Minimally Supervised Methods to Correct Optical Character Recognition

Andrew Stromme
Computer Science Dept
Swarthmore College
Swarthmore, PA 19081
astromm1@swarthmore.edu

Ryan Carlson
Computer Science Dept
Swarthmore Colege
Swarthmore, PA 19081
rcarlso1@swarthmore.edu

Abstract

Optical character recognition (OCR) is the transformation of an image of handwritten or typed text to raw text. It is used in a range of modern applications that can make a significant impact, so it is important to have robust OCR programs. There are a variety of such programs today, both propriety and open-source.

We have implemented a post-processing layer over OCR output from GOCR to reduce errors and resolve ambiguities. We first use an minimally supervised algorithm to identify pairs of characters that are often incorrectly identified with the other. We then identify words that the OCR program could not translate and use an *n-gram* model to fix the errors. Our testing reveals high precision and recall, showing we can successfully modify the OCR output.

1 Introduction

There is wide-reaching interest in OCR. The U.S. Post Office uses the technology to identify postal codes, and Google uses it to translate text from book scans into searchable text. OCR can even be used to catch image based spam. The average consumer can use OCR to index bills or letters, digitizing and making them available long after the ink fades. OCR typically acts on document scans, which are often subject to significant noise. Not all OCR programs are created equally, with different paradigms pervading over different programs. These approaches are discussed in Section 2.

We have chosen a relatively bare-bones OCR program called GOCR (Schulenburg, 2010) in order to best see the impact of our changes. GOCR reads and classifies one character at a time and allows us to set a confidence threshold for each character. Higher thresholds result in higher precision, because the system is more certain of its guesses. It also results in lower recall since there are more characters that have a lower confidence than the cutoff. GOCR also outputs an unknown marker for every unrecognized character.

Thus, we can raise the confidence threshold such that the characters GOCR does output are correct with very high probability. This allows us to post-process the output under the assumption that most characters are correct. We soon found out that this assumption is not always true, since even at very high confidence there exist characters that are often mistaken for others. To remedy this problem we created a simple supervised learning algorithm that identifies characters that are commonly confused. Whenever we see any of these characters, we can swap them and check if the swap results in a better choice.

We define an ambiguous word as one that contains either the unknown marker from GOCR or an ambiguous character as defined above. In our algorithm, each unknown marker is expanded to any character and we check which of the resulting words are valid. Given this set of possible valid words, we then use an n-gram model to choose the most likely

word from the choices. Using this model gives syntactic context to a model that has none otherwise. We further describe our process in Section 3.

While a quantitative evaluation (see Section 4) of our data is certainly useful, we also perform some qualitative analysis. We compare our processed OCR output against both the original text and the baseline unmodified OCR output. This gives a sense of improvement over GOCR. We examine common cases that our algorithm does not correct and other issues in Section 5. There are a number of interesting questions that have arisen from our results, which we discuss in Section 6.

2 Related Work

To motivate the use of OCR, we offer a small sampling of interesting applications for the technology in this section. We then describe some different approaches used and paradigms taken by OCR programs today.

2.1 Applications

In the early 1980's the U.S. Postal Service (USPS) started using OCR as part of a modular system to identify addresses and spray the appropriate POSTNET bar code onto the letter. This bar code was then read, sorted, and sent to the recipient. The system could process approximately 30,000 pieces of mail per hour. In contrast, manual identification had previously moved at between 500 and 1,000 pieces per hour. The technology has improved since then, and importantly has allowed the USPS to make large volumes of mail manageable (Srihari and Kuebert, 1997).

Another interesting project uses OCR to detect image-based spam. To bypass filters, distributors are disseminating spam with images of their product (e.g. Viagra) but otherwise no mention of the product. Ma et al. (2007) plug OCR into a popular spam filter called SpamAssassin. They pre-process the email and translate the image to text which the filter can then act on. Since the OCR system was still subject to some error, Markov models were used on the output to make the system resistant to misspellings.

OCR has also made Google co-founders Larry Page and Sergey Brin's dreams come true. The Google Books project has the goal of scanning and indexing the world's books and making them available online. To make these books and other documents searchable online, OCR has been employed to great effect. Having the raw text of scanned documents allows Google to use their search algorithms and make these books more accessible by returning relevant results to user queries (Vincent, 2007).

