Notes on Grobid

Joseph Boyd

Grobid (GeneRatiOn of BIbliographic Data) is a Java wrapper for applying conditional random fields (CRF) to the problem of metadata extraction of scientific papers. The wrapper sits atop a CRF engine that performs the abstract CRF computations. Wrapper and engine are combined through the use of the Java Native Interface (JNI). Grobid is divided into four packages: grobid-home, grobid-core, grobid-trainer, and grobid-server. Grobid produces a set of models, each addressing a different part of the metadata extraction, and each trained separately on its own spectrum of features. In the case of citation processing, the models are applied according to a hierarchy, starting with the segmentation of the reference list, followed by the tagging of each part of each reference, and finally a set of micro-models to further classify these reference parts.

0.1 Engines

Grobid allows for a choice between two alternative linear-chain CRF engines (both written in C++): CRF++ (developed by Taku Kudo) and Wapiti (developed by Thomas Lavergne). The default option is Wapiti, a more recent engine that allegedly performs better. Both engines follow similar input and output formats, allowing an easy abstraction within Grobid over the choice of engine. Though we have access to the source code, the CRF engine will be largely black-boxed in this project, and we will instead concentrate on tuning and configuring Grobid.

0.2 grobid-home

The grobid-home package houses the models, configuration files and additional software components such as pdftoxml, which is used to extract both plaintext and OCR data from the PDF inputs. This package does not contain any source code.

0.3 grobid-core

The grobid-core package contains the majority of the source code, including that for modelling documents and interacting with the CRF engine. grobid-core also supplies the main entry points for using Grobid as a command line tool. Through this interface, we may use Grobid to batch process collections of PDF files.

0.4 grobid-trainer

grobid-trainer contains the source code for coordinating the training of the models, including preparing the training data from the TEI inputs. grobid-trainer provides the main entry point for training models

and can also be used for training models through the command line.

0.5 grobid-server

grobid-server onsists of a web interface for Grobid, running on Jetty (Java) HTTP server. It allows a user to upload a PDF through a web form for processing. This feature is unlikely to be of any interest to this project.

