Indian Statistical Institute, Kolkata


M. Tech. (Computer Science) Dissertation

# No-Reference Quality Assessment for OCR'd Documents

A dissertation submitted in partial fulfillment of the requirements
for the award of Master of Technology
in
Computer Science

Author:
Arnab Biswas
Roll No: MTCS-1415

Supervisor:
Dr. Utpal Garain
CVPR Unit, ISI

**M.Tech(CS) DISSERTATION THESIS COMPLETION CERTIFICATE**
**Student: Arnab Biswas (MTCS1415)**
**Topic: No-Reference Quality Assessment for OCR'd Documents**
**Supervisor: Dr. Utpal Garain**

This is to certify that the thesis titled **No-Reference Quality Assessment for OCR'd Documents** submitted by Arnab Biswas in partial fulfilment for the award of the degree of Master of Technology is a bonafide record of work carried out by him under my supervision. The thesis has fulfilled all the requirements as per the regulations of this Institute and, in my opinion, has reached the standard needed for submission. The results contained in this thesis have not been submitted to any other university for the award of any degree or diploma.

Date: Dr. Utpal Garain

# Dedicated To

I dedicate this to all the people who wishes me well in life including my brother, mother, uncle, friends and last but not least my project guide.

# Acknowledgements

I would like to thank my dissertation supervisor Dr. Utpal Garain for agreeing to guide me and for helping me to undertake work in the topic.

I would also like to thank Mr. Abhisek Chakrabarty, currently pursuing his Ph.D. under the guidance of Dr. Garain for helping me in many ways for understanding the problem and doing this work.

Last but not the least I am grateful to Indian Statistical Institute and its CVPR unit which provided me a very good research infrastructure that helped me all the way along.

# Abstract

This thesis deals with predicting quality of a text document that has been generated by an OCR system. As OCR systems are prone to make mistakes while converting an imaged document to machine readable form, this research concerns with finding errors in an OCR'd text and classify the OCR document accordingly. So far OCR community has dealt with this problem by following either of these two methods: (i) manual labeling of the errors or (ii) comparing the OCR'd document against the true text file. Manual counting of errors is infeasible in commercial situation whereas the true text file if often not available to compare with. This work attempts to develop methods for automatic prediction of OCR'd documents under no-reference condition. Bengali has been taken as the reference language. Use of lexicons and language models has been explored in several directions. Experiment with a large corpus of OCR'd documents shows that a lexicon based approach coupled with a suitable edit distance measure could be a viable method for no-reference quality assessment of OCR'd documents.

# Contents

# Chapter 1

# Introduction

The job of an OCR system is to recognize each character from the a given input file or input stream, failing to which introduces error and in this thesis we are going to discuss different approaches of estimating those error in order to predict the accuracy of the documents produced by that OCR system.

The first question may arise in people's mind is that if the same OCR is producing a set of documents how could they differ in quality or accuracy? The answer to this question lies in the fact that the same OCR system may have difficulty in recognizing one particular character but not so for another. Different documents with difference in number of letters or words of each type plays essential role determining the accuracy of the document.

The metric we used is WER(Word Error Rate). The focus of most of the experiments are how to predict correct words in place of wrong ones in a given OCR text file [1]. We have used Bengali language. The algorithms and procedures discussed here can be incorporated in any other language and in some cases result might differ a little.
Classifying OCR documents according to its accuracy involves scanning the entire file word by word and find the following errors,-

(A) Spelling mistake,-
  (i)True Negative.
  (ii)False Positive.
(B) Unlikely combination of words.
(C) Grammatical anomalies.

If we consider OCR files are taken from be some standard corpus (as in this case the local News Daily) then problem of having error of type (B) and (C) are rare. So our main focus is now on error of type (A).

There can be two kinds of spelling mistakes, firstly True negative; which means the word is correct but our program detects it as wrong word because it does not have enough

[1]Here OCR text file denotes those files which are scanned by an OCR system i.e files whose accuracy we are going to predict.

vocabulary strength. Also in languages like Bengali it is difficult to detect named entities and that can also be out of vocabulary word and would be considered as error. For languages rich with inflections, it may not always be possible to have each and every word in dictionary. So there exist possibility of True negative errors.

The other form of spelling mistake is False positive, where a wrong word can be passed as a correct one, which happens when you have error in your training corpus. The Bengali corpus we used as correct text documents and build our lexicon from, also contains these errors. Some, but not all of them are manually taken care of.

## 1.1 Problem Definition

The purpose of the research is to build a method by which the OCR documents quality can be measured when we have no idea about the corresponding original correct text document.

First we wrote a program that actually compares both set of files side by side and calculates the error known as Word Error. By using that program we assign each OCR document a class label. That class label is used as the target class for calculating accuracy of those classifiers we build. The $WER$ is calculated in the following fashion,

$$WER = (S + D + I)/N * 100 \tag{1.1}$$

where,
$S$ = Number of Substitutions.
$D$ = Number of deletions.
$I$= Number of Insertions.
$N$ = Number of characters in the document

Depending on the error we can classify the documents into 3 classes $A, B$ and $C$. Which is defined as-

$$Class = \begin{cases} A, & \text{if} \quad \text{accuracy} \geq 90\% \\ B, & \text{if} \quad 90\% > \text{accuracy} \geq 85\% \\ C, \ Otherwise \end{cases} \tag{1.2}$$

So for each OCR document we have these class labels as target class label which we have calculated as mentioned above using some a certain program. Our goal is to calculate the $WER$ and assign a class to the OCR documents without comparing it to its corresponding correct text document. At the end we compare the assigned class by our method with the already assigned (target) class and calculate how much accurately we can do this classification.

## 1.2 Our work

Our work involves reading an input OCR file, scanning the entire file, finding errors and finally assign a class to the document. The scanning and the measurement of error involves the use of appropriate Data Structure and Algorithms, sometimes Pattern Recognition technique and last but not least Natural Language Processing.

