Notes and approaches to OCR

Rohan Alexander, John Tang, Diego Mamanche Castellanos

08 August 2020

```
## Warning: package 'reticulate' was built under R version 3.6.3
```

Introduction

A crucial aspect of the data science workflow is gathering data. It involves the tools and methods used to collect information in an established systematic fashion and identify the variables being measured. It also describes the methods used to obtain the data. Although the data collection component of research remains the same regardless of the field of study, the methods used vary among those fields of study as they have different interests. Moreover, the diversity of data types adds complexity to the process of gathering data. Structured data is organized in a highly regular manner or a pre-defined data model where the regularities apply to all the data in a particular dataset. Some examples are tables and relations. Semi-structured data contains this same information, but instead of having regular structures applied to all items in the dataset, the data might be interpreted with structural information. It can be supplied as tags e.g. name = "Bob" but also other markers to separate semantic elements and enforce hierarchies of records and fields within the data. Finally, Unstructured data, such as texts or images, holds information with no explicit structured data, such as tags. However, these tags may be assigned using manual or automatic techniques, converting the unstructured data to semi-structured data ($RobertM.Losee, 2005$)[1]. This diversity and complexity of data structures complicate, even more, the data gathering process as each data type may require a specific gathering approach(es) when needed.

The Portable Document Format (PDF) is an example of unstructured data created by the company Adobe in the 90s. It is widely used for documents. According to the $ISO32000-2:2017$[2], the latest edition of the PDF Reference (2.0), PDF enables users to exchange and view electronic documents easily and reliably, independent of the environment in which they were created or the environment in which they are viewed or printed. It is intended to fulfill the following requirements:

- preservation of document fidelity independent of the device, platform, and software,
- merging of content from diverse sources — Web sites, word processing and spreadsheet programs, scanned documents, photos, and graphics — into one self-contained document while maintaining the integrity of all original source documents,
- an extensible metadata model at the document and object level,
- collaborative editing of documents from multiple locations or platforms,
- digital signatures to certify authenticity,
- security and permissions to allow the creator to retain control of the document and associated rights,
- accessibility of content to those with disabilities,
- extraction and reuse of content for use with other file formats and applications, and
- electronic forms to gather and/or represent data within business systems.

To mention some of the features incorporated in this version we have:

- 12.10, "Geospatial features";

- 13.7, “Rich media” annotations;
- 14.7.4, “Namespaces” for tagged PDF;
- 14.9.6, “Pronunciation hints”;
- 14.12, “Document parts”;
- 14.13, “Associated files”;
- Support for PRC (see 13.6, “3D Artwork”);
- Support for UTF-8.

However, these advantages mean that the data in PDFs cannot be used for quantitative statistical analysis. The reason is that it needs measurable and verifiable data. But PDF documents, as it was mentioned previously, can have different objects, some of them non-static objects compressed in the same document. This situation makes it difficult to extract valuable information from the paper.

Sometimes just by copying and pasting the information directly from the PDF is possible when it contains simple texts or regular tables. In that case, there is no barrier in the process of extracting data. But other times it is not as easy if it is an image, a survey form, or complex shapes containing relevant information, . Furthermore, there is a linguistic component that adds more complexity. For instance, it is well known that non-Latin characters such as Japanese, Chinese, Arabic, among others, generate several difficulties in different stages of the text mining process. It is because those characters have complicated structures, a huge number of categories, and resemblance among characters, font unevenness, or writing styles. In those cases, the need for reliable tools or methods with the ability to elicit data becomes extremely important.

Optical character recognition (OCR), a process that transforms a bitmapped image of printed or handwritten text into text code, and thereby making it machine-readable, has been widely used for scientists trying to capture the images of characters and texts back in the 50s. First, by mechanical and optical means of rotating disks and photomultiplier flying spot scanner with a cathode ray tube lens, followed by photocells and arrays of them. The OCR process was slow, and one line of characters could be digitized at a time. Nowadays, in OCR, once a printed or handwritten text has been captured optically by a scanner or some other optical means, the digital image goes through the following stages of a computer recognition system $(Cheriet, Mohamed; Kharma, 2007)$[3]:

- The preprocessing stage that enhances the quality of the input image and locates the data of interest.
- The feature extraction stage that captures the distinctive characteristics of the digitized characters for recognition.
- The classification stage that processes the feature vectors to identify the characters and words.

