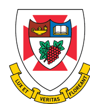
Access to Information and Optical Character Recognition (OCR): A Step-by-Step Guide to Tesseract

Part one of the CAIJ Computer Literacy Series

Alex Luscombe, Kevin Dick, Jamie Duncan, & Kevin Walby

June 3, 2020



THE UNIVERSITY OF WINNIPEG

Access to Information and Optical Character Recognition (OCR): A Step-by-Step Guide to Tesseract

CAIJ
Centre for Access to Information and Justice

THE UNIVERSITY OF WINNIPEG

Recommended citation: Luscombe, Alex, Kevin Dick, Jamie Duncan, and Kevin Walby. 2020. *Access to Information and Optical Character Recognition (OCR): A Step-by-Step Guide to Tesseract*. Winnipeg, MB: Centre for Access to Information and Justice.

Report design: Alex Luscombe.

Cover photo: ©Michael Dziedzic/Unsplash.

Please direct inquiries to:
Centre for Access to Information and Justice
University of Winnipeg
Department of Criminal Justice
Centennial Hall, 3rd Floor
515 Portage Avenue
Winnipeg, Manitoba
Canada R3B 2E9

www.uwinnipeg.ca/caij

Contents

About the Authors 2

About the CAIJ 3

Executive Summary 4

Introduction 5

Making Computational Social Science Accessible 5

The Pros and Cons of Tesseract OCR 6

Tutorial (MacOS/Linux only) 7

Moving Ahead 12

Appendix 15

About the Authors

Alex Luscombe is a PhD student at the University of Toronto's Centre for Criminology, a Junior Fellow at Massey College, and a CAIJ Researcher.

Kevin Dick is a PhD candidate and Queen Elizabeth II Scholar at Carleton University's Department of Systems and Computer Engineering and a CAIJ Researcher.

Jamie Duncan is an affiliate of the Ethics of AI Lab at the University of Toronto's Centre for Ethics and a CAIJ Researcher.

Kevin Walby is Associate Professor of Criminal Justice at the University of Winnipeg and the CAIJ Director.

The authors thank Marcus Sibley, Fernando Avila, and Andrea Sterling for their help in testing and improving the usability of the tutorial.

About the CAIJ

The Centre for Access to Information and Justice (CAIJ) at the University of Winnipeg aims to be a leading international collaboratory for public interest research on matters of freedom of information (FOI) and access to justice in Canada and beyond. Through local and international collaborative projects, the CAIJ promotes a multi-disciplinary and critical approach to research and policy engagement. The CAIJ investigates government practices, tracks general trends in FOI and access to justice, as well as charts national and regional variations in these practices. The CAIJ advances theoretical, empirical, and policy-oriented studies of FOI and access to justice in the form of workshops, reports, articles, and books produced by its members.

The CAIJ's mission and goals include:

- Advancing knowledge of FOI and access to justice practices through multi-disciplinary and critical collaborative research projects;
- Organizing knowledge mobilization and research-driven working groups, workshops, seminars, training, and conferences on FOI and access to justice;
- Serving as a welcoming and enabling context for students and visiting scholars working in the areas of FOI and access to justice in Canada and beyond;
- Engaging in outreach with a community and public interest focus.

For more information, please visit the Centre's website.

Executive Summary

It is a perennial problem in Canada that municipal, provincial, and federal government agencies disclose records under Access to Information (ATI) / Freedom of Information (FOI) law in non-machine readable (image) format by default. The same problem regularly emerges in historical and archival research. The inability to machine read these texts limits the analytic techniques that may be applied. It is also a barrier to access. Fortunately, there exist a number of free and open-source solutions to this problem. In the field of computer science, transforming scanned images into machine readable text is considered to be a "solved" problem. One state-of-the-art solution is the *Tesseract Optical Character Recognition* (OCR) engine, which is considered to be one of the best OCR engines available. This report will teach you how to use Tesseract OCR, which is made easily accessible with some simple Python code. Our larger goal is to improve access to open-source tools that can eliminate barriers to accessing information. The ability to convert a document into a format that can be searched for keywords, phrases, and possibly studied using natural language processing (NLP) or corpus linguistic methods alongside more traditional qualitative ones promises to revolutionize social science research. We hope this tool will help ATI/FOI system users as well as historians and archivists render their files more accessible. The discoverability of texts is a crucial element of access to information.

Link to full tutorial: click here

Link to companion video: click here

Introduction

It is a longstanding practice in Canada for municipal, provincial, and federal government agencies to disclose records under Access to Information (ATI) / Freedom of Information (FOI) law in non-machine readable (image) formats. Lengthy reports, emails, and excel files are often printed and scanned by access coordinators before they are released to the requester. Typically, if a user requests records in digital form, they receive .pdfs of scanned record copies via email or on a data stick or CD ROM. Coordinators may be willing to release the data in a "raw" format, however, this is not always the case, and inexperienced requesters may not even realize that this is something they can ask for (indeed, they may not even be aware that the files they have requested are coming in image format before it is too late).