- The Data Structure based approach is mostly involving lexicon, bi-Gram dictionary and/or Trie data structure.
- The pattern recognition based approach involves feature selection and extraction and train a Random Forest classifier.
- We also tried one approach called word probability approach where we calculated the occurrence and probability of each word in an OCR document and tried to get something out of it. Two methods we tried there. Results are not so good.
- The Natural Language Processing based approach involves training and using the Recurrent Neural Network Language Models.

# Chapter 2

# Related works

## 2.1 Spell Checking

The OCR documents accuracy prediction problem is till now unexplored territory of computer science research. The first of the three approaches we made as mentioned above mostly involves spell checking and spelling correction. Spell checking is a well-known task in computational linguistics, started back in 1960s. One of the most notably work was of Damerau (1964). The very first idea of spell checking comes from dictionary based approach. From the corpus they build a dictionary/lexicon and given any input word, it is searched throughout the dictionary. If not found, then it is considered as error. But the problem of this is that dictionary can be of huge size. So people started thinking alternative approaches.

Riseman and Hanson in 1976 proposed a technique [10] that uses dictionary indirectly. It was a trigram based approach. They have a trigram dictionary and any trigram from a particular word not appearing in the dictionary is considered as error. To detect typographical errors, Morris and Cherry in 1975 proposed a technique [15] that does not use dictionary at all. The algorithm described here divides the text into trigrams and creates a table of these, noting how often each one occurs in this particular piece of text. Depending on this frequency it assigns some peculiarity index. Based on that index it detects spelling mistakes.

Researchers so far have published a lot of papers related to this spell checking and correction. Renato Corderio and Marcos Zampieri proposed clustering algorithm based checking [5] in 2013 which provides insights of a very new approach.Knuth 1973, and Davidson 1963 devised SOUNDEX system [8] and Pollock and Zamora in 1984 proposed SPEEDCOP system [17] both of which are very good in spell checking and correction.

## 2.2 Language Model

Language Models are now a days famous for word prediction and context based spell checking process. Applications of statistical language modelling includes speech recognition (Jelinek, 1998 [20]; Schwenk, 2010 [12]), text generation (de Novais et al., 2010 [6]), machine translation (Brown et al., 1993 [3]; Och and Ney, 2002 [16]; Kirchhoff and Yang, 2005: [11]), spelling

correction (Ahmed et al., 2009 [1]),optical character recognition and handwriting recognition (Vinciarelli et al., 2004 [4]), syntactic (Huang et al., 2014) and semantic processing (Deschacht et al., 2012).

The recent approaches of language model based on NGram is available as SRILM toolkit [19]. It can be used for evaluating statistical language models and can also be used in spelling correction or word prediction. Michal Righter and Pavel Stranak's Korektor system [18] is developed for context sensitive spelling correction and diacritics completion.

# Chapter 3

# Data Set Description

Selecting a good corpus is always challenging. It is a general practice to have a perfect corpus and properly labeled training data. Also the corpus should be as big as possible or more importantly it should be as complete as possible. The corpus we used is taken from a collection of our local news daily "Anandabazar Patrika". Each file has some detailed report based on any particular headlines appearing in a certain day. Each file is scanned later using a faulty OCR system to generate corresponding erroneous OCR file. As mentioned earlier we have to sets of files - correct and OCR. Their description is given below -

Total correct text Files = 62824.

Total OCR text Files = 62790.

Total number of unique words (i.e vocabulary size) = 451486.

The corpus contains a total of = 23545051 words.

The file size varies from 1KB to 23 KB having 21 as lowest number of words in a file to maximum of over 2000 words. As the data is in UTF-8 encoding format we have 111 differnt unicode characters in correct text files compared to only 73 different unicodes over all OCR documents.

The data set described above is huge in terms of text document. But the first challenge to overcome is to make sure that our training data set (in this case correct text files) is error free and correctly labeled. But in reality this is hardly the case. For example there are still misspelled words in correct texts-

শিলিগুড়িপ্রতিযোগিতার/ shiligu.DipratiyogitAra
রঘুনাথগঞ্জরঘুনাথগঞ্জ/raghunAthaga njaraghunAthaga nja
কাকোদকরেরসংবাদসংস্থা/kAkodakarerasa.nbAdasa.nsthA
পাঁচখেলাজয়ড্রহারগোলপার্থক্যপয়েন্টইস্টবেঙ্গল/pA.NchakhelAjaYaDrahAragolapArthakyapaYenTaisTabe Nga

We manually corrected some of them. But with this huge corpus, it was practically impossible to correct all. So a lot of misspelled words still remained and we had to sacrifice accuracy a little.

## 3.1 Unicode and UTF-8

Research based on text processing or NLP has to deal with different languages. The data structures and algorithms proposed while working on a language most of the time should work on any other language. But sometimes results we get may differ a little. As we choose a Bengali corpus we first have to understand how Bengali letters are stored in a text file. Both sets of data (correct text and OCR) are of UTF-8 encoding. Now question may arise about the difference between Unicode and UTF-8.

Actually, comparing UTF-8 and Unicode is like comparing apples and oranges: Unicode is a character - UTF-8 is an encoding. A character set is a list of characters each assigned with a unique numbers (these numbers are sometimes referred to as "code points"). For example, in the Unicode character set, the number for A is 41. An encoding on the other hand is an algorithm that translates a list of numbers to binary so it can be stored on disk. For example UTF-8 would translate the number sequence 1, 2, 3, 4 like this:
00000001 00000010 00000011 00000100
Once the data is translated into binary it can be saved in disk.
Lets go through an example. Say an application reads the following from the disk:
1101000 1100101 1101100 1101100 1101111
The application knows this data represent a Unicode string encoded with UTF-8 and will show this as text to the user. First step, is to convert the binary data to numbers. The application uses the UTF-8 algorithm to decode the data. In this case, the decoder returns this:
104 101 108 108 111
Since the application knows this is a Unicode string, it can assume each number represents a character. It uses the Unicode character set to translate each number to a corresponding character. The resulting string is "hello".