0.6 Models

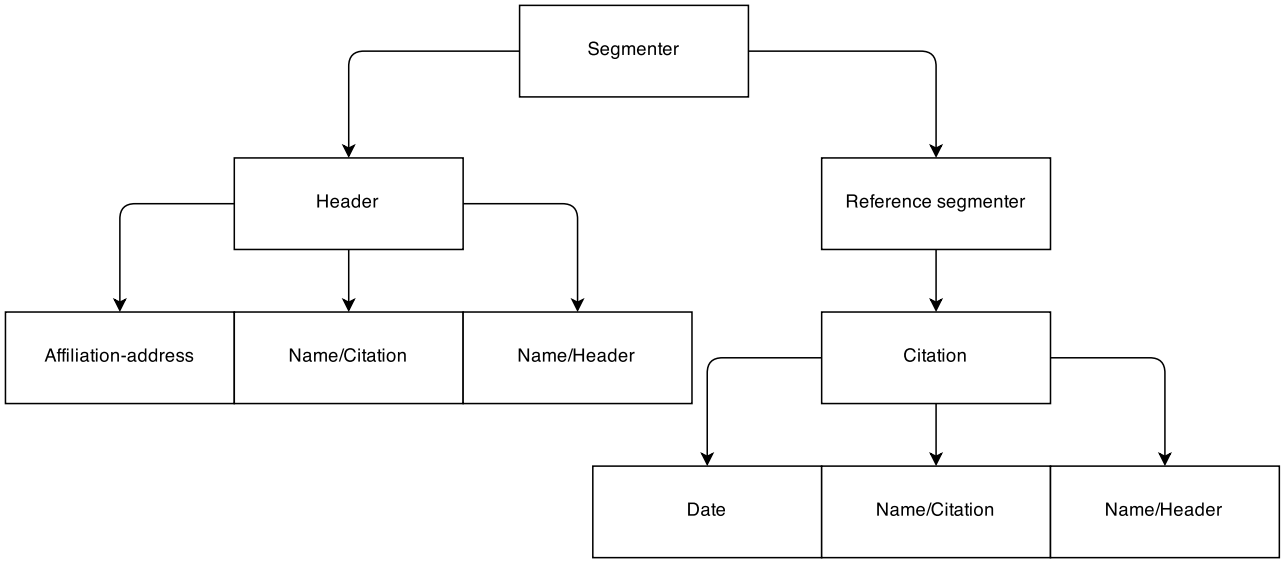

Figure 1: Cascade of models used by Grobid

Finding the header is typically easy, as we can assume it always to constitute the first page or so of the document. For this reason, when we call `processHeader` on its own, the Segmentation model is not called. On the other hand, though references are almost invariably at the end of an article, it is difficult to guess where it begins, and the reference model would suffer more from garbage than the header model. For this reason, the Segmentation model is called when we call `processReferences`, before calling the Reference-segmenter model to slice the reference list (identified by Segmentation) into individual citations.

Grobid manages the training, evaluation and usage of a set of models, each addressing a part of the information extraction. Higher level models such as the Header, Segmentation, and Reference Segmentation models may applied directly to PDFs, while the other, more specific models, such as the Date model, operate only on plaintext inputs. Though they have much in common, the models vary in the labels they assign, the features they exploit, and, due to the varying size of the vocabulary (compare say, the number of possible month names to the number of possible author names), the size (dimensionality) of the models. Calling `processFullText` runs all available models on a batch of PDF documents. The output of training is a model, which takes takes the form of a text or binary file, depending on the engine. These models are then "loaded" at prediction time for the labelling of new documents. Some models are in practice not used independently, rather, they form part of a "cascade" of models that progressively address finer subcomponents of the classification problem. As shown in Figure 1, reference extraction begins with segmentation models, which classify each line of a document, resulting in homogenous blocks of lines e.g.

```
[...]

January january J Ja Jan Janu y ry ary uary LINESTART INITCAP NODIGIT 0 0 1 NOPUNCT I-<month>
1994 1994 1 19 199 1994 4 94 994 1994 LINEEND NOCAPS ALLDIGIT 0 1 0 NOPUNCT I-<year>

July july J Ju Jul July y ly uly July LINESTART INITCAP NODIGIT 0 0 1 NOPUNCT I-<month>
1996 1996 1 19 199 1996 6 96 996 1996 LINEEND NOCAPS ALLDIGIT 0 1 0 NOPUNCT I-<year>

[...]
```

Figure 2: Training input for Wapiti

```
# Capitalization
U50:%x[0,11]
U51:%x[1,11]
U52:%x[-1,11]
U53:%x[0,11]/%x[1,11]
U54:%x[-1,11]/%x[0,11]
```

Figure 3: Feature templates for Wapiti

header, paragraph, figure, references. This information is then distributed to the other models, for example the Reference Segmentation model, which further breaks down the reference list into individual references. The Citation model then classifies the parts of each reference into classes, for example, date, affiliation, and author. Finally, the atomic subcomponents of these are classified by their respective models. Note that the citation branch of the hierarchy has the option of further cross-checking extracted references with the third-party CrossRef web service. Thus, the overall accuracy of the system is dependent on the combined accuracy of models.

0.7 Training

Both training and evaluation are performed on sets of XML documents following the Text Encoding Initiative (TEI) standard for representing electronic texts (http://www.tei-c.org/index.xml). This is also the output format for prediction, so there is consistency between input and output formats. It may appear paradoxical to *evaluate* on well-structured data, when the tool is ultimately intended to operate on unstructured PDF documents, that is, at prediction time. However, with closer inspection of the source code, an equivalence can be seen between:

1. applying an OCR tool (pdftoxml), tokenising the output and transforming to CRF input data
2. extracting tokens from TEI documents, and transforming to CRF input data

