A System for Classification of Crowdsourced Text Correction (CCTC)

Megha Gupta
Dept. of Computer Science
IIIT-Delhi, India.
meghag@iiitd.ac.in

Haimonti Dutta*
Department of Management Science and Systems
State University of New York, Buffalo
New York, 14260
haimonti@buffalo.edu

Brian Geiger
Center for Bibliographical Studies and Research
University of California, Riverside.
brian.geiger@ucr.edu

ABSTRACT

Optical Character Recognition (OCR) is a commonly used technique for digitizing printed material enabling them to be displayed online, searched and used in text mining applications. The text generated from OCR devices is often garbled due to variations in quality of the input paper, size and style of the font and column layout. This adversely affects retrieval effectiveness and hence techniques for cleaning the garbled text need to be improvised. Often such techniques involve laborious and time consuming manual processing of data. This paper presents a prototype system for **C**lassification of **C**rowdsourced **T**ext **C**orrection (CCTC) which takes as input log files containing garbled and manually corrected OCR text, parses and tokenizes them and builds models for categorizing the corrections using state-of-the-art machine learning algorithms. Retrieval effectiveness on the California Digital Newspaper Collection is measured using mean reciprocal rank. This prototype system is expected to be deployed on historical newspaper archives that make extensive use of user text corrections.

1. INTRODUCTION

Crowdsourcing is used extensively in cultural heritage and digital history related projects in recent years to digitize, create and process content and provide editorial or processing interventions. For example, the Australian Newspapers Digitization Program [2] allows communities to explore their rich newspaper heritage by enabling free online public access to over 830,000 newspaper pages containing 8.4 million articles. The public enhanced the data by correcting over 7 million lines of text and adding 200,000 tags and 4600 comments [14]. FamilySearch [18] made available handwritten digital images of births, deaths and marriage records for transcription by the public. The New York Public Library has 1,277,616 dishes transcribed to date from 17,079 menus.

In all of the above crowdsourcing projects, large volumes of data are generated by users. These include tags, folksonomies, flagged content, information on history, relationship and preference data, structured labels describing objects and creative responses [23]. However, little statistical analysis is done of the user generated content in most cases. Assessment of data quality obtained by leveraging the "wisdom of the crowd" remains an open problem.

In this paper, we focus on understanding the nature of text corrections done by users of an old historic newspaper archive. The newspapers are made available for searching on the Internet after the following processes take place: (1) the microfilm copy or paper original is scanned; (2) metadata is assigned for each page to improve the search capability of the newspaper; (3) OCR software is run over high resolution images to create searchable full text. The OCR scanning process is far from perfect and the documents generated from it contains a large amount of garbled text. A user is presented with a high resolution image of a newspaper page along with erroneous or distorted text from the OCR and is expected to rectify the garbled words as appropriate. A prototype for a system that can be used for **C**lassification of **C**rowdsourced **T**ext **C**orrection (CCTC) is presented which can answer simple questions such as "What are the different kinds corrections proposed by users?" and provide statistics generated from the correction process. The output from the system can be used to enhance search and retrieval.

The study used log files generated from text correction software in use at the California Digital Newspaper Collection (CDNC)[1], which contains over 400,000 pages of newspapers published in California between 1846-1922. The work done on reCAPTCHA [30] channelizes the old printed text which OCR devices failed to recognize. This work aims to correct the the printed books with high accuracy apart from differentiating between computers and humans by asking users to correct suspicious distorted words. To the best of our knowledge, this is the first attempt to statistically analyze, model and classify OCR error corrections provided by the crowd. We posit that such a classification system will be beneficial when attempting to compensate the annotators; it can also be used for task allocation if some users are more

*The author is also affiliated to the Institute of Data Science and Engineering (IDSE), Columbia University and is an adjunct professor at IIIT-Delhi.