So in conclusion we can say :: UTF-8 and Unicode cannot be compared. UTF-8 is an encoding used to translate numbers into binary data. Unicode is a character set used to translate characters into numbers.

## 3.2 Windows naming convention of Unicode text

Windows uses UTF-16LE encoding internally as the memory storage format for Unicode strings. It considers this to be the natural encoding of Unicode text. In the Windows world, there are ANSI strings (system's default encoding) and there are Unicode strings (stored internally as UTF-16LE). This was all devised in the early days of Unicode(before UTF-8 was invented). Before that we had UCS-2 which later proved to be insufficient. So Windows started Unicode strings(stored internally as UTF-16LE) . This is why Windows's support for UTF-8 is all-round poor. This misguided naming scheme became part of the user interface. A text editor that uses Windows's encoding support to provide this encodings will automatically and inappropriately describe UTF-16LE as "Unicode", and UTF-16BE

as “Unicode big-endian”.

### Challenge of Handling Unicodes

Unlike English language, Bengali has a lot of mixed characters (also called yuktakshar) and different fonts like Akash, Lohit Bangla, Mukti etc. Some characters in one font is defined differently than another. For that reason sometimes two Bengali characters can be combined to form a third character but having different number of unicodes. For example -

অ (<U0985>) + া (<U09BE>) = আ (<0986>)
ে (<U09C7>)+ া (<U09BE>) = ো (<U09CB>)
ে (<U09C7>) + ৗ (<U09D7>) = ৌ (<U09CC>)

Figure 3.1: Unicode Combinations

We have to take care of them manually and we made sure each character has only one representation by replacing each consecutive appearances of such unicodes by the corresponding unicodes written in the right hand side.

# Chapter 4

# Our Approaches

## 4.1 Predicting OCR's behavior

The first approach is about creating a dictionary from where we can get the idea of the behavior of OCR and thus can figure out those particular kind of errors that were generated. For that we need to compare both the files side by side and generate a (wrong,correct) pair of word list. We write that pair in our dictionary. We used Hash map for this purpose. It may use more space but it is really fast and reduces the time complexity by a huge margin. Once we create the dictionary, we can use it to test OCR files.

The algorithm for first part i.e creation of (wrong,correct) pair dictionary is as follows,-

**Algorithm 1** Predicting OCR behavior

**Input:** Pairs of OCR document $R$ and its corresponding correct text documents $S$.

**Output:** A one to one mapping containing of pair of words in the form (wrong,correct).

1: **for** each word $w_i$ in $R$ check whether that word occurs in $S$ **do**
2: **if** yes **then**
3: that is a correct word
4: **else**
5: That is a wrong word
6: find a word $w_j$ from $S$ having minimum edit distance and assign them as a pair ($w_i$,$w_j$) and put them into a hash map and keep a count of how many time that pair appears.
7: **end if**
8: **end for**
9: now for each wrong word $w_i$ we find a word $w_j$ that has the maximum count i.e with each occurrence of misspelled $w_i$ we got $w_j$ as its closest word maximum number of times. We choose ($w_i$,$w_j$) and put them into dictionary.

While making the pairs we assigned maximum allowable edit distance as 6 (which is chosen empirically), i.e if the pair has edit distance more than 6 we don't choose them. The Bengali scripts are represented in UTF-8 format and average length of a word is more than

that English word. So an Edit distance of 6 is very very rare in English OCR but in Bengali it can be unlikely but still possible.

Once we have the result with us the pre-processing is done. Say dictionary is written in the file $T$. The dictionary [1] looks like (the 1st column is the wrong word and the 3rd column is the corresponding correct word) -

Table 4.1: Dictionary

| | | | |
|---|---|---|---|
| মবষ্যগণ | /mabaShyagaNa | মনুষ্যগণ | /manuShyagaNa |
| তিমিরবান্ব | /timirabAnba | তিমিরবাবু | /timirabAbu |
| কৌন. | /kauna. | কৌন’ | /kauna’ |
| ছিলেন. | /chhilena. | ছিলেন | /chhilena |
| যায়ে. | /yAYe. | যায়ে’ | /yAYe’ |
| তফাসলই | /taphAsalai | তফসিলই | /taphasilai |
| মিথিলার | /mithilAra | মুসলিম | /musalima |
| শিক্ষক.অভিভাবকেরা | /shikShaka.abhibhAbakerA | শিক্ষক-অভিভাবকেরা | /shikShaka-abhibhAbakerA |
| ছিলেন1 | /chhilena1 | পড়লেন | /pa.Dalena |
| প্রশ্র..হান্ধা | /prashra..hAndhA | প্রশ্ন:হাল্কা | /prashna:hAlkA |
| অঙ্গভঙ্গি.সহ | /a Ngabha Ngi.saha | অঙ্গভঙ্গি-সহ | /a Ngabha Ngi-saha |
| ওাষবাসের | /AShabAsera | ’’চাষবাসের | /’’chAShabAsera |
| অগুন | /aguna | আগুন | /Aguna |
| আর্যভমিতে | /Aryabhamite | আর্যভূমিতে | /AryabhUmite |
| ব্রড়িই | /bra.Dii | গুঁড়িই | /gu.N.Dii |
| অঘঢন | /aghaDhana | অঘটন | /aghaTana |

One interesting thing top notice here is that, from the pair (wrong,correct) word it would be a common practice to take the node whose edit distance is minimum as that can be the closest word. But when we deal with texts generated by OCR, it may not be true always. So we took the pair which has maximum frequency i.e max occurrence indicating that particular pair of (wrong, correct) word has appeared most number of times together and is a proper pair. If we take minimum edit distance then following is an example of what can go wrong-

[1] Usually in UTF-8 encoding, the mixed letters appear good in text documents but not in latex. None of the latex fonts we used actually could print the mixed words properly.

