Classification of Crowdsourced Text Correction

Haimonti Dutta*
Center for Computational Learning Systems,
Columbia University,
New York, NY 10115
haimonti@ccls.columbia.edu

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

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 Spearman's rank correlation metric. 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 (ADNP 2008) 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 (Holley 2009). FamilySearch (LDS 2005) 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 (Ridge 2011). 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. To the best of our knowledge, this is the first attempt to statistically analyze and model 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 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.

*The author is also a visiting assistant professor at IIIT-Delhi.

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

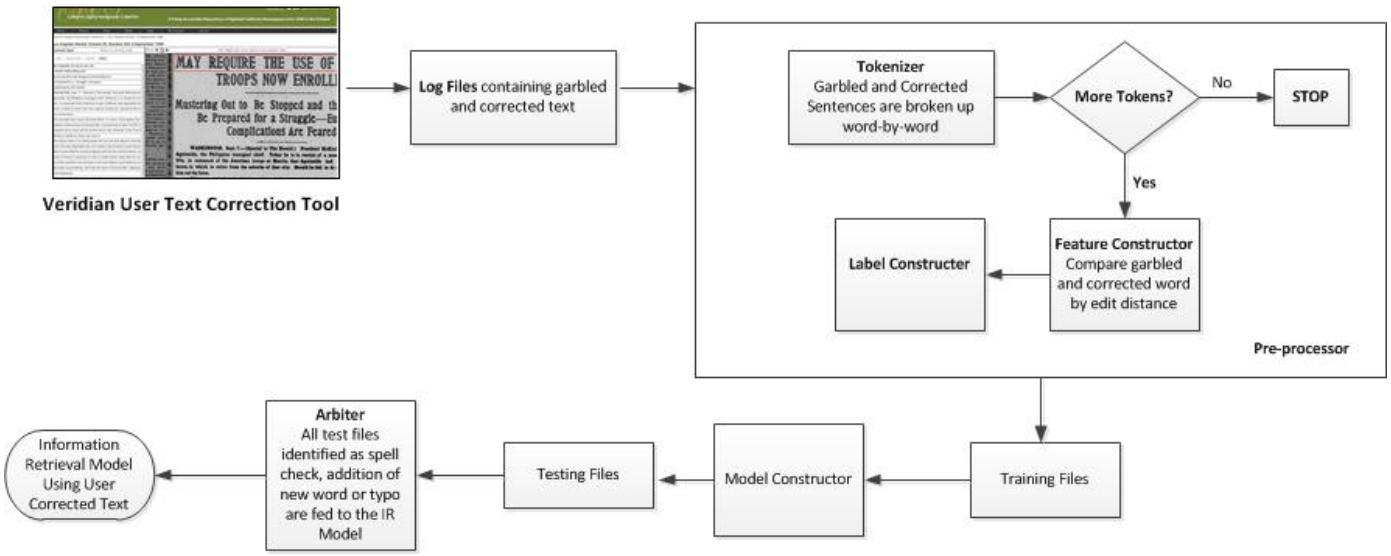


Figure 1: The Architecture of the Proposed System

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 3(a) 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 3(b).

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.

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 20,137 are corrected by the annotators.

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

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

Table 1: A segment of the log file

   - **Feature Constructor:** Features are crafted by computing the Levenshtein edit distance between the old word and its correction. The Levenshtein edit distance (Wagner and Fischer 1974) is defined 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) **Same Length** : 1, if both the old word and new word have same length and 0 otherwise. (b) **Edit Distance Zero**: 1, if both the words are exactly same and 0 otherwise. (c) **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. (d) **Edit Distance Below Three**: 1, if less than three edit operations are required to convert old word to new word and 0 otherwise. (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

