ARC Collection Script Process Overview

Evan Fioritto

8/1/24

General Overview (readme.md)

 # ARC Collection Script

Written by Evan Fioritto, May to August 2024

For MSU College of Business - Management Department

Requirements

- Download StanfordNLP jar folder@ https://drive.google.com/drive/folders/1nLSX6i8wG5HWzsVFjF0l22Wj6ET7Graj

assign your system the environment variable “STANFORD_NLP_HOME” with the destination as the directory you downloaded

-Download Java @

https://www.java.com/download/ie_manual.jsp

set the environment variable “JAVA_HOME” at Java’s bin folder (in ProgramFilesx86/Java/jre-…/bin by default)

-Download glove files (for icgauge) @

https://drive.google.com/drive/folders/1Zts0rkp-Te8oYutpRZe1Y3A8vzFSMD8q

assign your system the environment variable “GLV_HOME” with the destination as the directory you downloaded

-Download GTK @

https://drive.google.com/drive/folders/1nFvK1aJJl02kSM2k4RtKfwcWCcjW9sgq

assign your system the environment variable “WEA$YPRINT_DLL_DIRECTORIES” with the destination as the gtk’s bin directory(in program files by default)

Restart PC after setting environment variables

- `alive-progress==3.1.5`

- `glib==1.0.0`

- icgauge (my abridged version)

@ https://drive.google.com/drive/folders/1QN5BpS9ZoCAmaQ3XkQXw9trTMI8va3iQ

download into root project folder (/ARC-COLLECTION-SCRIPT)

- `httplib2==0.22.0`

- `nltk==3.8.1`

- `numpy==1.26.4`

- `scikit-learn==1.5.1`

- `panda==0.3.1`

-`mord`

-`requests`

- `pango==0.0.1`

- `pycairo==1.26.1`

- `PyMuPDF==1.24.9`

- `PyPDF2==3.0.1`

- `python-doctr==0.8.1`

- `thefuzz==0.22.1`

- `selenium==4.23.1`

- `stanfordcorenlp==3.9.1.1`

- `torch==2.4.0 torchvision torchaudio --index-url https://download.pytorch.org/whl/test/cu118`

-`webdriver-manager==4.0.2`

CSV Explanations

- **ziptracker.csv**: Information scraped from each search, along with the status of the bulk download. This file is more of a history than an account of downloaded files. For example, files correctly downloaded in a previous run may err and be marked incorrectly even with a prior correct entry. For an account of files on disk, consult filestracker.csv. Of format: [Global Company Key, Company Name, Page Found On]

- **filestracker.csv**: Information regarding every successfully downloaded file (not verified). Of format: [Global Company Key, Company Name, Row Number, Page Number, Filing Date, Doctype]

- **metadata.csv**: Extrapolation off `filestracker.csv` and `ziptracker.csv`, pairing each entry with its location on disk. Of format: [Filename, GVKey, HH Name, Mergent Name, Year, Date, DocType, Parent Zip]

- **matching.csv**: Relation of `filestracker.csv` and `ziptracker.csv` to `ARC_missing.csv`. Of format: [GVKey, Company Name, Year, Data Date, Status, Year Match, File Count]

- **found_firms.csv**: Collection of all desired entries with a GVKey and year match with at least one file. Of format: [GVKey, Company Name, Year, Data Date, File Count]

- **missing_firms.csv**: Collection of all desired entries either not on Mergent or not with a year match. Of format: [GVKey, Company Name, Year, Data Date, Status]

- **confirmations.csv**: Basic OCR results validating actual contents of PDFs derived from `metadata.csv`. [Path, GVKey, HH Name, Mergent Name, Year, Doctype Confirmed, Name Confirmed, Year Confirmed, Index in missing.csv]

Folder Explanations

- **/zips**: zips downloaded from mergent by `downloadscript.py`

- **/folders**: all zips unzipped, populated in `verifyscript.py`

- **/trackers**: csv files containing information about all zips and files downloaded by `downloadscript.py`

- **/matched_folders**: folders with a proved correlation to `ARC_mising.csv`

- **/sample_data**: folder containing json training data `toy.json` for icgauge to compare against in `OCRscript.py`

Other

- **lastindex.txt**: the index of unique firm last reached in `downloadscript.py`, used to store globally across runs

- **temp.json**: file to hold NLP parsed data for icgauge validation in `OCRscript.py`

Abbreviations

- **OK**: A GVKey match

- **Y**: A year match

- **N**: No year match

- **NA**: No GVKey Match

- **TL**: zip skipped due to being over 1.8 GB

- **SK**: zip skipped due to bug in webpage processing

Process Overview

1. **Downloadscript.py**: Runs the initial bulk download based off entries in `ARC_HH_OK_AK_missing.csv`, storing information about each search and successful download in `ziptracker.csv` and `filestracker.csv`.

2. **Verifyscript.py**: Details what information we believe the files hold in `metadata.csv`, as well as rerunning a secondary search for skipped or missing files using functions from `downloadscript.py`. Basic fuzzy matching is conducted to verify the contents of PDFs.

3. ****Analyzeresults.py****: Uses data stored in `matching.csv` from `verifyscript.py` to produce statistics as well as store in separate files `found_firms.csv` and `missing_firms.csv`.

4. ****Copyfoundfirms.py****: Copies all downloaded files correlated with an entry in `ARC_HH_OK_AK_missing.csv` to the `matched_folders` directory.

