CandidateBios Documentation

Table of Contents

Directory Structure

Root

RevolvingDoor/Code/Scraping/Victor/

./DataTests

A subdirectory containing all of the CSVs and visuals produced while manually comparing the pipeline-produced biodata with the true biodata available on the internet.

./DataTests/Samples

A subdirectory containing some sample search and scraping results to reference and use.

./Documentation

A subdirectory containing all of the documentation necessary to run the CandidateBios Data Processing Pipeline and understand all of the files in the root directory.

./Newspapers

A subdirectory containing all of the files relevant to gathering the roll call votes for lobbying legislation passed after 1998.

./Old

A subdirectory containing the old source code for the CandidateBios Data Processing Pipeline.

./Pipeline

A subdirectory containing the up-to-date source code for the CandidateBios Data Processing Pipeline.

./Results

A subdirectory containing the results of the CandidateBios Data Processing Pipeline.

./Results/Accuracy

A subdirectory containing the manual verification accuracy of a sample of 100 candidates from the final output.

./Results/All

A subdirectory containing the final output CSV with all of the gathered candidate biodata.

./Results/BaseOutputs

A subdirectory containing all of the merged output and error files generated from the initial run of the CandidateBios Data Processing Pipeline.

./Results/ChatGPT

A subdirectory containing all of the intermediary save files generated by the data extraction phase during the initial run of the CandidateBios Data Processing Pipeline.

./Results/GoogleSearch

A subdirectory containing all of the intermediary save files generated by the data search phase during the initial run of the CandidateBios Data Processing Pipeline.

./Results/Missing

A subdirectory containing all of the files relevant to candidates that were not scraped during the initial run of the CandidateBios Data Processing Pipeline.

./Results/Missing/Scraping

A subdirectory containing information about all of the candidates whose ChatGPT prompts were longer than the 4096 token context window of the gpt-turbo-3.5-0613 API model.

./Results/Missing/Summary

A subdirectory containing information and output files for candidates whose ChatGPT responses were incorrectly parsed on the initial run of the CandidateBios Data Processing Pipeline, but eventually gathered on a later run of the pipeline.

./Results/Outputs

A subdirectory containing all output files generated by combining the intermediary save files created by the data extraction phase during the initial run of the CandidateBios Data Processing Pipeline.

./Results/Scraping

A subdirectory containing all of the intermediary save files generated by the data retrieval phase during the initial run of the CandidateBios Data Processing Pipeline.

Reproducibility

Ensuring the reproducibility of research is of great importance, and numerous measures were undertaken to maximize the replicability of the candidate biographical data generated through a custom Python data processing pipeline. During the Data Search phase of the pipeline, the *Google Custom Search JSON API* was employed, with results restricted to English-language content originating from the United States, and a *Google Custom Search Engine (CSE)* was utilized. Owing to the API's reliance on the ever-evolving Google Search algorithm and a *CSE*, the four source URLs used to gather each candidate's biodata were recorded in the "Sources" column of the final data output, "outputs-all.csv.", to mitigate the potential loss of access to a source or alterations in the page content In the Data Retrieval phase, the source pages were scraped to create a ChatGPT prompt for each candidate. All prompts were stored in the "ChatGPT Prompt" column of the file "retrievals-base-all.csv."

The Data Extraction phase leveraged the *Chat Completions Endpoint* and the *gpt-3.5-turbo-0613* model of the *OpenAI API* to summarize and extract the desired candidate biodata from the given ChatGPT prompt for each candidate. At the time of executing the pipeline and gathering the data (Fall 2023), these were the latest and most cost-efficient endpoints and models available. However, it is noteworthy that the *gpt-3.5-turbo-0613* model is scheduled for deprecation on June 13th, 2024. Newer, more powerful *OpenAI API* models may exhibit enhanced efficacy in summarizing and extracting the desired candidate biodata, potentially producing output with fewer false positives and more comprehensive information. Additionally, as with all large-language models, ChatGPT is non-deterministic, implying that the same input can produce different outputs due to the inherent structure of the technology. To maximize the deterministic output, the *temperature* parameter of the API was set to 0. In retrospect, defining the *seed* parameter in the API would have been prudent, as repeated requests with the same *seed* and parameters would then return identical results. However, it is worth noting that OpenAI does not guarantee determinism in this configuration, so it may not have significantly impacted the outcomes. Given the relatively small prompt size for each candidate (a maximum length of 4096 tokens) and the objective nature of the desired candidate biodata, it is anticipated that the ChatGPT responses should typically remain consistent. Nevertheless, fields with more extensive information were observed to exhibit slight variations in their responses; in particular, when the "Work History" consisted of an extensive list of jobs, there were inconsistencies in the number of jobs listed in the output.

