OCR Formal Model Definition

Zifei Shan

1 Input and Output

1.1 Input

Input is a relation of OCR outputs:

```
OCR(ocrid, docid, wordid, word)
```

Each row in the relation correspond to a word output by a certain OCR, a row `ocrid, docid, wordid, word` means that the *ocrid*'th OCR believes that in document *docid*, the *wordid*'th word is *word*.

There can be structural information and visual layout information in the input listed in following relations:

```
OCR_sentence(ocrid, docid, sentid, wordid_from, wordid_to)
OCR_box(ocrid, docid, wordid, word_box)
```

1.2 Output

We want to finally populate a relation of documents predicted by our system, which contains words for each position in each document:

```
Doc(docid, wordid, word)
```

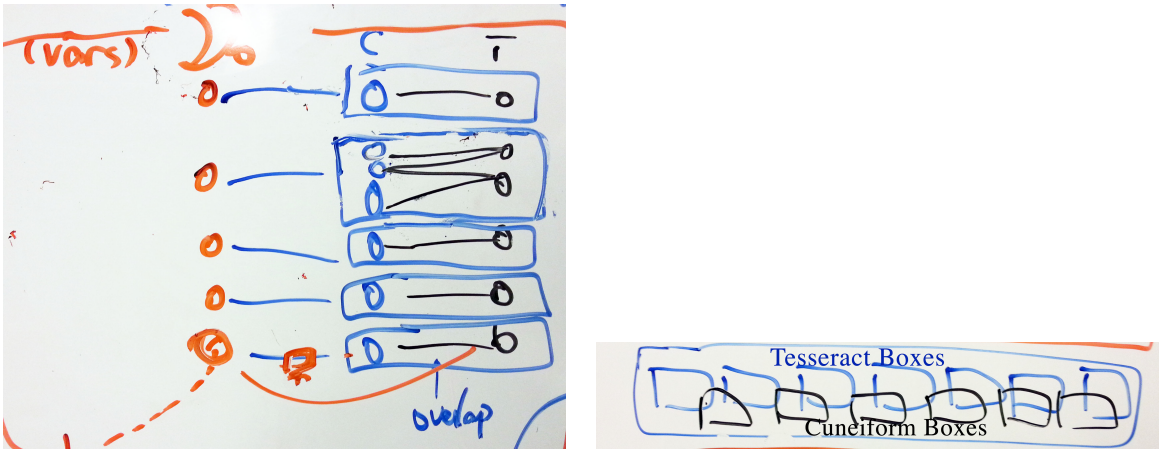
2 Procedure

2.1 Factor Graph generation

To generate the factor graph, we generate a multinomial variable for each word, whose domain is all its possible candidates. To collect all candidates for each variable, first we need to decide the number of words in the document, and how each word in a document maps to one word in OCRs' output.

For each document, each OCR gives a sequence of output words, with bounding boxes for each word. Different OCRs give different bounding boxes, and there might be multiple ways to generate variables:

1. An naive way is using bounding boxes of one OCR as a baseline, and align all other OCRs to it. This has the problem of how to choose the baseline OCR, and whether this decreases the recall.



(a) Sample G with Bounding boxes for 2 OCRs (b) Sample case of a large connected component

Figure 1: Factor Graph Generation

2. To better approach it, we first construct a graph G: each node in G is one output word by an OCR, and there exists an edge between two words if their bounding boxes overlap. See Figure 1(a).

    - Then we find all connected components in G. For each connected component, we create one variable v in our factor graph.
    - This has a problem when there are a lot of large connected components, as in Figure 1(b). To verify whether this problem matters, we need to plot the distribution of *#connected components*.

Denote the generated variables as V_O. Assume that we choose the 2nd approach to generate variables. For each variable $v \in V_O$, v is a multinomial variable, and its value ranges from all possible candidates. Each variable $v \in V_O$ corresponds to an output word in relation `Doc`:

```
Doc(docid, wordid, word)
```

Since we preserve the mapping between OCR outputs and variables based on bounding box overlappings, we are able to get candidates for each word given by OCR outputs, and generate further candidates based on edit distance, etc. In our implementation, we store all possible candidates for each word into relation `Candidate`:

```
Candidate(docid, wordid, candid, word)
```

2.2 Feature extraction

We extract features for each variable, with information about all candidates:

```
Feature(docid, wordid, fname, fval)
```

2.3 Supervision

Our training data is

```
LabeledDoc(docid, wordid, word)
```

which is extracted from hand-labeled documents (HTML files) that we assume correct.

For each word variable $v \in V_O$, denote its domain to be $\{c_1, c_2, \ldots, c_k\}$.

Supervision is a function f that labels a candidate for a variable:

$$f : (v, c_i) \to \{True, False, Unknown\}$$

where True labels give positive examples, False labels give negative examples.

Naive direct supervision would wrongly assume that `Doc` is aligned with `LabeledDoc`. Therefore we might need distant supervision.

2.3.1 Distant Supervision by 1-Gram in Labeled Document

First we construct $1gram$, a set of 1-grams (words) from LabeledDoc.