Both approaches yield the same input data for the CRF engine, and so evaluation is in fact equivalent to prediction, despite the initial difference in input formats. Figure 7 shows an excerpt from an input file to the CRF engine for training. These features are for inputs “January 1994” and “July 1996”, for training the Date model. The features range from token identity, to a variety of prefixes and punctuation features. It should be noted that OCR information is only used in higher level models, that is, the Header and Segmentation models. The input for lower-level models such as Date is plaintext, and so features are typically simple, but dictionary-based features, where information about a token is referenced in a

dictionary resource within Grobid, are also used. Note the features shown are only those pertaining to the token itself. The full range of features (including those involving concatenations of the token's neighbours etc.) are defined by a set of feature templates. The feature templates for each model are contained in a separate file. An excerpt of this is shown in Figure 3. These are given as a separate input to the CRF engine, and it is with these that the engine constructs all feature functions for the model. It is therefore vital that the feature extraction, which is generated by Grobid, is aligned with the template file, which is manually configured by the developer. As depicted in Figure 6, there is a strong coupling between these two parts of Grobid. The excerpt shown is from the Wapiti model, but the notation is the same for CRF++, which first standardised the syntax. This subset of five feature templates capture information about the capitalisation of a token and its neighbours. The notation has the structure, [identifier]:[%x][row, col], where row is the offset from the current token, and col indicates the feature index. Thus, "U50:%[0,11]", denotes that the feature template identified as "U50" takes the 11th feature for the current token (0 offset). This feature will be equal to 1 if a token is capitalised, and 0 otherwise. "U52:%[-1,11]" indicates the same thing, but based on the capitalisation of the *previous* token. "U54:%x[-1,11]/%x[0,11]" is a binary function for detecting the capitalisation of the current *and* the following token.

Now we may see an alignment with the mathematical model. Recall a linear chain CRF is expressed in the simplest case as,

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{y})}{\sum_{y'} p(\mathbf{x}, \mathbf{y}')}, \tag{1}$$

where,

$$p(\mathbf{x}, \mathbf{y}) = \exp\left\{ \sum_t \sum_{i,j \in S} \lambda_{ij} \mathbb{1}_{\{y_t = i\}} \mathbb{1}_{\{y_{t-1} = j\}} + \sum_t \sum_{i \in S} \sum_{o \in O} \mu_{io} \mathbb{1}_{\{y_t = i\}} \mathbb{1}_{\{x_t = o\}} \right\}, \tag{2}$$

Here $\mathbf{x}$ is a sequence of observations and $\mathbf{y}$ is a sequence of labels. S is the set of all labels, O is set of observations (the vocabulary of the tokens to be labelled). When the coefficients $\lambda_{ij} = \log p(y_t = i, y_{t-1} = j)$ and $\mu_{ij} = \log p(y_t = i, x_t = o)$, this joint distribution is equivalent to a Hidden Markov Model (HMM), with coefficients, λ_{ij} as transition probabilities and μ_{ij} emission probabilities. In this simple case, features are based solely on the token's identity, i.e. feature functions are an indicator function. For clarity, we may write,

$$p(\mathbf{x}, \mathbf{y}) = \exp\left\{ \sum_{i \in S} \sum_{j \in S} \lambda_{ij} F_{ij}(\mathbf{y}) + \sum_{i \in S} \sum_{o \in O} \mu_{io} F_{io}(\mathbf{x}, \mathbf{y}) \right\}, \tag{3}$$