Sources
https://platform.openai.com/docs/api-reference

Dependencies

APIs

- google_api_python_client==2.125.0
- openai==1.16.2

Efficiency

- tenacity==8.2.3

Machine-Learning

- The dependencies in accuracy.ipynb can be directly installed from within the file.

Scraping

- beautifulsoup4==4.11.2
- PyPDF2==3.0.1
- requests-html==0.10.0

Utility

- pandas==2.2.1
- python-dotenv==1.0.1

.env File

There is already a .env file in the "Pipeline" directory ("../Pipeline"), but the variables have been reset to empty strings for security reasons. The following steps explain how to configure your own .env variables.

1. *engine*
    1.1. Go to https://programmablesearchengine.google.com/controlpanel/all.
    1.2. Create a new search engine.
    1.3. For the *What to search* parameter, select *Search the entire web*.
    1.4. Click *Create* and then click *Customize*.
    1.5. In the section titled *Search Features*, select the *United States* as the *Region*.
    1.6. Turn on *Region restricted results*.
    1.7. For the *Sites to search* parameter, add the following sites:
        1.7.1. www.ballotpedia.org/*
        1.7.2. www.wikipedia.org/*

        1.7.3. justfacts.votesmart.org/*
    1.8. Set the *engine* variable in the .env file equal to the *Search engine ID* found in the section titled *Basic*.
2. *google_api_key*:
    2.1. Go to https://developers.google.com/custom-search/v1/introduction.
    2.2. Click on *Get a key* and follow the instructions.
    2.3. Set the *google_api_key* variable in the .env file equal to the api key from step 2.2.
3. *openai_api_key*:
    3.1. Go to https://platform.openai.com/api-keys.
    3.2. Click on *view user api keys*.
    3.3. Create a new secret key.
    3.4. Set the *openai_api_key* variable in the .env file equal to the api key from step 3.3.

Data Pipeline

a_pipeline.py

Created by Victor Verma
Last edited May 7, 2024

This file was used to run the data processing pipeline from end to end, starting with the Google search for information and ending with parsing the ChatGPT response and exporting the information into an output CSV file.

Setup

- *testRows*: a dictionary that stores some row numbers that are useful for pipeline testing.
- *outputFiles*: a list containing the names of all of the files that are produced through the program.
- *groups*: a list containing the groups of candidates that have been processed.

Functions

gatherData(n, r = 4, read = “random”)
Description

- Wrapper function used to run the program.

Parameters

- *n*: an integer that indicates the number of unique candidates for whom to gather biodata.
- *r*: an integer that specifies the number of Google API search results to use during the gathering process. 1 <= *r* <= 4, and r is set to 4 by default.

- *read*: a string that defines how the *n* unique candidates should be chosen. If *read* is set to "random", then n unique random candidates are used, and if *read* = "order", then the first *n* unique candidates in order are used. *read* is set to random by default.

Return

- A dataframe containing each candidate's full name, state, min year, candid, college major, undergraduate institution, highest degree, work history, sources, and ChatGPT confidence. This dataframe is also output to d1_extractions.csv.

gatherRow(r = 4, rows = [])

Description

- Wrapper function used to run the program on specified candidates.

Parameters

- *r*: an integer that specifies the number of Google API search results to use during the gathering process. 1 <= *r* <= 4, and *r* is set to 4 by default.
- *rows*: an integer array containing the candidates' row numbers for whom to gather biodata. The row number passed into the array for a candidate should be equal to the row number for that candidate in ldata_R_unique.csv - 2 to account for the indexing in pandas dataframes.

Return

- A dataframe containing each candidate's full name, state, min year, candid, college major, undergraduate institution, highest degree, work history, sources, and ChatGPT confidence. This dataframe is also output to d1_extractions.csv.

deleteOutput(outputFiles)

Description

- Deletes existing program output files from the local directory.