In this paper we review various OCR options for researchers who need to gather data from PDFs in which the text must be extracted to be reused, edited, or reformatted; the text should be available for full-text information retrieval; the text is to be coded in HTML or SGML; the text should be available to adaptive equipment for the visually impaired; the file size is of concern (in terms of storage or bandwidth to transmit); or the resources are available to perform OCR and correct the output.

When they consider a primer on what OCR options exist these days depending on your technical ability and then how they perform for an easy example and then how they perform for a hard example (the Japanese extract) and finally some suggestions for what is needed in the future....

Our paper represents...

The remainder of this paper is structured as follows...

Data

Test Samples

Japanese dictionaries…

OCR options

The following list contains some of the OCR APIs and libraries available that will be evaluated in this study:

OCR Tool	Company	Type	Monthly Price Aug/2020	Tech Requierements
Azure Cognitive Services (OCR)	Microsoft	API	Free Web Container: 5000/Free; S1 Web Container: 0-1M/$1.50 per 1,000 trans, 1M-5M/$1 per 1,000 trans, 5M-10M/$0.65 per 1,000 trans, 10M-100M/$0.65 per 1,000 trans, +100M/$0.65 per 1,000 trans	

OCR Tool	Company	Type	Monthly Price Aug/2020	Tech Requierements
Amazon Rekognition	Amazon	API	First 1 million images: $0.001 per image Next 9 million images $0.0008 per image Next 90 million images $0.0006 per image Over 100 million images $0.0004 per image	
Tesseract		R library	Free	
PyTesseract (Wrapper Google's OCR Engine)		Python Library	Free	
OpenCV		Python Library	Free	
Watson Visual Recognition	IBM	API	Lite: 1000/Free, Standard: $0.002 per trans	
Kraken (Linux OS)		Python Library	Free	
Google Vision (for Text recognition)	Google	API	1000/Free, 5 mill/$1.5, >5mill/$0.6	
SwiftOCR (iOS)	Apple	Swift Library		

OCR Tool	Company	Type	Monthly Price Aug/2020	Tech Requierements
Amazon Textract	Amazon	API	First 1 Million pages: $0.015 - $0.05 per page, Over 1 Million pages: $0.01 - $0.04 per page	
Tensorflow	Google	Python Library	Free	

Azure Corgnitive Services - Computer Vision OCR

Among the Microsoft Azure's portfolio, Azure Cognitive Services offers Computer Vision, an API that processes images and returns information based on the visual features the user is interested in. Those services comprise object detection of an image, visual features tagging, image categorization, Optical Character Recognition (OCR), among *others*[4]. The OCR API processes an image and returns the language of the document, orientation, regions, lines, and finally, the characters.

In python we need several libraries for calling the API.

```
import os
import sys
import requests
# If you are using a Jupyter notebook, uncomment the following line.
#%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from PIL import Image
from io import BytesIO
import matplotlib
import matplotlib.font_manager as font_manager
from os import path
import ast
```

Then, we need to create headers and parameters necessary for calling the API. The library **requests** is commonly used in python to do that. The method **post** is needed as the API uses the HTTP method POST.

```
#Create the URL for the API call. The endpoint corresponds to the Azure's
# endpoint of the account.
ocr_url = endpoint + "vision/v3.0/ocr"
```

```
#Create the header using the Azure's subscription key.
headers = {'Ocp-Apim-Subscription-Key': subscription_key}

#In params language': 'ja' for japanese only. 'unk' means self detection
params = {'language': 'unk', 'detectOrientation': 'true'}
data = {'url': image_url}

#Call and save the response using the method post from the library request
response = requests.post(ocr_url, headers=headers, params=params, json=data)
```