where $F_{ij} = \sum_t \mathbb{1}_{\{y_t = i\}} \mathbb{1}_{\{y_{t-1} = j\}}$ and $F_{io} = \sum_t \mathbb{1}_{\{y_t = i\}} \mathbb{1}_{\{x_t = o\}}$. In a CRF, however, we may replace the indicator function for observations with any sort of function, typically binary, extracting rich features from a token. Thus, $F_{io} = \sum_t \mathbb{1}_{\{y_t = i\}} f_{io}(\mathbf{x})$. The set of functions, $\{f_{io}\}$, are the functions that we define in the feature template files. Note that, unlike an HMM, the vocabulary is not pre-defined, it is "discovered" through training on samples. Therefore, the number of actual features depends on the training set itself, whereas the feature template is fixed. Since we use indicator functions, which produce a feature for every observation, we may end up with an enormous number of features. Take the Date model for example: 5815 features are produced for a single block (not counting the one representing the label), and there are seven labels. As per our formulation in (3) we therefore have 7 * 7 "transition" features and 5815 * 7 "emission" features, totalling 40754 features. This is corroborated by the model output in Figure 4. Wapiti automatically constructs this vast feature space from the inputs we provide. In the Date model,

```
* Initialize the model
* Summary
    nb train:    493
    nb labels:   7
    nb blocks:   5816
    nb features: 40754
* Train the model with l-bfgs
  [   1] obj=1688,58    act=16482    err=25,80%/50,91% time=0,08s/0,08s
  [   2] obj=1221,30    act=15580    err=19,11%/35,50% time=0,05s/0,12s
  [   3] obj=922,15     act=13869    err=17,20%/33,67% time=0,04s/0,17s
  [   4] obj=638,04     act=10845    err= 6,53%/15,21% time=0,04s/0,20s
  [   5] obj=478,72     act=10582    err= 5,68%/13,59% time=0,04s/0,24s
  [   6] obj=416,15     act=9926     err= 3,77%/ 9,53% time=0,04s/0,28s
```

Figure 4: Output from training date model

```
8:u00:_x-3,
8:u01:_x-2,
8:u02:_x-1,
12:u03:November,
6:u04:19,
5:u05:,,
8:u06:2001,
17:u07:_x-1/November,
15:u08:November/19,
8:u09:19/,,
13:u0A:_x-2/_x-1,
8:u10:_x-2,
8:u11:_x-1,
```

Figure 5: Excerpt of model contents–the realisation of feature functions

the labels are I-<day>, I-<month>, I-<year>, I-<other>, <day>, <month>, and <other>. The I (probably) stands for "initial", as in training these are assigned to the first tokens of this class found in the string.

A model is typically a large file (as much as 100Mb). At the top of the file, the feature templates are declared, just as they are in the input. Because of this, that file is not required at prediction time. Following this the labels are declared. Then come two longer sections: first, the feature functions themselves as defined. Figure 5 shows the first 12 features produced from the first token in the first sample in the training set–"November". Because this is the first token in the string, we see the first three feature macros, which relate to the identity of the token's predecessors, remain unresolved. The fourth, however, shows the indicator for the token. This function will be true if a token is equal to "November". The fifth function is an indicator for if the token's successor is equal to "19", and so on. The final (and usually largest) section of the model file defines the non-zero weights for the feature functions. The weights are represented in scientific notation and in hexadecimal representation, presumably to avoid arithmetic underflow (a common problem when dealing with with the computation of HMMs and related models).

0.8 Evaluation

Training may be done with a split defined by the developer, which Grobid will use to set aside a proportion of the training data for evaluation. The evaluation of a model produced by training follows identical procedures, preparing the same input data. The output, however, is not a model but a the input data with labels. Grobid compares its predictions with the ground truth and outputs *precision*, *accuracy* and *F1 scores* as performance indicators, at the token, field, and instance levels. A token refers to a single contiguous string of characters (without spaces), a field is a block of contiguous tokens, and an instance is an entire document. Accuracy of an instance is therefore judged by the correctness of all tagging for the whole document, a difficult thing to achieve without any mistakes.

0.9 Prediction

Figure 6 shows the flow of information from input to output, as well as the relationship between training and prediction. When it comes to labelling (prediction), the starting point is a PDF document. With a third-party tool, pdftoxml, this is transformed into an XML file containing OCR information (font, style, orientation) for every token in the document. This information is stored in LayoutToken objects within Grobid. These tokens are arranged into blocks and features are extracted as they were for training and evaluation. Excepting the absence of tokens, the input is the same. The model created in the training phase is first loaded, and then the EngineTagger calls the CRF engine to label the inputs. Unlike for training, the feature template file is not required, as these have already been absorbed into the model file. After processing, Wapiti returns the same file with tags inserted. Grobid then further processes this information to transform it into the final TEI format.

