Workflow of metadata extraction from retro-born-digital documents

Dominika Tkaczyk and Łukasz Bolikowski

ICM, University of Warsaw

DML 2011: Towards a Digital Mathematics Library
20-21 July 2011

The problem

Our goal is to be able to **extract metadata** from scientific articles:

- **title**,
- **authors** and **affiliations**,
- **abstract**,
- **parsed bibliographic references**,
- **journal**, **volume**, **issue** and **pages**,
- **year of publication**,
- etc.

The motivation

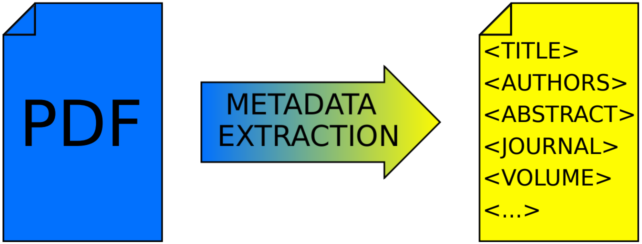
Why do we need to extract metadata from digital documents?

- There are **documents without metadata**.
- **Metadata information** may be **incomplete** or **incorrect**.

Previous work

- The **Medical Article Records System** (MARS) works on document scans in the form of TIFF images.
- **Flynn** *et al.* perform OCR on the document images and convert them to XML. The metadata is extracted based on rule-based approach.
- **Giuffrida** *et al.* propose a system for extracting metadata from PostScript files. They use `pstotext`-based tool for content extraction, while the metadata is extracted using a set of features and rules.
- **Dejéan** *et al.* extract structured content from PDF files. XY-cut-based approach is used for segmentation and detecting the reading order, while the logical structure of the document is obtained based on the table of contents.
- **Marinai** uses JPedal package for extracting characters from PDF documents, performs rule-based page segmentation, and finally employs neural classifier for zone classification.
- **Cui and Chen** employ a Hidden Markov Model to extract metadata from PDF documents, while page segmentation and text extraction is done by `pdftohtml`.

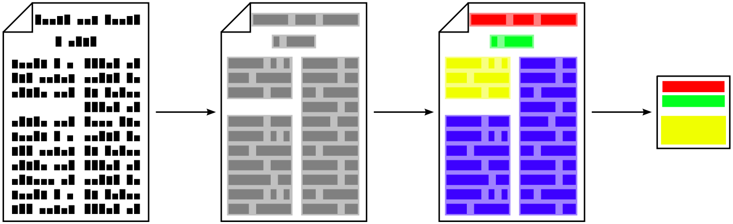
The solution



The **metadata extraction process**:

- the input is a **document in PDF format**,
- the output is a **metadata record** containing all extracted information.

The process



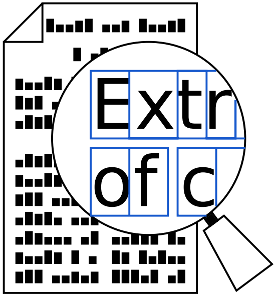
The metadata extraction process consists of three main stages:

1. **building a tree structure** storing the document's content
2. **analysing and enhancing the document's content** based on its structure
3. **generating the document's metadata record** based on enhanced content

Character extraction

The purpose of the **character extraction** step is to build an initial, flat structure storing the document's content. As a result we get a set of individual characters with their bounding boxes.

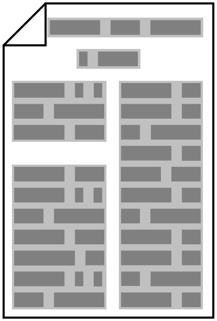
The implementation is based on **iText** library.



Page segmentation

In the **page segmentation** step we group individual characters into words, lines and zones to build a tree structure representing the document's content.

