Ensemble Optical Character Recognition Systems via Machine Learning

Zifei Shan, and Haowen Cao
Department of Computer Science, Stanford University
{zifei,caohw}@stanford.edu

ABSTRACT

Optical Character Recognition (OCR) Systems are widely used to process scanned text into text usable by computers. We observe that current OCR systems have bad performance on domain-specific papers, even generating lots of incorrect words; besides, different OCR systems make relatively independent mistakes. Based on these observations, we train an ensemble system from multiple open-source OCR systems, which chooses outputs from candidates generated by each OCR, and train the system with machine learning techniques. We implement Softmax Regression and multi-class SVM. Our system achieve over 80% accuracy selecting between different outputs on our training set of 1,011 words. We further explore ways to improve the performance by suggesting new options, and use domain knowledge to improve its performance.

Our contribution lies in following aspects: (1) We show the great potential of treating OCR systems as black-boxes and correct their outputs from each other. (2) Our system build on best open-source OCRs and achieve significant improvement on their accuracy. (3) Moreover, our work explore the possibility to make use of rich semantic knowledge to craft a better OCR system, and cast insight to a general approach to ensemble systems as black-boxes.

Keywords

Machine Learning, Optical Character Recognition, Ensemble Learning

1. INTRODUCTION

Optical character recognition (OCR) is a common technology to convert scanned images of handwritten or typewritten text into machine-encoded text. It is widely used in digitizing books and texts so that they can be used in electronically search, machine translation and text mining. Usually OCR will do pre-processing, character recognition and post- processing.

Tesseract [7] is a well-known open-source OCR engine that was developed by HP between 1984 and 1994. Tesseract assumes that its input is a binary image with optional polygonal text regions defined. Tesseact processing follows a traditional step-by-step pipeline. It includes connected component analysis, recognition and resolving fuzzy spaces. It is the first OCR able to handle white-on- black text trivially.

Cuneiform [8] is another powerful open-source OCR software that we use. The algorithms used in Cuneiform come from the rules of writing letters and do not require pattern recognition learning. It can recognize more than twenty different languages including English, French, Russian and etc.

In the usage of these two systems, we found following interesting observations that inspire our work:

(1) Different OCR systems make independent mistakes. Based on studying the outputs of three open-source OCRs on processing geo-science papers, we found an interesting fact that they seldom make same mistakes; instead, they fail on different word. For example, Cuneiform is bad at recognizing cases that all characters are upper-case, while Tesseract can give correct outputs in this case; another example is that Tesseract often read *"fi"* as "?", while Cuneiform doesn't.

(2) If the output does not exist in a knowledge base, it can hardly be correct. The OCRs we use sometimes generate words that does not exist in dictionaries or web pages. To generalize, if the sentence cannot be mapped into a correct grammar, it is also likely to be a mistake.

These observations inspire us with the following ideas: what if we combine these OCR systems together and use them to correct each other? Can we give suggestions to achieve the correct output even when all OCRs fail? Can we further combine semantics information to do a better job? Can we use domain knowledge of the corpus to improve the results?

The core idea of our work is following: we treat each OCR system as a black box, which takes images as input and generates text as output. Instead of understanding what is happening inside these black boxes, we directly use their outputs to train a classifier that selects among their outputs.

2. RELATED WORK

Ensemble learning [2] is a popular machine learning method, which aggregates a set of individually trained classifiers, and make a combined prediction. Previous research has shown that an ensemble is often more accurate than any of the single classifiers in the ensemble. Our system is like this, to combine the two results of OCR together and make one decision.

As a result of Ensemble learning, *Ensemble systems*, or known as multiple classifier systems, have shown to produce favorable results compared to those of single-expert systems for a broad range of applications under a variety of scenarios. In [5], it tells specifically the conditions under which ensemble based systems are more beneficial than their single classifier counterparts. Ensemble based systems have advantages in statistics, large data utilizing, dividing and conquering, etc.

In our case, different OCR softwares have different complex algorithms, and it is impossible to merge them from inside, treating them as white- boxes. However when we treat them as black-boxes and learn a ensemble system based on their outputs, we have a significant chance to achieve an efficient error correction rate, thus improving the overall accuracy of OCR.