WOODSTOCK '97 El Paso, Texas USA

[1]http://cdnc.ucr.edu/cgi-bin/cdnc

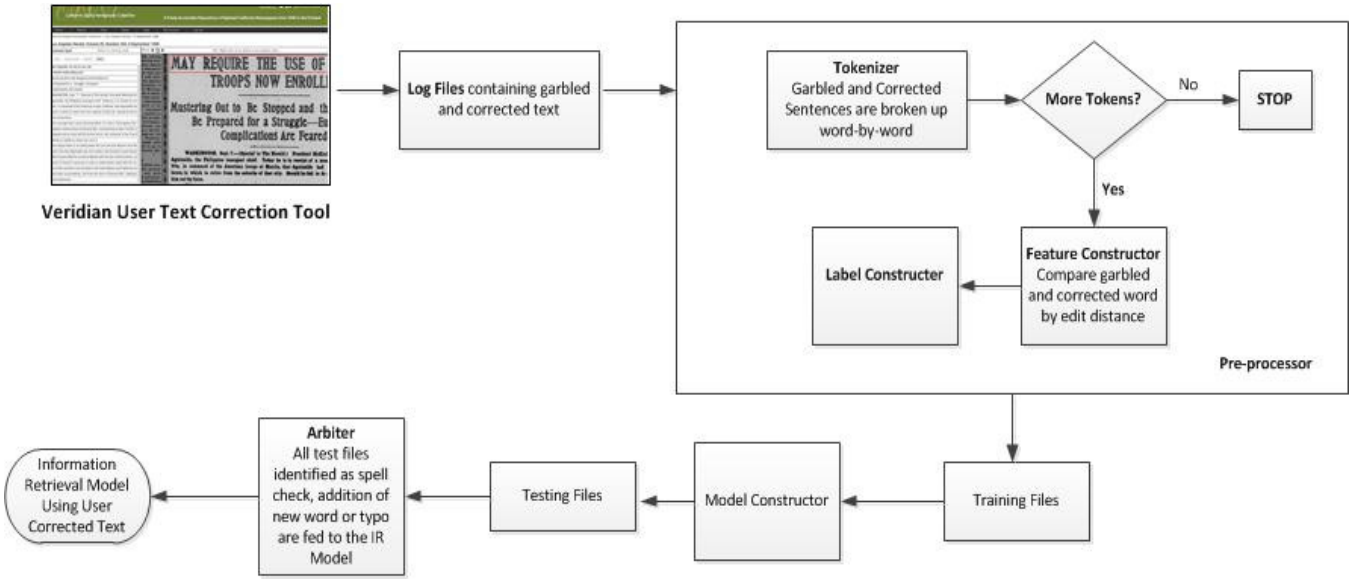

Figure 1: The Architecture of the Proposed System

comfortable with certain type of corrections than others.
Organization: Section 2 describes the architecture of the proposed system; Section 3 presents empirical and scalability results on real-world data collected at CDNC; Section 4 presents information retrieval techniques; Section 5 discusses related work. Finally, Section 6 concludes the paper.

2. ARCHITECTURE OF THE PROPOSED SYSTEM

The **C**lassification of **C**rowdsourced **T**ext **C**orrection (CCTC) system has the following components:

1. The **Veridian User Text Correction**[2] tool which takes as input a scanned page of the newspaper and enables users to correct OCR errors as they come across them. Figure 2a shows an example of a scanned page from "The Amador Ledger" published on January 26, 1900. The article to be corrected by a user is highlighted. The raw OCR text from the article and the tool used by patrons to correct text is shown in figure 2b.

2. **Log Files:** All corrections performed by the annotators are recorded in log files. To date approximately 1,705,149 lines have been edited by 848 annotators which resulted in 235 log files. A sample of 191 files has been used for this work. Each log file is generated at the issue-level and contains XML data about the pages in the issue. Table 1 describes the structure of the log file. The following information is provided about the corrections made by the patrons: (a) *Page Id:* The id of the page in which editing was done. (b) *Block Id:* The id of the paragraph containing the line corrected by the user. (c) *Line Id:* The id of the line edited by the user. (d) *Old Text Value* is the garbled text generated by the OCR device and replaced by the user. (e) *New Text Value* is the corrected text with which the old text was replaced.

<TextCorrectedLine lineID="P2_TL00800">
<OldTextValue>**Spll, Stales**</OldTextValue>
<NewTextValue>**Union Stables**</NewTextValue>
</TextCorrectedLine>
<TextCorrectedLine lineID="P2_TL00801">
<OldTextValue>****?ï£¡** Under Webb Hall ***
</OldTextValue>
<NewTextValue>**Under Webb Hall** </NewTextValue>
</TextCorrectedLine>

Table 1: A segment of the log file

3. **Preprocessor:** The preprocessor has three main components:

   - **Tokenizer:** The old text and the corresponding new text from the log file is tokenized by white-spaces. There are 44,022 tokens of which 21,108 are corrected by the annotators.
   - **Feature Constructor**
     Two different kinds of features are generated from the raw text - (1) word level and (2) character level features. Word level features are carefully crafted by hand and form the baseline for the classification task described in item 4 of Section 2. Character level features are automatically generated from the text corrections using machine learning algorithms. We believe these character level features will help us understand the process of automation and classification of user corrections.
     Word-level feature construction:
     Features are crafted by computing the Levenshtein edit distance between the old word and its correction. The Levenshtein edit distance [32] is defined

