Extracting Paragraphs from PDF Files

PDF Format

PDFs do not retain paragraph, heading, etc information. Instead, a PDF encodes a rendered form of a document, rather like the individual characters that are rendered on a screen. A PDF file can be thought of as containing a sequence of pieces of information. Each piece of text is located at a particular (x,y,z) position, along with font information.

Depending on the application writing the PDF file, a word may be added as a single word, or it may be added as several substrings. Some applications tend to add each of the characters of a word separately.

Space characters are not (usually) added to the file, as they don't add anything to the rendering.

pdfbox

PDFDocument uses *pdfbox* to do much of the work. pdfbox extracts the text elements from a PDF file, taking account of columns of text and pages. *pdfbox* works out where to add spaces, and is configured by *PDFDocument* to add a space at the end of each line.

Once pdfbox has extracted the text for the selected pages, *PDFDocument* then uses that to try and extract paragraphs, based on the location and font information provided.

Paragraph Extraction Heuristics

PDFDocument uses two heuristics to extract paragraphs. The result is far from perfect. To extract paragraphs in a complete and consistent way would be a extremely difficult (most probably impossible in general).

The heuristics used are as follows. A line is treated as starting a new paragraph:

- If the space between that line and the previous one is larger than a space threshhold (as defined below)
- If the font and font size is sufficiently smaller in the previous line. This is based on the start of current line and the end of the previous line.

The space threshhold is calculated as follows:

- *basicParagraphDrop* + max(previousLineHeight, currentLineHeight) * *heightSpaceFactor*
- The *basicParagraphDrop* and *heightSpaceFactor* may be configured, to take account of different spacings used in different PDFs.

Extraction Limitations

We use a simple approach that may be helpful when testing the textual contents of PDF files. However, minor formatting changes of a PDF may lead to different paragraphs being extracted, so it needs to be used with some care.

After experimenting with a range of approaches to try and improve the results, we've decided it's best to keep it simple. Example issues include:

1. The spacing between lines in a paragraph and between paragraphs may vary considerably.
2. A paragraph may flow from one column to the next, or one page to the next
3. Text may wrap around images
4. A heading may be obvious to the reader, but may not differ in terms of spacing or font from following text.
5. Headers, footers and footnotes.
6. A new paragraph being signalled by an indent on the first line, with no difference in line spacing.

Matching text in a PDF

Perhaps the simplest approach is to use pattern matching on the raw text. As this eliminates line break information, by adding a space at the end of each line, it means that you can ignore the details of the layout.

Some examples of this matching approach are shown in the storytest example that uses this PDF file as an example.

On the other hand, if you need to base matching on text relative to headings, you may be lucky with the current heuristics. You may need to customise the values of *basicParagraphDrop* and *heightSpaceFactor*, as explained above, to suit your particular document. However, we recommend that you don't spend too long on this, as it's likely to be a lost cause.

Conclusions

PDFDocument tries to make it easier to test the text in PDF files. It's rather rudimentary, due to the constraints that naturally arise with PDF, so the paragraph structure that is extracted is likely to be an approximation of the intended paragraph structure.