Here we have an example of the response:

```
{'language': 'ja',
 'textAngle': 0.0,
 'orientation': 'Up',
 'regions': [{'boundingBox': '31,30,1521,2721',
   'lines': [{'boundingBox': '730,30,44,2718',
     'words': [{'boundingBox': '735,30,34,34', 'text': ' '},
      {'boundingBox': '736,71,34,26', 'text': ' '},
      {'boundingBox': '749,100,6,11', 'text': ' '},
      {'boundingBox': '735,116,34,27', 'text': ' '},
      {'boundingBox': '736,149,33,33', 'text': ' '},
```

Language and Orientation

Figure 1 presents a sample that corresponds to a Japanese text, but it also contains English text in some fragments of the text. The API offers the possibility to do self-detection, which in this case is helpful, but also the opportunity to specify the language in advance. Moreover, the orientation can also be self-detected or not.

Figure 1

Regions

The blue line figure 2 denotes the segment from the image that includes the characters. In this example, there is only one blue line because the document is only text. In other cases, it may contain images requiring more than one region.

Figure 2

Lines

Once the region is defined, the API provides in figure 3 (red rectangles) the subsets of characters of the region called Lines. From the example, we can see that some characters are not included in any of the lines. Those characters are not recognized by the API.

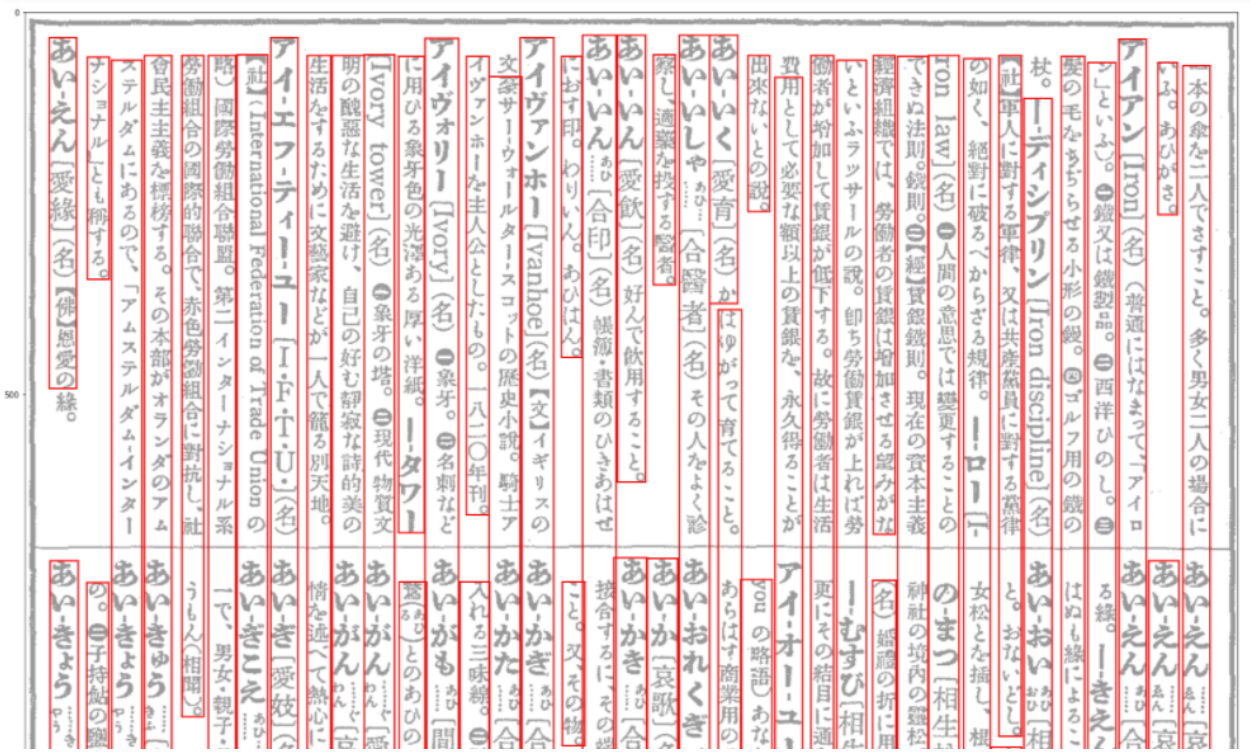

Figure 3

The Character Boundaries

After the lines are identified, the API provides in figure 4 each recognized character bounding boxes. Those without a yellow square were not recognized by the API in this example.