Parameters

- *outputFiles*: a string array that contains the names of all output files that should be deleted. Globally defined as ["b1_searches.csv", "c1_retrievals.csv", "c2_scrapingTimeouts.csv", "d1_extractions.csv", "d2_promptErrors.csv", "d3_parseErrors.csv", "d4_reruns.csv", "errors.txt"].

Return

- No return value.

combineCSV(comboType, group)

Description

- Combines the specified set of CSVs into one large CSV.

Parameters

- *comboType*: a string that indicates whether the prompt, timeout, or output CSVs should be combined.
- *group*: a string that specifies which group's candidates should be used for pathing purposes.

Return

- No return value.

Data Search

b_search.py

Created by Victor Verma
Last edited May 7, 2024

This file was used to read the given candidate information from ldata_R_unique.csv and run it through the Google Custom Search JSON API to find websites containing biodata information for the candidates. The intermediate results were stored in b1_searches.csv.

Setup

- *sourceData*: a global variable that stores the relative path to ldata_R_unique, which serves as the source data for the candidate information.
- *google_api_key*: a global variable that reads the Google Custom Search JSON API key from the .env file.
- *engine*: a global variable that reads the Google Custom Search Engine ID from the .env file.
- *resource*: a global variable that integrates the Google Custom Search JSON API key with the Google Custom Search Engine.
- *states*: a global dictionary that stores U.S. state abbreviations as keys and the full state name as the corresponding pairs.
- *testRows*: a global dictionary of candidate rows that are useful for testing.

Functions

search(n = 1, r = 4, read = “random”)

Description

- Wrapper function used to run the data search phase.

Parameters

- *n*: an integer that indicates the number of unique candidates for whom to gather biodata.
- *r*: an integer that specifies the number of Google API search results to use during the gathering process. 1 <= *r* <= 4, and *r* is set to 4 by default.
- *read*: a string that defines how the *n* unique candidates should be chosen. If *read* is set to “random”, then n unique random candidates are used, and if *read* = “order”, then the first *n* unique candidates in order are used. *read* is set to random by default.

Return

- A dataframe containing each candidate’s Google Search results, first name, middle name, last name, full name, min year, state, and candid. This dataframe is also output to b1_searches.csv.

searchRow(r = 4, rows = [])

Description

- Wrapper function used to run the data search phase on specified candidates.

Parameters

- *r*: an integer that specifies the number of Google API search results to use during the gathering process. 1 <= *r* <= 4, and *r* is set to 4 by default.
- *rows*: an integer array containing the candidates' row numbers for whom to gather biodata. The row number passed into the array for a candidate should be equal to the row number for that candidate in ldata_R_unique.csv - 2 in order to account for the indexing in pandas dataframes.

Return

- A dataframe containing each candidate’s Google Search results, first name, middle name, last name, full name, min year, state, and candid. This dataframe is also output to b1_searches.csv. This dataframe is also output to b1_searches.csv.

randomRead(file, n)

Description

- Reads the relevant candidate information from ldata_R_unique.csv for n randomly chosen candidates.

Parameters

- *file*: a string representing the file path to ldata_R_unique. The global variable *sourceData* is always passed in as *file*.
- *n*: an integer that indicates the number of unique candidates for whom to read relevant information from ldata_R_unique.csv. The candidates are chosen randomly.

Return

- An array containing the relevant candidate information for each candidate as the elements in the array. Each element in the array is a dictionary containing the full name delimited by quotes, state, first name, last name, full name, and candid for the chosen candidate.

orderRead(file, n)

Description

- Reads the relevant candidate information from ldata_R_unique.csv for the first n candidates in order.

Parameters

- *file*: a string representing the file path to ldata_R_unique. The global variable *sourceData* is always passed in as *file*.
- *n*: an integer that indicates the number of unique candidates for whom to read relevant information from ldata_R_unique.csv. The candidates are chosen in order.

Return

- An array containing the relevant candidate information for each candidate as the elements in the array. Each element in the array is a dictionary containing the full name delimited by quotes, state, first name, last name, full name, and candid for the chosen candidate.

rowRead(file, rows)

Description

- Reads the relevant candidate information from ldata_R_unique.csv for the candidates who correspond to the specified rows.

Parameters

- *file*: a string representing the file path to ldata_R_unique. The global variable *sourceData* is always passed in as *file*.
- *rows*: an integer array containing the candidates' row numbers for whom to gather biodata. The row number passed into the array for a candidate should be equal to the row number for that candidate in ldata_R_unique.csv - 2 to account for the indexing in pandas dataframes.

