1 Overview

All code used to produce this analysis is available on Github. Code is written in Python 3.7.10 and was run on a Lenovo T480s with Intel Core i7-8550U CPU @ 1.80GHz with 4 cores, 16 GB RAM. Details for creating a conda environment with the same package versions as used in this analysis can be found on in the `requirements.txt` file in the main Github directory. Throughout, the `random_state` parameter when fitting Gensim LDA models was set to 175.

The code for this analysis is modular. It is broken into 16 scripts of general functions contained in `src\func` together with 15 scripts that implement these functions for the *Demography* data, 12 scripts that implement these functions for the Sociology data, and a *filepaths.py* script which contains relative filepaths for accessing code and data and saving results, all contained in `src\scripts`. Tables in the following sections provide summaries of what each of these files contains and Figure 2 summarizes how they fit together.

Two demonstration jupyter notebooks are available in the `src\demos` folder. The first demonstrates the use of the modelling scripts `LdaLogging` and `LdaGridSearch` on a toy example. The second demonstrates the use of the functions in `LdaOutput` and its subsidiaries on the actual data.

Data availability: Data are available upon request to academic authors with approved access to the Scopus API. Due to Scopus use policies, it cannot be posted publicly. The trained models are, at this time, also not posted publicly but can, if one has access to the data, be replicated with random state in Gensim set to 175. All this means that much of the code (especially the scripts) will not actually run with what is available in the public GitHub alone. That said, the general functions should still be applicable to other data. EdaHelper.py is the one function in that directory that is more specific to the data, but it will work for Scopus data in general.

2 Computing Time

The time it takes to train a `Gensim` LDA model depends on a variety of factors. It clearly takes longer to train a model with more documents and more passes (epochs). The values of η and K and the number of iterations per document matter less. Logging the model substantially increases runtimes, though this is largely due to the topic convergence metric being slow. These relationships are plotted for the *Demography* data in Figure 1. Examining the y axis of these plots shows that while with logging, training a model took as long as $\approx$ 10-20 minutes, without logging, each model took under a minute.

