PDF

Contents

> Portable Document Format (PDF), standardized as ISO 32000, is a file format developed by Adobe in 1992 to present documents, including text formatting and images, in a manner independent of application software, hardware, and operating systems.

This covers how to load `PDF` documents into the Document format that we use downstream.

Using PyPDF

Load PDF using `pypdf` into array of documents, where each document contains the page content and metadata with `page` number.

```
!pip install pypdf
```

```
from langchain.document_loaders import PyPDFLoader

loader = PyPDFLoader("example_data/layout-parser-paper.pdf")
pages = loader.load_and_split()
```

```
pages[0]
```

An advantage of this approach is that documents can be retrieved with page numbers.

We want to use `OpenAIEmbeddings` so we have to get the OpenAI API Key.

```
import os
import getpass

os.environ['OPENAI_API_KEY'] = getpass.getpass('OpenAI API Key:')
```

```
from langchain.vectorstores import FAISS
from langchain.embeddings.openai import OpenAIEmbeddings

faiss_index = FAISS.from_documents(pages, OpenAIEmbeddings())
docs = faiss_index.similarity_search("How will the community be engaged?", k=2)
for doc in docs:
    print(str(doc.metadata["page"]) + ":", doc.page_content[:300])
```

Using MathPix

Inspired by Daniel Gross's

https://gist.github.com/danielgross/3ab4104e14faccc12b49200843adab21

```
from langchain.document_loaders import MathpixPDFLoader
```

```
loader = MathpixPDFLoader("example_data/layout-parser-paper.pdf")
```

```
data = loader.load()
```

Using Unstructured

```
from langchain.document_loaders import UnstructuredPDFLoader
```

```
loader = UnstructuredPDFLoader("example_data/layout-parser-paper.pdf")
```

```
data = loader.load()
```

Retain Elements

Under the hood, Unstructured creates different “elements” for different chunks of text. By default we combine those together, but you can easily keep that separation by specifying `mode="elements"`.

```
loader = UnstructuredPDFLoader("example_data/layout-parser-paper.pdf",
mode="elements")
```

```
data = loader.load()
```

```
data[0]
```

Fetching remote PDFs using Unstructured

This covers how to load online pdfs into a document format that we can use downstream. This can be used for various online pdf sites such as https://open.umn.edu/opentextbooks/textbooks/ and https://arxiv.org/archive/
Note: all other pdf loaders can also be used to fetch remote PDFs, but `OnlinePDFLoader` is a legacy function, and works specifically with `UnstructuredPDFLoader`.

```
from langchain.document_loaders import OnlinePDFLoader
```

```
loader = OnlinePDFLoader("https://arxiv.org/pdf/2302.03803.pdf")
```

```
data = loader.load()
```

```
print(data)
```

Using PyPDFium2

```
from langchain.document_loaders import PyPDFium2Loader
```

```
loader = PyPDFium2Loader("example_data/layout-parser-paper.pdf")
```

```
data = loader.load()
```

Using PDFMiner

```
from langchain.document_loaders import PDFMinerLoader
```

```
loader = PDFMinerLoader("example_data/layout-parser-paper.pdf")
```

```
data = loader.load()
```

Using PDFMiner to generate HTML text

This can be helpful for chunking texts semantically into sections as the output html content can be parsed via `BeautifulSoup` to get more structured and rich information about font size, page numbers, pdf headers/footers, etc.

```
from langchain.document_loaders import PDFMinerPDFasHTMLLoader
```

```
loader = PDFMinerPDFasHTMLLoader("example_data/layout-parser-paper.pdf")
```

```
data = loader.load()[0]   # entire pdf is loaded as a single Document
```

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(data.page_content,'html.parser')
content = soup.find_all('div')
```

```
import re
cur_fs = None
cur_text = ''
snippets = []   # first collect all snippets that have the same font size
for c in content:
    sp = c.find('span')
    if not sp:
        continue
    st = sp.get('style')
    if not st:
        continue
    fs = re.findall('font-size:(\d+)px',st)
    if not fs:
        continue
    fs = int(fs[0])
    if not cur_fs:
        cur_fs = fs
    if fs == cur_fs:
        cur_text += c.text
    else:
        snippets.append((cur_text,cur_fs))
        cur_fs = fs
        cur_text = c.text
snippets.append((cur_text,cur_fs))
# Note: The above logic is very straightforward. One can also add more
strategies such as removing duplicate snippets (as
# headers/footers in a PDF appear on multiple pages so if we find duplicatess
safe to assume that it is redundant info)
```

```
from langchain.docstore.document import Document
cur_idx = -1
semantic_snippets = []
# Assumption: headings have higher font size than their respective content
for s in snippets:
    # if current snippet's font size > previous section's heading => it is a
new heading
    if not semantic_snippets or s[1] >
semantic_snippets[cur_idx].metadata['heading_font']:
        metadata={'heading':s[0], 'content_font': 0, 'heading_font': s[1]}
        metadata.update(data.metadata)
        semantic_snippets.append(Document(page_content='',metadata=metadata))
        cur_idx += 1
        continue

    # if current snippet's font size <= previous section's content => content
belongs to the same section (one can also create
    # a tree like structure for sub sections if needed but that may require
some more thinking and may be data specific)
    if not semantic_snippets[cur_idx].metadata['content_font'] or s[1] <=
semantic_snippets[cur_idx].metadata['content_font']:
        semantic_snippets[cur_idx].page_content += s[0]
        semantic_snippets[cur_idx].metadata['content_font'] = max(s[1],
semantic_snippets[cur_idx].metadata['content_font'])
        continue

    # if current snippet's font size > previous section's content but less tha
previous section's heading than also make a new
    # section (e.g. title of a pdf will have the highest font size but we don't
want it to subsume all sections)
    metadata={'heading':s[0], 'content_font': 0, 'heading_font': s[1]}
    metadata.update(data.metadata)
    semantic_snippets.append(Document(page_content='',metadata=metadata))
    cur_idx += 1
```

```
semantic_snippets[4]
```

Using PyMuPDF

This is the fastest of the PDF parsing options, and contains detailed metadata about the PDF and its pages, as well as returns one document per page.

```
from langchain.document_loaders import PyMuPDFLoader
```

```
loader = PyMuPDFLoader("example_data/layout-parser-paper.pdf")
```

```
data = loader.load()
```

```
data[0]
```

Additionally, you can pass along any of the options from the PyMuPDF documentation as keyword arguments in the `load` call, and it will be pass along to the `get_text()` call.

PyPDF Directory

Load PDFs from directory

```
from langchain.document_loaders import PyPDFDirectoryLoader
```

```
loader = PyPDFDirectoryLoader("example_data/")
```

```
docs = loader.load()
```

Using pdfplumber

Like PyMuPDF, the output Documents contain detailed metadata about the PDF and its pages, and returns one document per page.

```
from langchain.document_loaders import PDFPlumberLoader
```

```
loader = PDFPlumberLoader("example_data/layout-parser-paper.pdf")
```

```
data = loader.load()
```

```
data[0]
```