3. MODEL

Figure 1: Example of a word in the document

3.1 Problem Definition

To craft an ensemble OCR system, we start by modeling the OCR-aggregation task as a classification problem. For each *word* in the document, OCRs will give *options* to this word. We can further *suggest* more options by other means such as spell-check tools. As options as words can be in infinity space, we extract features from all these *options* to serve as inputs for this *word*. Denote k as the number of options for a word, and treat k as constant, then we have the following classification problem:

PROBLEM 1. *Documents (D) consist of words (W). For each word w in W, k options are given by outputs of k OCRs.*

Each word has n features extracted from all its options. Denote n as the number of features. $x = \{x_1, x_2, \ldots, x_n\}$ *is a set of features, which is the input of the classifier.*

Our classifier outputs a label $y \in [1, k+1]$ *for each word. Class* $y = i$ *(*$i \in [1,k]$*) means that the ith option is the correct output for this word, and* $y = k+1$ *means that none of the OCR outputs are correct.*

For example, for a word in the document shown in Figure 1, our option set is {*Pataeoctimatology, Palaeoclimatology*}, where the first option is given by Cuneiform and second given by Tesseract. $y = 2$ is the correct output label.

3.2 Dataset Selection

Now that we have the classification problem, we need a dataset to work on. However, getting accurately labeled scan of documents for training is not trivial. We looked into IAM dataset [3], which is an labeled English handwritten collection. We tried to use our OCR systems on these datasets, however we found that these open-source OCRs do an extremely bad job in handwritten images.

To better fit in our use case, we craft our own dataset by picking three (26141 words in total, ∼30 pages each) geology papers and hand- labeling them.

3.2.1 Data Preprocessing

We managed to run a complex pre-processing alignment algorithm to align each word in output of two OCRs, based on geometric position (boxes) of words on documents, given by OCR systems. After alignment, our dataset contains *21652* words that are correctly aligned.

3.2.2 Hand-Labeling Assumption

For most of aligned words, options given by Tesseract and Cuniform agreed with each other. We looked through them and cannot find any error in them. To make sure, we checked the dictionary for word correctness, and output the cases where words do not exist in dictionary. However, we found that words outside dictionary are actually valid words in geology domain, and are actually correct on papers.

Based on above observation, we make an assumption before hand-labeling: **outputs that are agreed by two OCRs are correct.** This dramatically reduces our workload to only *1011* words, where OCRs give different results.

This assumption also helps us in the classification task: for words that options agree, we directly give the agreed output; only for words that options do not agree, we run the classifier to judge which options to output.

Table 1: Error Analysis

OCR	Error Case	Occurrence
Tesseract	Recognize *fi* as *?*	44
Tesseract	Recognize *fl* as *?*	30
Tesseract	Recognize *rn* as *m*	13
Tesseract	Recognize *rm* as *nn*	3
Tesseract	Recognize *l* as *]*	2
Tesseract	Recognize *j* as *?*	2
Tesseract	Recognize *"* as *?*	5
Cuneiform	Fail on all-uppercase words [1]	16
Cuneiform	Recognize *e* as *c*	8

3.3 Feature Design

Recall that we have mapped the problem into a classification problem by extracting features of all options, the selection of features is critical. We conduct an error analysis on our dataset, comparing what errors do OCRs make independently, to indicate what features are useful in the classification.

3.3.1 Initial Error Analysis

In hand-labeling process, we are able to analyze the errors that OCRs individually make.

We output all the cases that OCRs give different answers, and list the major ones in Table 1.

3.3.2 Features

Based on errors in Table 1, we design following features for classification. Note that these features are adopted to *each option*, e.g. *DictValid(1)* and *DictValid(2)* are two different features.

- $WL(i)$: Word length.
- $WebOccur(i)$: Number of Search result on Internet.
- $DictValid(i)$: whether option i is valid in dictionary.
- $UpperPunish(i) = -4x(x-1)$, where x is upper case percentage in the option.
- $LowerToUpperChange(i)$: how many times in the option is a lowercase character followed by an uppercase character.
- Count of each character occurrence: $Cnt_x(i)$, where $x \in \{a, b, c, \ldots, z, SpecialChars\}$, and *SpecialChars* is characters like *[,], ?, !, ", '*, etc.
- Count of occurrence of two consequent character: $Cnt_{xy}(i)$, where $x, y \in \{a, b, c, \ldots, z, SpecialChars\}$.

This gives us more than 2,000 features, which is larger than our training set. To reduce the number of features, we only pick $Cnt_{xy}(i)$ where combination *xy* appears in our error analysis. Specifically, we pick *xy* in *fi, fl, rn, nn, rm*.

4. ALGORITHMS

Given the problem model and the feature set, we now have a multi-class classification problem. We try two learning algorithms to achieve an ensemble system: *Softmax regression*, and *Multi-class Support Vector Machine*.

To select most useful features and prevent overfitting, we implement a feature selection algorithm based on a *Forward Search* with K-fold Cross-Validation.

[1] For example, recognize *EOCENE* as *EocENE*.