Table 4.2: Mistake while taking minimum edit distance

| Misspelled Word | Actual Word | Closest Word |
|---|---|---|
| থলিশ/thalisha | পুলিশ/pulisha | থলি/thali |
| অবসন্ধানের/abasandhAnera | অনুসন্ধানের/anusandhAnera | পথসন্ধানের/pathasandhAnera |
| সিঙ্গাথর/si NgAthara | সিঙ্গাপুর/si NgApura | সিঙ্গার/si NgAra |
| আলিথর/Alithara | আলিপুর/Alipura | আলির/Alira |

Now we can use that to calculate the $WER$ in test phase and assign the class to our OCR files. The algorithm,-

**Algorithm 2** Algorithm for calculating accuracy

**Input:** OCR file $f$ and preprocessed text $T$ file.

**Output:** $WER$ as $E$ of OCR file and class.

1: from $T$ build a hash map $H$ having key as the 1st column i.e wrong word and value as the edit distance between the pair of (wrong,correct) word.
2: set error as $E \leftarrow 0$
3: **for** Each word $w_i$ in $f$ **do**
4:  **if** $w_i \notin H$ **then**
5:  $E \leftarrow E + H.get(w_i)$
6:  **end if**
7: **end for**
8: $E \leftarrow E+$ C(other unknown symbols)
9: $E \leftarrow E/$ (Total Characters in f)
10: Assign class A,B or C to $F$ according to the value of $E$ and conditions mentioned in equation (1.2).

Here C(other unknown symbols) is the count of all other symbols like .,{ ,*,}, # etc that usually do not appear in a Bengali text file.

## 4.2 When OCR data not available during training

The first procedure described above, we built the dictionary of pair of words using both OCR and correct text files. So we had OCR files as training data and from there we guessed the OCR system's behaviour. The following procedures however, are suitable when we do not have the OCR file beforehand and all we have is the correct text files.

### 4.2.1 Using Lexicon only

From the given set of correct files we build a lexicon.

**Algorithm 3** Accuracy Calculation using Lexicon

**Input:** OCR file $f$ and Lexicon $L$ ( which is stored in a Hash map or hash set).

**Output:** $WER$ as $E$ of OCR file and class.

set error as $E \leftarrow 0$
**for** Each word $w_i$ in $f$ **do**
  **if** $w_i \notin L$ **then**
    **for** Each word $w_j$ in $L$ **do**
      find the $w_j$ for which the $EditDistance(w_i, w_j)$ is minimum, which is (say) denoted by $Dist_{min}$
    **end for**
    $E \leftarrow E + Dist_{min}$
  **end if**
**end for**
$E \leftarrow E+$ C(other unknown symbols)
$E \leftarrow E/$ (Total Characters in f)
Assign class A,B or C to $F$ according to the value of $E$ and conditions mentioned in equation (1.2).

The above algorithm has time complexity $O(n * k)$ where $n$ is the size of the dictionary and $k$ is the number of words in OCR file. Obviously $n \gg k$ and time complexity is huge. In the following section we aim to reduce this time complexity.

### 4.2.2 Using Lexicon vocabulary and Trie Data Structure

We used two Trie structures, one is forward Trie $T_f$ and another is Reverse Trie $T_r$ (where each word is stored in reverse order). In case when a word from OCR file is not found in Lexicon, we can conclude that the word is wrong. Then it is searched in both $T_f$ and $T_r$ . Both searches will be terminated before reaching leaf node but would return a set of words from there. We take the union of these two set as it has very high probability of having the corresponding correct word. Now the word which belongs to the set and has minimum edit distance from the misspelled word is considered as our desired word.

**Algorithm 4** Accuracy calculation using Lexicon and bi directional Trie search

**Input:** OCR file $f$ and Lexicon $L$ ( which is stored in a Hash map or hash set) and two $Trie$ structures $T_f$ and $T_r$.

**Output:** $WER$ as $E$ of OCR file and class.

1: **for** Each word $w_i$ in $f$ **do**
2:  **if** $w_i \notin L$ **then**
3:  search in $T_f$ which will end in a non leaf node & will return a set of words $F$.
4:  search in $T_r$ which will end in a non leaf node & will return a set of words $R$.
5:  $S \leftarrow F \cup R$
6:  **for** each word $w_j$ in $S$ **do**
7:  Find the $w_j$ for which the $EditDistance(w_i, w_j)$ is minimum. The min distance is (say) denoted by $Dist_{min}$
8:  **end for**
9:  $E \leftarrow E + Dist_{min}$
10:  **end if**
11: **end for**
12: $E \leftarrow E+$ C(other unknown symbols)
13: $E \leftarrow E/$ (Total Characters in f)
14: Assign class A,B or C to $F$ according to the value of $E$ and conditions mentioned in equation (1.2).

The creation of $L$,$T_f$ and $T_r$ can be considered as preprocessing and once loaded into memory, can be used in as many OCR file as desired. Then all we care about is processing all the words of a single OCR file. It may bug some people why we did not take the intersection between $F$ and $R$ as it could make the set $S$ much smaller, because in that case $S$ may not contain the particular word we are looking for. The above algorithm runs in $O(|S| * k)$ time., which is sure an improvement as $|S| \ll n$. The procedure is best when,-

(i) We have a single letter error in word.

(ii) The substring containing erroneous letters does not form a proper valid substring of another correct word, in which the set of words returned after unsuccessful Trie search may not contain the actual correct word.

The algorithm can still be improved with respect to time complexity but in the cost of space.