5. ****OCRscript.py****: Uses docTr and icgauge packages to extract text and weigh semantic complexity for each file specified from `found_firms.csv`, will run significantly slower on machines will low specifications

Running the Scripts

Running `main.py` will execute all scripts in order. If one errs, they are all fit to be rerun individually and repeatedly

---

File By File Logic Analysis

Downloadscript.py

Purpose: Execute, track and manage bulk downloads of firms described in ARC_HH_OK_AR_missing.csv

Main():

1. Create dictionary of unique firms from ARC_HH_OK_AR_missing.csv (958 firms)
2. Initialize selenium WebDriver and file trackers
3. Gather range of firms to query and MSU login details from user via CMD.
4. Iterate through dictionary, searching by key (GVKEY) and value (Company Name), checking if downloads in the last hour have exceeded 1800000 kb (1.8GB) as Mergent's lockout threshold is 2GB per hour.

The search process: Step 4's functionality is split between 2 major functions, SearchActions() and ResultActions().

SearchActions():

1. Check for search page load, if load takes over 15 seconds the zip is marked as "SK" in ziptracker.csv and firm is skipped.
2. Company name is entered and Annual Report is selected from drop-down menu, the doctype selection here is finicky, but the script will attempt 10 times to select the desired doctype, after 10 attempts, it will proceed will selected doctype. However, this almost never happened in any runs.
3. ResultActions() is now called. Company name and doctype are submitted, Mergent will redirect to results page.

ResultActions():

1. Results page sometimes hangs on redirection, so attempt refresh and reload, if second attempt fails, firm is marked as "SK" and skipped

2. Check for total number of pages. If it is greater than 1, step 3 will be performed iteratively. If no results, SearchActions() will be called again, this time searching for all doctypes*, not just Annual Reports. This is effectively a recursive call for both SearchActions() and ResultActions()
3. Check if download will cross threshold using CheckAndWait(), resets if over
4. Scrape columns for file dates and sizes, generate a file name (ending in _altreport if procured from a secondary search), download will only proceed if the tentative download + the total download are smaller than threshold. If it is greater, firm is marked "TL" in ziptracker.csv. File is stored in zips.
5. After download is done redirect back to search page for next search

**"All doctypes" means alternative doctypes, namely "Annual/10K Report" "10K or Int'l Equivalent"*

Verifyscript.py

Purpose: Quantify accuracy and output of downloadscript.py

Main():

1. Unzip all files in' /zips' subdirectory to folders of the same name in './folders' subdirectory
2. Iterate through each zip in ziptracker.csv, If zip has an integer value in the zip_page column (page of download, index 2 of ziptracker.csv), iterate through files and find a matching file in './folders' that correlates with zip_page value gvkey, and row value (derived from number in pdf name, assigned by mergent accurately). Store file path.
3. Compile all correlation data gathered in step 2 in metadata.csv.
4. Iterate through ARC_HH_OK_AR_missing.csv, then compare each entry against entries from ziptracker.csv and entries from filestracker.csv from valid doctypes.
5. If a GVKey is found the firms status in matching.csv (index 4) will be marked "OK", if there is no GVKey match it will be marked "NA", If a file is found with a matching GVKey and Mergent listed year, index 6 will be marked "Y", index 7 specifies how many year matches were found.
6. Next, a tertiary search is conducted on all firms with an "OK" status, yet still have years marked "N", this search repeats the search process described in downloadscript.py, even using its functions, except targeting these specific "gap years".
7. Repeats steps 1-5 again to analyze new data added from tertiary search.
8. VerifyPdfs(), PrintStatistics() and CountMatches() functions are called to print basic overview of results of the previous 7 steps. VerifyPdfs() conducts basic OCR + fuzzymatching to verify file contents and stores in confirmations.csv.

Analyzeresults.py

Purpose: Analyze matching.csv to quantify missing firms, found firms and other statistics. Does not change any files, except adding found_firms.csv and missing_firms.csv. Mainly for insight.

Main():

1. Analyze matching.csv for files that returned no Mergent result or no matching year
2. Analyze matching.csv for files that OK and have a year match and at least one file
3. Counts complete firms and total entries, matches, sends statistics to terminal.

Copyfoundfirms.py

Purpose: Separate desired folders from irrelevant folders

Main():

1. Derive unique found firm GVKeys from found_firms.csv
2. Iterate through each folder in './folders', if the GVkey is matched and the file hasn't been copied yet, the folder will be copied to './matched_folders'

OCRscript.py

Purpose: Extract text from desired files using docTR and weigh their semantic complexity using ICgauge

Main():

1. Setup OCR Model using "linknet_resnet18" dataset (trained on rotated files) from docTR documentation
2. Start Stanford NLP parsing server for ICgauge, hosted locally as a subprocess using 6GB of allocated memory, can be changed according to your machine's max RAM.
3. Pass desired documents into OCR model
4. Format OCR'd text using FormatResult() function, which only passes lines with 5 or more words. This eliminates tabular data or incomplete lines.
5. Pass formatted text into GetICGaugeScore function which returns semantic complexity average.

GetICGaugeScore():

1. Connect to Stanford NLP via http://localhost:9000
2. Parse the body of text. This begins by splitting text into smaller segments via .split(\n"). This will split the body into sentences.
3. Iterate through these sentences, appending each group of 5(our decided length for a paragraph) to the "paragraphs" dictionary with the key being raw text and the value being the StanfordNLP analyzed text. If a paragraph taxes over 10 seconds to parse it will be skipped.
4. This dictionary is parsed to "temp.json" in CreateTempJson(), ICgauge requires its training and assessment data of this format with each entry holding a parse, paragraph, and score. Since we are creating an assessment set, our score for each paragraph is irrelevant, so it is set to a random integer.
5. The ICgauge experiment_Features function is now ran using their sample data (located in sampledata/toy.jso) as a trainer and ours as an assessment.
6. The experiment function returns a list of each paragraph with its predicted value in the last index of its return array. The semantic complexity is derived from an average of all predicted values.
7. Server is terminated.
