u	3
l->f	3
t->l	1
r->x	5
z->s	2
a->z	1
c->e	7
r->l	3
f->t	4

f->l	6
h->b	2
g->c	2
a->u	3
v->r	1
r->e	2
e->r	2
x->z	1
v->y	1
l->j	2
e->t	2
f->i	5
a->o	8
l->i	9
n->m	3
e->u	1
u->y	1
g->s	3
e->r	2
t->i	2
y->s	1
a->f	1

r->i	2
s->i	1
c->o	2
o->c	2
m->n	3
v->u	2
j->i	1
e->o	1
f->i	5
i->l	1
a->c	1
a->e	1
c->o	2
g->s	3
v->y	1
s->a	2
r->x	5
f->l	6
u->i	5
m->n	3
r->d	1
s->e	1

d->a	3
u->v	1
v->u	2
e->o	1
f->i	5
u->m	1
h->i	1
a->d	2

Table A.4: List of Substitution errors in Training Data

BIBLIOGRAPHY

[1] “The Boston Public Library”,(Janurary,2010)
http://www.bpl.org/general/about/bpl_an_overview_2010

[2]Vincent, L. (2007, September). *Google* Book Search: Document understanding on a massive scale. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on* (Vol. 2, pp. 819-823). IEEE.

[3] Klein, S. T., Ben-Nissan, M., & Kopel, M. (2002). A voting system for automatic OCR correction.

[4] Cheriet, M., Kharma, N., Liu, C. L., & Suen, C. (2007). *Character recognition systems: a guide for students and practitioners*. Wiley-Interscience.

[5] Taghva, K., Beckley, R., & Coombs, J. (2006). The effects of OCR error on the extraction of private information. In *Document Analysis Systems VII* (pp. 348-357). Springer Berlin Heidelberg..

[6] Miller, D., Boisen, S., Schwartz, R., Stone, R., & Weischedel, R. (2000, April). Named entity extraction from noisy input: speech and OCR. In *Proceedings of the sixth conference on Applied natural language processing* (pp. 316-324). Association for Computational Linguistics.
.
[7] Lebert, M. (2008). *Project Gutenberg (1971-2008)*. Project Gutenberg.).

[8] Bokser, M. (1992). Omnidocument technologies. *Proceedings of the IEEE*,*80*(7), 1066-1078.

[9] Levenshtein, V.I., Binary codes capable of correcting deletions, insertions, and reversals, Cybernetics and Control Theory, 10(8), 707-710, (1966).

[10] Niwa, N., Kayashima, K., & Shimeki, Y. (1992). Postprocessing for character recognition using keyword information. In *IAPR Workshop Machine Vision Applications* (pp. 519-522)

[11] Taghva, K., Borsack, J., & Condit, A. (1994, August). Results of applying probabilistic IR to OCR text. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*(pp. 202-211). Springer-Verlag New York, Inc.

[12] Kise, K., Shiraishi, T., Takamatsu, S., & Fukunaga, K. (1996). A method of post-processing for character recognition based on syntactic and semantic analysis of sentences. *Systems and computers in Japan*, *27*(9), 94-107.

[13] Hull, J. J. (1996). Incorporating language syntax in visual text recognition with a statistical model. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *18*(12), 1251-1255.

[14] Dictionaries, O. (2011). Oxford Dictionaries Online.

[15] Tong, X., & Evans, D. A. (1996, August). A statistical approach to automatic OCR error correction in context. In *Proceedings of the fourth workshop on very large corpora* (pp. 88-100).

[16] Bassil, Y., & Alwani, M. (2012). Ocr post-processing error correction algorithm using *Google* online spelling suggestion. *arXiv preprint arXiv:1204.0191*.

[17] Bassil, Y., & Alwani, M. (2012). OCR Context-Sensitive Error Correction Based on *Google* Web 1T 5-Gram Data Set. *arXiv preprint arXiv:1204.0188*.

[18] Mihov, S., Koeva, S., Ringlstetter, C., Schulz, K. U., & Strohmaier, C. (2004). Precise and efficient text correction using Levenshtein automata, dynamic Web dictionaries and optimized correction models. In *Proceedings of Workshop on International Proofing Tools and Language Technologies*.

[19] Ringlstetter, C., Schulz, K. U., & Mihov, S. (2007). Adaptive text correction with Web-crawled domain-dependent dictionaries. *ACM Transactions on Speech and Language Processing (TSLP)*, *4*(4), 9.

[20] Strohmaier, C., Ringlstetter, C., Schulz, K. U., & Mihov, S. (2003, August). Lexical postcorrection of OCR-results: The web as a dynamic secondary Dictionary. In *Proceedings of the 7th International Conference on Document Analysis and Recognition (ICDAR)* (pp. 1133-1137).

[21] Taghva, K., & Stofsky, E. (2001). OCRSpell: an interactive spelling correction system for OCR errors in text. *International Journal on Document Analysis and Recognition*, *3*(3), 125-137.

.

[22] Esakov, J., Lopresti, D. P., & Sandberg, J. S. (1994, March). Classification and distribution of optical character recognition errors. In *IS&T/SPIE 1994 International Symposium on Electronic Imaging: Science and Technology* (pp. 204-216). International Society for Optics and Photonics.

[23] Croft, W. B., Harding, S. M., Taghva, K., & Borsack, J. (1994, April). An evaluation of information retrieval accuracy with simulated OCR output. In*Symposium on Document Analysis and Information Retrieval* (pp. 115-126).

.[24] Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). 1 99 0. Introduction to Algorithms. *MIT Press, Cambridge, MA*..

[25] Reynaert, M. (2008, May). All, and only, the errors: more complete and consistent spelling and ocr-error correction evaluation. In *6th International Conference on Language Resources and Evaluation* (pp. 1867-1872).

VITA

Graduate College

University of Nevada, Las Vegas

Shivam Agarwal

Degrees:

Bachelor of Technology in Computer Science, 2011

Indian Institute of Information Technology

Master of Science in computer science, 2013

University of Nevada Las Vegas

Thesis Title: Utilizing Big Data in Identification and Correction of OCR Errors

Thesis Examination Committee:

Chair Person, Dr. Kazem Taghva, Ph.D.

Committee Member, Dr. Ajoy K. Datta, Ph.D.

Committee Member, Dr. Laxmi P. Gewali, Ph.D

Graduate College Representative, Dr. Emma Regentova, Ph.D.