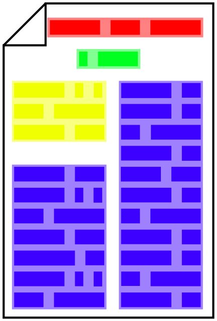
Current implementation is based on top-down **X-Y cut** algorithm. In the future we plan to replace it with a bottom-up approach based on **Docstrum**.



Zone classification

The purpose of the **zone classification** step is to classify the document's zones as *title*, *author*, *affiliation*, *abstract*, *body*, *footer*, *header*, etc.

The implementation is based on **Hidden Markov Models** with all probablity information obtained from a training set. We use **Viterbi algorithm** to calculate the most probable labels of a sequence of zones based on zones' feature vectors.



Zones' feature vectors

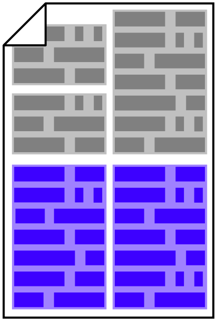
We use **37 features** to describe the document's zones:

- features refering to the **zone's bounding box** and its **position on the page**, e.g. zone's absolute and relative dimensions, horizontal and vertical position,
- features related to **the inner structure of the zone**, e.g. absolute and relative number of text lines and words, mean text line height, width and position,
- features based on **the text of the zone**, e.g. the number of characters, digits, lowercase/uppercase letters, punctuation marks.

Citation extraction

In the **citation extraction** step we locate the bibliographic references block in the document's body.

Current implementation processes only the text content of the document and is based on simple character frequency heuristic. In the future we plan to implement a citation extractor that makes use not only of the text content, but also of the document's tree structure.



Citation parsing

[8] Y. Wang, I.T. Phillips and R.M. Haralick, Document zone content classification and its performance evaluation, Pattern Recognition 39 (1) (2006), pp. 57–73.

The purpose of the **citation parsing** step is to identify the citation's fragments containing *author*, *title*, *journal*, *volume*, *series*, *number*, *publisher*, *pages*, *year*, etc.

The implementation is based on **Hidden Markov Models** with all probablity information obtained from a training set. We use **Viterbi algorithm** to calculate the most probable tags of a sequence of citation's tokens based on tokens' feature vectors.

Citations' feature vectors

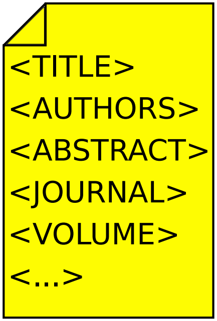
We use **48 features** to describe citation's tokens:

- features measuring relative number of **a particular character type**, e.g. digits or lowercase/uppercase letters,
- features checking whether the token is **a particular character** (a comma, a dot, a quote, etc.),
- features checking whether the token is **a particular word** ('and', 'http', 'vol', etc.),
- features based on **dictionaries built from the training set**, e.g. a dictionary of cities or words commonly occuring in the journal title.

Metadata extraction

During the final step, the metadata is extracted from the document's labelled content and the resulting metadata record is formed.

This is still to be implemented.



Experiments

We measured the performance of our implementations of zone classifier and citation parser:

- In the case of the **zone classifier** we used documents obtained from **MARG** repository for both training and test sets. The training set we used consisted of 317 elements and 1379 zones. The tests we performed on 1236 documents with 5359 zones gave the accuracy rate **95.5%**.
- Both training and test sets for **citation parser** were obtained from digital collections **NUMDAM** and **CEDRAM**. The training set we used consisted of 2318 bibliographic references. The tests we performed on 2532 references gave the accuracy rate **85.8%** of correctly identified fragments of bibliographic references.

Future work

Our plans for the future include:

- replacing current implementation of the page segmenter with a version based on Docstrum algorithm,
- an implementation of a better bibliographic reference extractor that makes use not only of the text content, but also of the document's tree structure,
- an implementation of the final step of the process, the metadata extraction,
- constructing semi-manually a better test set for page segmenter and zone classifier.

Thank you

Thank you!
Questions?