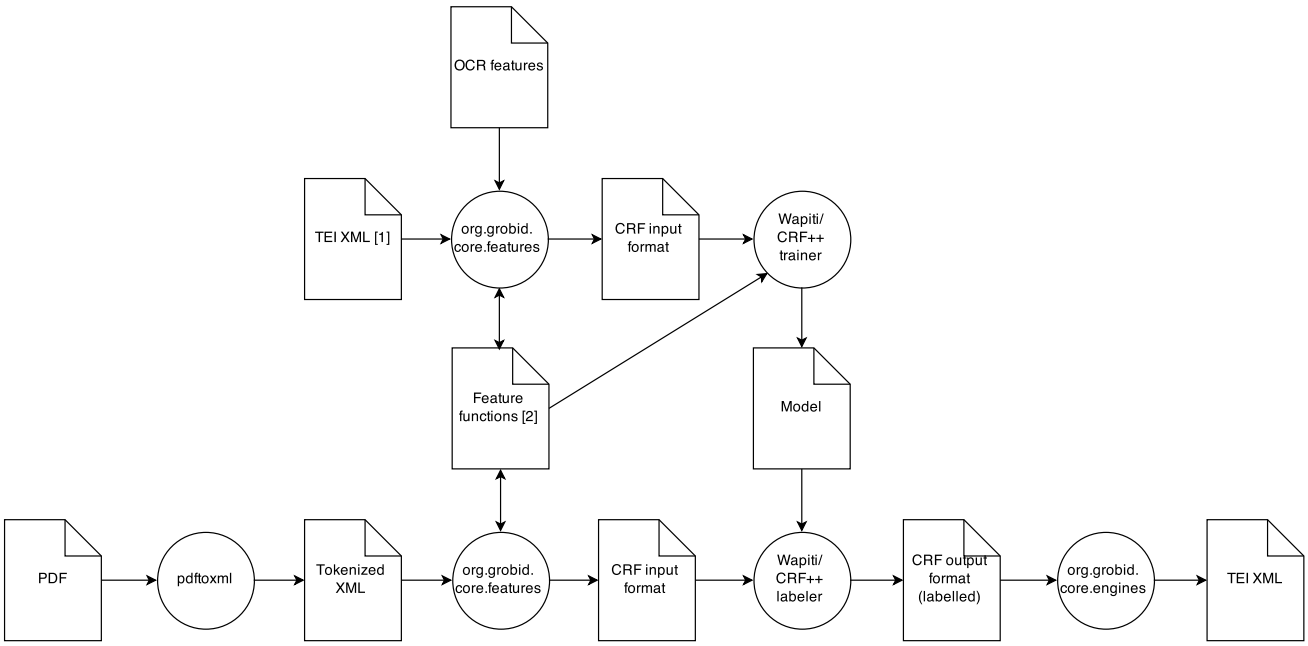

Figure 6: The flow of information and the interaction of training and prediction. [1] Training inputs follow and TEI standard. [2] There is a strong coupling between the features module and the feature functions file.

0.10 Other Functionality

In addition to the above, Grobid provides a means of producing training sets semi-automatically. This consists of applying the existing models on the training set to produce the XML inputs. Of course, each field must be checked against a ground truth and errors corrected before it is used as a training set. We intend to use this functionality to generate training data for benchmarking Grobid on HEP papers.

