

README to replicate Revisiting Group Differences in High-Dimensional Choices: Methods and Application to Congressional Speech, (JAE, 2025)

Paul Hofmarcher, Jan Vávra, Sourav Adhikari, and Bettina Grün

Gentzkow, Shapiro, and Taddy (GST, [Econometrica](#), 2019) (henceforth GST) use a supervised text-based regression model to assess changes in partisanship in U.S. congressional speech over time. Their estimates imply that partisanship is far greater in recent years than in the past, and that it increased sharply in the early 1990s. We provide a replication in the wide sense of GST by complementing their analysis in three ways. First, we propose an alternative unsupervised language model, which combines ideas of topic models and ideal point models, to analyze the change in partisanship over time. We apply this model to the Senate speech data used in GST ranging from 1981-2017. Using our model we replicate their results on the specific evolution of partisanship. Second, our model provides additional insights such as the data-driven estimation of evolvement of topical contents over time. Third, we identify key phrases of partisanship on topic level.

You can also find this project on [github](#).

Content

This file contains information on the steps to reproduce the mentioned paper and apply our model to your own dataset. The following steps are described:

- preprocessing the Hein-Daily dataset of congressional speeches,
- estimating TBIP provided by [Keyon Vafa](#),
- estimating our Time-Varying version of TBIP,
- performing the simulation study that imitates the original dataset.

1 Data Preparation

In order to perform the analysis of the congressional speeches, please:

- Download the data `hein-daily.zip` from [Hein-Daily](#),
- Unzip them into the directory `data/hein-daily`,
- Add your own file with stopwords. (ours is copied from [Keyon Vafa](#)),
- Run the two scripts in `code/preprocessing`:
 - `01.supervocab.py` creates the vocabulary that spans all sessions.
 - `02.inputmatrices.py` creates the count matrices and other required files.

If you wish to use TVTBIP for your own dataset, you need to keep the following structure:

- `data/your-data-name` should be the directory containing the data in the original format.
- `data/your-data-name-number` should store both the clean inputs and the estimates from TVTBIP for the time period with this number:
 - For `hein-daily` we have 97, ...,114.
- Within the subdirectory for a time period there should be two subdirectories to store:
 - `clean` with the clean inputs for the TVTBIP, which should contain:
 - * `counts.npz` - the document-term matrix in sparse format,
 - * `author_indices.npy` - vector of author indices declaring who is the author of the corresponding document,
 - * `author_map.txt` - name and surname of the speaker followed by party label in parentheses (one per line),
 - * `vocabulary.txt` - list of words (one per line) that should be common to all sessions for TVTBIP,
 - `pf-fits` with the initial parameter values estimated with Poisson factorization for the first time period,
 - `tbip-fits` with outputs of the TVTBIP, containing subdirectory `param` with estimated parameter values.

2 Estimating TVTBIP

The estimation is performed in `code/tbip/` by `tbip-different_init.py`, which runs only one single session (time period). It is the original `tbip.py` model by [Keyon Vafa](#) but the initialization process is adjusted to reflect our time-varying version. Both were designed for Tensorflow version 1.15, which substantially differs from the newer version.

The key libraries and their versions used in the analysis are:

- `absl-py` 1.4.0
- `matplotlib` 3.3.4
- `numpy` 1.19.5
- `pandas` 1.1.5
- `pip` 21.3.1
- `scikit-learn` 0.24.2
- `scipy` 1.5.2
- `seaborn` 0.11.2
- `tensorboard` 1.15.0
- `tensorflow` 1.15.0
- `tensorflow-estimator` 1.15.1
- `tensorflow-gpu` 1.15.0
- `tensorflow-probability` 0.8.0rc0
- `wordcloud` 1.9.2

More details could be found in `requirements_tf_1.15.txt`.

2.1 Parameter Initialization

The flag `pre_initialize_parameters` provides four options:

- **random** - Initialize model parameters completely at random, which we do not recommend.
- **NMF** - Initialize the location parameters for documents and objective topics by performing Non-negative Matrix Factorisation (NMF), using NMF from `sklearn.decomposition`:
 - Used for the first time period (session).
- **PF** - Initialize the location parameters for documents and objective topics by performing Poisson Factorization (PF), where the implementation from `tbip` is used:
 - Alternative initialization of the first time period (session).
- **previous** - For time period t , initialize model parameters by estimates from the previous time period $t - 1$:
 - Determined by the argument `previous_data`.
 - Objective topic location and ideological topic location are taken.
 - Use objective topic locations as a fixed parameter in `non_negative_factorization` from `sklearn.decomposition` to find reasonable initial values for document locations.

2.2 Computational Resources

The Congressional speeches dataset is quite large, making it difficult to run the analysis on a personal computer. A compute cluster with GPU support has been used to perform the estimation. In directory `code/create_slurms`, we provide code to make `.slurm` files for submitting the individual jobs. In these scripts, we carefully specify that the first session has to be initialized with NMF/PF and the other sessions with the results from the previous session. A file for submitting all jobs at once uses `--dependency=singleton` to compute only one job at a time so that results from the previous session are ready for initialization.