The inability to search and analyze texts with a computer limits the analytic techniques that may be applied to them, presenting barriers to access. Access to Information Officers often "over produce" when processing requests by including mounds of irrelevant text as part of one's disclosure package. Manually sifting through thousands of pages of image format documents disclosed under ATI/FOI in search of one or two lines or key words becomes the equivalent of finding a needle in a haystack.

Fortunately, there are several free and open-source solutions to this problem. In the field of computer science, transforming scanned images into machine readable text is widely considered to be a "solved" problem. One state-of-the-art solution is the *Tesseract Optical Character Recognition* (OCR) engine, considered to be one of the best OCR engines available.

This report will walk you through how to use Tesseract OCR, which we have made easily accessible with some simple Python code. It is part of a larger series of projects we intend to launch to promote computer literacy and the accessibility of computational tools and methods among non-computer scientists. In doing so, we hope to improve access to open-source tools that can eliminate barriers to accessing information. The ability to convert a .pdf document of any size into a format that can be searched for keywords, phrases, and studied using natural language processing (NLP) or corpus linguistic methods alongside more traditional qualitative ones promises to revolutionize social science research. Before the tutorial, we discuss how this tool might help researchers and other ATI/FOI users integrate computational methods into their work.

Making Computational Social Science Accessible

Following digital humanities scholars, social sciences researchers are increasingly adopting computational methods to complement traditional disciplinary approaches. Computational methods have now been applied in disciplines from sociology, to criminology, political science, geography, media studies, and beyond (van Atteveldt & Peng, 2018; Lucas et al., 2015; Torrens, 2010). Some entail searching for and analyzing big data – a term simply defined as the processing of massive data sets (Kitchin & McArdle, 2016). More complex definitions incorporate consideration of the extent to which data can be collected and processed in real time (velocity), the incorporation of data that is "structured, semi-structured, and unstructured" (variety), its capacity to "point to" or pair with other data (indexicality and relationality), and its ability to describe holistic systemic processes with a high level of detail (exhaustivity and resolution) (Kitchin & McArdle, 2016, 1).

Reflexively applying technology to improve access to previously inaccessible texts can open up social science to new innovative, analytic techniques and approaches (Kitchin, 2017). Drawing from and contributing to the field of computational social science, this report provides one such tool. Optical Character Recognition (OCR) is a technique for converting texts to a digital, machine-readable form. The typical assumption among non-technical researchers is that only proprietary software tools for OCR are accessible to them.

We demonstrate how open-source OCR alternatives can be equally if not more effective (not to mention free). In the context of ATI/FOI, computational methods can help researchers efficiently surface patterns and relationships within large repositories of textual records of varied types, structures, and sources.

Corpus linguistics, which emerged in the late 1950s, is arguably the first form of computational text analysis (McEnery & Hardie, 2013). At a high level, corpus analysis involves assembling a body (corpus) of machine-readable text to answer a specific research question. The corpus is analyzed (and in many cases annotated) according to frequency (how common particular words or meanings are) and concordances (the context in which specific words are used) (McEnery & Hardie, 2011). While these discussions are not entirely new in the social sciences (Leetaru, 2012; Franzosi & Roberto, 2004), the technologies and techniques available to researchers have become significantly more advanced in the past decade.

Lewis et al. (2013) argue in favour of blending computational and traditional social science methods depending on the research question. Traditional analytic methods using 'small data' remain useful in the big data world (Kitchin & Lauriault, 2015; Mason et al., 2014). As Jemielniak (2020) argues, researchers collecting and analyzing big data should incorporate and describe nuanced contexts and relationships, something that combining computational and qualitative methods can help with.

There is no standard approach to computational methods or automated text analysis, so one needs to constantly validate and adapt their approach to coding (Grimmer & Stewart, 2013). Advanced computational methods have begun to allow researchers to go far beyond counting words and analyzing their immediate surroundings in a text (Wiedemann, 2013, 343). For example, natural language processing (NLP) techniques like Latent Dirichlet Allocation (LDA) Topic Modeling can classify and sort texts according to abstract algorithmically generated topics (Jacobi et al., 2016). However, there remains much work to be done. Longstanding epistemological issues such as criteria for research quality are as crucial as ever.

In order for the texts we obtain to be analyzed using computational methods, they need to be machine readable. The means of achieving this is OCR. We designed this OCR tool and step-by-step tutorial with two types of non-technical users in mind, although its utility is by no means limited to them. The first group is users of ATI/FOI systems, which allow citizens by law to request documents and data held by their governments (and in many cases foreign governments as well). In Canada, these records are often voluminous, and the records are usually not provided in machine readable formats compatible with computational methods. Users of ATI/FOI systems often lack these tools. The other group we envision finding this guide useful are historians and archivists. While these users may already have digital tools available to them, this typically comes in the form of costly, proprietary software.