These methods are,

1. createTrainingHeader
2. createTrainingFulltext
3. createTrainingSegmentation
4. createTrainingReferenceSegmentation
5. createTrainingCitationPatent

Now, for example, when we run `createTrainingHeader` to create a header training set, grobid creates four files per pdf:

1. PDF-NAME.affiliation.tei.xml
2. PDF-NAME.authors.tei.xml
3. PDF-NAME.header.tei.xml
4. PDF-NAME.header

Files (1) and (2), are the training data for the affiliation-address and author models respectively. Files (3) and (4) are jointly required for training the Header model: file (3) simply contains the tagged data, but file (4) contains the feature (untagged) for each token, including OCR information. We must additionally store (4) as the OCR information cannot be derived from the plaintext TEI file. That the processing does not tag the blocks allows the programmer to intervens and correct and discrepancies with the ground truth. Thus the header files need only be corrected then moved to a separate folder for training. Notice training data for the segmentation models may be harder to generate automatically.

Notice that there is no method for creating citation training data, $\mathcal{D}_{tr}$ directly. This data is created as part of `createTrainingFulltext`. This creates four files also:

1. PDF-NAME.training.citations.authors.tei.xml
2. PDF-NAME.training.references.tei.xml
3. PDF-NAME.training.fulltext.tei.xml
4. PDF-NAME.training.fulltext

File (1) contains a refinement of the author names found in file (2), thus can be used as training data for the Name-citation model. File (2) contains the training data for the Citation model, in the format apparent in the training corpus. As should be expected, the name and date fields for this data are not resolved into their subcomponents, this task being left to the smaller models dedicated to these tasks. Files (3) and (4) are the files required for training the citation model.

Model	Labels
Header	<title>, <author>, <affiliation>, <reference>, <submission>, <abstract>, <address>, <keyword>, <degree>, <pubnum>, <email>, <date>, <copyright>, <intro>, <web>, <note>, <phone>, <dedication>, <entitle>, <grant>, <date-submission>
Affiliation address	<institution>, <other>, <settlement>, <department>, <postcode>, <contry>, <marker>, <region>, <addrLine>, <laboratory>, <postbox>, <other>, <null>
Name/Header	<forname>, <surname>, <marker>, <middlename>, <other>, <suffix>, <title>
Name/Citation	<surname>, <forname>, <other>, <middlename>
Citation	<journal>, <volume>, <other>, <issue>, <pages>, <date>, <author>, <title>, <booktitle>, <location>, <pubnum>, <note>, <publisher>, <editor>, <institution>, <tech>, <web>, <issue>
Date	<other>, <day>, <month>, <year>
Segmentation	<headnote>, <header>, <body>, <page>, <references>, <footnote>, <cover>, <acknowledgement>, <annex>
Reference-Segmenter	<label>, <reference>, <other>
Fulltext	<section>, <paragraph>, <citation_marker>, <other>, <table_marker>, <figure_marker>, <figure_head>, <trash>, <figDesc>, <equation>, <item>

Table 1: We have here excluded the Patent, Entities, and E-book models as these are experimental models not currently used by Grobid. Of the models listed, we will be focusing on Header, probably Segmentation, and perhaps Reference-Segmenter and Citation also. This is because these are likely to require special training and configuration for HEP papers. Others such as Name and Date are unlikely to improve with training on HEP papers.

```
<?xml version="1.0" encoding="UTF-8"?>
<tei xmlns="http://www.tei-c.org/ns/1.0" xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:mml="http://www.w3.org/1998/Math/MathML">
<teiHeader><fileDesc><sourceDesc><biblStruct><analytic>
<author>
<persName>
<forename>B</forename>.
<surname>GARRIGUES</surname>
</persName>
ET
<persName>
<forename>A</forename>.
<surname>MUNOZ</surname>
</persName>
</author>
</analytic></biblStruct></sourceDesc></fileDesc></teiHeader>
</tei>
```

Figure 7: Sample TEI (input and output) format for the Name-header model.

```
<bibl>
<label>[20]</label>
D. A. Hill, " Fields of horizontal currents located above the earth, " IEEE Transactions
on Geoscience and Remote Sensing, vol. 26, no. 6, pp. 726-732, 1988. [21]
A. M. Badescu, V. Savu, and O. Fratu, " Preliminary tests in Unirea salt mine (Slanic
Prahova, Romania), " Report 4, University Politehnica of Bucharest, Bucharest,
Romania, 2011.
</bibl>
<bibl>
<label>[22]</label>
M. Stefanescu, O. Dicea, and G. Tari, " Influence of extension and compression on
```

Figure 8: Sample error made by the (cora-trained) `reference-segmenter` model: the model fails to spot the citation label for reference [21], and the references are consequently merged. The `citation` model will then fail to match the ground truth.