2.3 Output Files

The output `param` directory will contain the following files:

- `document_loc.npy` - Location parameter for document intensities (θ).
- `document_scale.npy` - Scale parameter for document intensities (θ).
- `objective_topic_loc.npy` - Location parameter for objective topics (β).
- `objective_topic_scale.npy` - Scale parameter for objective topics (β).
- `ideological_topic_loc.npy` - Location parameter for ideological topics (η).
- `ideological_topic_scale.npy` - Scale parameter for ideological topics (η).
- `ideal_point_loc.npy` - Location parameter for ideal points.
- `ideal_point_scale.npy` - Scale parameter for ideal points.
- `document_mean.npy` - $\log(E\theta) = \text{loc} + 0.5 \times \text{var}$.
- `neutral_topic_mean.npy` - $\log(E\beta) = \text{loc} + 0.5 \times \text{var}$.
- `negative_topic_mean.npy` - $\log(E\beta \times \exp(-\eta)) = (\beta_{\text{loc}} - \eta_{\text{loc}}) + 0.5 \times (\beta_{\text{var}} + \eta_{\text{var}})$.
- `positive_topic_mean.npy` - $\log(E\beta \times \exp(+\eta)) = (\beta_{\text{loc}} + \eta_{\text{loc}}) + 0.5 \times (\beta_{\text{var}} + \eta_{\text{var}})$.

3 Post-processing the Results

The directory `code/analysis` contains numbered Python scripts that process the outputs from all sessions.

The first file, `01_analysis.py`, resaves the outputs into CSV files: **thetas** (log scale), **betas** (exp scale), **etas** (log scale), but sorted according to the importance for each topic. Moreover, it gathers some data from all sessions into one dataset:

- `ideal_point_speakers.csv` - Ideal points for all speakers separately for each session.
- `ideal_points_all_sessions.csv` - Ideal points for all speakers and all sessions combined (empty if a speaker is not present).
- `speeches_by_speaker.csv` - Data on speeches included within the analysis, including information about the speakers.
- `speeches_by_preprocessed_speakers.csv` - Number of speeches given by a speaker in each session.
- `posneg.cs.csv` - Cosine similarity between positive and negative topics for each topic and session.

Python script `02_plots.py` creates descriptive plots such as:

- Boxplots of Democratic and Republican senator ideological positions over time.
- The evolution of the average partisanship (difference between means of Democratic and Republican positions) over time.
- Averaged cosine similarities of positive, neutral, and negative topics over time.
- Wordclouds containing the top 20 terms used by Republicans and Democrats for each topic.
- Wordclouds of the top 20 neutral terms.

R script `02_plots.R` creates refined plots using `ggplot2`:

- Boxplots of ideological positions for each session, distinguished by political party.
- Average partisanship over time (difference between positions of Democrats and Republicans).

File `03_influential_speeches.py` finds the most influential speeches for selected senators. The influence is measured using the log-likelihood ratio test statistic for testing whether the ideal point is zero.

File `04_list_top_bigrams.py` creates CSV files containing the top 10 terms for each topic, based on objective topics + (-1,0,1) ideological topics:

- `negative_10_bigrams.csv` - Terms used by speakers with ideological position -1.
- `neutral_10_bigrams.csv` - Terms used by neutral speakers (zero ideological position).
- `positive_10_bigrams.csv` - Terms used by speakers with ideological position 1.

4 Simulation Study

Our simulation study uses the estimated parameters from the analysis of Congressional speeches to generate datasets of the same size under different scenarios for the ideological positions of the speakers.

4.1 Generating Counts and Exploration

The directory `code/simulation` contains the script `simulate_counts.py` for generating the counts and `explore_sampled_counts.py` to judge the reasonability of the sampled counts. After some trial and error, we settled on the following approach.

For each time period, we use the corresponding document intensities (θ) from file `thetas.csv`, objective topics (β) from file `neutral_topic_mean.npy` (not `betas.csv` due to shuffling), ideological topics (η) either from `ideological_topic_loc.npy` or restored from `negative_topic_mean.npy` and `positive_topic_mean.npy` (not `etas.csv` due to shuffling).

Since some values of η were quite extreme, we decided to winsorize them to the interval $[-1, 1]$, i.e., values lower than -1 became -1 and values higher than 1 became 1 . All of these are on the log scale, so the Poisson rates are reconstructed as:

$$\exp(\theta + \beta + \eta \cdot \text{ideal})$$

where `ideal` is constructed according to four different scenarios:

- **zero** - All ideal points are zero.
- **party** - Ideal point is -0.5 for Republicans, 0.5 for Democrats, and 0 for Independent senators.
- **diverge** - Zero ideal points until session 100, then increase or decrease by 0.05 for each additional session towards the political party.
- **estimate** - Use the estimated ideological positions for the corresponding session.

Note that for this part, we already used a newer version of Tensorflow. See `requirements_tf_TBIP.txt`.

4.2 Estimating TVTBIP on Simulated Data

All computations were performed on a compute cluster. Hence, we provide the file `simulation.py` to create all `slurm` files needed to submit a job. These are saved in the directory `slurm/simulation-scenario/`. To run all of them, submit `run_all_97_114.slurm`, which estimates TBIP for a given session only after TBIP for the previous session has been estimated.

4.3 Post-processing Results on Simulated Data

First, we needed to collect all the estimated ideological positions into one file, see `01_ideal_to_csv.py`. However, we noticed that the ranges of ideological positions differed from the ranges of ideal points in the original data. Thus, we took the estimates of ideological topics (η) from the original dataset and the simulated dataset, computed the interquartile range (difference between Q3 and Q1) of η , and scaled the estimated ideological positions using:

$$\text{scale_coefficient} = \frac{\text{sim_eta_scl}}{\text{orig_eta_scl}}$$

This adjustment ensured that the summaries and plots were on comparable scales. Rescaled ideal points were saved for each session separately into `ideal_point_speakers_rescaledIQR.csv` and then combined into a single CSV file for all sessions: `scenario_ideal_point_all_sessions_rescaledIQR.csv`, which was saved in the common directory `data/simulation`.

Having all the estimated ideological positions in one place, we proceeded with plotting the differences between Democratic and Republican Senators using `02_plots.py`. Additionally, we used R and `ggplot2` to generate refined plots with `03_plots.R`.