2.2 Program Approaches

We have so far discussed applications that use of build on OCR programs. But the underlying structure of the technology is not a given. We explore three programs in this section: Tesseract, OCRopus, and GOCR.

2.2.1 Tesseract

This project was initially developed at Hewlett-Packard and is currently in development by Google. Tesseract first analyzes connected components to generate *blobs*. To determine the type of spacing (fixed or proportional) the blobs are broken into text lines and analyzed. The process now becomes two-pass operation. Any words identified with high enough confidence are logged and passed to be learned by an adaptive classifier. The second pass gives the classifier a chance to use its newfound knowledge to classify the text again. The adaptive classifier uses isotropic baseline/x-height normalization which allows it to distinguish capital from lowercase words. The engine can correctly identify about 99% of characters, which translates to approximately 95% word accuracy (Smith, 2007).

2.2.2 OCRopus

OCRopus is notable because of its modular design. The system is composed of a set of plugins for layout analysis, text line recognition, and language modeling. Before all this, a preprocessing step readies the engine to analyze the image by adjusting for skew and cleaning the image. The layout analysis divides the image into non-text regions and text lines. Also, it is responsible for determining the reading order of the lines. Interestingly, the text line recognition process uses Tesseract. Finally, language models are applied to correct the output. These models could be anything from n-grams to stochastic grammars. One common representation for these models is a fi-

nite state transducer, which comes as a backend with the program (Breuel, 2008).

2.2.3 GOCR

GOCR is a much simpler OCR program than those we have discussed. It is simple and fast, does not require (or take) training data, and only works on non-rotated, black and white images. Some simple preprocessing uses threshold value detection, box detection, zoning, and line detection. An OCR engine is then called twice, the first for the entire document and the second for any unknown characters. This second pass offers more context than the first allowing more characters to get classified. This simple structure makes the system both very accessible and wide open for improvement. Our post-processing contributions to this system over others are more necessary and make a larger impact (Schulenburg, 2005).

3 Methods

We use a two-phase minimally supervised learning approach to fixing OCR. The first phase identifies a set of oft-confused characters. The second phase expands words with those confusable characters and unknown markers from GOCR to possible words and then evaluates the choices using an n-gram model to pick the best.

3.1 Learning Ambiguous Characters

Soon after we began our experiments, we realized there are a set of character pairs that often get confused. For example, the letter *o* and the number 0 are often mistaken for each other by the OCR engine. Instead of hard coding these pairs, we decided to learn them given training data.

We first create a document with all alphanumeric characters in various fonts. We convert that document to both an image and raw text. The image is fed into GOCR and the output is recorded. We then compare the output to the raw text transcription character by character and record any disparities. Since the image is essentially free of noise the OCR translation should perform close to its best, recording these as ambiguous characters is valid. We note that this method is not specific to English, but will adapt to any domain where the set of possible characters is known. As we scan through our test corpora, we are careful to be suspicious of words containing any of these characters.

3.2 Expanding Ambiguous Words

In Section 1 we defined an ambiguous word as one containing ambiguous characters or the unknown marker output by GOCR. Going forward, we will denote the unknown marker as '_'. For example, if the source word is *own* and OCR could not recognize the w and mistook the o for a zero, our output would be `0_n`. Given this (and not knowing that the correct word is *own*), we expand the output to a set of possible words.

First we toggle the 0 to be the letter *o*, since it is in our list of ambiguous characters. Next we expand the number of unknown markers to zero, one, and two markers such that we have the following set of possibilities:

`0n, 0_n, 0__n, on, o_n, o__n`

We allow any unknown marker to expand to one alphanumeric character[1] and then check the resulting string against a dictionary of valid English words. To get these words, we gather data from the Brown corpus, the Reuters corpus, and the Gutenberg corpus. We also compute counts of these words and select the top ten words with the highest counts to move on to the next stage. In our example, the above would expand to

`on, own, open, oxen, oven, omen`

Note that there are only six total options, but in general there can be thousands or hundreds of thousands of options. In this case we limit ourselves to the ten most likely options as defined by our smaller corpus. We can now use these options as part of an n-gram model to determine which option is most likely given the previous words in the test document.