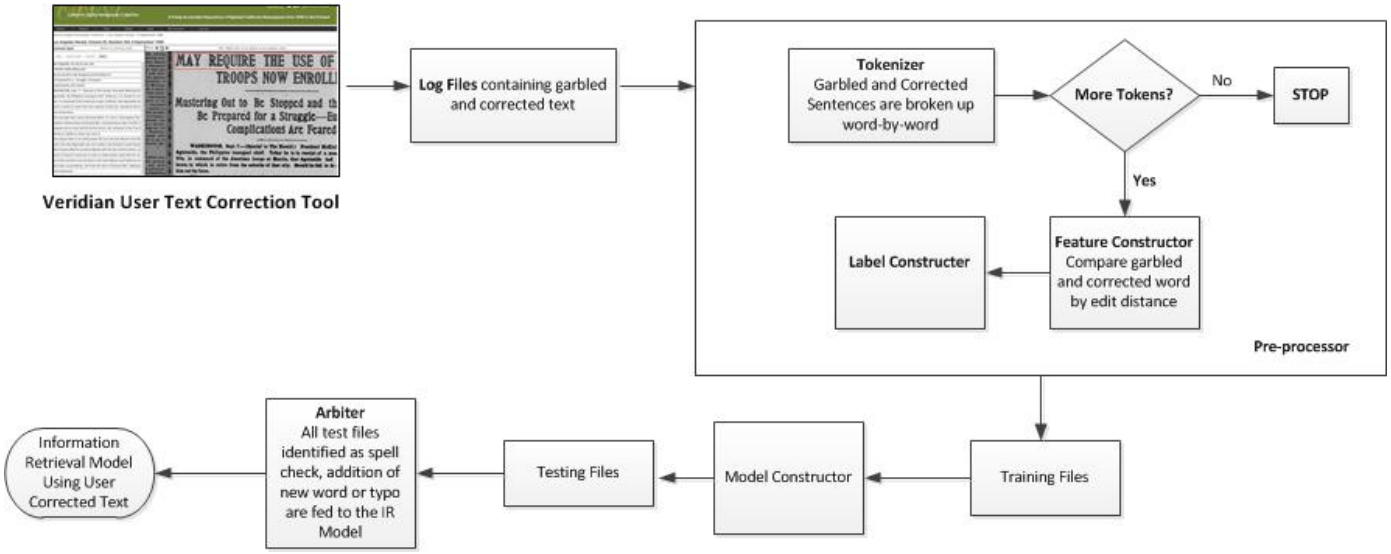
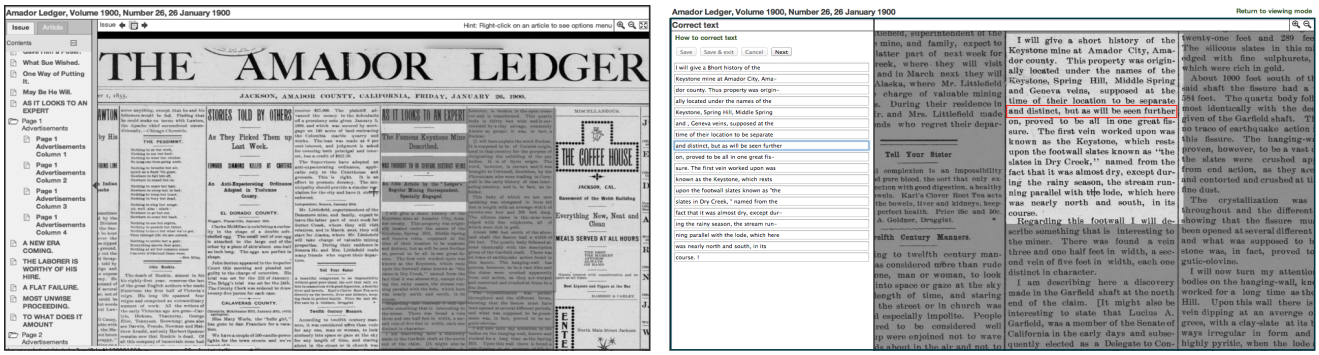


Figure 2: The Architecture of the Proposed System



(a) Scanned newspaper highlighting an article to be corrected by a user.

(b) The tool used by patrons to annotate articles.

Figure 3: The Amador Ledger, Jan. 26, 1900.

(!"#$%&'*+,-./:;<=>?@[\]^_‘{|}~).

- **Label Constructor:** The errors rectified by the users are categorized as Spellcheck Error, Addition of a new word, Capitalization Error, Typo and Punctuation error. Specifically, (a)**Spellcheck Error:** When the edit distance is between 1 and 3. For example, “mounten” and “mountain”. (b)**Addition of a new word:** When the edit distance is more than 3. For example, “at” and “attend”. (c)**Capitalization error:** When the edit distance of two strings is exactly 1 and first letter of both the strings changes from upper to lower case or vice versa. For example, “largest” and “Largest”. (d)**Typographical Error:** When the edit distance is exactly one and case change is 0. For example, “teh” and “the” (e)**Punctuation Error:** When the two strings differ by special characters contained in the set (!"#$%&'()*+,-./:;<=>?@[\]^_‘{|}~). (f)**No Correction:** When the old and new text are same.

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
Spellcheck	1970
Addition of new word	8732
Capitalization	261
Typo	2572
Punctuation	6602
No correction	23885
Total	44022

Table 2: Class Distribution

[3]For e.g. a correction of “t$e” to “the” could be either a Typo or a Punctuation Error correction.

4. **Model Construction:** The model for classifying crowdsourced text correction is built using a Multiclass Support Vector Machine algorithm (Tsochantaridis et al. 2004). 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 (Crammer and Singer 2002) 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 (Hsu and Lin 2002).

   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 of a new word", and "Typo" 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

Experiments are performed by randomly selecting 70% of the data[4] as the train set. For each experiment, the regularization parameter C and the type of kernel t are varied as $0.001, 0.01, 0.1, 1, 10, 100$ and Linear, Polynomial and Radial Basis Function kernel respectively. The experiments are performed on two machines, one of which is a linux server and the other a dual core Mac machine with Intel Core i7 processor, 8GB RAM, 2.9 GHz of processor speed. Each experiment is repeated 5 times and the average loss on the test set and CPU runtime are reported. Table 3 presents the average loss on test set for different values of C and t; AE_L, AE_P, AE_R represents the average loss on the linear, polynomial and RBF kernels respectively. Table 4 shows the average runtime (cpu sec) required to learn a model for various values of C and for different kernels. Here, AT_L, AT_P, AT_R refers to the average run time of linear, polynomial and rbf kernels respectively. The loss is least when the $C = 100$ for the polynomial kernel. However, the time taken to train the model is considerably higher at 106753s. It must also be noted that one can learn less accurate models with linear or RBF kernels faster. 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 from `https://github.com/megha89/`. It must be noted that the $SVM^{multiclass}$V2.2 package (Joachims 2008) 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.