The Pros and Cons of Tesseract OCR

Tesseract OCR is completely free. Proprietary OCR software by contrast can be quite expensive. There is also no restriction to how many documents you can process using Tesseract OCR, whereas many proprietary tools will limit the number of pages you can process (often until you pay more). Depending on the software you purchase, it may also be less effective than Tesseract, which is recognized as one of the best OCR engines available. Finally, by using an open source tool we generate transparent, reproducible code that can be shared with others or adapted for future purposes.

Like many computational tools, Tesseract is highly effective but it is not perfect. ATI/FOI disclosures are typically printed and scanned copies of records so depending on their quality, not all characters may be properly recognized (though you may be surprised how effective it really is). Additionally, ATI/FOI disclosures generally contain redactions in the form of white, grey, or black boxes covering undisclosed information. Finally, complex character layouts (like in some tables) may not come out logically ordered relative to the original record, a limitation of any OCR engine, not just Tesseract.

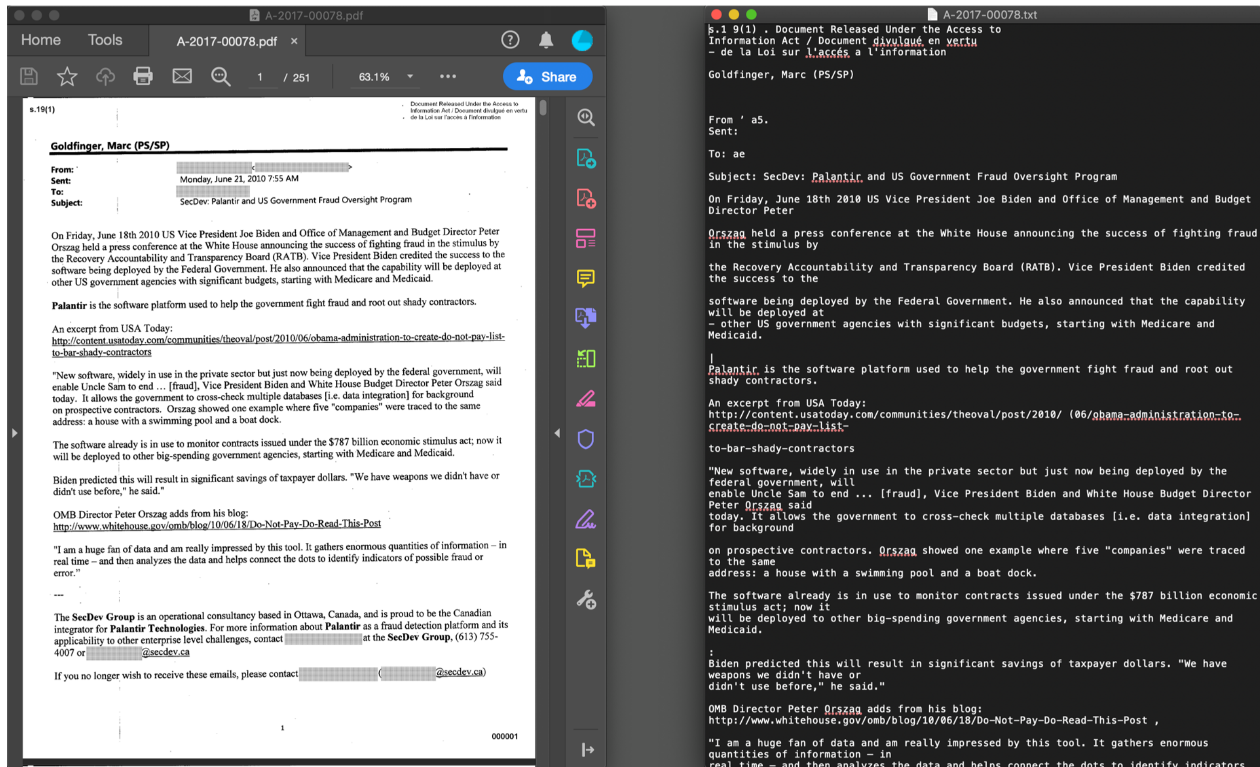


Figure 1: Original .pdf (left) compared to .txt file output post OCR processing (right)

Depending on the quality, layout, and format of the ATI/FOI records, it is likely that some degree of document "cleaning" will be required after processing them. Cleaning can be effectively streamlined by following what computer scientists call the "human-in-the-loop" paradigm, which we plan to cover in a future tutorial.