Return

- An array containing the relevant candidate information for each candidate as the elements in the array. Each element in the array is a dictionary containing the full name delimited by quotes, state, first name, last name, full name, and candid for the chosen candidate.

googleSearch(rep, r = 4)

Description

- Uses the Google Custom Search JSON API and a Google Custom Search Engine to gather the top URLs from the Google Search of each candidate.

Parameters

- rep: a dictionary containing the full name delimited by quotes, state, first name, last name, full name, and candid of a candidate. This is exactly an element from the output of randomRead, orderRead, or rowRead, depending on how the candidates were chosen.
- r: an integer that specifies the number of Google API search results to use during the gathering process. 1 <= r <= 4, and r is set to 4 by default.\

Return

- A dictionary containing the top r URLs from the Google Search, first name, middle name, last name, full name, min year, state, and candid of a candidate as keys. The value containing the top r URLs is a string array.

searchCSV(urls)

Description

- Processes the data gathered in the data search phrase and converts it into a pandas dataframe and CSV file.

Parameters

- *urls:* an array containing the relevant candidate information for each candidate as the elements in the array. Each element is itself a dictionary containing the top r URLs from the Google Search, first name, middle name, last name, full name, min year, state, and candid of the candidate as keys. The value containing the top r URLs is a string array.

Return

- A dataframe containing each candidate's Google Search results, first name, middle name, last name, full name, min year, state, and candid. This dataframe is also output to b1_searches.csv.

Data Retrieval

c_retrieval.py

Created by Victor Verma
Last edited May 7, 2024

This file was used to scrape the web pages that were temporarily stored in b1_searches.csv, which was produced in b_search.py. The scraped texts were converted into ChatGPT prompts, which were intermediately stored in c1_retrievals.csv. The elements in b1_searches.csv that could not be scraped were stored in c2_scrapingTimeouts.csv. The CSV file containing the scraping timeouts can also be passed in as the *searchData* to retry scraping them.

Setup

- *timeoutCandidates*: a global variable that stores the data of candidates that took longer than 500 seconds to scrape.
- *searchData*: a global variable that stores the relative path to the CSV file containing the relevant candidate search results.

Functions

retrieve(searchData = searchData, timeout = 200)

Description

- Wrapper function used to run the data retrieval phase.

Parameters

- *searchData*: a global variable that stores the relative path to b1_searches.csv, which contains all of the information gathered in the data search phase. This can optionally be configured to another CSV of the proper format.
- *timeout*: specifies the time allotted for each instance of the biodata() function to complete before raising a TimeoutError. This is set to 200 seconds by default.

Return

- A dataframe containing each candidate’s ChatGPT prompt, sources, full name, min year, state, and candid. This dataframe is also output to c1_retrievals.csv.

splitCandidates(urls, batchSize)

Description

- Splits the array of candidates to be scraped into subarrays of size batchSize, with the last array containing any leftovers.

Parameters

- *urls*: An array whose elements are dictionaries containing each candidate’s Google Search results, first name, middle name, last name, full name, min year, state, and candid as keys.
- *batchSize*: an integer that specifies the size of each batch of candidates to be scraped in a singular instance of the ThreadPoolExecutor().

Return

- A 2-D array containing the batches as elements. Each batch is itself an array of maximum size batchSize whose elements are dictionaries containing each candidate’s Google Search results, first name, middle name, last name, full name, min year, state, and candid as keys.

sourceParser(sources)

Description

- Converts a string representing an array of source URLs into a string array with each source URL as an element within the array.

Parameters

- *sources*: a string representing an array of source URLs for a candidate.

Return

- A string array with each source URL as an element within the array.

bioData(link)

Description

- Wrapper function used to scrape the source URLs for each candidate.

Parameters

- *link*: A dictionary containing the top r URLs from the Google Search, first name, middle name, last name, full name, min year, state, and candid of the candidate. The value containing the top r URLs is a string array.

Return

- A dictionary containing plain text scraped from the source URLs, the source URLs, full name, year, state, and candid of a candidate. The value containing the source URLs is a string array.

pdfReader(url)

Description

- Scrapes up to the first 3 pages of a pdf.