C	AE_L	AE_P	AE_R
.001	32.06 ±0.149	32.06±0.149	11.96±0.101
.01	32.06±0.149	32.06±0.149	12.04±0.199
.1	32.06±0.149	29.10±0.133	12.04±0.199
1	29.10±0.133	28.99±0.264	6.36±0.142
10	29.20±0.133	8.338±0.107	6.36±0.142
100	29.20±0.133	0.58±.034	6.35±0.16

Table 3: Experiment Results : Average loss error (test set)

C	AT_L	AT_P	AT_R
.001	0.218	5192	4596
.01	0.202	5166	4460
.1	0.208	72225	4377
1	0.678	81251	16468
10	0.672	260316	55308
100	0.998	106753	48734

Table 4: Experiment Results : Training time (cpu sec)

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 60 keywords for search was prepared by randomly sampling from the corrected words.
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.

[4] Approximately 30,815 instances

Evaluation: The main aim of this system is to improve the quality of text for use with an IR system. The metric used for our evaluation include the standard IR techniques of recall and precision. Recall is defined as the ratio of number of relevant documents returned to the total number of relevant documents for a query, in the collection. Precision is defined as the ratio of number of relevant documents returned to the total number of documents returned for a given query. The document to query relevance is manually determined by other computer science students. The table shows the precision at ten standard recall points for corrected OCR text over the raw OCR text.

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.

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 (Fujisawa and Liu 2008), (Esakov et al. 1994), (Lecun et al. 1989). 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. (Niwa, Kayashima, and Shimeki 1992) 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 (Yannakoudakis and Fawthrop 1983) conducted a study to create a set of rules based on common misspelling pattern and used them to correct errors. Cherkassky and Vassilas (Cherkassky and Vassilas 1989) 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 (Tong and Evans 1996) 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. (Golding and Schabes 1996) 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 (Reynaert 2008) 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. (Bassil and Alwani 2012b; 2012a) 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. (Abdulkader and Casey 2009) 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 (Velagapudi 2005) 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. (Esakov, Lopresti, and Sandberg 1994) 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 (Laroum1 et al. 2011), 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 (Daoason 2012) 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 (von Ahn et al. 2008), ESP (von Ahn and Dabbish. 2004)) and Games with a Purpose (von Ahn 2008). A number of recent papers have evaluated the effectiveness of using Mechanical Turk to create annotated data for text and natural language processing applications (Snow et al. 2008). 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) (Franks 2000) is a web-based project de-

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

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

signed 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. (Yamangil and Nelken 2008) 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 and is currently stored in an online database making them accessible to 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 modeling 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. Information retrieval metrics are used to quantify the effectiveness of the user text correction. 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.

References

Abdulkader, A., and Casey, M. R. 2009. 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, 576–580.

ADNP. 2008. Australian newspapers digitization program.

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

Bassil, Y., and Alwani, M. 2012b. Ocr post-processing error correction algorithm using google online spelling suggestion. *CoRR* abs/1204.0191.

Cherkassky, V., and Vassilas, N. 1989. Performance of Back Propagation Networks for Associative Database Retrieval. *In Proc. of the International Joint Conference on Neural Networks* 1:77–84.

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

Daoason, J. F. 2012. *Post-Correction of Icelandic OCR Text*. Faculty of Industrial Engineering, Mechanical Engineering and Computer Science, University of Iceland.

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

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

Franks, C. 2000. Distributed proofreaders.

Fujisawa, H., and Liu, C. L. 2008. 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.

Golding, A. R., and Schabes, Y. 1996. 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, 71–78. Stroudsburg, PA, USA: Association for Computational Linguistics.

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

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

Joachims, T. 2008. Multi-class support vector machine.

Laroum1, S.; Bchet2, N.; Hamza3, H.; and Roche, M. 2011. Hybred: An ocr document representation for classification task. *IJCSI International Journal of Computer Science Issues* 8.

LDS. 2005. Family search.

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

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

Reynaert, M. 2008. Non-interactive OCR post-correction for giga-scale digitization projects. In *Computational Linguistics and Intelligent Text Processing*, volume 4919. chapter 53, 617–630.

Ridge, M. 2011. Frequently asked questions about crowdsourcing in cultural heritage.

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

Snow, R.; O'Connor, B.; Jurafsky, D.; and Ng, A. Y. 2008. 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, 254–263.

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

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

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

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

von Ahn, L.; Maurer, B.; Mcmillen, C.; Abraham, D.; and Blum., M. 2008. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. Science 321:5895–.

von Ahn, L. 2008. Designing Games with a Purpose. *Communications of the ACM* 51.

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

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

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