Tutorial (MacOS/Linux only)

This tutorial will walk you through how to render your scanned, image-based documents .pdf documents into a machine-readable, text-based format (figure 1). Throughout, we encourage you to follow along with our (companion how-to video on YouTube). The tutorial is designed to build skills for the use of open-source software to improve access to information generally. We cover the high-level steps required to convert a large scanned .pdf format document into a machine-readable and searchable .txt format. We recommend beginning by following the steps using the sample .pdf file provided in the "Sample-Files" folder in the repository.

Step one: download the repository

Download the repository from the CAIJ GitHub page, unzip the folder, and save it locally on your computer's hard drive (figure 2).

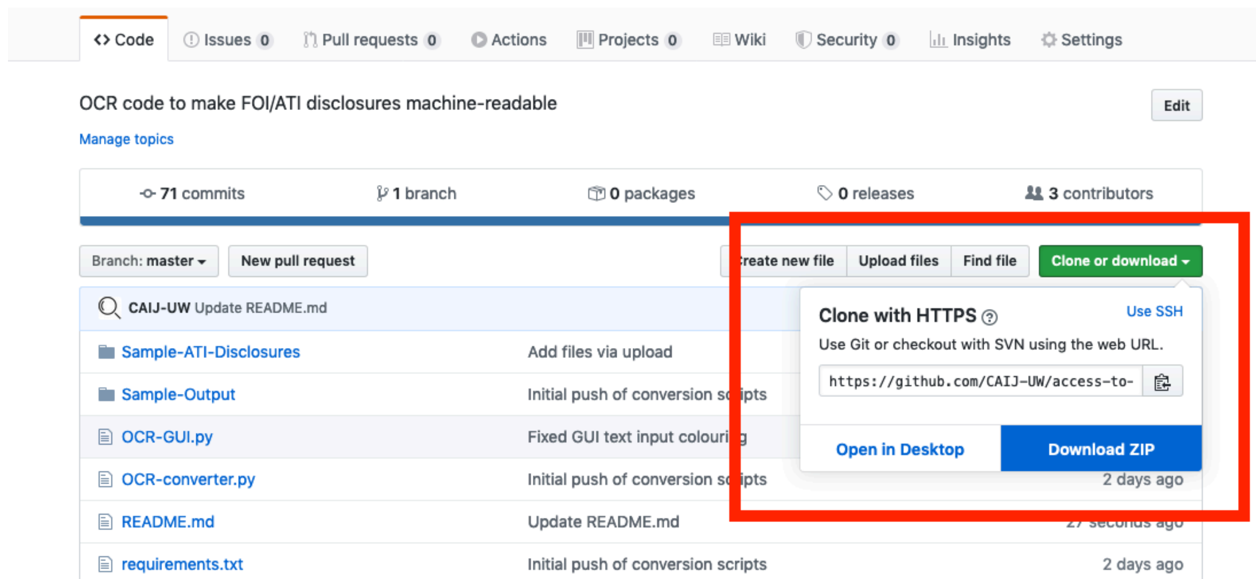


Figure 2: Screenshot of downloading the full repository from GitHub

Step two: open your computer's command prompt

Open your computer's command prompt. On MacOS, this is called the Terminal. To open the Terminal, simply press Command + Space and type the word "Terminal" in the search bar. Double click the Terminal application listed under Top Hit to open it.

Step three: install Homebrew

If you already have Homebrew installed on your computer, skip this step.

If you unsure if you have homebrew installed, type the following line into your computer's Terminal:

```
brew help
```

If it returns "command not found", you do not have homebrew installed on your computer.

You can download Homebrew by entering the following command into your computer's Terminal:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
install/master/install.sh)"
```

Step four: install Ghostscript using Homebrew

If you already have Ghostscript installed on your computer, skip this step.

If you unsure if you have Ghostscript installed, type the following line into your computer's Terminal:

```
ghostscript help
```

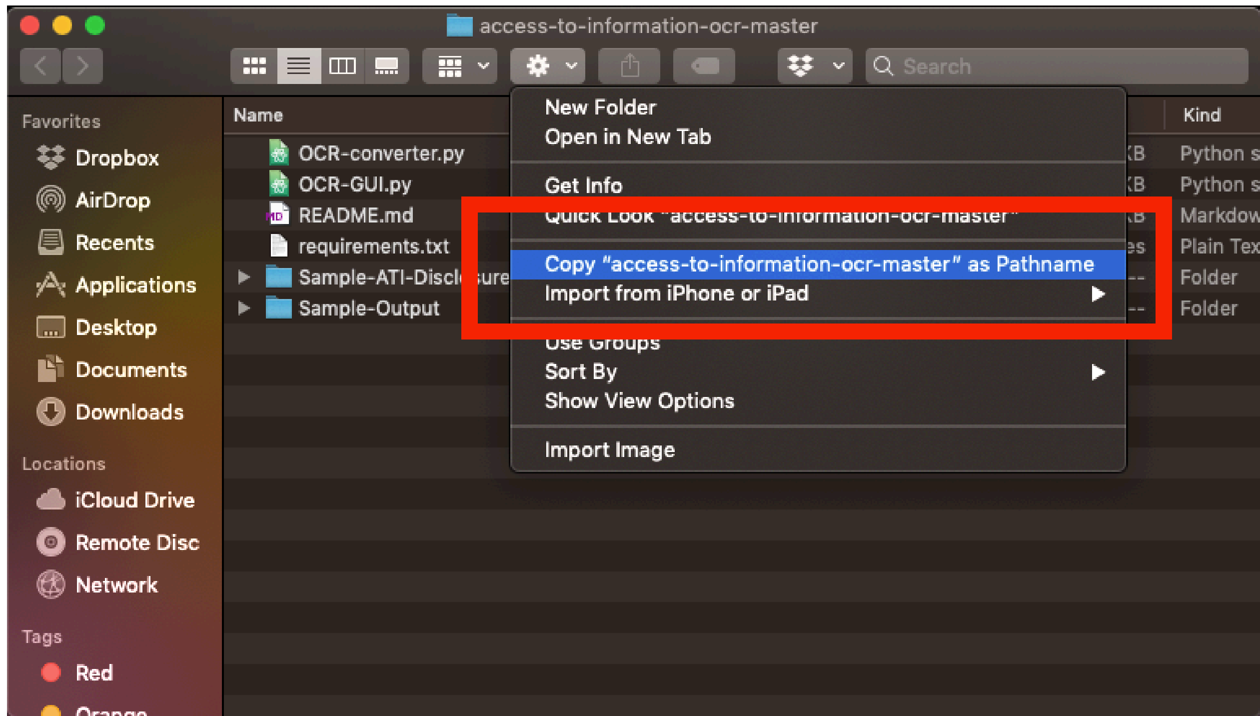


Figure 3: Screenshot of how to obtain file pathname on MacOS

If it returns "command not found", you do not have Ghostscript installed on your computer.

You can download Ghostscript by entering the following command into your computer's Terminal:

```
brew install ghostscript
```

Step five: install Tesseract using Homebrew

If you already have Tesseract installed on your computer, skip this step.

If you unsure about whether you have Tesseract installed, type the following line into your computer's Terminal:

```
tesseract help
```

If it returns "command not found", you do not have Tesseract installed on your computer.

You can download Tesseract by entering the following command into your computer's Terminal:

```
brew install tesseract
```

Step six: copy the pathname of the folder you downloaded, unzipped, and saved in step one

We need to obtain the full directory pathname to the folder that you downloaded, unzipped, and saved in step one. This is a copy of the repository and contains the sample file and Python script we will need to run the Tesseract OCR engine. To obtain the exact pathname to this folder, simply open the folder on your computer, click the gear icon at the top of the window, and select the "copy as Pathname" option from the list (figure 3). This will copy the full pathname to your clipboard.

Step seven: change your working directory in the Terminal

We will now change our working directory using the pathname you just copied to your clipboard. To do this, return to the Terminal and type "cd" followed by the pathname we just copied (you can paste it with command + V). The end result will look something like this (think of this as the basic formula):

```
cd path/name
```

So, if your name is Jane Doe, and you saved the folder on your desktop, it should look something like this:

```
cd /users/janedoe/desktop/access-to-information-ocr-master
```

To double check that you are in the correct directory, you can enter the following command into your Terminal, which will tell you the name of your current directory and display the name of the files contained in it:

```
ls
```

If you are in the correct directory, the "ls" command should show the name of all of the files stored in the master folder, including the most important ones, ORC-converter.py and requirements.txt. If this is not the case, you may need to ensure the "cd" command worked properly, or return to step six.

Step eight: create an output folder

To process our sample ATI/FOI disclosure file to make it machine readable, we are going to first parse the file into individual pages, run each of these through the Tesseract OCR engine, and finally recompile the .txt files generated from each page into a single .txt file that we can then clean, search, and analyze. Throughout these processing stages, a lot individual .png and .txt files are going to be generated, and these need to be stored somewhere.

Inside the access-to-information-ocr-master folder, our working directory, create a new folder. You can call this folder whatever you'd like. If you are using one of the sample data files in the Sample-Files folder, you might name the folder after the sample record you are processing by calling it "A-2017-00078".

The structure of the command is (again, think of this as the formula):

```
mkdir subfolder-name
```

So, if you are going to call this folder "A-2017-00078", you would enter the following into your computer's Terminal:

```
mkdir A-2017-00078
```

To verify that this worked, enter the “ls” command that we learned earlier into the Terminal:

```
ls
```

You should now see the name of your new subfolder listed with the other files in your working directory.

Step nine: ensure you have Python 3 installed on your computer

Python is available on all MacOS computers by default, but depending on the age of your computer, you may need to update it to version 3. To do this, use homebrew to install python3 by entering the follow into your Terminal:

```
brew install python3
```

Step ten: load in the requirements.txt file

Before we can run our python script, we need to install two key Python libraries. To do this, enter the following into the Terminal:

```
pip3 install -r requirements.txt
```

Step eleven: run the script

We are now ready to process our file! This stage can take several minutes (or hours) depending on the size of the file. Processing will happen in three stages. First, the PDF will be parsed into individual page elements (using Ghostscript). Second, each page will be processed into a machine readable .txt file with Tesseract. Third, each page (the .txt files) will be recompiled into a single .txt file named after the PDF.

To do this, we will run a simple Python script (in your working directory, this is the OCR-converter.py file). Same as the previous steps, we will run this script in the Terminal.

To run the Python script, all we need to do is obtain the correct pathname for our input file (if you are following along with the example, one of the sample ATI/FOI disclosure files) and the correct pathname for our output file (the folder we created in step eight) and we are ready to go.

The basic command formula is:

```
python3 OCR-converter.py -i input/file/pathname -o output/folder/pathname
```

So, let’s say we are going to run the script on the A-2017-00078.pdf file in the Sample-Files subfolder, and we are going to store the results in the subfolder we created in step eight called A-2017-00078. The result would look like this:

```
python3 OCR-converter.py -i Sample-Files/A-2017-00078.pdf -o A-2017-00078
```

Enter this into your Terminal, sit back, and relax (but don’t change anything in the master folder until the code is completely finished running!). The end result will be a single .txt file in the output subfolder you created by the same name as the PDF file you processed (e.g., A-2017-00078.txt).

Moving Ahead

When reflecting on the state of his discipline, Immanuel Wallerstein (2001) often suggested it was necessary for sociology to move beyond 19th century paradigms of thought. Wallerstein (1996) also regularly argued it was necessary for contemporary sociologists to understand multiple languages, as failure to do so would result in an inability to comprehend our complex world-system. We would suggest that engaging with computational social science begins to achieve both of these goals. Engaging with computational social science forces scholars to think differently about data. It is not hyperbolic to suggest that the emergence of "big" and new digital forms of data are ushering in the greatest shift in social science since the emergence of the survey (Kent, 1985). Engaging with computational social science also forces scholars to learn what is literally a new language, the language of code.

At the very least, we hope that the tool and step-by-step tutorial we have provided here will help users of ATI/FOI law as well as historians and archivists render big files more searchable, processable, and analyzable. The discoverability of texts (Yarkoni et al., 2019) is a crucial element of access to information. More broadly, we hope this report and our tool will help entice social scientists in Canada and beyond to dip their toes into the computational social science pool. It is fun, it is generally free (assuming one has access to a computer and the Internet, of course, which we acknowledge not everyone does), it is interesting, and it is extremely useful.

References

Franzosi, R., & Roberto, F. (2004). *From words to numbers: Narrative, data, and social science* (Vol. 22). Cambridge University Press. Retrieved from http://services.cambridge.org/us/academic/subjects/sociology/sociology-general-interest/words-numbers-narrative-data-and-social-science?format=PB

Grimmer, J., & Stewart, B. M. (2013). Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis*, *21*(3), 267–297. Retrieved from https://www.cambridge.org/core/journals/political-analysis/article/text-as-data-the-promise-and-pitfalls-of-automatic-content-analysis-methods-for-political-texts/F7AAC8B2909441603FEB25C156448F20

Jacobi, C., Van Atteveldt, W., & Welbers, K. (2016). Quantitative analysis of large amounts of journalistic texts using topic modelling. *Digital Journalism*, *4*(1), 89–106. Retrieved from https://www.tandfonline.com/doi/abs/10.1080/21670811.2015.1093271

Jemielniak, D. (2020). *Thick big data: Doing digital social sciences*. Oxford University Press. Retrieved from https://global.oup.com/academic/product/thick-big-data-9780198839705?cc=us&lang=en&

Kent, R. (1985). The emergence of the sociological survey, 1887-1939. In M. Bulmer (Ed.), *Essays on the History of British Sociological Research* (p. 52-69). Cambridge University Press. Retrieved from https://www.cambridge.org/core/books/essays-on-the-history-of-british-sociological-research/720B29760F67BAB665E8AF6FDC3393BF

Kitchin, R. (2017). Thinking critically about and researching algorithms. *Information, Communication & Society*, *20*(1), 14–29. Retrieved from https://www.tandfonline.com/doi/abs/10.1080/1369118X.2016.1154087

Kitchin, R., & Lauriault, T. P. (2015). Small data in the era of big data. *GeoJournal*, *80*(4), 463–475. Retrieved from https://link.springer.com/article/10.1007/s10708-014-9601-7

Kitchin, R., & McArdle, G. (2016). What makes big data, big data? Exploring the ontological characteristics of 26 datasets. *Big Data & Society*, *3*(1), 2053951716631130. Retrieved from https://journals.sagepub.com/doi/10.1177/2053951716631130

Leetaru, K. (2012). *Data mining methods for the content analyst: An introduction to the computational analysis of content*. Routledge. Retrieved from https://www.routledge.com/Data-Mining-Methods-for-the-Content-Analyst-An-Introduction-to-the-Computational/Leetaru/p/book/9780415895149

Lewis, S. C., Zamith, R., & Hermida, A. (2013). Content analysis in an era of big data: A hybrid approach to computational and manual methods. *Journal of Broadcasting & Electronic Media*, *57*(1), 34–52. Retrieved from https://www.tandfonline.com/doi/abs/10.1080/08838151.2012.761702

Lucas, C., Nielsen, R. A., Roberts, M. E., Stewart, B. M., Storer, A., & Tingley, D. (2015). Computer-assisted text analysis for comparative politics. *Political Analysis*, *23*(2), 254–277. Retrieved from https://www.cambridge.org/core/journals/political-analysis/article/computerassisted-text-analysis-for-comparative-politics/CC8B2CF63A8CC36FE00A13F9839F92BB

Mason, W., Vaughan, J. W., & Wallach, H. (2014). Computational social science and social computing. *Machine Learning*, *95*, 257-260. Retrieved from https://link.springer.com/article/10.1007/s10994-013-5426-8

McEnery, T., & Hardie, A. (2011). *Corpus linguistics: Method, theory and practice*. Cambridge University Press. Retrieved from https://www.cambridge.org/ca/academic/subjects/languages-linguistics/applied-linguistics-and-second-language-acquisition/corpus-linguistics-method-theory-and-practice?format=PB&isbn=9780521547369

McEnery, T., & Hardie, A. (2013). The history of corpus linguistics. In K. Allan (Ed.), *The Oxford Handbook of the History of Linguistics* (pp. 727–745). Oxford University Press Oxford. Retrieved from https://www.oxfordhandbooks.com/view/10.1093/oxfordhb/9780199585847.001.0001/oxfordhb-9780199585847-e-34

Torrens, P. M. (2010). Geography and computational social science. *GeoJournal*, *75*(2), 133–148. Retrieved from https://link.springer.com/article/10.1007/s10708-010-9361-y

van Atteveldt, W., & Peng, T.-Q. (2018). When communication meets computation: Opportunities, challenges, and pitfalls in computational communication science. *Communication Methods and Measures*, *12*(2-3), 81–92. Retrieved from https://www.tandfonline.com/doi/full/10.1080/19312458.2018.1458084

Wallerstein, I. M. (1996). *Open the social sciences: Report of the Gulbenkian Commission on the restructuring of the social sciences*. Stanford University Press. Retrieved from https://www.sup.org/books/title/?id=792

Wallerstein, I. M. (2001). *Unthinking social science: The limits of nineteenth-century paradigms*. Temple University Press. Retrieved from http://tupress.temple.edu/book/3582

Wiedemann, G. (2013). Opening up to big data: Computer-assisted analysis of textual data in social sciences. *Historical Social Research/Historische Sozialforschung*, 332–357. Retrieved from http://www.qualitative-research.net/index.php/fqs/article/view/1949

Yarkoni, T., Eckles, D., Heathers, J., Levenstein, M., Smaldino, P., & Lane, J. I. (2019). Enhancing and accelerating social science via automation: Challenges and opportunities. Retrieved from https://osf.io/preprints/socarxiv/vncwe/

Appendix

Full python script (available in the GitHub repository as OCR-Converter.py):

```
""" OCR-converter.py
Author: Kevin Dick
Date: 2020-05-18
---
Description: Optical Character Recognition (OCR) script
that takes in a scanned PDF document, splits it into individual
PNG pages, applies the Tesseract OCR to each, and compiles together
the result in a machine-readable and searchable text-document.
"""
import os, sys
import locale
import ghostscript
import argparse

parser = argparse.ArgumentParser(description='')
parser.add_argument('-i', '--input_pdf', required=True,
                 help='path to the scanned PDF document')
parser.add_argument('-o', '--output_dir', required=True,
                 help='the directory where the output will be saved')
parser.add_argument('-v', '--verbose', action='store_true',
                 help='increase verbosity')
args = parser.parse_args()