0.11 Pipeline

0.12 Training - References

To prepare Grobid for practical reference extraction, our task is twofold, involving the training of both the `reference-segmenter` and `citation` models. The overall success of the citation model in tagging an entire reference list depends on the reliability of the citation blocks fed to it by its parent model. To illustrate, even a citation model that is perfect in its own right will not help very much in the reference extraction if the `reference-segmenter` above it is incapable of finding the references. Note that, to be precise, there is a further dependency on the accuracy of the `segmenter` model, the most high-level model in Grobid's arsenal, which is responsible for identifying the reference section for the before it is segmented, however we will assume temporarily that this is accurate enough in practice. Similarly, the `date` and `name` models are assumed to be sufficiently accurate, insofar as they are supplied with the correct text by the `citation` model, and so improvement of these will not be a focus of the project. Suffice to say, these models are unlikely to require special training on HEP papers. *Instance-level* accuracy for the citation model, measured over a whole document, will be the number of correctly tagged references *as a fraction of* the references correctly found by the `reference-segmenter`.

By contrast, the training of these two models may be performed independently. It does not matter to the citation model if the references in its training set are not contiguous members of the same reference list; the `citation` model operates at the reference string level. In the same way, it does not matter to the `date` or `name` models whether the dates or names they are training on or tagging belong to the same document. For this reason, creating our training set (as per the pipeline) for the citation model is simplified: if, when preparing the set, we notice a ground truth reference is missing from the output, we can just ignore this and continue. On the other hand, it matters very much that the `reference-segmenter` identifies the correct number of references, as this is the goal of the model, and it has a knock-on effect for the `citation` model below it. See for example Figure 8, showing an error made by the pre-trained (Cora) `reference-segmenter` model shipped with Grobid. It is rather surprising that such obviously strong indicators as the square brackets in the citation label, "[21]", are insufficient to classify it as such, especially when this error seems to happen rather often. This may be the result of poor feature engineering, a limited test set, or a mixture of the two. To see how citation model would tag the correctly segmented reference "[20]", see Figure 9.

If we are to correct the segmentation done when creating the reference segmenter training set, it is useful

```
<bibl><author>D. A. Hill</author>, &quot; <title level="a">Fields of horizontal
currents located above the earth</title>, &quot; <title level="j">IEEE Transactions
on Geoscience and Remote Sensing</title>, vol. <biblScope type="vol">26
</biblScope>, no. <biblScope type="issue">6</biblScope>, pp. <biblScope type="pp">
726âĂŞ732</biblScope>, <date>1988</date>. [<editor>21] A. M. Badescu, V. Savu, and
O. Fratu</editor>, &quot; <title level="a">Preliminary tests in Unirea salt mine</title>
(<pubPlace>Slanic Prahova, Romania</pubPlace>), &quot; <note type="report">
Report 4</note>, <orgName>University Politehnica of Bucharest</orgName>,
 <pubPlace>Bucharest, Romania</pubPlace>, <date>2011</date>.</bibl>
```

Figure 9: Citation tagging of the correctly identified reference number 20 as shown in Figure 8.

to note segmentation is done simply by placing the delimeters, i.e. the labels. If the delimeter is correctly placed, the entire reference is correctly segmented. We may therefore proceed by checking the delimeters. If one of them is missing, it indicates a problem such as shown in Figure 8, and further processing is required. Again, we are assuming the reliability of the OCR tool and the initial `segmentation` model to provide the `reference-segmenter` with the correct block of text as reference list.