3.3 N-gram model

Suppose we are given the following OCR output

`you can brew your o_n beer`

and we want to find which option `o_n` expands to. Looking at just the word counts alone, we might find that `on` is the most likely candidate. But given the

[1] we additionally allow expansion to single quotes to account for contractions.

context, own is clearly the best choice. We can use an n-gram model to disambiguate the words.

Given some unknown word w_i, we want to predict its likelihood given every word that came before it. Unfortunately, it is fairly trivial to come up with a valid sentence that has never before been uttered[2] and it is difficult to assign accurate probabilities to novel strings of words. Even given a corpus with vast coverage, trying to compute the probability of a word given *all* the previous is not possible.

We can use an *n-gram* model to make this process feasible. The model assumes that the probability of some word w_i given all the previous words w_1^{i-1} can be represented as the probability of that word given the previous $n-1$ words. Formally, we have

$$P\left(w_i \mid w_1^{i-1}\right) = P\left(w_i \mid w_{i-(n-1)}^{i-1}\right)$$

For example, in a bigram model we have $n = 2$ so $P\left(w_i \mid w_1^{i-1}\right) = P\left(w_i \mid w_{i-1}\right)$. This method proves to provide some syntactic context and allows our algorithm to make better decisions. Google has released their indexed n-grams counts for $n = 1, \ldots, 5$ (Franz and Brants, 2006) which contains over one trillion tokens with over thirteen million types. This serves as our n-gram corpus.

We have empirically determined that bigram counts with a backoff to unigram counts are most effective. Additionally, we use unigram counts if the preceding word is left unfixed in previous passes. As we traverse the test document, any time we have a known word followed by an unknown word (in our example, your o_n) we can generate the top ten bigrams and query our n-gram corpus to see which is the most likely. If none of the options appear as bigrams in Google's index, we drop back to unigram counts.

To see the effect of using bigrams before resorting to unigrams, consider our example word pair your o_n. Just using the unigram counts, the word on (which we get by collapsing the unknown indicator) was seen over 3.4 billion times wheres as the word own was only seen about 200 million times. However, looking at the bigrams your on and your own we see that the counts become 300 thousand against 63 million, an order of magnitude difference in the other direction. Simply selecting the higher unigram count is rarely sufficient, while the bigram count tends to provide enough context to select the correct word.

When we are given OCR output, we first search through the output and find any known words followed by an unknown one and compile a list. Any time that no expansions of unknown markers in a given word result in a bigram in our index, we drop to unigram counts. Thus, if none of the expansions of your o_n resulted in a valid bigram, we drop to counting the unigram counts for o_n and we would select on. This means that a chain of unknowns takes a long time to compute. If our test corpus had instead been

you _an brew you_ o_n beer

we process you _an and brew you_ simultaneously, deciding the corrections are you can and brew your, respectively. Then we make another pass on your o_n to choose your own. Thus, our runtime is determined primarily by the longest string of *consecutive* ambiguous words since the algorithm is primarily I/O bound. Scanning through large files that contain the n-grams has a high overhead, and as we execute more passes we are querying fewer and fewer n-grams from each file, which results in diminished returns.

4 Results

We tested our algorithm on seven documents, examining various statistics about the results. In this section we discuss our testing methods and present quantitative results.

4.1 Corpora

As described in Section 3.2, the dictionary we use to pick the top ten expansions of unknown markers is based on the Brown, Reuters, and Gutenberg corpora. This combined corpus contains five million tokens and over one hundred thousand types. We have experimentally determined that most words in our test documents are found in this initial corpus. This is important, since this first pass word extraction is then sent to our Google corpus to find the best n-grams, and if significant words were missing from our initial corpus they would never make it to the next phase.

[2] "cry like an algebraic field" may be such an example.

We introduced Google's n-gram corpus, which contains over one trillion tokens and thirteen million types, in Section 3.3. Acting as our second phase, this corpus represents excellent coverage of the English language. It also represents how people write today, since it the data is culled from the Internet at large. This is important when analyzing recent documents because it may contain many phrases common today that are not accounted for in other corpora.