# EXAMPLE: python3 OCR-converter.py -i /path/to/the/scanned/pdf-file.pdf -o /path/to/the/
    directory/where/files/will/be/saved/ -v

# Subdirectories for the individual pages
PAGE_IMG = 'page-img'
PAGE_TXT = 'page-txt'

def check_paths():
       """ check_paths
           This function will verify that the PDF exists and that the output paths are
               correctly formatted.
           If the sub-directories for the pages are not present, they will be made.
           Input: None
           Output: <str, None>, error message if input file or output dir don't exist.
               None, if everything ok.
       """
       if args.verbose: print('Checking validity of input and creating subdirectories..')
       if not os.path.exists(args.input_pdf): return f"Error: File {args.input_pdf} doesn'
           t exist."
       if not os.path.exists(args.output_dir): return f"Error: Directory {args.output_dir}
            doesn't exist."

       # If the page image directory exists, remove everything inside & remake it
```

```
        if os.path.exists(os.path.join(args.output_dir, PAGE_IMG)): os.system(f'rm -rf {os.
            path.join(args.output_dir, PAGE_IMG)}')
        os.mkdir(os.path.join(args.output_dir, PAGE_IMG))

        # If the page text directory exists, remove everything inside and remake it
        if os.path.exists(os.path.join(args.output_dir, PAGE_TXT)): os.system(f'rm -rf {os.
            path.join(args.output_dir, PAGE_TXT)}')
        os.mkdir(os.path.join(args.output_dir, PAGE_TXT))

def main():
        """ main function """
        # Step 0: We must ensure that the PDF and directory specified exist and the
            subdirectories for individual pages exist.
        # We make the subirectories if they dont already exist.
        setup = check_paths()
        if setup != None:
                # If we get an error message, we print it and exit
                print(setup)
                sys.exit(0)

        # These are the two locations where individual files will be saved as well as where
             the output txt will be saved
        png_dir = os.path.join(os.path.abspath(args.output_dir), PAGE_IMG)
        txt_dir = os.path.join(os.path.abspath(args.output_dir), PAGE_TXT)
        output_txt = os.path.join(os.path.abspath(args.output_dir), os.path.basename(args.
            input_pdf).replace('.pdf', '.txt'))

        # Step 1: First the PDF is split into individual pages, with one PNG image
            representing each page of the scanned PDF.
        # This is achieved using Ghostscript, a suite of software capable of manipulateing
            and transforming the contents
        # of PDFs. We move to the directory where Ghostscript will output the individual
            pages as PNG files: png_dir.
        # We then run the following command:
        gs_args = [arg.encode(locale.getpreferredencoding()) for arg in ['',
                                                                          '-dBATCH',
                                                                          '-dNOPAUSE',
                                                                          '-sDEVICE=png16m',
                                                                          '-r256',
                                                                          '-sOutputFile=page%d.png',
                                                                          '-f', os.path.abspath(args.
                                                                              input_pdf)]]
        # Equivalent to this command: gs -dBATCH -dNOPAUSE -sDEVICE=png16m -r256 -
            sOutputFile=page%d.png {os.path.abspath(args.input_pdf)}
        os.chdir(png_dir)
        ghostscript.Ghostscript(*gs_args)
        #if args.verbose: print(cmd)
        #os.system(cmd)

        # Step 2: Secondly, we run the Tesseract OCR on each PNG image and it produces the
```

```
            detected text-based output for each image.
        # Since we are currently in the PNG subdirectory, we will run over all files in
            this directory and save them to the
        # txt_dir.
        for page_png in os.listdir(png_dir):  # extract all of the filenames in this
            directory (e.g. "page001.png")
                if 'page' not in page_png: continue # skip any files that don't have 'page'
                    in the name (e.g. .DS_Store)
                page_txt = page_png.replace('.png', '') # convert the .png to .txt for the
                    output file (the .txt gets added)
                cmd = f'tesseract {page_png} {os.path.join(txt_dir, page_txt)}'
                if args.verbose: print(cmd)
                os.system(cmd)

        # Step 3: Finally, we iterate over all txt files in sorted order to compile them
            into a final document of all pages.
        # The output file name will be the same as the input file, only with .txt instead
            of .pdf)
        open(output_txt, 'w').write('') # Create an empty file to save txt data to
        for page_txt in sorted(os.listdir(txt_dir)):
                if 'page' not in page_txt: continue # skip any files that don't have 'page'
                    in the name (e.g. .DS_Store)
                # the 'cat' command prints out the contents of a file
                # the '>>' operator redirects the result of the 'cat' command to the END of
                    the file
                # together, this command simply adds the text of each page to the end of the
                     output txt file.
                cmd = f'cat {os.path.join(txt_dir, page_txt)} >> {output_txt}'
                if args.verbose: print(cmd)
                os.system(cmd)

if __name__ == "__main__": main()
```