### 4.2.3 Using Lexicon vocabulary and Bi-Gram voting method

The preprocessing part here includes first creating an Index over *Lexicon*, where each word is assigned an unique integer number known as its index. Then we create another dictionary called *BiGram* dictionary which contains one bi unique bigram and the index numbers of all those words in which it has occurred. We used a simple hash map with key as bigram string and value as Hash set of integers. This dictionary along with Index on *Lexicon* are

stored in text file. When we want to test OCR files we just need to read the file and load the data into memory. After that we can test as many files as we wish.

The algorithm here uses more space than previous algorithm, but is much faster. Here it goes,-

**Algorithm 5** OCR accuracy prediction using Bigram Voting

**Input:** OCR file $f$ and Lexicon $L$ ( which is stored in a Hash map or hash set) and a bigram dictionary $B$ (which is stored in a Hash map whose key is a unique bigram and value is a Hash set of integers).

**Output:** $WER$ as $E$ of OCR file and class.

1: **for** Each word $w_i$ in $f$ **do**
2: **if** $w_i \notin L$ **then**
3: $E \leftarrow E+$ **CalculateError**$(w_i)$
4: **end if**
5: **end for**
6: $E \leftarrow E+$ C(other unknown symbols)
7: $E \leftarrow E/$ (Total Characters in f)
8: Assign class A,B or C to $F$ according to the value of $E$ and conditions mentioned in equation (1.2)

The function **CalculateError**$(W_i)$ is defined below,-

**Algorithm 6** Algorithm for Calculating Error

**Input:** Misspelled word $w$

**Output:** an integer *dist* indicating the error or the min edit distance of wrong word $w$ from its corresponding correct word.

1: Break $w$ into bigrams. For each bigram get the set of indexes. Those indexes appearing maximum number of times are put into a candidate set $S$.
2: From $S$ select the word that has minimum edit distance from $w$. That word is our corresponding correct word and the distance is the minimum edit distance.
3: Return that minimum distance.

The above algorithm is quickest. It uses extra space as we had to use Hash sets and Hash maps. But once processing for one word is done the allocated memory is freed and can be used in next word processing. It does not have problem (iii) of previous double Trie method.

All our approaches discussed here needs to calculate $EditDistance$ (also called Lavenstein distance). So our main aim was to prune the set of words and create as small set as possible at the end and select the one with min distance.

## 4.3 Pattern Recognition Approaches

Pattern recognition is a branch of machine learning that focuses on the recognition of patterns and regularities in data, although it is in some cases considered to be nearly synonymous with machine learning. Usually pattern recognition deals with recognition of different patters or features of training data and based on that extracting information from the data or building suitable classifier.

The OCR document's accuracy prediction is actually a classification problem. We classify the documents according to some properties. Till now the approaches discussed was to find the closest word for each wrong word and calculate the Edit distance which sum up to WER(word error rate). The next approach is more of getting different features from the OCR files and train a classifier for our classification purpose. We are going to use a very well know classifier "Random Forest".

### 4.3.1 What is Random Forest

Random forests is a notion of the general technique of random decision forests that are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

### 4.3.2 Building a Random Forest

#### Unicode Approach

This technique uses a random forest to measure the occurrences of unicodes in OCR document.OCR documents have 73 distinct unicodes. Here we are only considering the percentage of each unicodes appearing in an OCR documents. Each of the 73 unicodes are considered as one feature that has been used to train the model.

We used 1200 files of each class A,B and C. The R package we used has some limitation in the size of feature vector and number of training samples for each forest. So we created 12 small Forests and combined them into a single one. Each forest is made of feature vector from 100 files of each class.

$$f_i = \frac{N_i}{\sum N} \tag{4.1}$$

where,

$N_i$ = Number of times $i$th unicode appeared in OCR document.

N = Total number of unicodes(characters) in that OCR document.

We tested the the above Random Forest on 2000 files of class A and B and 500 class C files.

**Top W words Approach**

It is almost same as before. First we have set a value for W. Then read all the correct text documents we have and calculated the unigram frequency of each word. Using that frequency we got the top W words. Each of those words acted as a single feature. While training the random Forest, this time we used top W words and their occurrences in our OCR documents. The feature is selected as -

$$f_i = \frac{N_i}{\sum N} \tag{4.2}$$

where,
$N_i$ = Number of times $i$th word appeared in OCR document.
N = Total number of characters in the OCR document.

Same as before, we tested the the above Random Forest on 2000 files of class A and B and 500 class C files.
The value of W is taken as 20 and 50 (any one can try this procedure by using different values of W).

**More added features**

Both the Pattern Recognition approaches discussed above are improved upon by adding two extra features. The first of the two is number of spaces and the other one is number of different of special characters (like ,.!?- etc) in a file.

### 4.3.3 Word Probability Approach

When you do not have a correct text file with you or you do not even have vocabulary knowledge then this method can work. At First, from the training data (OCR document and its assigned class) we generated a matrix called Word Statistics ($W_{stat}$) for each class of documents. For each class(A,B or C) of documents the matrix contained each word($w_i$) in its first column followed by other metrics in the next column described below,-

Total Count($N_{w_i}$) = Number of times $w_i$ appeared in the Training set.
Document Count($D_{w_i}$) = Number of different documents in which $w_i$ appeared.
Total Number of words in the corpus = $(\sum w_i)$
Total Number of Documents = $(\sum D)$
Document Probability of each word P($w_i$) = $\frac{N_{w_i}}{D_{w_i}}$

$$\text{Corpus Probability} = \frac{N_{w_i}}{\sum w_i}$$

$$\text{Document Corpus Probability} = \frac{D_{w_i}}{\sum D}$$

$$\text{Word Corpus Probability} = \frac{N_{w_i}}{\sum D}$$

Here, "Total Number of Documents" means number of documents trained for any particular class. So we had $W_{stat}$ matrix for each class of documents. Then we tested these metrics on two separate algorithm as follows,-

**Algorithm 7** Accuracy Prediction using Document Probability Calculation

**Input:** An OCR document $D$ and $W_{stat}$ matrix for each class A,B and C, denoted by $W_{stat}(A)$, $W_{stat}(B)$ and $W_{stat}(C)$ .