To test our ability to correctly fix OCR output, we used the first page of seven distinct essays from an introductory Cognitive Science class. We had access to the original PDF's which we converted to image files, which means our tests were as free from noise as possible. Additionally, the essays were written using Latex, which creates single glyphs from two distinct letters when adjacent (fi, for example). Finally, the content is that of an academic paper – it is prose that contains mostly common words, with some occasional technical jargon thrown in. It also contains a high number of authors' names which are less likely to show up in our five million token corpus. This means our initial corpus is sufficient for most of the words, but there are some that will throw off our algorithm and leave room for improvement.

4.2 Tests

Before we present our results, it is interesting to look at the effects of increasing the confidence levels. GOCR has its own internal confidence score associated with each letter that it converts, however this information is inaccessible to the end-user. What is accessible is a confidence threshold, adjustable per document. We experimented with various confidence levels, from 95% (the default) up to 99%. When GOCR has a match for a letter it checks the probability and only outputs the letter if the probability is above the threshold. Initially we worked with the default confidence level but found it to be limiting. Often letters were wrong or merged together in ways that our ambiguous character detection could not fix. Increasing the confidence level meant more words contained unknown markers, but this is beneficial because our work is designed to fix those words. The percent of words that the GOCR incorrectly classified for various confidence levels is plotted in Figure 1.

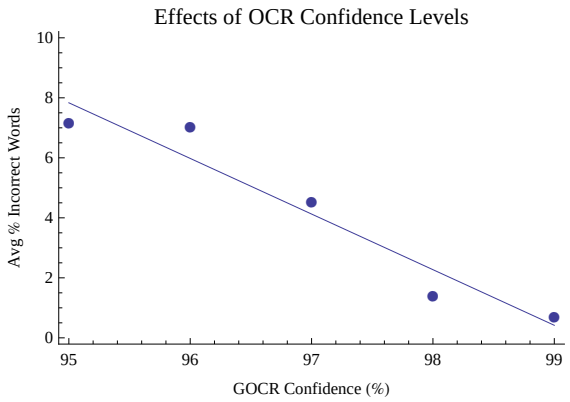

Figure 1: The percent of words GOCR believed it got correct but were actually wrong, averaged across 7 test documents for various confidence levels. Note that as we increase the confidence, we see a marked decrease in incorrect words.

We ran tests over each of the seven test documents and calculated statistics for each. These have been averaged together and are shown in Table 1. We see that recall and precision both increase along with confidence until 98 percent, at which point they both drop sharply. This is because at 99 percent confidence, too few characters are being classified so there is not enough context for our bigram model to have effect. We see that the best results are achieved with GOCR set to 98 percent confidence.

Because our work is post-processing on top of GOCR, we must think carefully about how to compute precision and recall. It is not valid to compare the final document with the original without considering the work GOCR does itself. In the raw GOCR output there are some number of unknown markers. We define the set of possible words to be corrected as those containing at least one unknown marker. From those words we can either choose to fill in a new word or leave it unchanged. Precision is thus the number words that we guess correctly out of the number words we chose to guess. Recall is defined as the number of words we correctly guess out of the total number of possible words to guess. We consider guesses to be correct when the words match exactly or if they only differ in case.

GOCR Confidence	Recall	Precision	Correct Fixed Words	Total Fixed Words	Unfixed Words	OCR Wrong Words
95 %	78.6 %	86.2 %	96.9	110.4	13.0	32.0
96 %	78.5 %	87.7 %	97.1	111.0	13.0	32.0
97 %	76.4 %	83.9 %	173.9	207.4	20.14	19.3
98 %	**81.9 %**	**87.8 %**	200.0	232.0	12.9	3.4
99 %	68.4 %	71.1 %	227.7	320.1	12.7	1.6

Table 1: Statistics on correction algorithm for five OCR confidence levels. Results averaged across 7 test documents. Highest precision and recall values are in bold.

5 Analysis

In addition to quantitative results, we also closely analyzed our results qualitatively. Here we present some key examples that point to the impact of our design decisions. We also discuss some limitations of those decisions.