[2]http://veridiansoftware.com/crowdsourcing/

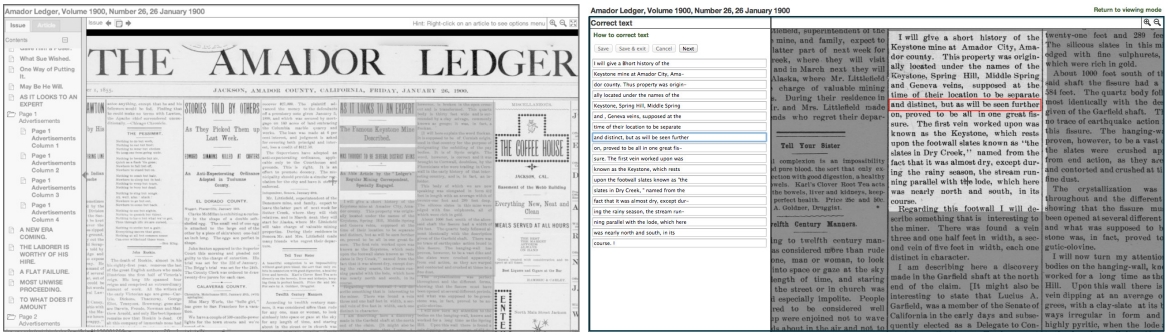

Figure 2: The Amador Ledger, Jan. 26, 1900

as the minimum number of single edit operations (insertions, deletions and substitutions) required to convert one string into another. Six binary features are generated as follows: (a) **Difference Length Zero** : 1, if both the old word and new word have same length and 0 otherwise. (b) **Difference Length Above One** : 1, if the length of new word exceeds the length of old word and 0 otherwise. (c) **Edit Distance One**: 1, if single edit operation is required to convert old word to new word and 0 otherwise. For example, the feature is 1 for tokens "Under" and "under" or "the" and "them". (d) **Edit Distance Above One**: 1, if more than one edit operation is required to convert old word to new word and 0 otherwise. For "Spll," and "Union", value is 1 as more than one edit operation is required to convert from old to new token. (e) **Edit Distance is 1 and Case Change**: 1, if the two words have edit distance is exactly 1 and the first character of one string change from upper case to lower case or vice versa. For example, for "Stales" and "Stables", the value is 0 as there is no case change. (f) **Punctuation Difference**: 1, if both the old text and new text differ in any of the following punctuation marks (!"#$%&'*+,-./:;<=>?@[\]^‘{|}~).

Character-level feature construction:
The tokenization process of the text in the log files was done at the character level. If the text was deleted by the user as a part of correction process then the character "∼" represented the new character denoting that the user performed the task of deletion, for example, t → ∼ means that the user deleted character "t" from the old token. Whereas if the user added a new text then the character "+" represented that the user added text that was initially missing in the OCR data, for example + → o means that the user added character "o" as a part of correction process.
The character-level dataset consists of 58, 963 instances of which we encountered that 55, 612 had same character-level corrections indicating that the OCR makes the same type of mistakes repeatedly. The maximum number of redundant corrections noted were deletion of punctuations and special characters. There were 4344 deletions of comma, 2831 deletions of *Ä*. Amongst character to character corrections, 635 cases of o→e, 470 cases of b→h, 299 cases of u→n contributed maximum to the redundancy. The graph in figure 3, 4, 5, 6, 7 (See Appendix) shows that the histograms of the different character level corrections. The x-axis in the graph denotes the examples of character-level corrections whereas the y-axis denotes the number of corrections made on the examples in the entire corpus. The performance of the proposed character level feature dataset was compared against the baseline method of manually crafted word level dataset.
Apriori algorithm [3] implementation of WEKA 3.7.10 was used to mine character-level rules. Mining association rules is to discover all the strong rules in the database using different measures of interestingness. To select interesting rules from the total set of possible rules, we use some user controlled measures like minimum support (*minsup*) and minimum confidence (*minconf*). They are formally defined as follows.
Support(X→Y) = P(X ∪ Y)
represents the percentage of instances from the database that contains both X and Y.
Confidence(X→Y) = P(Y|X)
represents the probability that the instance containing X also contains Y. In our case, the value of *minconf* and the the lower bound *minsup* was set to 0.0001, and the upper bound of *minsup* was set to 1. The upper bound of *minsup* was iteratively decreased by a factor of *delta* until either the lower bound of *minsup* was reached or the required number of rules were generated.
The number of features and instances are 6631 and 3351 respectively in the character-level data set.
Example rules mined from the character level data

- old=b ==> new=h
- old=J ==> new=j
- old=K ==> new=R
- old=0 ==> new=6
- old=G ==> new=6

The above examples show that the old character is corrected to a new character by the patrons. As can be seen, the transformations done by the

OCR device were very close to the original word, for example, character 'h' can be easily misinterpreted as character 'b'.

- **Label Constructor:** The errors rectified by the users are categorized as addition, deletion, punctuation error, capitalization error, and spellcheck error. Specifically, (a) **Addition:** When the length of new string exceeds the length of old given the difference is made by alphanumeric characters and the edit distance is above one. For example, "6RAVR" and "GRAVEL". (b) **Deletion:** When the length of old string exceeds the length of new given the difference is made by alphanumeric characters and the edit distance is above one. For example, "VVe" and "We". (c) **Punctuation:** When the difference in the length of strings is non-zero and they differ by special characters contained in the set (!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~). For example, "Ladies!" and "Ladies". (d) **Capitalization:** When both the strings have equal length, edit distance is exactly 1 and first letter of both the strings change from upper to lower case or vice versa. For example, "largest" and "Largest". (e) **Spellcheck:** When the difference between string is above zero and the edit distance contributed by alphanumeric character is exactly one or when the strings have same length and the edit distance is one or above one irrespective of the involvement of special characters. For example, "the" and "them" or "hanger" and "banger".

  The distribution of the these classes in the dataset is shown in Table 2. It must be noted that by design tokens are always assigned to one class, although in principle it may be possible to assign them to multiple classes[3].

Class	No. of Instances
Addition	4575
Deletion	5024
Punctuation	6401
Capitalization	299
Spellcheck	4809
Total	21108

Table 2: Class Distribution

4. **Model Construction:** The model for classifying crowdsourced text correction is built using a Multiclass Support Vector Machine algorithm [26]. Each training point belongs to one of k different classes. The goal is to construct a function, which given a new data point, will correctly predict the class to which it belongs. Different methods have been proposed in literature for solving the SVM multi-class classification problem. Some popular techniques include: a) *One-Versus-All (OVA) classification* Build k different binary classifiers; for the i^{th} classifier, let the positive examples be all points in class i and negative examples not in class i. Let $f(x) = \arg\, max_i f_i(x)$. b) *one-versus-one* uses the majority voting strategy where each classifier assigns the new instance one of two classes. The class with the majority votes is assigned to the instance. Crammer and Singer [7] pose the multi-class classification problem as a single optimization problem, rather than decomposing it into multiple binary classification problems. A comparison of the above approaches can be found here [15].

