PDF Table Extraction Tool

Extract tables from non-OCR PDFs and generate pandas DataFrames (stored on disk as CSVs)

What does it do?

1. OCR PDFs
2. Extract any tables as pandas DataFrames
3. Write CSVs of each table found

Dependencies

- **PyPDF2** (split PDFs by page)
- **GhostScript** (PDFs --> TIFFs)
- **Tesseract** (TIFFs --> OCR PDFs)
- **Tabula** (OCR PDFs --> DataFrames)
- **Java** (tabula dep)

Installation

In a Python 3.6 virtual environment:

```
pip install PyPDF2
pip install tabula-py
brew cask install java
brew install ghostscript
brew install tesseract
```

Optionally, see `/scripts/users/keng/cltools/pdf_tables` for my environments `python36_conda_virtual_env.yml` and `requirements.txt`:

```
# in python 3.6 venv
pip install -r requirements.txt

# in python 3.6 conda venv
conda env create --name NAME --file=python36_conda_virtual_env.yml
```

Simple Usage:

Run script on entire non-OCR PDF.

1. `./pdf_tables.sh non_ocr_input.pdf`

Detailed Usage:

Step by step execution (without reading).

1. `python script.py --pdf non_ocr_input.pdf`
2. `cd working/`
3. Remove any non-table PDF images (optional, but recommended for large PDFs, >400 pages)
4. `python script.py --gs`
5. `python script.py --tess`
6. `python script.py --csv`

Optional:

7. `mkdir csvs`
8. `mv *.csv csvs/`

Detailed Usage:

Step by step execution (with reading).

```
# chop up PDF per page
python pdf_table_to_dataframe.py --pdf input.pdf

# manually remove any non-table PDF files

# convert each non-OCR PDF to TIFF image, by looping over
# each file in the current working directory
python pdf_table_to_dataframe.py --gs

# run tesseract on each TIFF in current working directory,
# to generate OCR'd PDFs for each image.
python pdf_table_to_dataframe.py --tess

# read in each PDF with tabula, generate a pandas dataframe,
# and save the df as CSV to current working dir.
python pdf_table_to_dataframe.py --csv
```

Additional Usage: extract from normal PDF

Extract from normal PDFs, e.g. pubmed, scopus, JACS article that is already OCR'd (and is not a patent).

1. `python script.py --pdf non_ocr_input.pdf`
2. `cd working/`
3. `python script.py --csv-no-ocr`

You can also run `python pdf_table_to_dataframe.py --help` for a list of commands.

Additional Notes

Next Steps

- After tesseract completes, use PyPDF2 to stitch together all the OCR'd PDFs, then use tabula to select the specific pages a user wants to extract from.
- Tabula has a JSON output option, we could upload JSON data to Curator via the backend? From Curator interface, we could edit the values and proceed with verification.
- We could additionally upload the stitched-OCR PDF to Curator, on a new page, that provides searchable, highlightable text searching (use tesseract to PDF --> .txt). (not very useful?)
- We can directly export CSVs with tabula, skipping the `df.to_csv()` step.

Benefits

1. OCR'd PDFs are text-searchable so keywords can be searched via Finder.

Prototype Workflow: 01/04/2019

STEP 1

Chop PDF up, by page using `PyPDF2`

```
from PyPDF2 import PdfFileWriter, PdfFileReader

inputpdf = PdfFileReader(open("document.pdf", "rb"))

for i in range(inputpdf.numPages):
    output = PdfFileWriter()
    output.addPage(inputpdf.getPage(i))
    with open("document-page%s.pdf" % i, "wb") as outputStream:
        output.write(outputStream)
```

STEP 2

Manually remove non-table PDF files

STEP 3

Convert each non-OCR'd PDF into TIFF files using GhostScript: `gs`

```
gs -q -dNOPAUSE -dBATCH -sDEVICE=tiffg4  -r300 -sOutputFile="output.tif"
input.pdf
```

STEP 4

OCR each TIFF file using `tesseract` CL tool

```
tesseract input.tif output-ocr pdf
```

STEP 5

Read PDF table contents into DataFrame using `tabula`

```
from tabula import read_pdf

df = tabula.read_pdf('output-ocr.pdf').dropna()
```

STEP 6

Usage..

```
In [9]: df_c
Out[9]:
   Example Compound Potency
0        1        1       C
1        1        2       B
2        2        3       B
4        3        4       B
5        4        5       B
6        5        6       B
7        6        7       C
9        7        8       B
10       8        9       C
11       9       10       A
12       9      1 1       B
14       9       12       A
15      10       13       C
16      10       14       C
17      10       15       C
19     1 1       16       A

In [10]: c = '> 10'
In [11]: df_c.replace({'C':c})
Out[11]:
   Example Compound Potency
0        1        1    > 10
1        1        2       B
2        2        3       B
4        3        4       B
5        4        5       B
6        5        6       B
7        6        7    > 10
9        7        8       B
10       8        9    > 10
```

```
11       9       10       A
12       9      1 1       B
14       9       12       A
15      10       13    > 10
16      10       14    > 10
17      10       15    > 10
19     1 1       16       A

In [12]:
```