**Output:** A Class assigned to the OCR document $D$.

1: **for** each word $w_i$ in $D$ **do**

2: From each class of $W_{stat}$ matrix get the **Document Probability** of $w_i$ and sum them up i.e

$SumA+ \leftarrow W_{stat}(A).getDocumentProbability(w_i)$

$SumB+ \leftarrow W_{stat}(B).getDocumentProbability(w_i)$

$SumC+ \leftarrow W_{stat}(C).getDocumentProbability(w_i)$

3: **end for**

4: The class of the document is assigned based on which class has maximum sum of **Document Probability** i.e based on out of $SumA$, $SumB$ and $SumC$ which one is maximum.

The second algorithms a bit different,-

**Algorithm 8** Accuracy Prediction using Word Classification

**Input:** An OCR document $D$ and $W_{stat}$ matrix for each class A,B and C, denoted by $W_{stat}(A)$, $W_{stat}(B)$ and $W_{stat}(C)$ .

**Output:** A Class assigned to the OCR document $D$.

1: **for** each word $w_i$ in $D$ **do**

2: assign class to $w_i$ as A,B or C based on which one of $W_{stat}(A)$, $W_{stat}(B)$ and $W_{stat}(C)$ has highest Document Probability.

3: **end for**

4: As each word in the document $D$ has been assigned a class, we just count which class has been assigned to most of the word in the document and that would be assigned as the class of the document.

We have written only **Document Probability** metric in the above algorithms. But we actually tested both the algorithms on all the other metrics as well. But only **Document Probability** has shown some positive result in our classification.

## 4.4 Recurrent Neural Network Language Model Approach

There are many RNN tools available these days in the internet. Among them Mikolov's recurrent neural network [14] is one of the best. It has faster training capabilities over large training data compared to Theano or Pybrain. Our next approach is based on Mikolov's RNN toolkit.

### 4.4.1 Advantage of RNN over n-gram Language Model

RNN do not make the Markov assumption, n-gram models do. RNN, in theory can take into account long-term dependencies when modeling natural language. According to Bengio et al [2] learning Long-Term Dependencies with Gradient Descent is difficult and, to my knowledge, Mikolov's work is not addressing this problem. His work focuses more on storing the entire history in the hidden layer as some kind of state which is useful in predicting probability of the next input pattern and the use of RNN for language modeling tasks by making them super fast to train in comparison to previous implementations. But sometimes it is suitable to have n-gram sequences as an input to the recurrent network as it would better learn long term dependencies.

N-gram techmique rely on simple smoothing like Kneser–Ney or Good–Turing [7]. With the increase in size of vocabulary a number of parameters would explode. RNN on the other hand has a greater representational power and intelligent smoothing power by taking into account syntactic and semantic features (see for example Turian et al. [13]).

The probability of a sequence of symbols (usually words) is computed using a chain rule as

$$P(w) = \sum P(w_i|\ (w_1.w_2.....w_{i-1})) \tag{4.3}$$

The most frequently used language models are based on the n-gram statistics, which are basically word co-occurrence frequencies. The maximum likelihood estimate of probability of word A in context H is then computed as

$$P(A|\ H) = \frac{C(HA)}{C(H)} \tag{4.4}$$

where C(HA) is the number of times that the HA sequence of words has occurred in the training data. The context H can consist of several words, for H = $\emptyset$, the model is called unigram, and it does not take into account history. For the usual trigram models $|H| = 2$. As many of these probability estimates are going to be zero (for all words that were not seen in the training data in a particular context H), we apply smoothing technique. This works by redistributing probabilities between seen and unseen (zero-frequency) events. The problem with this approach is that we assign probability based on a single observation, which sometimes leads to over/under estimation.

According to Mikolov's work, the input layer of RNNLM consists of a vector w(t) that represents the current word w(t) encoded as 1 of V (thus size of w(t) is equal to the size of the vocabulary), and of vector s(t - 1) that represents output values in the hidden layer from the previous time step. After the network is trained, the output layer y(t) represents , -

$$P(w) = P(w_{t+1}|\ w_t, s(t-1)) \tag{4.5}$$

### 4.4.2 Word level vs Character Level language Model

OCR documents accuracy prediction using language model again comes back to word prediction and WER calculation but this time focusing more on the context of the word rather than other means. The word level model is perhaps more suitable choice as it will predict the probability distribution of a word given its history or context. If, on the other hand, we apply character level model, we can only predict next character given the history, which raises the difficulty of detecting the occurrence of a wrong character in a wrong word (the same limitation as the a bi-directional Trie). Word level models are easier to apply in algorithms but have limitations when it comes to practically running the code. The reason is described as following, -

According to Mikolov the traditional training algorithm of RNNLM denoted as normal backpropagation, as the RNN is trained in the same way as normal feed forward network [9], with one hidden layer, with the only exception that the state of the input layer depends on the state of the hidden layer from previous time step. In Mikolov's paper [14], it was describe that such training approach is not optimal - the network tries to optimize prediction of the next word given the previous word and previous state of the hidden layer, but no effort is devoted towards actually storing in the hidden layer state some information that can be actually useful in the future. This kind of design will not be able to remember long context information in the state of the hidden layer. However, a simple extension of the training algorithm can ensure that the network will learn what information to store in the hidden layer - this is the so-called Backpropagation through time algorithm.

After applying the Backpropagation through time algorithm, the output layer 'y' represents a probability distribution of the next word $w_{t+1}$ given the history. The time complexity of one training or test step is proportional to

$$O = H \times H + H \times V = H \times (H + V) \tag{4.6}$$

Where H is size of the hidden layer and V is size of the vocabulary.
For Word level Language Model, $|V| = 451486$ .
For Character level Language Model, $|V| = 95$

So, after training the word level model (in spite of using frequency banning [21]) we end up getting a model of size 2.7 GB. For each word in a document(which may contain up to 1000 or more words) loading that model in memory and running the code takes a huge time (almost 1hr to process each document). So, in order to avoid this technical limitation we trained a char level language model.