   For a training set $(x_1, y_1)...(x_n, y_n)$ with labels $y_i \in \{1, \cdots, k\}$, the multi class problem can be posed as a constrained optimization problem with a quadratic objective function: $\min \frac{1}{2}\sum_{i=1}^{k} \parallel w_i \parallel^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i$

   s.t. $\forall y \leq k : [x_1 \cdot w_{yi}] \geq [x_1 \cdot w_y] + 100 * \Delta(y_1, y) - \xi_1 \cdots$
   s.t. $\forall y \leq k : [x_n \cdot w_{yn}] \geq [x_n \cdot w_y] + 100 * \Delta(y_n, y) - \xi_n$

   Here C is the regularization parameter, $\Delta(y_j, y), 1 \leq j \leq n$ is the loss function that returns 0 if y_j equals y, and 1 otherwise and $\xi_i, 1 \leq i \leq n$ are the non negative slack variables which measure the degree of misclassification of the instance x_i. It must be noted that when the data is not linearly separable, the following kernel functions are used for classification: a) Linear Kernel: $K(x, y) = x^T y + c$ b) Polynomial Kernel : $K(x, y) = (\alpha x^T y + c)^d$ c) Radial Basis Kernel : $K(x, y) = exp(-||x - y||^2/2\sigma^2)$

5. **Information Retrieval Techniques:** The tokens identified by the model as "spell check","addition", "capitalization", and "punctuation" play an important role in trying to enhance search and retrieval on the archive. This is presented here for the completeness of the architecture, but described in detail in Section 4.

3. EMPIRICAL EVALUATION

The goal of our experiments is to understand whether character level features or word level features are better predictors of the class of text correction. The experiments are performed on a linux HPC cluster with 20 CPU(s) and 10 cores per socket, 99GB RAM and 2.50GHz of processor speed. In these experiments, the regularization parameter C is varied from 0.0001 to 10000 for the linear kernel. The performance of the algorithm is evaluated using 10-fold cross validation technique. The **A**verage (loss) **E**rror (AE) and **A**verage (running) **T**ime (AT) from cross validation for baseline (word-level) and proposed (character-level) dataset respectively are represented as AE_b, AE_p. Table 3 shows the result of the experiment, error and running time (cpu seconds) using linear kernel for different values of C. Table 4 shows the result of the experiment, error and running time (cpu minutes) using polynomial and radial basis kernel. As it can be seen from the results, the word level features perform better than the character level.
The multi-class SVM algorithm using polynomial kernel gives the best results in both the datasets, word-level and character-level. In case of automatically generated character-level dataset, the performance of the algorithm does not improve by varying the regularization constant in contrast to the performance of the manually crafted word-level dataset. In case of word-level dataset, as the value of regularization constant increases, the average loss of the learning model decreases. However, the time taken to train the model is considerably

[3]For e.g. a correction of "t$e" to "the" could be either a Spellcheck or a Punctuation Error correction but we assign it to Spellcheck

C	AE_p	AE_b	AT_p	AT_b
.0001	99.43±.09	49.47±0.25	1.268±.06	0.11±.01
.1	99.43±.09	49.464±.47	2.512±0.01	0.061±0.01
10	99.43±.09	4.974±.5	2.512±0.01	0.17
1000	99.43±.09	1.743±.14	3.635±0.08	0.382±0.04
10000	99.43±.09	0	6.126±0.02	0.303±0.02

Table 3: Results using linear kernel

	Polynomial		RBF	
C	AE_p	AE_b	AE_p	AE_b
.0001	42±.13	50±.47	63±.15	27±.5
100	42±.13	.33±.2	63±.16	3±.4
1000	42±.13	0±0	63±.16	1.7±.2
C	AT_p	AT_b	AT_p	AT_b
.0001	37±1.4	10±0.2	25±0.7	6±0.2
100	326±11.8	1239±118	540 ± 110	404±34
1000	942±17.4	764±93	537±111.72	957±184

Table 4: Results using polynomial and rbf kernels

high. It must also be noted that one can learn less accurate but faster models with linear or RBF kernels. Thus, it may be useful to consider a trade-off between accuracy of learnt models versus time taken for classification in large scale deployments.

The code used to build the prototype system along with data generated are available at the github account[4]. It must be noted that the $SVM^{multiclass}$V2.2 package [16] was used for the implementation of Multi-class SVMs. For linear kernels, $SVM^{multiclass}$V2.20 is very fast and runtime scales linearly with the number of training examples. Training of non-linear kernels is very slow using the algorithm described in Section 2.

4. INFORMATION RETRIEVAL TECHNIQUES

The following details are relevant:

Query Set: To measure the retrieval effectiveness of the corrected text versus the garbled OCR, a list of 20 keywords for search was prepared by randomly sampling from the corrected words. The table 5 (See Appendix) presents the list of 20 keywords that were used to analyse the retrieval effectiveness of raw OCR versus corrected OCR text.