5.1 Impact of Confidence Levels

Let us examine the recognition of the word `understand` in one of our testing documents. When set to a 95 percent confidence threshold, GOCR converted this word in the source image to the word `urlderstarld`. GOCR incorrectly classified the consecutive characters `nd` as `rld` twice. This is a word that we can't correct because `rl` is not in our ambiguous characters dictionary and there are no unknown indicators in the word. However, when we run GOCR with a confidence threshold of 98 percent on the same source image we get `u_derst__d` as the output. GOCR was unsure about more than just the `n` characters but importantly it did not misclassify any characters, so our algorithm is able to select `understand` as the best candidate. While there are far more unknown characters, roughly on the order of 3 times as many across our testing documents, our models work well in such environments. Thus, there is significant value in setting GOCR's confidence threshold high.

5.2 Expanding Unknown Markers

Initially we were unsure how many words to match in the initial pass against the nltk corpus. For words with unknown characters at the beginning of the word there can be a huge number of potential matches, somewhere in the range of 300,000 or more for three unknown markers (`___`). We initially chose 5 matches as a reasonable constraint, but this proved too limiting as the correct word was often not in the window of matches. This is due to our local corpus and the Google corpus having different unigram count accuracies. However, there was not a difference in the precision of our results regardless of whether the window was set to 10 options or 100 options. In the results shown the window is set to 10 options. The runtime of our algorithm did not increase noticeably when the window was increased to 10 or even 100 options, indicating the largest cost is in file access of the Google n-grams.

Interestingly, we discovered a case where we do correct misspellings. The word *consciousness* was spelled incorrectly as *conciousness* in one of our testing documents and because its OCR output contained an unknown character (`co_ciousness`) this got expanded to two unknown characters and then was matched against the correct word `consciousness`.

5.3 Limitations

There are a few places where the model still fails consistently. If a number appears before a word with unknown characters (i.e. `there are 45 clo_es`) the bigram model is thrown completely off because the word following the number rarely depends on value of that number. We also have a hard time with proper nouns where even Google has not seen the word we are looking for. Proper nouns, such as authors' names, are often document-specific and can be repeated many times throughout a single document. For instance, in one of our testing documents the proper noun *Premacks* occurred 10 times. In that same document, we chose to ignore

only 19 words which had unknown characters within those words. Thus that one proper noun accounted for over half of our non-fixed words.

We are also often expanding or shrinking an unknown marker and then selecting a word based on the new match which is more popular but is incorrect. For example, in `unintentionally give_` the marker was reduced to nothing and the word was changed to `give` because the bigram `unintentionally give` occurs more than `unintentionally given`. A possible response might be to weight the bigram or unigram counts based on how the unknown markers in the word were expanded, contracted or left the same. The expansion and contraction was implemented because GOCR sometimes misreads two characters as one unknown, or one character as an unknown and the correct character.

The process currently takes a long time to run, in the range of 8 minutes for our documents with 450 words, 225 of which are ambiguous. This stems from two factors, the first is the large number of matches which can potentially be generated for each ambiguous word and the second is the large amount of disk access needed to use the google corpus. Possible improvements to this are discussed in Section 6.1. However, the currently large runtimes prevent this from being used as a reasonable addition to OCR in cases where speed of results is at all important.

6 Future Work

There are several ways to extend this work. The biggest problems involve the speed of the program. It takes an unreasonable amount of time to process the output and make corrections. We can also consider the way we use n-grams and ways we might want to extend this. We then turn to other uses for our program aside from OCR.

6.1 Speedup

Since all the n-gram data is stored externally in large files, random access is very slow. We have identified two simple ways of speeding up the process. The first is to load the k most popular bigrams and unigrams from disk into memory. We expect that most of the unigrams are contained in the top million entries and the unigrams are perhaps in the top five million entries. If an n-gram is not found in memory, we still have the option of backing off to disk. Taking an initial step to calculate these would be expensive, but we can later save only the top entries to disk which can be loaded quickly.

Another method of reducing runtime is to break Google's n-gram files into smaller files. Currently, the files are broken into sets of large (approximately 160 MB) files and for each set there is an index with the head n-gram of each file and where it is found. The index allows us to quickly seek to the correct file, but the files are still so large that scanning for an n-gram towards the end of the file is still too expensive. A set of smaller files and an updated index will greatly speed up the program. These two approaches are not mutually exclusive. Instead, they should be combined such that we can check memory and if the check fails we can get faster lookups on disk.