Figure 4

Characters

Once the bounding boxes are located, each character takes its corresponding place, as it is shown in figure 5.

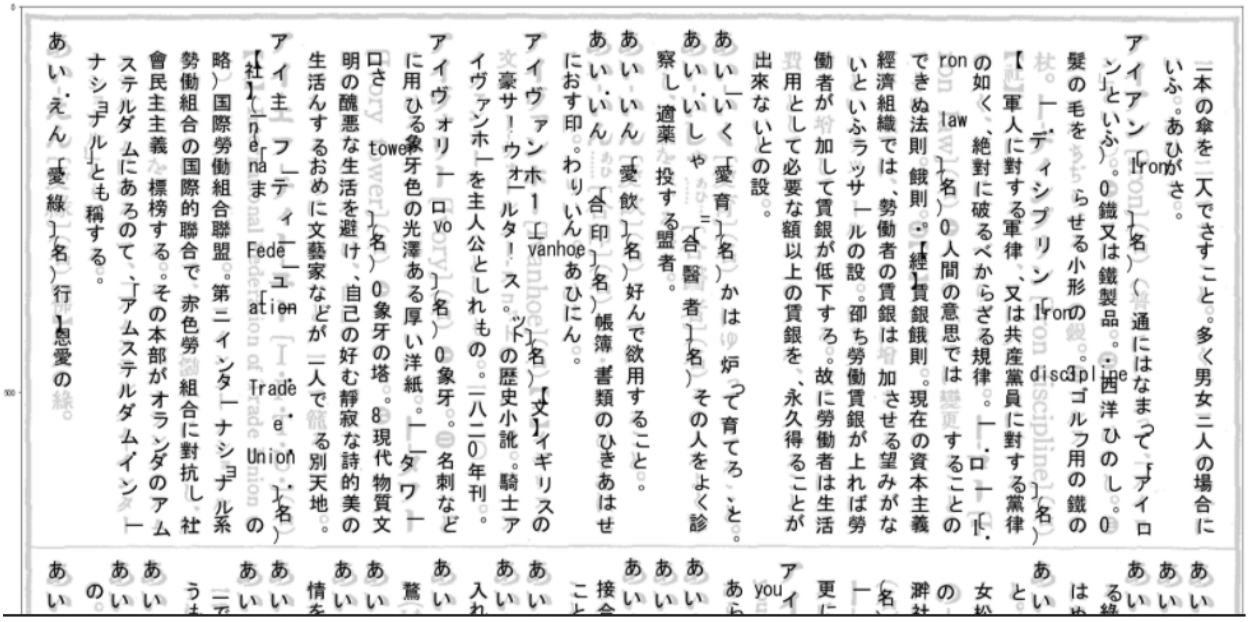

Figure 5

Advantages and Disadvantages

Advantages The API is easy to use after reading the documentation. It offers an example of how to use it in different programming languages. It also offers the possibility to self-detect languages and the orientation of the document. The response is clear, and extract the information from the JSON file is not complicated.

Disadvantages Even though the API offers the possibility to self-detect the language, It changes the order in which the characters should be in the output. In figures 6 we can see that the first line provided by the self-detection approach is located in the middle. This situation does not allow a correct interpretation of the text because it does not follow the reading pattern. In this case, it is a vertical orientation, which means we have to read it from right to left.