Dominika Tkaczyk
d.tkaczyk@icm.edu.pl

Docstrum

Docstrum

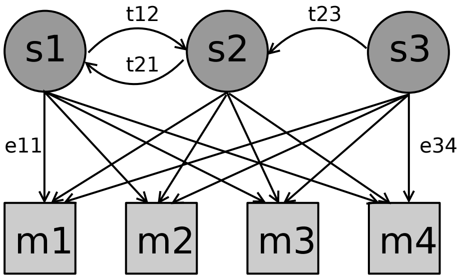
Docstrum

The main stages of **Docstrum** algorithm:

1. calculating **nearest-neighbour pairs** of individual characters
2. generating **angles histogram**
3. estimating **text line orientation**
4. generating **within-line** and **between-line distance histograms**
5. estimating the **character spacing** and **text line spacing**
6. **finding text lines**
7. calculating the final estimation of text line orientation
8. **grouping text lines into zones**

Hidden Markov Models

Hidden Markov Models

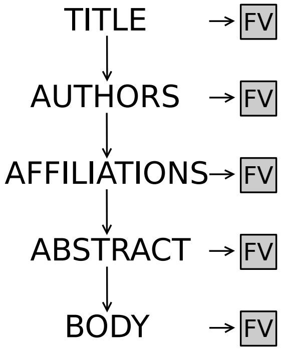


- **s** – states
- **m** – emitted messages
- **t** – transition probabilities
- **e** – emission probabilities

Zone HMM example

Hidden Markov Model for **zone classifier**:

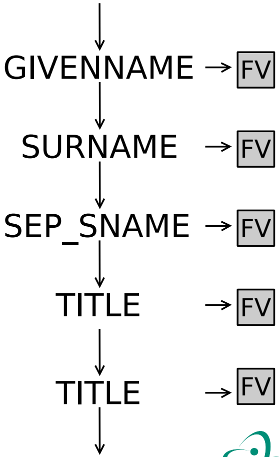
- states – **zones** with **unknown labels**
- state sequence – a sequence of **all zones from one page sorted** accordingly to their positions
- emitted messages – **feature vectors**



Citation HMM example

Hidden Markov Model for **citation parser**:

- states – **citation tokens** with **unknown tags**
- state sequence – a sequence of **all tokens forming a citation**
- emitted messages – **feature vectors**



Probability information

Hidden Markov Model's probabilities:

- **initial probability** $i(s)$, where $s \in \mathcal{S}$
- **transition probability** $t(s_1, s_2)$, where $s_1, s_2 \in \mathcal{S}$
- **emission probability** $e(s, m)$, where $s \in \mathcal{S}, m \in \mathcal{M}$

Viterbi algorithm

Input: a sequence of emitted messages $m[1..n]$, $m[t] \in \mathcal{M}$
Output: a sequence of the most probable states $s[1..n]$, $s[t] \in \mathcal{S}$

```
 1: for $s \in \mathcal{S}$ do
 2:     $V[1, s] \leftarrow i(s) \cdot e(s, m[1])$
 3: end for
 4: for $t \in (2..n)$ do
 5:     for $s \in \mathcal{S}$ do
 6:         $V[t, s] \leftarrow \max_{p \in \mathcal{S}}\{V[t-1, p] \cdot t(p, s) \cdot e(s, m[t])\}$
 7:         $B[t, s] \leftarrow$ previous state $p \in \mathcal{S}$, that gave the best probability $V[t, s]$
 8:     end for
 9: end for
10: $highest \leftarrow \max_{p \in \mathcal{S}}\{V[n, p]\}$
11: return the best path obtained from B by backtracking
```

The implementation

- All **HMM parameters** were obtained from **a set of training elements** (sequences of objects with unknown labels and calculated feature vectors).
- **Initial** and **transition probabilities** were calculated directly from the training set.
- **Emission probability** can be calculated for any feature vector using a **decision tree** built from training elements' feature vectors.