Software: PyLucene 3.6.2, a Python extension for accessing Java Apache Lucene was used as an IR software library for enabling full text indexing and search capabilites. Inverted indices are built for both the original and corrected corpus. Keywords from the query set are tested on both corpora and documents containing words in the query set are retrieved. The documents are ranked according to the frequency of the keyword present – higher the frequency, higher the rank.

Evaluation: The main aim of this system is to improve the quality of text for use with an IR system. The metric used for evaluation here is mean reciprocal rank (MRR) [20] which produces a list of possible ranked responses to a sample of queries, Q. The reciprocal rank is the multiplicative inverse of the rank of the first correct answer. The first correct answer is decided by the users. The average of the reciprocal ranks over a sample of queries is the mean reciprocal rank. MRR = $\frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$

The same set of queries were applied on both the datasets, that is raw OCR and corrected OCR individually. For each of the of 20 queries, we chose the most relevant document from the retrieved results and observed the rank of the most relevant document in both the set of results. The MRR for the raw OCR data was noted to be 0.355 whereas for corrected data it was 0.65415. The higher value of MRR denotes that relevant results get higher priority in corrected corpus. The metric MRR was chosen because it did not require relevance feedback from the users as in precision/recall [20].

Deployment: The full-scale deployment of this prototype system is being considered on old historic newspaper archives at California Digital Newspaper Collection and the New York Public Library among others, where user text corrections are used extensively for cleaning garbled OCR.

[4]https://github.com/megha89/

5. RELATED WORK

Optical Character Recognition (OCR) is a commonly used method of digitizing printed texts so that it can be searched and displayed online, stored compactly and used in text mining applications. The text generated from OCR devices is often garbled due to variations in quality of the input paper, size and style of the font and column layout, its condition at the time of microfilming, choice of scanner, and the quality of the OCR software. Several techniques for post processing garbled OCR have been designed [12], [10], [19]. These include:

Dictionary based schemes: These algorithms use a dictionary to spellcheck misspelled OCR recognized words. They correct non-word errors - words that are recognized by the OCR device but do not correspond to any entry in the lexicon. Niwa et al. [21] proposed an OCR post error correction method based on pattern learning where a list of suitable candidates is generated from the lexicon and the best candidate is selected as a correction. Yannakoudakis and Fawthrop [34] conducted a study to create a set of rules based on common misspelling pattern and used them to correct errors. Cherkassky and Vassilas [6] use back propagation algorithms for correction.

Context based schemes: These algorithms perform error detection and correction based on the grammatical error and semantic context. They are able to correct real-word errors – words that are recognized by the OCR system and correspond to an entry in the lexicon. Tong and Evans [25] describe an automatic, context-sensitive, word-error correction system based on Statistical Language Modeling (SLM). The system exploits information from letter n-grams, character confusion probabilities and word bi-gram probabilities. Golding et al. [13] applies a part-of-speech (POS) tagger enhanced by word trigram model and a statistical Bayesian classifier to correct real-word errors in OCR text. Reynaert [22] presents a system for reducing the level of OCR-induced typographical variation in large text collections called Text-Induced Corpus Clean-up (TICCL). The system focuses on high-frequency words to be cleaned and gathers all typographical variants for any particular focus word that lie within the predefined Levenshtein distance.

Bassil et al. [4, 5] propose a post-processing context-based error correction algorithm for detecting and correcting OCR non-word and real-word errors. The proposed algorithm is based on Google's online spelling suggestion. Abdulkader et al. [1] present a method for digitizing textual data by using neural network classifiers to estimate OCR errors, clustering similar errors, designing a user interface and using active learning to tune the error estimation from user labeled data. Velagapudi [27] uses a combination of classifiers – kNN classifier, multilayer perceptrons and SVM to discuss the effects of error correction on the classification accuracy of each method.

Very little work has been done on automatic *classification* of OCR error corrections[5] to get a clear understanding of the *nature* of errors encountered – Esakov et al. [9] suggest classification of output from the OCR engine as simple substitutions ($e \rightarrow c$), improper segmentation or multiple substitutions ($T \rightarrow' l, m \rightarrow rn, he \rightarrow b$), deletions and insertions (involving space) and unrecognized characters ($u \rightarrow \sim$). They use a new variant of the dynamic programming algorithm for classification. In [17], OCR documents are classified into fixed number of categories based on their content. The accuracy of their approach was best when evaluated using SVM among other algorithms. Daoason [8] posits that OCR errors are not random – for example, it is extremely unlikely that the letter o will be misrecognized as x since they are very dissimilar. He classifies OCR errors as character errors (characters in the input document that are replaced with other characters), word errors (word in the input document is not correct or not present in the OCR generated text), or zoning errors (OCR software is unable to decipher zones correctly). Our work primarily focusses on *manual* error correction classification where the task of correcting OCR errors is outsourced to a crowd of workers.

Crowdsourced OCR correction Recruiting users and paying them small sums of money to tag and annotate text and images in large digital archives has become common practice (e.g. Amazon's Mechanical Turk, reCAPTCHA [31], ESP [29]) and Games with a Purpose [28]. A number of recent papers have evaluated the effectiveness of using Mechanical Turk to create annotated data for text and natural language processing applications [24]. In the same vein, many workshops and conferences have been organized on the theme of machine learning in human computation, crowdsourcing and collective intelligence[6].