The supervision rule is:

$$f(v, c_i) = \begin{cases} True & \text{if } c_i \in 1gram \\ False & \text{if } c_i \notin 1gram \vee \exists c_j, \text{s.t. } f(v, c_j) = True \end{cases}$$

When conflict, we can break ties by:

1. Select a random candidate;
2. Select candidate with larger 1-gram frequency;
3. Break ties using 2-gram.

We can see that 1-gram distant supervision does not preserve order of document. To better capture the order, we may use 2-gram or n-gram.

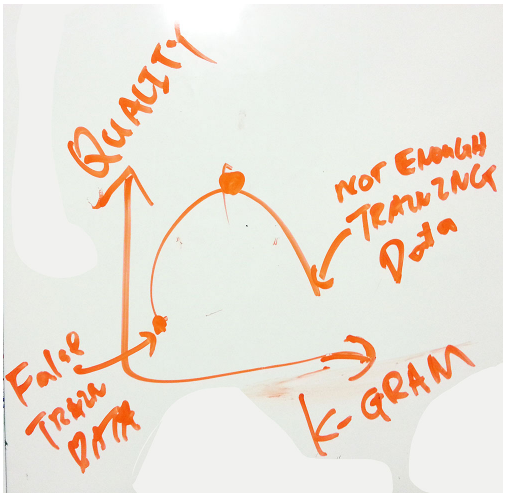


Figure 2: N-gram Tradeoff

2.3.2 Distant Supervision by n-Gram ($n > 1$) in Labeled Document

First we construct $Ngram$, a set of n-grams from LabeledDoc. N-gram can be used to break ties for *(N-1)*-gram, or to generate labels directly.

For generating labels, basically, we enumerate every n neighboring words, examine combinations of candidates to generate positive and negative labels.

When conflict, break ties randomly, by larger n-gram frequency, or using *(n+1)*-gram.

2.3.3 Tradeoffs of using N-gram for different N's

We will experiment on using different N's for supervision, and generate a plot similar as Figure 2. X-asis is N, and y-axis is supervision quality, on the left side quality is low because of false training examples, on the right side quality goes down because of not enough training data.

Similarly, we can also try supervision with other n-grams rather than only n-grams for this document, e.g. Google Ngram, Ngram from the whole LabeledDoc corpus, or a dirty Ngram from raw OCR outputs for Paleo documents.

2.3.4 Order-aware Supervision (open question)

N-gram supervision can only preserve partial order. To capture a global order, we might be able to implement an order-aware supervision, by finding an optimal mapping between predicted words and LabeledDoc (HTML) words, that maximizes recall of LabeledDoc words,

under constraint that the mapping preserves order. This method is similar as our evaluation method described in Section 2.5.

For every possible world I, we calculate the optimal mapping and assign *True* labels to the mapped candidates. The calculation of optimal mapping is $O(N^2)$, where N is the number of words in one document. To reduce N, we might partition the document into small chunks and find the mapping in each chunk.

2.4 Inference

We use the `Feature` relation and additional inference rules to populate the relation `Doc`, predicting multinomial variable `word`. When implementing with DeepDive, we transform this problem into predicting multiple boolean variables with constraints. Specifically, we transform `Doc` into relation `DocCand`, and predict boolean variable `isTrue`:

```
DocCand(docid, wordid, candid, isTrue)
```

We can populate relation `Doc` by picking the most possible candidate for each word, with predictions in relation `DocCand` and candidate-word mappings in relation `Candidate`.

2.5 Evaluation

For evaluation, we hold out a fraction of supervision set `LabeledDoc` as `Test`, and calculate the recall of predicted words on the testing set. We always hold out **entire documents**, e.g. all rows with $docid = 1, 4, 7$ in `LabeledDoc`.

For convenience, we select `Predicted` relation for each `docid` in Test:

$$Predicted = Doc \bowtie \pi_{docid}(Test)$$

The naive evaluation assuming perfect alignment is:

$$Recall = \frac{|Predicted \bowtie Test|}{|Test|}$$

However the naive evaluation would wrongly punish misalignments. To fix it, we enable any shifting of words while preserving orders, that can maximize the number of matched words in the testing set.

Evaluation for each single document: For each $docid = did$ in Test, denote:

$$Test_{did} = \sigma_{docid=did}(Test)$$

$$Predicted_{did} = \sigma_{docid=did}(Predicted)$$

Calculate the optimal mapping for document `did`, denoted as Map_{did}, which generates the relation MappedPred that has the maximal number of matched words with $Test_{did}$:

$$\begin{aligned} Map_{did} = \ & argmax_{Map_{did}}(|MappedPred \bowtie Test_{did}|) \\ & \text{s.t. } Map_{did}(i) = j \iff \pi_{word}\sigma_{wordid=i}MappedPred = \pi_{word}\sigma_{wordid=j}Pred_{did} \\ & \text{and } i < j \rightarrow Map_{did}(i) < Map_{did}(j), \forall i, j \in \pi_{wordid}MappedPred \end{aligned}$$

The optimal mapping can be calculated by dynamic programming, with a time complexity of $O(n^2)$.

We can therefore define the Precision and Recall:

$$Precision_{did} = \frac{|Map_{did}|}{|Predicted_{did}|}$$

$$Recall_{did} = \frac{|Map_{did}|}{|Test_{did}|}$$

And we can use the F1 score for evaluating the overall performance:

$$F1_{did} = \frac{2Precision_{did}Recall_{did}}{Precision_{did} + Recall_{did}}$$