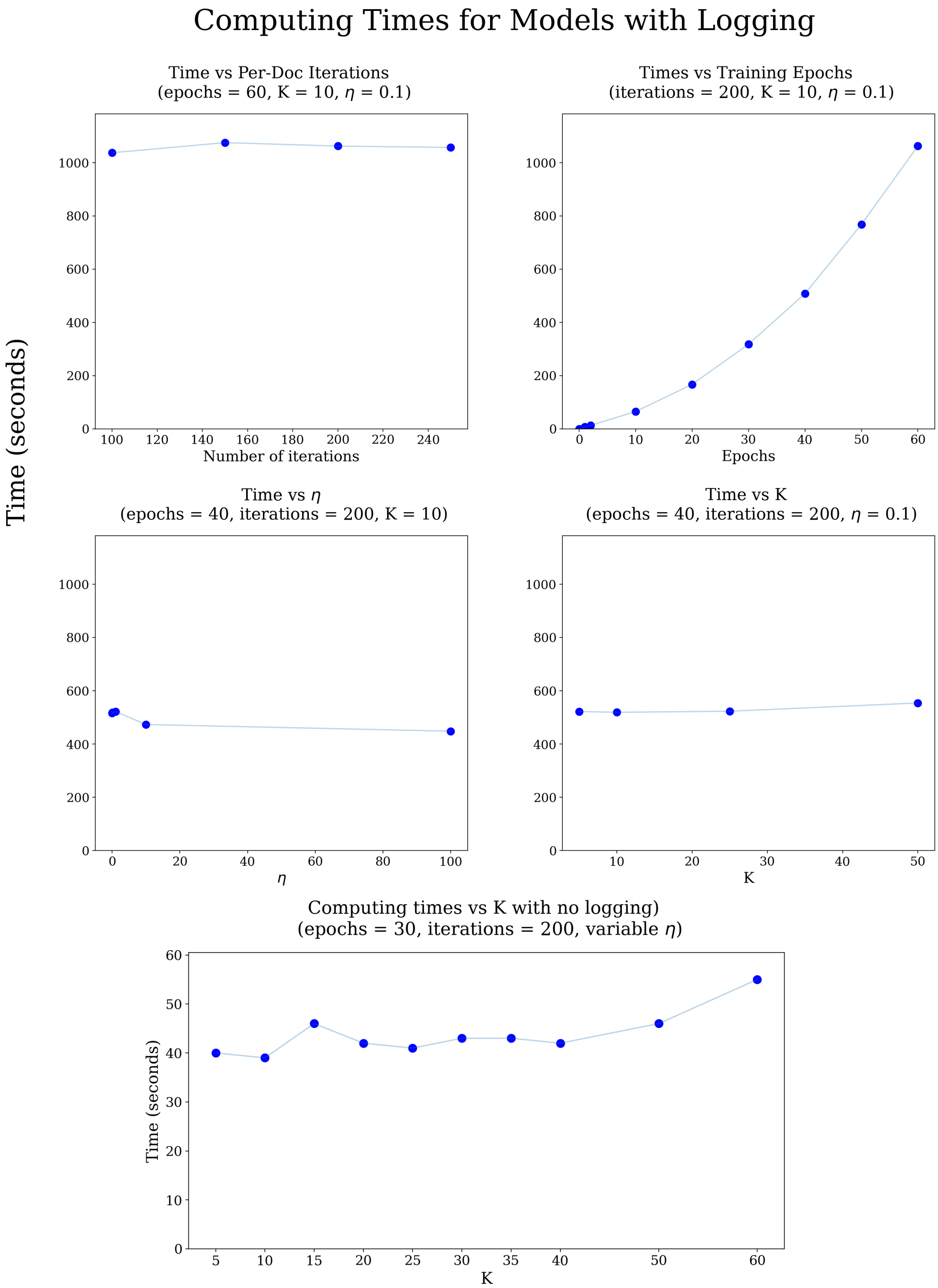

Figure 1: *Summary of Demography computing times for corpus with 2719 documents, vocabulary of 3257 words and 247,319 words total.*

3 Data Processing

File	Description
`EdaHelper.py`	Extract years and detect language for abstracts from original Scopus data; Examine missingness; observation counts by year; abstract missingness by year
`AbstractCleaner.py`	Preprocessing steps 0–5: Copyright removal, basics, "ly" word stemming, bigrams, and low-frequency word removal. Also functions for creating corpus and dictionary, extracting vocabulary from corpus, examing vocabulary and corpus by group, and plotting abstract length boxplots overall or by group

Table 1: *Key functions from data processing scripts: some helper functions excluded*

4 Fitting LDA Models and Grid Search

File	Description
`LdaLogging.py`	`log_setup` creates a log file for LDA and sets up user-specified convergence metric loggers to track coherence, perplexity, topic convergence, and document convergence over epochs (warning: slows down training). `LdaModelLogged` is a wrapper for `gensim`'s `LdaModel` function that includes `log_setup` step. `metric_comparison_plot` can be used to plot metrics over epochs – either individually or all together. A few additional functions allow you to plot mean coherence versus K and create the runtime plots presented above.
`LdaGridSearch.py`	`GridEta` fits LDA models for multiple η values for given K but it is mainly a helper function for `GridEtaTopics`, which fits LDA models for a grid of η and K (can choose a grid of only one K if desired). B both output detailed summary dictionaries, including per-topic metrics, which can be saved using `save_GridEta` and `save_GridEtaTopics` and used for weighted grid search. Avoids need to save all the models in the grid. `WeightedEtaSearch` does weighted grid search for η and fixed K using `GridEta` output and specified weights. `WeightedEtaTopicsSearch` applies `WeightedEtaSearch` to all K-values in output from `GridEtaTopics` and outputs a summary dictionary, which can be fed into `get_overall_best`, along with K-weights, to get a best overall model. `visualize_weighted_eta` produces plots visualizing the impact of weight choice on model selection for η search and `visualize_weighted_eta_plots_grid` does the same for K grid search. `GridEta_scatterbox` and `GridEtaTopics_scatterbox` produce scatterbox comparison plots for different η or different K using output from `GridEta` and `WeightedEtaTopicsSearch` respectively. The second uses the grid search output because it needs to pick an η for each K.