Intuitively, the easiest way to correct garbled OCR text is to hire a group of people to edit it manually. Distributed Proofreaders (DP) [11] is a web-based project designed to facilitate proofreading of paper books into e-books and was meant to assist the project Gutenberg[7]. Wikipedia is yet another example of a large scale crowdsourcing project. Yamangil et al. [33] proposed a learning algorithm for mining wikipedia edit history using baseline Hidden Markov Model augmented with perceptron re-ranking. The model was trained on wikipedia edits and hence incorporated human corrections.

6. CONCLUSION & FUTURE WORK

The California Digital Newspaper Collection has an archive of 400,000 pages of historical California newspapers published between 1846 to 1922. This archive which has been subjected to OCR is currently stored in an online database accessible to the patrons. The OCR scanning process generates lot of garbled text which needs to be corrected to make the online newspaper repository more accessible to general public.

In this paper, we present a system for Classification of Crowdsourced Text Correction which is capable of modelling user corrections using state-of-the-art machine learning techniques and retrieve categories which are likely to enhance search on the archive such as addition of words, elimination of typographical errors or addition of content. We also compared the two datasets, word-level and character-level by applying the machine learning algorithms on each. The manually crafted word-level dataset performed better in terms of average loss error. However, the running time of the algorithm was considerably high. Thus, there is a need to improve the algorithms in order to make them compatible with large scale datasets.

Information retrieval metric, mean reciprocal rank is used to quantify the effectiveness of the user text correction. The higher value of the MRR in the case of corrected corpus indicates that the relevant results are ranked higher in the retrieved set of documents. The prototype system is being considered for deployment on historic newspaper archives at the California Digital Newspaper Collection and the New York Public Library.

Acknowledgment

This work is supported by funding from the National Endowment for Humanities, Grant No: NEH HD-51153-10. The authors would like to thank Stefan Boddie, DL Consulting, Ltd., New Zealand for sharing experiences with the Veridian software and Luis. C. Baquera, University of California, Riverside for providing data generated from the log files at CDNC.

7. REFERENCES

[1] A. Abdulkader and M. R. Casey. Low cost correction of ocr errors using learning in a multi-engine environment. In *Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*, ICDAR '09, pages 576–580, 2009.

[2] ADNP. Australian newspapers digitization program, 2008.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *20th International Conference on Very Large Data Bases*, pages 478–499. Morgan Kaufmann, Los Altos, CA, 1994.

[4] Bassil and Alwani. Ocr post-processing error correction algorithm using google online spelling suggestion. *CoRR*, abs/1204.0191, 2012.

[5] Y. Bassil and M. Alwani. Ocr context-sensitive error correction based on google web 1t 5-gram data set. *CoRR*, abs/1204.0188, 2012.

[6] V. Cherkassky and N. Vassilas. Performance of back propagation networks for associative database

[5]Most prior work has dealt with evaluation of candidate corrections for OCR errors.

[6]See http://ir.ischool.utexas.edu/crowd/

[7]http://www.gutenberg.org/

retrieval. *In Proc. of the International Joint Conference on Neural Networks*, 1:77–84, 1989.

[7] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2:265–292, Mar. 2002.

[8] J. F. Daoason. *Post-correction of icelandic OCR text.* Faculty of Industrial Engineering, Mechanical Engineering and Computer Science, University of Iceland, 2012.

[9] J. Esakov, D. P. Lopresti, and J. S. Sandberg. Classification and distribution of optical character recognition errors. *SPIE, Document Recognition*, 2181:204–216, 1994.

[10] J. Esakov, D. P. Lopresti, J. S. Sandberg, and J. Zhou. Issues in automatic ocr error classification. *Proc. Third Annual Symposium on Document Analysis and Information Retrieval*, 1994.

[11] C. Franks. Distributed proofreaders, 2000.

[12] H. Fujisawa and C. L. Liu. Classification and learning for character recognition: Comparison of methods and remaining problems in neural networks and learning in document analysis and recognition. *Machine Learning in Document Analysis and Recognition, Studies in Computational Intelligence*, 90:139–161, 2008.

[13] A. R. Golding and Y. Schabes. Combining trigram-based and feature-based methods for context-sensitive spelling correction. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pages 71–78, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.

[14] R. Holley. Crowdsourcing and social engagement: potential, power and freedom for libraries and users., november 2009.

[15] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *Trans. Neur. Netw.*, 13(2):415–425, Mar. 2002.

[16] T. Joachims. Multi-class support vector machine, Aug. 2008.

[17] S. Laroum1, N. BÈchet2, H. Hamza3, and M. Roche. Hybred: An ocr document representation for classification task. *IJCSI International Journal of Computer Science Issues*, 8, 2011.

[18] LDS. Family search, 2005.

[19] Y. Lecun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. *NIPS*, 1989.

[20] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, Cambridge, UK, 2008.

[21] H. Niwa, K. Kayashima, and Y. Shimeki. Postprocessing for character recognition using keyword information. In *IAPR Workshop on Machine Vision Applications*, pages 519–522, Dec 1992.

[22] M. Reynaert. Non-interactive OCR post-correction for giga-scale digitization projects. In *Computational Linguistics and Intelligent Text Processing*, volume 4919, pages 617–630, 2008.

