CandidateBios Documentation

Table of Contents

Dependencies

APIs

- google_api_python_client==2.80.0
- openai==0.27.0

Efficiency

- tenacity==8.0.1

Machine Learning

- scikit_learn==1.4.0
- tensorflow==2.15.0
- xgboost==2.0.3

Scraping

- beautifulsoup4==4.12.2
- PyPDF2==3.0.1
- requests-html==0.10.0

Utility

- pandas==1.5.3
- python-dotenv==1.0.0

Data Pipeline

pipeline.py

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 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 in order 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 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 ["errors.txt", "extractions.csv", "parseErrors.csv", "promptErrors.csv", "reruns.csv", "retrievals.csv", "searches.csv"].

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

search.py

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 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 searches.csv. This dataframe is also output to 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

- A 2D array containing the relevant candidate information for each candidate as the elements in the array. Each element is itself an array 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

- A 2D array containing the relevant candidate information for each candidate as the elements in the array. Each element is itself an array 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 in order to account for the indexing in pandas dataframes.

Return

- A 2D array containing the relevant candidate information for each candidate as the elements in the array. Each element is itself an array 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*: an array 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 2D array 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. The element 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*: a 2D array containing the relevant candidate information for each candidate as the elements in the array. Each element is itself an array 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 element 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 searches.csv.

Data Retrieval

retrieval.py

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 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 retrievals.csv.

splitCandidates(urls, batchSize)

Description

- Splits the list of candidates to be scraped into sublists of size *batchSize*, with the last sublist containing any leftovers.

Parameters

- *urls*: A 2-D list containing each candidate's Google Search results, first name, middle name, last name, full name, min year, state, and candid as elements.
- *batchSize*: an integer that specifies the size of each batch of candidates to be scraped in a singular instance of the ThreadPoolExecutor().

Return

- A 3-D list containing the batches as elements. Each batch is itself a 2-D list of maximum size *batchSize* containing each candidate's Google Search results, first name, middle name, last name, full name, min year, state, and candid as elements.

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 2D array 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 element containing the top *r* URLs is a string array.

Return

- An array containing plain text scraped from the source URLs, the source URLs, full name, year, state, and candid of a candidate. The element 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*: An array 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 element containing the source URLs is a string array.

Return

- An array containing the ChatGPT prompt, source URLs, full name, min year, state, and candid of a candidate. The element 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 retrievals.csv and scrapingTimeouts.csv, respectively.

Parameters

- *prompt*: a 2D array containing the relevant candidate information for each candidate as the elements in the array. Each element is itself an array containing the ChatGPT prompt, source URLs, full name, min year, state, and candid of the candidate. The element 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 retrievals.csv.

Data Extraction

extraction.py

Setup

- *retrievalData*: a global variable that stores the relative path to 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 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 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 reruns.csv is created from scratch. If *attempt* is set to “later”, the new results are appended to an already existing 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 reruns.csv.

chatFeed(p)

Description

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

Parameters

- *p*: An array containing the ChatGPT prompt, source URLs, full name, min year, state, and candid of a candidate. The element containing the source URLs is a string array.

Return

- An array containing the ChatGPT response, source URLs, full name, min year, state, and candid of a candidate. The element containing the source URLs is a string array.

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 extractions.csv, promptErrors.csv, and parseErrors.csv, respectively.

Parameters

- *outputs*: a 2D array containing the relevant candidate information for each candidate as the elements in the array. Each element is itself an array containing the ChatGPT response, source URLs, full name, min year, state, and candid of a candidate. The element containing the source URLs is a string array.
- *promptErrors*: a 2D array containing all candidates that encountered prompt errors during the chatFeed function. Each element is itself an array containing the ChatGPT prompt, source URLs, full name, min year, state, and candid of a candidate. The element 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 extractions.csv. If *variant* is set to "rerun", the dataframe containing the final results will be output to 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 reruns.csv is created from scratch. If *attempt* is set to "later", the new results are appended to an already existing 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 extractions.csv. If *variant* is set to "rerun", this dataframe is instead output to 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*: an array containing the ChatGPT response, source URLs, full name, min year, state, and candid of a candidate. The element containing the source URLs is a string array.

Return

- If successful, an array 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. The element containing the source URLs is a string array. If unsuccessful, the return value is an array whose first element is -1.

Data Output

Results

- *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.
- *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.
- *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.
- *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 *extractions.csv*.

Errors

- *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).

- *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.
- *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). As of right now, these incorrectly formatted responses are simply stored in the CSV, but eventually, there will be a method to process them again.
- *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 *retrievals.csv*, which allows the user to run these candidates again directly using the extractAgain() function instead of redoing the Google Searches and scraping.

Data Accuracy

accuracy.py

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 and number of trials to perform for the corresponding model as values.

Functions

averageAccuracy(modelType, include, sourceData = sourceData, modelMap = modelMap)

Description

- Calculates the average accuracy of the specified machine-learning model in assessing whether the given output for a candidate row is true or false.

Parameters

- *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 candidate output rows are included. If *include* is set to “output”, only the candidate rows with some output present are included.
- *sourceData*: a string representing the path to the csv containing the source data. This is set to the global definition by default.
- *modelMap*: a dictionary that stores the string abbreviation of machine-learning models as keys, and a dictionary indicating the full name and number of trials to perform for the corresponding model as values. This is set to the global definition by default.

Return

- No return value.

accuracy(modelType, include, sourceData = sourceData)

Description

- Determines the accuracy of the specified machine-learning model in assessing whether the given output for a candidate row is true or false.

Parameters

- *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 candidate output rows are included. If *include* is set to “output”, only the candidate rows with some output present are included.
- *sourceData*: a string representing the path to the csv containing the source data. This is set to the global definition by default.

Return

- A float indicating the accuracy of the model.

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 candidate output rows are included. If *include* is set to “output”, only the candidate rows with some output present 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, dropping unnecessary columns, creating a train-test split, and scaling the feature data.

Parameters

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

Return

- A list containing the training data for the features, a list containing the testing data for the features, a list of binary values representing the output of the training data, and a list of binary values representing the output of the testing data.

trainModel(X_train, y_train, modelType)

Description

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

Parameters

- *X_train*: a list containing the training data for the features.
- *y_train*: a list of binary values representing the training data output.
- *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.

Return

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

evaluateModel(model, modelType, X_test, y_test)

Description

- Evaluates the accuracy of the machine-learning model and prints the test accuracy and classification report (if present).

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*: a list containing the testing data for the features.
- *y_test*: a list of binary values representing the output of the testing data.

Return

- A float indicating the accuracy of the model.

Classification Accuracy

Neural Network Classifier

- The average of 100 for the Neural Network Classifier is slightly above 71% if all rows are included. If only rows with candidate output are included, the accuracy is slightly above 70%.

Random Forest Classifier

- The average of 500 for the Random Forest Classifier is slightly above 74% if all rows are included. If only rows with candidate output are included, the accuracy is slightly above 71%.

XGBoost Classifier

- The average of 1000 for the XGBoost Classifier is around 74% if all rows are included. If only rows with candidate output are included, the accuracy is slightly above 70%.

K-Nearest Neighbors Classifier

- The average of 1000 for the K-Nearest Neighbors Classifier is slightly above 71% if all rows are included. If only rows with candidate output are included, the accuracy is slightly above 70%.