### 4.4.3 Training

#### Typical Choice of Hyper-Parameters

Due to huge computational complexity of neural network based language models, only a person with some experience can train the large scale model. Due to the fact that certain parameter combinations are too expensive to explore we choose the following parameters as suggested by Mikolov.

We ran BPTT(Backpropagation through time) algorithm for at 10 steps. Size of the hidden layer = 400. BPTT algorithm has been run through block mode(block size of 20). Using frequency binning approach it is desirable to make sure that class size does not exceed sqrt($|V|$). But while training a character level model we can achieve highest accuracy by turning off the class based training by setting the -class parameter to 1.

The algorithm for predicting correct word given a wrong word from RNN model is,-

**Algorithm 9** Predicting correct word using RNNLM

**Input:** A misspelled word $w_i$, its previous two context words $w_{i-1}$ and $w_{i-2}$, the model $M$ and two Trie data structure $T_{forward}$ and $T_{reverse}$

**Output:** The correct output word.

1: Generate the candidate set from $S$ from $w_i$ using $T_{forward}$ and $T_{reverse}$.

2: for each word $w$ in $S$ put $w_{i-1}$, $w_{i-2}$ and $w$ in character array and feed them into $M$. Get the top 10 words having the highest probability. From them select the word having minimum edit distance from $w_i$.

3: Return that word.

# Chapter 5

# Results and Analysis

The results for all three approaches are shown separately.

Table 5.1: Approach1 Result

| | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| OCR Prediction | 0.789 | 0.7875 | 0.789 | 0.787 |
| Only Lexicon | 0.7962 | 0.7948 | 0.7962 | 0.7947 |
| Double Trie | 0.6966 | 0.6991 | 0.6970 | 0.6921 |
| Bigram Voting | 0.7842 | 0.7826 | 0.7842 | 0.7825 |

OCR Prediction method is the fastest among all four methods described above. The average processing speed for each file is given below ,

- OCR Prediction = 2.948 ms
- Only Lexicon = 11.254 sec
- Double Tri = 993 ms
- Bigram Voting = 310 ms

OCR prediction method is on the other hand takes the least memory resources while running because it just loads lexicon and (wrong,correct) word pair dictionary in memory(which contains 365987 pairs). The better approach should be if we just store the pair character wise i.e just keeping track of which character or bigram or trigram gets change into which corresponding character, bigram or trigram. The dictionary would be much smaller. But that approach won't work as we have found the following example while attempting that method,-

Table 5.2: Example

| | | | |
|---|---|---|---|
| নাছগুলি | /nAchhaguli | গাছগুলি | /gAchhaguli |
| বন.না | /bana.nA | বর্ণনা | /barNanA |

We can clearly see that two misspelled words নাছগুলি and বন.না we have the ন. The changes have taken place accordingly

- In the first example ন has become গ.
- In the second example ন has become র্

There are many more example not shown here. So from that we can conclude that not all characters or character pairs change into same thing every time they occur in OCR document. An interesting fact to notice is that পুলিশ has been changed to থলিশ many times but not always. Also each occurrence of পু does not always change to থ in every misspelled word.

The following table shows a comparison between the closest words found corresponding to a misspelled word using "Only Lexicon", "Double Trie" and "Bigram Voting" approach.

Table 5.3: Comparison of different approaches while predicting the correct word

| Misspelled Word | Lexicon Word | biGram Word | Double Trie Word |
|---|---|---|---|
| ফিলিপিল | জিলিপিই | ফিলিপিনস | ফিলিপিনস |
| রুডপ্রয়াগ | সুপ্রয়াস | রুদ্রপ্রয়াগ | রুদ্রপ্রয়াগ |
| পরিবর্যন | পরিবর্জন | পরিবর্দন | পরিবর্ধন |
| সবষ্টি | সমষ্টি | সূষ্টি | সূষ্টি |
| নানাাবধ | নানাবিধ | জানাানো | নানাভাবে |
| কবাক্য | কূবাক্য | একবাক্য | একবাক্য |
| থটিনাটি | রটনাটি | খুঁটিনাটি | খুঁটিনাটি |
| ছড়তে | ছিড়তে | ঝড়তে | লড়তে |
| জুলাহ | জুলা | জুলা | জুলাই |
| উতসব | উসব | এতসব | উৎসব |

The following is the result for our **pattern recognition approach** using **Random Forest**

Table 5.4: Results:: Random Forest

| | Accuracy | Precision | Recall | F Score |
|---|---|---|---|---|
| Unicode Approach | 0.7740 | 0.7369 | 0.7843 | 0.7542 |
| Top 20 Words | 0.7380 | 0.7221 | 0.7556 | 0.7356 |
| Top 50 Words | 0.7523 | 0.7265 | 0.7668 | 0.7415 |

The following is the result for **Word Probability** based approach,-

Table 5.5: Results:: Word Probability approach

| | Accuracy | Precision | Recall | F Score |
|---|---|---|---|---|
| Method 1 | 0.6111 | 0.5097 | 0.6510 | 0.50 |
| Method 2 | 0.6148 | 0.5348 | 0.6395 | 0.5266 |

The following is the result for our **character level RNNLM**,-

Table 5.6: Result:: Character Level RNNLM

| | Accuracy | Precision | Recall | F Score |
|---|---|---|---|---|
| RNNLM | 0.7088 | 0.6623 | 0.665 | 0.6634 |

The word level RNNLM would give us more accurate results but we could not use it due to its size. The word level model is better due to the fact that we only had to predict(or in other words, calculate the probability of) a single word. In character level RNNLM we had to predict(calculate probability of) a sequence of characters. Here also we get a candidate set from double Trie data structure to make sure we do not have to calculate the probability over the entire lexicon. Each word from the candidate set is converted into a character sequence and then fed into our trained RNN model to get the probability(more accurately, to prevent floating point underflow we used *log* probability). Our accuracy is low here because Double Trie may not always return a proper candidate set which contains our desired correct word. So the error from Double Trie method is actually added here. To get more accuracy we had to calculate probability over the entire lexicon which will eventually come at the cost of higher time complexity.