[23] M. Ridge. Frequently asked questions about crowdsourcing in cultural heritage, June 2011.

[24] R. Snow, B. O'Connor, D. Jurafsky, and A. Y. Ng. Cheap and fast—but is it good: evaluating non-expert annotations for natural language tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 254–263, 2008.

[25] X. Tong and A. D. Evans. A statistical approach to automatic ocr error correction in context. In *Proceedings of the Fourth Workshop on Very Large Corpora (WVLC-4*, page 13, 1996.

[26] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 104–, 2004.

[27] P. Velagapudi. Using hmms to boost accuracy in optical character recognition, 2005.

[28] L. von Ahn. Designing games with a purpose. *Communications of the ACM*, 51, 2008.

[29] L. von Ahn and L. Dabbish. Labeling images with a computer game. *In* Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04*, pages 319–326, 2004.

[30] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. reCAPTCHA: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[31] L. von Ahn, B. Maurer, C. Mcmillen, D. Abraham, and M. Blum. reCAPTCHA: Human-based character recognition via web security Measures. Science, 321:5895–, 2008.

[32] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, Jan. 1974.

[33] E. Yamangil and R. Nelken. Scalable lexical correction from wikipedia edits using perceptron reranking. *unpublished*, 2008.

[34] E. J. Yannakoudakis and D. Fawthrop. The rules of spelling errors. *Information Processing and Management*, 19:87–99, 1983.

APPENDIX

S.no	Keywords
1	prices
2	trade
3	character
4	Development
5	Aldrich
7	Sutter
8	south
9	Sacramento
10	compelled
11	insertion
12	estate
13	Pacific
14	missed
15	cough
16	kidneys
17	CALAVERAS
18	sheep
19	aces
20	controlled

Table 5: List of keywords containing upper and lower case words, randomly sampled from the corrected corpus are used for testing the information retrieval techniques.

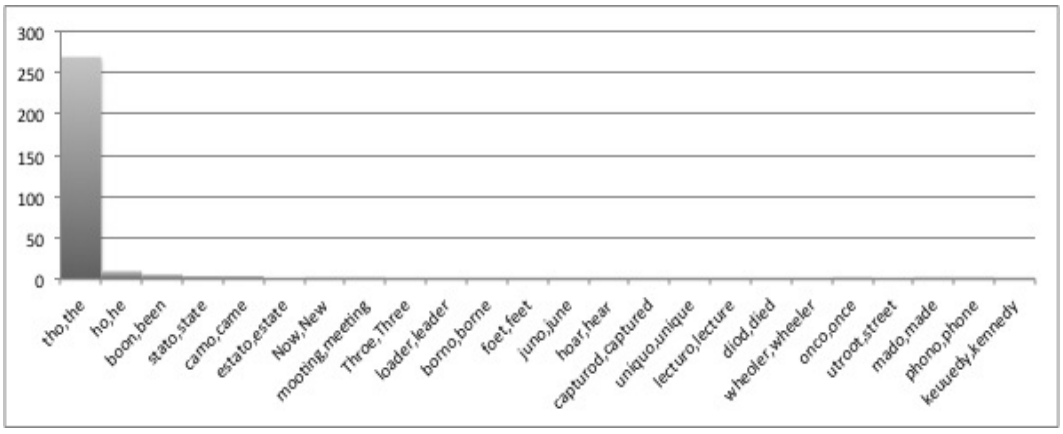

Figure 3: Examples of o → e corrections, (tho,the; ho,he; boon,been; stato,state; camo,came; estato,estate; Now,New; mooting,meeting; throe,three; loader,leader; borno,borne, foet,feet; juno,june; hoar,hear; capturod,captured; uniquo,unique; lecturo,lecture; diod,died; wheoler,wheeler; onco,once; utroot,street; mado,made; phono,phone)

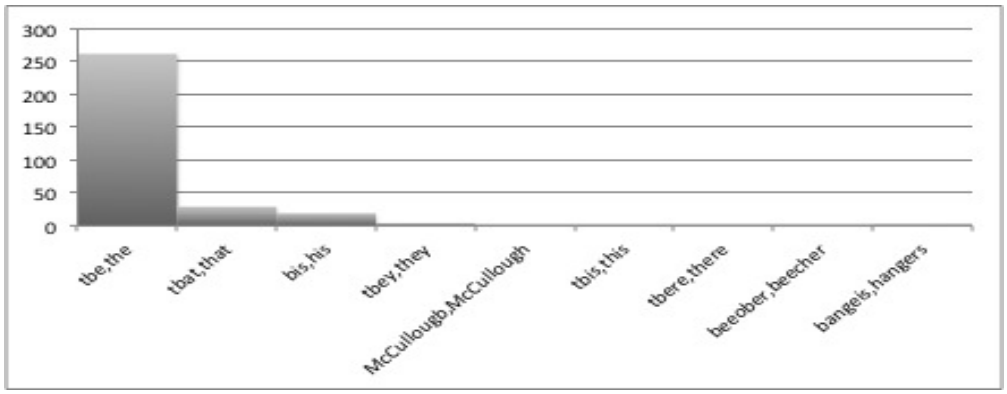

Figure 4: Examples of b → h corrections, (tbe,the; tbat,that; bis,his; tbey,they; McCullougb,McCullough; tbis,this; tbere,there; beeober,beecher; banger,hanger)