Table 2: *Key functions from model fitting and grid search. Helper functions excluded*

5 Output Analysis

File	Description
LdaOutput.py	core output processing functions. **See below.**
LdaOutput PerTopicMetrics.py	visualize_spread can be used to create scatterboxes for one or all four per-topic metrics available. Uses output of LdaOutput.topic_summarizer() multi_model_pairplot creates grids of scatterplots for understanding relationship between per-topic metrics and topic_metric_order_plot plots the topics of a given model ranked by a metric (e.g. coherence). evaluate_coherence_sensitivity produces plot to examine how coherence scores change with choice of M
LdaOutput TopicSimilarity.py	visualize_topic_difs creates scatterboxes to compare magnitudes of topic differences across omdels. plot_topic_heatmap uses JS or KL divergence to plot heatmap of topic similarity and get_most_sim_topics uses JS or KL to find most similar topics in one model to topics in another. topic_correlations creates topic correlation heatmap
LdaOutput WordPlots.py	topic_relevance_barplot plots top topn most relevant words for a single topic while topic_relevance_grid does this for all topics or for a custom list of topics. Plots inspired by LdaViz
LdaOutput TopicSizePlots.py	topic_size_comparison_plot compares per-topic size options either via barplots or a scatterplot grid. alpha_comparison_plot compares topic sizes to learned α values and metric_size_comparison and metric_size_comparison_grid compare topic sizes to other per-topic-metrics.

Table 3: *Key functions from output analysis (1/2). Helper functions excluded*

File	Description
LdaOutput TimePlots.py	plot_topic_sizes_by_year plots a grid of all topics or a custom subset of them over time for any of the three size calculations. plot_barplot_and_timeplot plots topic size over time for a single topic and also plots the top word barplot for that topic. topic_size_by_year_model_comparison plots time trends for topics of a main model *and* the time trends of the most similar topic (using KL or JS) from a list of other models. plot_topic_by_year plots words within a topic over time using circles of different sizes to represent their per-year probabilities within the topic.
LdaOutput GroupPlots.py	plot_topic_size_by_group can plot barplots grouped by topic, barplots grouped by group, or a scatter-plot grid comparing topic sizes for pairs of groups. plot_topic_by_group plots probability of words within topics for the overall most relevant words in a topic (Termite inspired). plot_topic_words_by_group plots barplots like those in LdaOutputWordPlots.py only now for topics calculated separately for each group. topic_by_group_time_plot plots topics size over time by group either for one topic, a custom list, or all topics in a model. plot_barplot_and_grouped_timeplot plots topic size over time by group for one topic in a grid with that topic's word bar plot from LdaOutputWordPlots.py
LdaOutputDocs.py	get_topic_topdoc_table gets top documents for a specified topic and get_topic_topdoc_by_journal does same only breaking it up by journal. topics_per_doc_summary plots histograms of the number of topics per document and topics_per_doc_summary_by_journal does this by journal. theta_hist plots a histogram of θ_{dk} values for a given topic and theta_hist_by_group does this by group. plot_doc_trajectories plots 'document trajectories' across models for documents from a given model and topic.

Table 4: *Key functions from output analysis (2/2). Helper functions excluded*

6 Helpers

File	Description
Helpers.py	Assorted functions for saving figures, resolving filenames if user attempts to save a file that already exists in directory, figuring out grid sizes when creating grid plots, and getting maximum values in dictionaries or nested lists
Scatterbox.py	Functions for plotting 'scatterboxes' (boxplots with overlayed scatter)
Groupbox.py	Functions for plotting and preparing input for grouped box plots