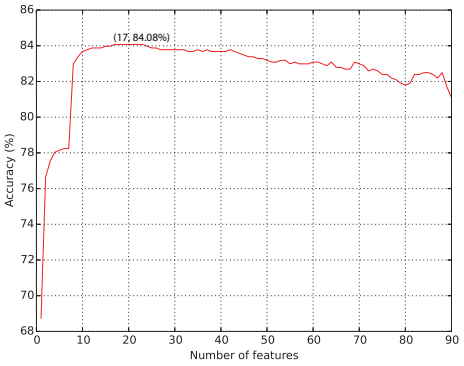

Figure 2: Foward Search Result

Table 2: Most Useful Features

Rank	Feature
1	Cnt(?,1)
2	Cnt(',1)
3	Cnt(rn, 0)
4	Cnt(nn, 1)
5	Cnt(nn, 0)
6	Cnt(v, 0)
7	WebOccur(0)
8	WebOccur(1)
9	Cnt(?,0)
10	UpperPunish(1)
11	Cnt(-,0)
12	Cnt(rn,1)
13	Cnt(d,0)
14	Cnt(b,0)
15	Cnt(",0)
16	Cnt(q,0)
17	Cnt(],1)

4.1 Feature Selection

Originally we set 90 features for classification. But too many features cause overfitting and have high variance. To fix the variance, we implement a Forward Search to select the most useful features.

We use K-fold cross validation on the *SVM with linear kernel*, on a dataset of 1011 examples, to greedily choose one feature with most gain in accuracy in each iteration. Figure 2 shows the result of the forward search. We see that when 17 features are used for SVM instead of 90 features, we can get a higher accuracy of 84.08% under SVM with a linear kernel.

We list the most useful features for SVM linear kernel, in Table 2, based on the order they are selected by the forward search. These results are in accordance with our initial error analysis and our feature design, e.g. *WebOccur, UpperPunish(1), Cnt(?,1), Cnt(rn, 0)* are quite expected.

5. EVALUATION

In this section we evaluate the performance of our ensemble system. We conduct Softmax Regression and Multi-class Support Vector Machine to train our classifier, on the dataset of 1011 examples.

We use a *K-fold Cross-Validation*, and pick $k = 5$, where each partition contains 202 examples.

We choose different parameters on the two training algorithms to experiment on. For Softmax, we use different learning rates and also

Table 3: Results of different classifiers and parameters

Classification Method	Accuracy
Softmax (learning rate = 0.01)	70.27%
Softmax (learning rate = 0.015)	76.23%
Softmax (learning rate = 0.015, decay = 0.005)	69.87%
SVM (linear kernel, #feature = 17)	84.08%
SVM (polynomial kernel, degree = 2, #feature = 90)	82.89%
SVM (polynomial kernel, degree = 3, #feature = 90)	82.69%
SVM (Gaussian kernel, #feature = 90)	81.70%

Table 4: Definite Accuracy of different Classifiers in terms of output text

Classifier	#Correct	Accuracy
Ensemble	766	75.77%
Cuneiform	361	35.71%
Tesseract	492	48.66%
Neither	158	15.63%
Total	1011	100%

Table 5: Class Distribution for SoftMax

Data\Predicted	1	2	3
1	30.36%	8.15%	0.33%
2	5.69%	39.73%	0.33%
3	5.47%	3.79%	6.14%

observe what influence will be if we use the *l1-regularization* on the classification. For the Support Vector Machine, we use different kernels to test its result.

In Table 3 we can see that different learning rates have some influence on the Softmax, while different kernels has small influence. L1-regularization does not give us a good performance. And different kernels has small influence on the accuracy of the cross validation. Among them, the Support Vector Machine with the linear kernel has the highest accuracy. And in conclusion, the accuracy of Support Vector Machine is apparently higher that the Softmax Regression.

For Softmax, we use all 90 features. For SVM, we use the selected 17 features. "Decay" refer to weight decay parameter in L1 regularization. In Table 5 and Table 6, we list the class distribution for Softmax and SVM. (Row i, column j) stands for times when data with real label i is has predicted label j. Results show that both system seldom have false positives in class 3. However, each system has a relatively high number of false positives in class 2. **When one OCR has the right answer (class is 1 or 2), SVM can achieve a total of 89.80% precision**. This exciting result indicates that our system have quite high confidence of choosing the correct output among two OCRs if one of them is correct (and the other is incorrect).

Another indicator shown in Table 4 is that we have a total correction rate (definite accuracy) of 75.77%, which means: **the chance that our system output the correct word, when two OCRs differ, is over 75%.** While for two single OCRs, this chance is respectively 35.71% and 48.66%. [2] These results show that our ensemble system achieves significant improvement over the single systems.

6. IMPROVEMENT BY SUGGESTIONS

[2]Note that this accuracy do not count class 3.

