**PDF Extraction**

**PDF Extraction Executable Summary:**

The current code uses the CERMINE JAR file to convert the pdf in to a Text file and an XML file, the xml follows the JATS-archivearticle1.dtd format.

The output extraction of the pdf follows the following structure:

article = {

        "Tittle" : string,

        "Year": integer,

        "ID": string

        "Author": [ List of dictionaries {

‘Name’ : string

‘Email’: string

‘Affiliations’ :[ List of affiliations ]

}],

        "Affiliations": {dictionary with all the affiliations present on the XML},

        "Abstract": string

    }

Currently all elements of the final data (except Abstract) are directly extracted from the initial XML produced by CERMINE, I have not done any cleaning and the outputs are not 100% perfect for all cases:  
  
**Tittle**: Works 97% of the time  
**Year**: Works 99%  
**ID**: Extracted when present  
**Author**: Noice is present in some cases with duplicates, affiliations are not always fully matched.  
**Affiliations**: I kept this option separate of the author, as in some cases not all the author affiliations are included, another issue I found is that it creates noise on the name of the affiliations:  
  
aff0: “University”  
aff1: “London”  
add2: “University of London”

**Abstract**

At the moment, I have 4 different methods to extract the abstract from the pdf, following a hierarchy of use as present next. With these methods applied on the PDF’s, from the 106 files, I can extract the abstract of 102 (not all 100% perfect but 90% yes), the remaining 4 don’t have an abstract on the text, on these cases the introduction or first paragraph is extracted.

After each method is used they follow an English test, this ensures that more than 60% of the abstract is English words, at the same time it checks for occurrences of really long words (abstract text gets corrupted). If any of this occurrences happens the code will it

**Method 1 – Element <abstract>**

The CERMINE XML, identifies the abstract and it creates an element <abstract> for it. This method identifies something as abstract 70% of the time, the text is accurate in most cases but sometimes is corrupted or noise is present at the start or end of the text.   
  
I have a applied a function to try to identify and remove noise, if the text is corrupted the code will not use it (English test)

The extraction is really straight forward, I look for <abstract> and If found, I extract the text. The text comes as one line with no breaks in between.  
  
 Advantage: This method is really good at identifying when is not Abstract tittle on the pdf, and the abstract is visually different from the main text in the pdf.  
  
Issue: It works when abstract is found as a tittle but in some cases it brings noise in to the abstract, either at the beginning or at the end. In a small number of cases it extracts different parts that are not consider part of the abstract.

**Method 2 – Parsing Cermine XML**file – XML produced by CERMINE

This method works 15-20% of the time, but is the most accurate when it does.   
  
for this method I look at the different sections in the body, the XML divides the text body in to multiple sections. The code will loop across each element of the section (<p>, <tittle>) until it finds any of the examples of word Abstract ("Abstract", "ABSTRACT", "TECHNICAL ABSTRACT", "A B S T R A C T", "Executive Summary").  
  
Once abstract is found, it will change the state of the method to abstract found and it will record the section where it found it.   
  
For this case, all the body of the abstract is on the same section number and on <p> elements, the code will stop adding to the abstract once:  
  
- finds an <tittle> element  
- changes section  
- finds any iteration of “Keywords”

Advantage: When the abstract is presented on this section of the XML it makes it really easy to extract and now when to stop. This method doesn’t add noise to the text.   
  
Issue: The XML is not formatted in this particular way most of the time.

**Method 3 – Text Extraction**

This method works with the text produced by the CERMINE file. This is the most labour-intensive method and the one with most constrains.  
  
This method returns an output 70% of the time. It only fails to return output when is no tittle for abstract, which it complements method 1 and 2 well.  
  
The loop starts by looking for any iteration of the word Abstract, then it sets the current state to abstract found and starts counting the number of sentences is iterating.   
  
From here is two mayors process the program will do:  
  
- Continue recording  
- Stop recording

**Continue recording:** this function, will try to exclude noise while recording without stopping to record. The Conditions at identifying noise:  
- finding @, email in the sentence  
- Excluding sentence with more than 4 numbers (for phone numbers)  
- Exclude names, when you have L. Simmons   
- Exclude sentence that starts with more than 2 numbers   
- Exclude sentences with "©", “#”  
  
**Stop recording:** This is the set of conditions that will make the program stop recording in the abstract. I kept a count on the number of sentences, as in some cases the order of the text gets move around and you can find “Kewords” just after the word Abstract. The stopping conditions I have:  
- if © is found on a sentence after 5 sentences recorded   
- if a sentence starts with \* after 5 sentences  
- if any form of “Keywords” is found after 5 sentences  
- if Fig. is found   
- if a sentence starts with any form of “Introduction”   
- If a sentence starts more than two words with only capital letters (NEW HEADING) after 5 sentences.  
- Set a maximum number of sentences to record (currently 50)

Advantage: This method seems to generate relatively ok results, is not full proof of noise particularly when the abstract is on two pages. It does well at removing most of the noise and stopping quiet well in most cases.

Issue: This method works perfectly about 70% of the time, in the other cases it can’t find abstract or the text contains noise. Most of the noise has been removed by our functions and it complements well when method 1 and 2 fail.

**Method 4 – Introduction/First Paragraph**

This method will be the last resource when Abstract is not present on the article, what the code will do, it will collect the first paragraph present on the XML file and stop when the section of the paragraph finishes.

As with the other sections, this will try to remove any noise present around the abstract text and stop before any new section is introduced.