6.2 Compare N-Gram Position

Our current method of using n-grams gives some context, but sometimes more is needed. Consider the examples

`he is _ beast`

and

`he is _ animal`

In the first case, the unknown indicator should expand to `a`. In the second case, it is clear the indicator should be `an`. However, in our current system, both cases will be examined with the bigram `is _` and will expand to the letter `a` since there are an order of magnitude more examples of `is a` than `is an`. To account for this, we can use a trigram model with the ambiguous word sandwiched between two known words. In this case, it becomes apparent which articles are needed. While there are still counts in Google's index for ungrammatical combinations (by the nature of their data source), they are dwarfed by the grammatical statements. Note that we could do this efficiently since we have the first word in the trigram, which allows us to index into the correct file and scan from there.

Another example may be instructive. Suppose we have the OCR output

`to _spire towards`

Using the current bigram model, `to inspire` has a count of 913,869 against `to aspire` at 95,204. On the other hand, if considering the full trigram, `to aspire towards` is seen 1,347 times, while `to inspire towards` is counted only 85 times. Again we see that the added structure can make a significant difference. However, it is important to note that as the OCR character-level confidence is increased the probability that an ambiguous word is found between two known words decreases. Pursuing this line of experiments will require carefully finding a balance between confidence and usefulness of the n-gram model.

6.3 Other Applications

Broadly speaking, we can apply our process to any transcription program that outputs text and indicates where it is unsure. Additionally, the input should not be random words thrown together, like one might find on a receipt or bill statement. Since we primarily rely on an n-gram model, we need text with uncertainty that benefits from the added syntactic context. Any type of lossy communication channel is an excellent application for our services. Here we have perhaps spoken word transcriptions with uncertainties. Moreover, we suspect Google's n-grams is more representative of spoken word than formal writing, so using their n-grams will be particularly effective.

7 Conclusions

We have presented a post-processing algorithm to edit OCR output from GOCR. The first step is to learn a list of character pairs that are commonly confused. This enables us to swap these characters and check if a word is created as a result. The second step is more involved, making use of Google's n-grams. We must check the output for ambiguous words preceded by a known word. We can then iteratively improve our guesses by choosing the best bigram (or unigram if necessary) in successive passes of the OCR output.

Despite prohibitively large runtimes, we have shown very promising results. Over different corpora, we have maintained high precision and recall. At 98 percent confidence levels, our algorithm achieves a precision of 87.8 percent and a recall of 81.9 percent. As we have discussed, many of the issues result from proper nouns that are not in our initial corpus and from insufficient context that would be supplied by the trigram model. With some of the modifications addressed earlier, this algorithm can significantly increase accuracy from original OCR transcription.

References

Thomas M. Breuel. 2008. The OCRopus open source OCR system. In *Proceedings of IS&T/SPIE 20th Annual Symposium*.

Alex Franz and Thorsten Brants. 2006. All our n-gram are belong to you. `http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html`, August.

Wanli Ma, Dat Tran, and Dharmendra Sharma. 2007. Detecting image based spam email. In *Proceedings of the 1st international conference on Advances in hybrid information technology*, ICHIT'06, pages 168–177, Berlin, Heidelberg. Springer-Verlag.

Joerg Schulenburg. 2005. GOCR LinuxTag Lecture. `http://www-e.uni-magdeburg.de/jschulen/ocr/linuxtag05/v_lxtg05.pdf`.

Joerg Schulenburg. 2010. GOCR. `http://jocr.sourceforge.net/`.

Ray Smith. 2007. An overview of the tesseract ocr engine. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition*, volume 2, pages 629–633, Washington, DC, USA. IEEE Computer Society.

Sargur. N. Srihari and Edward. J. Kuebert. 1997. Integration of hand-written address interpretation technology into the united states postal service remote computer reader system. In *Proc. Int. Conf. Document Analysis and Recognition (ICDAR)*, pages 892–896. IEEE-CS Press.

Luc Vincent. 2007. Google book search: Document understanding on a massive scale. In *In Proceedings, IAPR 9th Intl Conf. on Document Analysis and Recognition (ICDAR07)*.