Parameters

- *url*: a string that represents the web URL of a pdf.

Return

- A string representing the plain text of up to the first 3 pages of the pdf.

grabber(information, phrase)

Description

- Scrapes the first 400 plain text words following the occurrence of a specified phrase on a webpage or pdf.

Parameters

- *information*: a string representing the plain text of a webpage or pdf.
- *phrase*: a string that indicates where to start scraping the 400 words within the plain text given by *information*. *phrase* can be either the last name, first name, or middle name of the candidate.

Return

- A string representing the first 400 plain text words following the occurrence of a specified phrase on a webpage or PDF.

chatPrompt(info)

Description

- Creates a ChatGPT prompt for the candidate using the scraped text from its source URLs.

Parameters

- info: A dictionary containing plain text scraped from the source URLs, the source URLs, full name, year, state, and candid of a candidate. This is exactly the output of the bioData() function. The value containing the source URLs is a string array.

Return

- A dictionary containing the ChatGPT prompt, source URLs, full name, min year, state, and candid of a candidate. The value containing the source URLs is a string array.

retrieveCSV(prompt)

Description

- Processes the data gathered in the data retrieval phrase and converts it into the corresponding pandas dataframes and CSVs. Handles both successfully and unsuccessfully scraped candidates, storing the information in c1_retrievals.csv and c2_scrapingTimeouts.csv, respectively.

Parameters

- *prompts*: an array containing the relevant candidate information for each candidate as the elements in the array. Each element is itself a dictionary containing the ChatGPT prompt, source URLs, full name, min year, state, and candid of the candidate. The value containing the source URLs is a string array.

Return

- A dataframe containing each candidate’s ChatGPT prompt, sources, full name, min year, state, and candid. This dataframe is also output to c1_retrievals.csv.

Data Extraction

d_extraction.py

Created by Victor Verma
Last edited May 7, 2024

This file was used to feed the scrape information that was temporarily stored in c1_retrievals.csv, which was produced in c_retrieval.py, into the OpenAI API for summarization. The OpenAI responses that were correctly parsed were stored in d1_extractions.csv. The responses in this file can be interpreted as the final biodata output. The ChatGPT prompts that ran into errors with the API were stored in d2_promptErrors.csv. The OpenAI responses that were incorrectly parsed were stored in d3_parseErrors.csv. The prompt errors in d2_promptErrors.csv could also be retried directly from the file, and the results were stored in d4_reruns.csv.

Setup

- *retrievalData*: a global variable that stores the relative path to c1_retrievals.csv, which contains all of the information gathered in the data retrieval phase. This can optionally be configured to another CSV of the proper format.
- *promptErrorData*: a global variable that stores the relative path to d2_promptErrors.csv, which contains all of the candidates who encountered prompt errors. This can optionally be configured to another CSV of the proper format.
- *openai.api_key*: a global variable that reads the OpenAI API key from the .env file

Functions

extract(csvColumns = “regular”)

Description

- Wrapper function used to run the data extraction phase.

Parameters

- *csvColumns*: a string that describes the names of the columns of the source CSV file. If *csvColumns* is set to regular, then the program parses the column names “ChatGPT Prompt”, “Sources”, “Full Name”, “Min Year”, “State”, and “Candid”. If *csvColumns* is set to “condensed”, then the program parses the column names “chatgptprompt”, “sources”, “fullname”, “minyear”, “state”, and “candid”.

Return

- A dataframe containing each candidate's name, state, min year, candid, college major, undergraduate institution, highest degree and institution, work history, sources, and ChatGPT confidence. This dataframe is also output to d1_extractions.csv.

extractAgain(attempt = "first")

Description

- Wrapper function used to rerun the data extraction phase for candidates who encountered prompt errors.

Parameters

- *attempt*: a string that indicates which type of rerun is being processed. If *attempt* is set to "first", a new CSV called d4_reruns.csv is created from scratch. If *attempt* is set to "later", the new results are appended to an already existing d4_reruns.csv. This allows the function to be called multiple times without erasing the progress from previous reruns. *attempt* is set to "first" by default.

Return

- A dataframe containing each rerun candidate's name, state, min year, candid, college major, undergraduate institution, highest degree and institution, work history, sources, and ChatGPT confidence. This dataframe is also output to d4_reruns.csv.

chatFeed(p)

Description

- Uses the ChatGPT API to summarize the biodata from the scraped text and provide a JSON response.