# Chapter 6

# Conclusion and Future work

We have approached the OCR document's accuracy prediction problem in three different ways, each of which has some strong point and some draw backs. But our goal always remained the same - more accuracy in less time using lesser memory resources.

- When predicting the correct word for a misspelled word we tried to reduce the number of times edit distance has been calculated. First we searched the entire dictionary which caused us for every misspelled word 451486 distance calculations.
- Next, we generated a candidate set (using double Trie approach first followed by bigram voting approach) from where we get the correct word and thus reducing the number of distance calculations. We did sacrifice the accuracy a little (as sometimes the candidate set may not have the required word) and used more memory space but reduced time complexity.

The next approach we tried is based on a Random Forest classifier. This is a completely different approach where we extracted different features from our OCR documents and created a feature vector for each file. Then we built the random forest and tested OCR documents with a certain level of accuracy.

- Though we know that Random Forest is almost a black box approach still we have left a space of improvement here. We can try to extract features differently.
- We can combine different features and assign different weight to them.
- We can also use other classifiers such as SVM or MLP classifiers.

There is another approach based on word statistics calculation. Where we calculated the probability or frequency of occurrence of each word(or you can also call it pattern) of training OCR documents(correct and misspelled word) and from that we applied two different approaches on test documents. To be honest, result might not always get to be the exact same if we have different documents instead of the test set used. Also there are other metrics we used that did not give us any positive results at all.

The last approach is based on RNNLM. Though in OCR documents, it is hardly the case of grammatical anamoly or unlikely combination of words, we can still train an RNNLM and get a good prediction.

- Usually RNNLM or any other language model is more suitable for diacritics completion. Using that in OCR document also gave us good results but in expense time complexity. As RNNLM is proved to be superior over other language models, using it efficiently is an issue.
- The algorithm we used here is predicting and correcting each word from the beginning and every time we have a candidate set corresponding to a misspelled word it will load the model in the memory and calculate the probabilities. Some how if we can pre-load the model once and for all in the memory and use it again again it would save us a huge lot of time and then it would be easier for us to use word level RNNLM instead of character level RNNLM.

# Bibliography


[1] Ahmed, F., Luca, E. W. D., and Nürnberger, A. Revised n-gram based automatic spelling correction tool to improve retrieval effectiveness. *Polibits*, 40 (2009), 39–48.

[2] Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks 5*, 2 (1994), 157–166.

[3] Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics 19*, 2 (1993), 263–311.

[4] Bunke, H., Bengio, S., and Vinciarelli, A. Offline recognition of unconstrained handwritten texts using hmms and statistical language models. *IEEE transactions on Pattern analysis and Machine intelligence 26*, 6 (2004), 709–720.

[5] Cordeiro De Amorim, R., and Zampieri, M. Effective spell checking methods using clustering algorithms. In *Proceedings of Recent Advances in Natural Language Processing* (2013), Association for Computational Linguistics.

[6] De Novais, E. M., Tadeu, T. D., and Paraboni, I. Improved text generation using n-gram statistics. In *Ibero-American Conference on Artificial Intelligence* (2010), Springer, pp. 316–325.

[7] Gale, W., and Sampson, G. Good-turing smoothing without tears. *Journal of Quantitative Linguistics 2*, 3 (1995), 217–237.

[8] Greenfield, R., et al. An experiment to measure the performance of phonetic key compression retrieval schemes. *Meth. Inform. Med 16*, 4 (1977), 230–233.

[9] Hagan, M. T., and Menhaj, M. B. Training feedforward networks with the marquardt algorithm. *IEEE transactions on Neural Networks 5*, 6 (1994), 989–993.

[10] Hanson, A. R., Riseman, E. M., and Fisher, E. Context in word recognition. *Pattern Recognition 8*, 1 (1976), 35–45.

[11] Kirchhoff, K., and Yang, M. Improved language modeling for statistical machine translation. In *Proceedings of the ACL Workshop on Building and Using Parallel Texts* (2005), Association for Computational Linguistics, pp. 125–128.

[12] Kuo, H.-K. J., Mangu, L., Emami, A., and Zitouni, I. Morphological and syntactic features for arabic speech recognition. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing* (2010), IEEE, pp. 5190–5193.

[13] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1* (2011), Association for Computational Linguistics, pp. 142–150.

[14] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. Recurrent neural network based language model. In *Interspeech* (2010), vol. 2, p. 3.

[15] Morris, R., and Cherry, L. L. Computer detection of typographical errors. *IEEE Transactions on Professional Communication*, 1 (1975), 54–56.

[16] Och, F. J., and Ney, H. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics* (2002), Association for Computational Linguistics, pp. 295–302.

[17] Pollock, J. J., and Zamora, A. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM 27*, 4 (1984), 358–368.

[18] Richter, M., StraÅák, P., and Rosen, A. Korektor-a system for contextual spell-checking and diacritics completion. In *COLING (Posters)* (2012), pp. 1019–1028.

[19] Stolcke, A., et al. Srilm-an extensible language modeling toolkit. In *Interspeech* (2002), vol. 2002, p. 2002.

[20] Zamora-Martinez, F., Castro-Bleda, M. J., and Schwenk, H. N-gram-based machine translation enhanced with neural networks for the french-english btec-iwslt'10 task. In *International Workshop on Spoken Language Translation (IWSLT) 2010* (2010).

[21] Zweig, G., and Makarychev, K. Speed regularization and optimality in word classing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), IEEE, pp. 8237–8241.