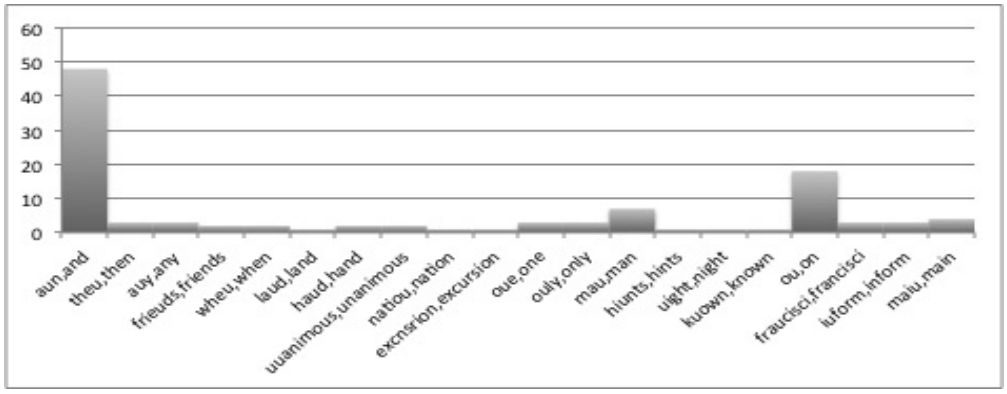

Figure 5: Examples of u → n corrections, (aud,and; theu,then; auy,any; frieuds,friends; wheu,when; laud,land; haud,hand; uunanimous,unanimous; natiou,nation; oue,one; ouly,only; mau,man; hiuts,hints; uight,night; kuown,known; ou,on; fraucisci,francisci; iuform,inform; maiu,main)

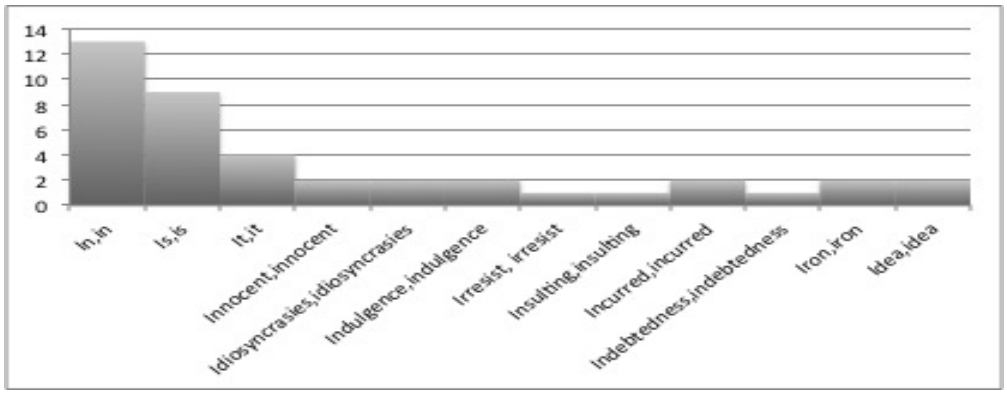

Figure 6: Examples of I → i corrections, (In,in; Is,is; It,it; Innocent,innocent; Idiosyncrasies,idiosyncrasies; Indulgence,indulgence; Irresist, irresist; Insulting,insulting, Incurred,incurred; Indebtedness,indebtedness; Iron,iron; Idea,idea)

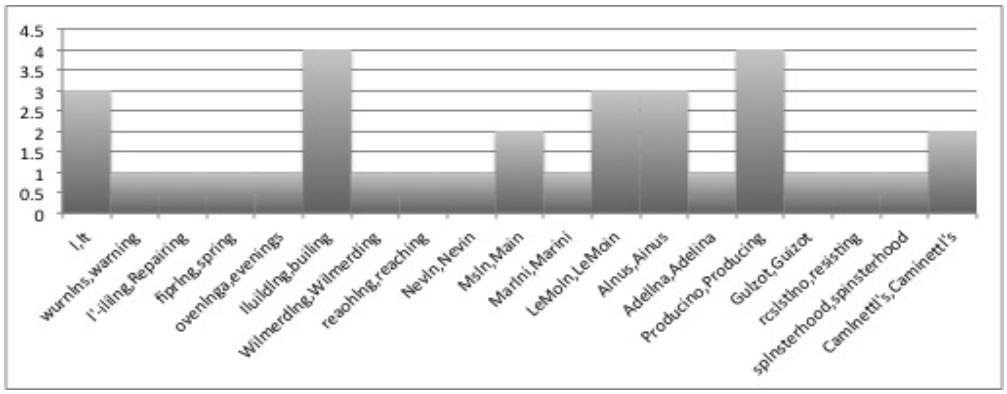

Figure 7: Examples of l → i corrections, (l,It; wurnlns,warning; l'-ililng,Repairing; fiprlng,spring; ovenlnga,evenings; Iluildlng,builing; Wilmerdlng,Wilmerding; reaohlng,reaching; Nevln,Nevin; Msln,Main; Marlni,Marini; LeMoln,LeMoin; Alnus,Ainus; Adellna,Adelina; Gulzot,Guizot; rcslstino,resisting; splnsterhood,spinsterhood; Camlnettl,Caminetti)