Parameters

- *p*: A dictionary containing the ChatGPT prompt, source URLs, full name, min year, state, and candid of a candidate as keys. The value containing the source URLs is a string array.

Return

- A dictionary containing the ChatGPT response, source URLs, full name, min year, state, and candid of a candidate as keys. The value containing the source URLs is a string array.

getBirthYear()

Description

- Processes the retrieval data, extracts the birth year of the candidate, and converts it into a pandas dataframe and CSV.

Parameters

- No input parameters.

Return

- A dataframe containing each candidate's candid and year of birth.

extractCSV(outputs, promptErrors, variant = "normal", attempt = "first")

Description

- Processes the data gathered in the data extraction stage and converts it into the corresponding pandas dataframes and CSVs. Handles normal responses, prompt errors, and parse errors, which are stored in d1_extractions.csv, d2_promptErrors.csv, and d3_parseErrors.csv, respectively.

Parameters

- *outputs:* an array containing the relevant candidate information for each candidate as the elements in the array. Each element is itself a dictionary containing the ChatGPT response, source URLs, full name, min year, state, and candid of a candidate as keys. The value containing the source URLs is a string array.
- *promptErrors:* an array containing all candidates that encountered prompt errors during the chatFeed function. Each element is itself a dictionary containing the ChatGPT prompt, source URLs, full name, min year, state, and candid of a candidate. The value containing the source URLs is a string array.
- *variant*: a string that specifies if the outputs are being processed normally or as part of a rerun. If *variant* is set to “normal”, the dataframe containing the final results will be output to d1_extractions.csv. If *variant* is set to “rerun”, the dataframe containing the final results will be output to d4_reruns.csv. *variant* is set to “normal” by default.
- *attempt*: a string that indicates which type of rerun is being processed. If *attempt* is set to “first”, a new CSV called d4_reruns.csv is created from scratch. If *attempt* is set to “later”, the new results are appended to an already existing d4_reruns.csv. This allows the function to be called multiple times without erasing the progress from previous reruns. *attempt* is set to “first” by default.

Return

- A dataframe containing each candidate’s name, state, min year, candid, college major, undergraduate institution, highest degree and institution, work history, sources, and ChatGPT confidence. If *variant* is set to “normal”, this dataframe is also output to d1_extractions.csv. If *variant* is set to “rerun”, this dataframe is instead output to d4_reruns.csv.

parse(output)

Description

- Reads the JSON formatted ChatGPT response of a candidate and extracts the full name, college major, undergraduate institution, highest degree and institution, and work history. Candidates whose responses get parsed incorrectly are appended to *parseErrors*.

Parameters

- *output*: a dictionary containing the ChatGPT response, source URLs, full name, min year, state, and candid of a candidate as keys. The value containing the source URLs is a string array.

Return

- If successful, a dictionary containing the full name, college major, undergraduate institution, highest degree and institution, work history, ChatGPT confidence, sources, min year, state, and candid of a candidate as keys. The value containing the source URLs is a string array. If unsuccessful, the return value is an array whose first element is -1.

Data Output

Results

- *b1_searches.csv*:
  - This file stores the results after completing the data search phrase (using the Google Search API). For each candidate, the first name, middle name, last name, full name, state, min year, and candid is recorded.
- *c1_retrievals.csv*:
  - This file stores the intermediary results after completing the data retrieval phase (scraping the sources) in a CSV format. For each candidate, the ChatGPT prompt, sources, full name, state, min year, and candid are recorded.
- *d1_extractions.csv*:
  - This file stores the final pipeline results in a CSV format. For each candidate, the full name, state, min year, candid, college major, undergraduate institution, highest degree and institution, work history, sources, and ChatGPT confidence are recorded.
- *d4_reruns.csv*:
  - This file stores the final results after rerunning candidates who encountered promptErrors [defined in the next section] in a CSV format. For each candidate, the full name, state, min year, candid, college major, undergraduate institution, highest degree and institution, work history, sources, and ChatGPT confidence is recorded, the same as in d1*_extractions.csv*.