Table 5: *Key functions from helper scripts*

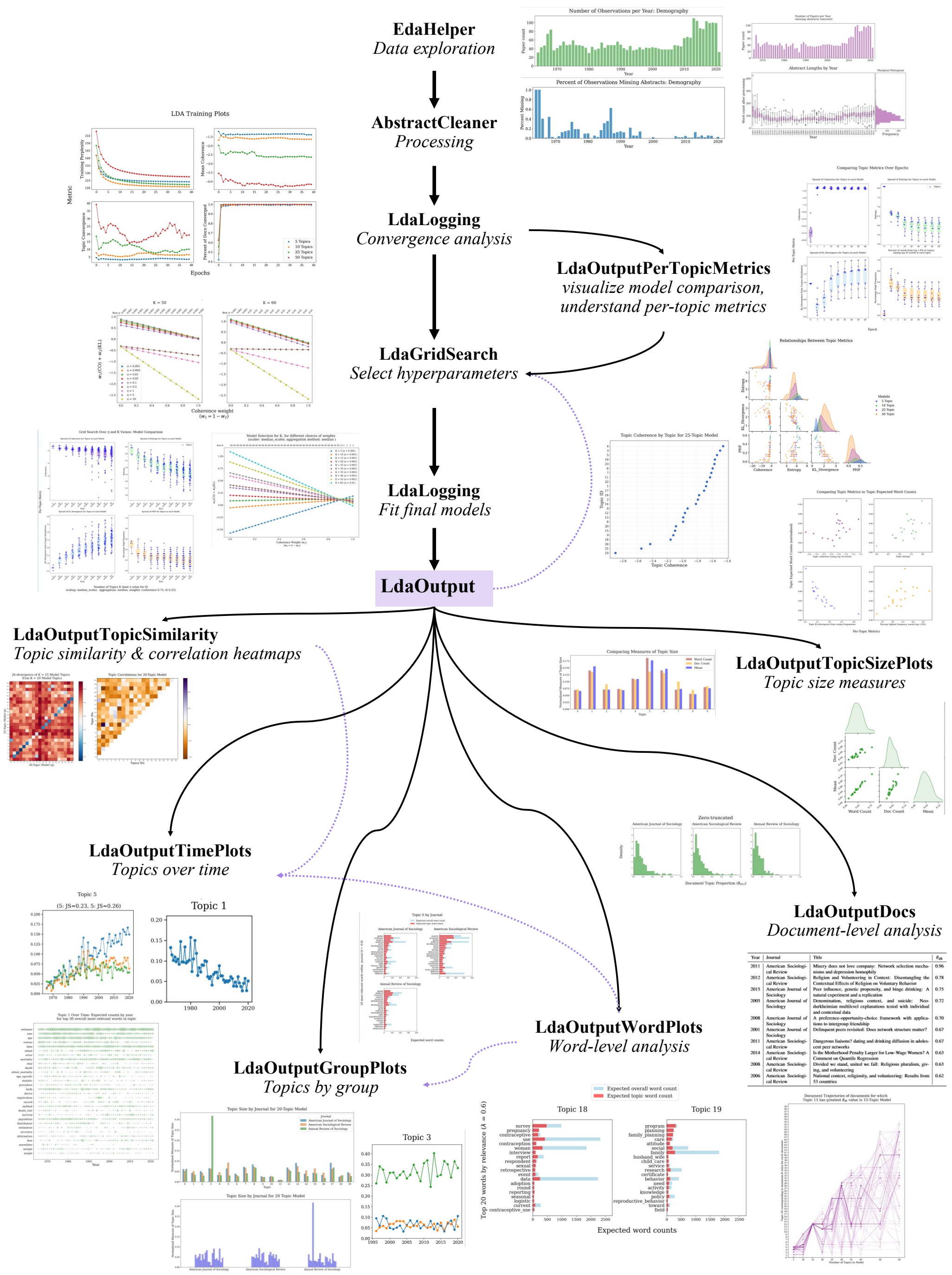

Figure 2: *Illustration of the main function scripts used in the analysis. Light purple lines indicate additional dependencies between the scripts.*

7 Analysis Scripts

The scripts for analyzing the demography and sociology data are similar in overall structure and function. The order in which the scripts should be run to produce the demography analysis is described in Table 6. For sociology, the scripts are named identically except with"Socio" replacing "Demog" in each file name. Scripts with * are not included for the sociology analysis.

Script	Description
`DemogGetData.py`	Extract journal-specific data from larger SCOPUS SOCI data file. *warning: this script does not use relative filepaths and loads data from an external drive
`DemogEDA.py`	Data exploration, year and language extraction
`DemogPreProcess.py`	Pre-processing steps and abstract length plots
`DemogConvergenceAnalysis_models.py`	Convergence analysis: fit models to produce logs
`DemogConvergenceAnalysis_plots.py`	Create plots from logs to analyze convergence
`*DemogCoherenceSensitivity.py`	Brief examination of how mean coherence rankings vary when vary number of top words used to calculate it.
`DemogGridSearch_initial_run.py`	Fit models over grid and saving summaries of each model for use in grid search
`DemogGridSearch_analyze _etaweights.py`	Examine impact of weight choice in η selection
`DemogGridSearch_analyze_Kweights.py`	Examine impact of weight choice in K selection
`DemogGridSearch_final_run`	Fit and save final model for each K in grid while setting η to best value from grid search
`DemogTopicWordPlots.py`	Produce grids of barplots representing all the topics in each model, plus some additional custom grids used in discussion
`DemogTopicAnalysis.py`	Create large assortment of different plots for topic analysis (size, over time, by group etc.)
`DemogExploreDocs.py`	Interactive script to explore top documents for each topic
`*DemogRuntimeAnalysis.py`	Examine time to fit different models with and without logging.
`*DemogDifRandState.py`	Fit model with a different random state to confirm that random state responsible for similar topics having same topic ids across models

Table 6: *Summary of scripts used to run the analysis.*