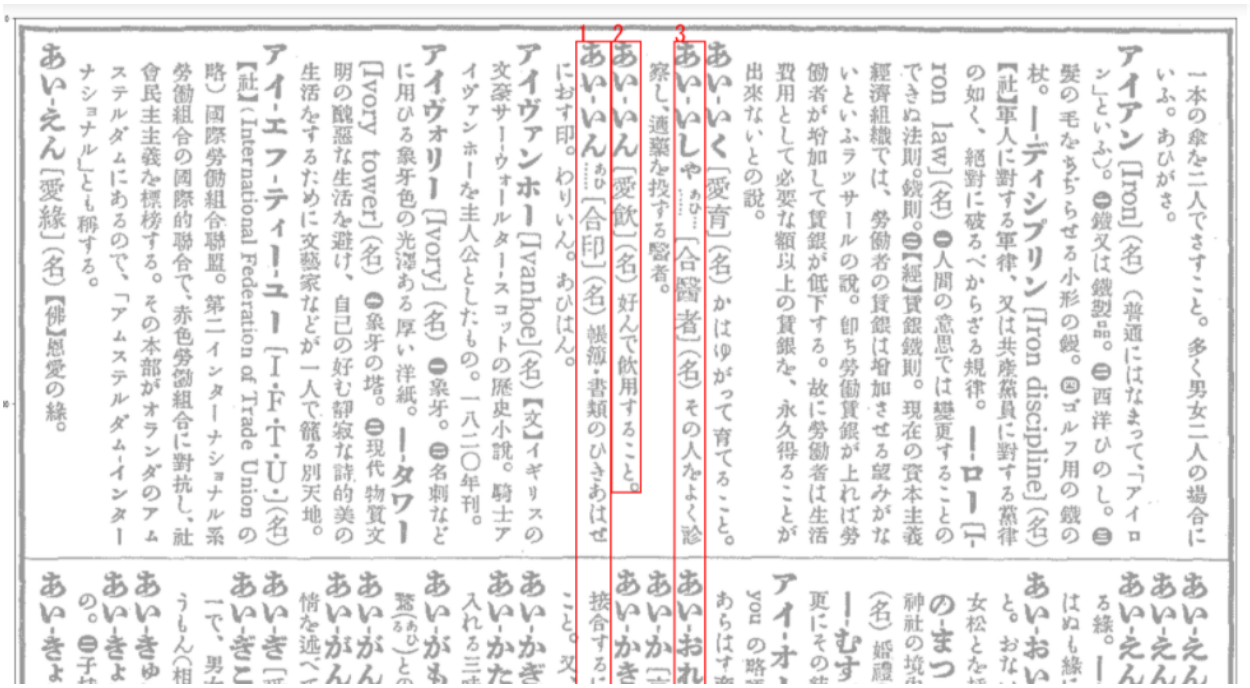

Figure 6

When the parameter language is stated as Japanese since the beginning, as we can see in figure 7, it recognizes the first line correctly.

Figure 7

In general, self-detection and the case with the language parameter as the Japanese do not generate the boundaries correctly. In Figures 6 and 7, we can observe that both cross the line, which is not correct for in this document. There is a line delimiting the boundaries of the text, but it has been ignored by both API calls. This situation makes it difficult for researchers to organize the information correctly as the order of the characters matter.

Future Analysis

By doing some image preprocessing before the API call, it might be possible to see improvements in terms of boundary identification….

PyTesseract - Python Library

Evaluation

For each option, we rank them based on two scales: results and difficulty. We need to work out a definition for each of these.

Diego's comment: From what I learned so far, it could be easiness (an API is more straightforward than a library), number of characters correctly recognized, flexibility (the opposite of easiness), and cost. Moreover, some APIs such as Google's groups more than one character, whereas Azure returns only one character per boundary.

The main aspect of the end product will be a graph with two axis, and dots for where each service is positioned, coloured or faceted by the test.

Discussion

References

1: https://www.researchgate.net/publication/267465115_An_Overview_and_Applications_of_Optical_Character_Recognition

2: https://books-scholarsportal-info.myaccess.library.utoronto.ca/en/read?id=/ebooks/ebooks2/wiley/2011-12-13/1/9780470176535

3:

4: https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home

5: https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/quickstarts/python-print-text