Errors

- *c2_scrapingTimeouts.csv*:
  - This file stores all of the candidates who took longer than 250 seconds to scrape. This file can be used directly as the input for the retrieve() function to retry the candidates.
- *d2_promptErrors.csv*:
  - This file stores all of the promptErrors in a CSV format. promptErrors are defined as instances where there was an issue getting a response from ChatGPT. This is usually due to a prompt going over the token limit. The candidate information for these errors is stored in the same format as c1*_retrievals.csv*, which allows the user to run these candidates again directly using the extractAgain() function instead of redoing the Google Searches and scraping.

- *d3_parseErrors.csv*:
  - This file stores all of the parseErrors in a CSV format. parseErrors are defined as instances where there was an issue parsing the ChatGPT response. This is usually due to incorrect formatting in the response (it is told to respond in JSON format, but occasionally doesn't do so). There is room here to get more candidate biodata if these parseErrors can be reused.
- *errors.txt*:
  - This file stores all of the error messages that occur while scraping the web pages and PDFs (biodata() function), creating the ChatGPT prompts (chatPrompt() function), and using the ChatGPT API to summarize the scraped text (chatFeed() function).

Data Accuracy

accuracy.ipynb

Created by Victor Verma
Last edited May 7, 2024

This file was used to assess the accuracy of the biodata outputs using various machine-learning techniques including random forests, k-nearest neighbors, xgboost, and neural networks.

Setup

- *sourceData*: a global variable that stores the relative path to the CSV file containing the relevant candidate output training data as a string.
- *modelMap*: a global dictionary that stores the string abbreviation of machine-learning models as keys, and a dictionary indicating the full name of the corresponding model as values.

Functions

readData(file, include)

Description

- Reads the csv containing the source data and creates a dataframe that contains the desired information.

Parameters

- *file*: a string representing the path to the csv containing the source data.
- *include*: a string that indicates whether to include candidate output rows with no data. If *include* is set to "all", then all output rows are included. If *include* is set to "output", only the rows with some output present are included. If *include* is set to "1998+", only the rows for candidates with min year of 1998 or later are included.

Return

- A pandas dataframe containing the accuracy training data.

prepareData(data)

Description

- Prepares the training data for the machine-learning models by encoding string datatypes and dropping unnecessary columns.

Parameters

- *data*: a pandas dataframe containing the accuracy training data.

Return

- The processed dataframe.

trainModel(X_train, y_train, modelType, include)

Description

- Trains the specified machine-learning model based on the training data.

Parameters

- *X_train*: an array containing the training data for the features.
- *y_train*: an array of binary values representing the training classifications.
- *modelType*: a string specifying the type of machine-learning model being trained. “NN” indicates neural network, “RF” indicates random forest, “XG” indicates XGBoost, and “KN” indicates k-nearest neighbors.
- *include*: a string that indicates whether to include candidate output rows with no data. If *include* is set to “all”, then all output rows are included. If *include* is set to “output”, only the rows with some output present are included. If *include* is set to “1998+”, only the rows for candidates with min year of 1998 or later are included.

Return

- A machine-learning model as specified by the *modelType* and trained on the accuracy data.

evaluateModel(model, modelType, X_test, y_test, confusionMatrix)

Description

- Evaluates the accuracy and f1 score of the model on the testing set and prints the classification report and confusion matrix.

Parameters

- *model*: a machine-learning model.
- *modelType*: a string specifying the type of machine-learning model being trained. “NN” indicates neural network, “RF” indicates random forest, “XG” indicates XGBoost, and “KN” indicates k-nearest neighbors.
- *X_test*: an array containing the testing data for the features.
- *y_test*: an array of binary values representing the testing classifications.
- *confusionMatrix*: a boolean value indicating if the confusion matrix should be printed.

Return

- A float indicating the accuracy of the model on the testing data.

validateModel(validationSet, model, modelType, confusionMatrix)

Description

- Evaluates the accuracy and f1 score of the model on the validation set and prints the classification report and confusion matrix.

Parameters

- *validationSet*: a pandas dataframe containing the validation data.
- *model*: a machine-learning model.
- *modelType*: a string specifying the type of machine-learning model being trained. “NN” indicates neural network, “RF” indicates random forest, “XG” indicates XGBoost, and “KN” indicates k-nearest neighbors.
- *confusionMatrix*: a boolean value indicating if the confusion matrix should be printed.

Return

- A float indicating the accuracy of the model on the validation data.

Classification Accuracy

For all of the models, the accuracy is approximately 72% and the f1 score is approximately 0.80.