Table 6: Class Distribution for SVM

Data\Predicted	1	2	3
1	28.78%	6.82%	0.10%
2	1.48%	46.98%	0.20%
3	3.86%	3.46%	8.31%

Table 7: Class Distribution for SVM with 2 suggestions

Data\Predicted	1	2	3	4	5
1	121	35	6	0	7
2	20	362	3	0	8
3	4	4	2	0	3
4	0	1	0	0	1
5	11	16	2	0	21

We further study ways to make suggestions to OCRs, to improve the classifier granularity. Previously when neither options are correct, the example is in class 3. With suggestions, the system may have a better chance to give a right answer, and improve the overall correction rate as shown in Table 4.

For each word, its M suggestions are chosen from words appearing in our corpus (130 documents), according to the Edit Distance [6] from options, and its word frequency in the corpus. Suggestions are different with each other.

The features for suggestions are:

- $Dist(i)$: Edit distance from each option i.
- *CorpusOccur*: Word frequency in corpus. (This feature is using the domain knowledge.)
- *WL*, *WebOccur*, *DictValid*, as defined in Section 3.3.2.

We create a labeled dataset of 627 examples with labels $y \in [1, k+M+1]$ and $M = 2$. We train SVM with polynomial kernel with degree$=3$, using 112 features. It achieves 80.70% accuracy in cross validation.

We further plot the class distribution in Table 7. Since our dataset is relatively small in terms of class 3 and 4, there is inevitably some overfitting. With a larger dataset, we can expect a better performance.

7. FUTURE WORK

We want to study how to leverage domain knowledge to help our task. Currently we have studied a set of additional features with domain knowledge: number of occurrence of option in the entire corpus (130 documents). Yet they do not increase the accuracy on the basis of 17 selected features.

In the future, we are more interested to explore how to use domain knowledge: can we have lexical and syntactical features using NLP techniques? Can we have semantic features using relations in the domain knowledge base? Can we use learning models with higher expressiveness, such as statistical inference on factor graphs [1], to model dependencies of predictions? In addition, we want to automatically generate training "silver-rule" examples using distant supervision [4], making use of word-level and sentence-level rules.

8. CONCLUSION

We study the methodology of ensemble learning on OCR systems, by adopting learning algorithms on aligned output. Our work is based on the observation that open-source OCR softwares make independent errors. We assume that when OCR outputs agree with each other they should be correct, therefore our ensemble system chooses among options when OCR results disagree with each other.

We select the known best open-source OCR systems, Cuneiform and Tesseract. We run these OCRs on our data corpus of 130 scanned geo- science papers, align them word-by-word, and craft a hand-labeled dataset of 1,011 words where outputs of OCRs disagree, out of 21,652 aligned words.

We implement Softmax Regression and Multi-class SVM to approach this classification problem. We conduct feature selection with a forward search, and pick 17 best features for SVM with linear kernel. In our evaluation, we found that SVM outperforms Softmax by about 10%.

For a detailed performance of SVM with 17 features:

1. Whole-document output accuracy is 98.87%, while Tesseract is 97.60%, and Cuneiform is 97.00%.
2. Accuracy when choosing among three classes (Cuneiform, Tesseract, and asserting neither of them is correct) is 84.08%.
3. Output accuracy when two OCRs disagree with each other is 75.77%, while Tesseract is 48.66%, and Cuneiform is 35.71%.

Our contribution includes: demonstrating the power of ensemble OCR systems to significantly improve individual OCR quality; building on best open-source OCRs and potentially creating a better software for users; exploring the possibility to make use of domain-specific knowledge to craft a better OCR system.

Acknolwegement

The authors would like to acknowledge Professor Andrew Ng for teaching the skills of machine learning and encouragements for the projects. Thanks to the teaching assistant Kyle Anderson for the detailed feedback and useful advice of our proposal. Thanks to Professor Christopher Re for providing us the data to enable this work.

9. REFERENCES

[1] M. I. Jordan. Graphical models. *Statistical Science*, pages 140–155, 2004.

[2] R. Maclin and D. Opitz. Popular ensemble methods: An empirical study. *arXiv preprint arXiv:1106.0257*, 2011.

[3] U.-V. Marti and H. Bunke. A full english sentence database for off-line handwriting recognition. In *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on*, pages 705–708. IEEE, 1999.

[4] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

[5] R. Polikar. Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE*, 6(3):21–45, 2006.

[6] E. S. Ristad and P. N. Yianilos. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(5):522–532, 1998.

[7] R. Smith. An overview of the tesseract ocr engine. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 629–633. IEEE, 2007.

[8] C. Technologies. Cuneiform ocr. `http://en.openocr.org/`.