Data from raw sources

Because of file size limits for files in the JHPE Dataverse, the whole /build/ folder would have been too large to upload as one zip folder, so we needed to upload the subfolders of the /build/ folder as separate 7z-zipped folders. E.g. "build_output_text.7z" means /build/output/text/ subfolder. Files in build_output.7z are ones that should be in "/build/output/" folder directly.

The required datasets are included in the package and thus it is not necessary to run any scripts in the /build/code folder. However, we provide here a short(ish) manual for cleaning and processing the plenary records of the Parliament of Finland from 1907-2018 to csv's and further to bag-of-words pickles. Tagged OCR outputs in text format (resulting from running "**`txt-tag-speeches.py`**" in step 2 below) are included in /build/output/text/full_text/tagged folder, and raw pdf's can be downloaded from the website of the Parliament of Finland. In our output csv's, each individual speech is cleaned and linked to a parliament member in year's Kansanedustajamatrikkeli.

1. Speaker-id mapping can be found in `build/output/key-speaker-id.csv`
2. The perhaps most useful intermediate files can be found in `build/output/speeches/linked/` with names `speeches-[YEAR].csv`
3. Bag-of-words data can be found in `analysis/input/bow-1907-2018-tf-100-df-10-ytf-10.csv` (trying to view this file may crash your system as it is very large).

Download data

- `input/pdf-links2.csv`: csv with direct links to pdf files
- `code/download-pdf2.py`: Download plenary record pdf's from Parliament web site for years before 2000. Uses the file above.
- `code/download-pdf-2000s.py`: Download plenary record pdf's from Parliament web site year 2000 onwards.

Processing text files

Parameters defined in the scripts include:

- `firstyear`: the first year to process
- `lastyear`: the last year to process
- `csc`: whether data is being processed in CSC environment; used to conditionally assign paths in the Python files
- `fin`: whether only Finnish speeches should be used in constructing Bag-of-words.

1. `txt-combine-parallel.py`

Description: Once you have downloaded the text files and used OCR, combine text pages output by the OCR process to pdf-level text files.

Input:

- for 1907-2015: `build/output/text/pages/raw/`
- for 2016-2018: `build/output/text/full_text/raw/`

Output:

- `build/output/text/full_text/combined/`

Details:

- 1907-2015: The previous OCR task processes pdf files page by page and outputs text files to the input paths. This program fetches pages with text from a given pdf file and combines it in the right order. It also adds a NEWPAGE_<PAGE_NUMBER> tag in the beginning of each page.
- 2016-2018: Pdfs from 2016 onwards are not being processed with OCR, but text is extracted with `pdf2text` command line tool.

OCR text outputs (currently years 1907-2015): Combine text output pages, tag page beginnings with page nrs pdf2text outputs (currently years 2016-2018): Tag page beginnings with page nrs

2. `txt-tag-speeches.py`

Description: Add tags to plenary record text files to help in extracting speeches in the later programs. See this file for regex patterns.

Input:

- `build/output/text/full_text/combined/[PDF-FILE-NAME].txt`, e.g. `PTK_1990_1.txt`

Output:

- `build/output/text/full_text/tagged/[PDF-FILE-NAME].txt`

Details:

First, clean text for improved tagging:

1. remove special characters (also pipes). Leave %$€§;:-()[] and punctuation.
2. replace text rows with only nrs with space
3. replace en and em dashes with '-'

4. remove line breaks in the middle of the word (i.e. remove character combinations like "-\n")

Then, tag sections of speech based on regex search: DISCUSSION_STARTS, DISCUSSION_ENDS, SUBTITLE, SESSION_ENDS, SESSION_BREAK

Tag beginning of speech: SPLIT SPEAKER, SPLIT SPEAKER CHAIR

3. `speech-extract-speeches.py`
Description: Split data from `.txt` files into speeches and save into csv's.
Input:

- `build/output/text/full_text/tagged/[PDF-FILE-NAME].txt`

Output:

- `build/output/speeches/raw/[PDF-FILE-NAME]-speeches-raw.csv`

Details:

1a. Split into speech sections based on DISCUSSION_STARTS - DISCUSSION_ENDS tags

- if other implicit speech end tags found (SESSION_BREAK, SESSION_ENDS, SUBTITLE), split on them 1b. Split into question hours based on QUESTION_HOUR - SESSION_ENDS tags

2. Find speechsection startpage nr
3. Print section to file
4. Remove parenthetical additions from text

- save to file with filename and speechsection startpage nr

5. Split to speeches based on SPLIT SPEAKER/SPEAKER CHAIR tags
6. Replace pagebreak with XXX and other whitespace characters with " "
7. Save to csv in temp

4. `speech-preprocess-speeches.py`
Description: Clean and preprocess text. Save output both in csv's (processed further) and txt files (used for viewing purposes).

Input:

- `build/output/speeches/raw/[PDF-FILE-NAME]-speeches-raw.csv`, e.g. `build/output/speeches/raw/PTK_1990_1-speeches-raw.csv`

Output:

- `build/output/speeches/preprocessed/[PDF-FILE-NAME].csv`, `build/output/speeches/preprocessed/[PDF-FILE-NAME].txt`

Details:

- Remove:
  - first row of each page if it does not include speaker tag
  - date rows (potentially footers) (regex)
  - procedural text, remaining self-added tags (e.g. `DISCUSSION_STARTS`), pagenrs
- Separate speaker name from the rest of the string
- Detect speech language
- Preprocess speech
  - expand §%€$ symbols to written
  - remove all nonalphanum
  - lowercase
  - remove stopwords
  - stem (with nltk Snowball stemmer)

Linking speeches to speaker ids

5. `speech-link-speaker-parallel.py`
Description:

Match speaker name to the list of year's parliament members. If name is not found, try to find a match from the set of spelling corrected versions of MP names.

Input:

- `build/output/speeches/preprocessed/[PDF-FILE-NAME].csv`

Output:

- `build/output/speeches/linked/speeches-[YEAR].csv`

Column name	Description	Example value
`year`	parliamentary year	`1907`
`file`	the pdf source file for the speech	`PTK_1907_II`

Column name	Description	Example value
speaker	speaker name as extracted from the text.	Ed. Kirves, Ed. Palmön
corr_name	suggested spelling correction for speaker name; empty if the given spelling of speaker's name is found from Kansanedustajamatrikkeli data.	, Palmén
speaker_id	speaker id in mps-ministers.csv.	753, 1431
speech_section_startpage	Start page of the speech in the pdf counterpart	120
language	Speech language as detected by langdetect library	fi
speech_raw	Speech without symbols and with line breaks replaced by "XXX"	SPEAKER Ed. Kirves: Minä olisin hyvällä syyllä toivonut, ettäXXXkaikki ne asiakirjat, jotka koskevat elinkeinolain uudistuksia,XXXtoisin sanoen myöskin entinen
speech_clean	Lowercased and stemmed speech without stopwords, line breaks, symbols and punctuation	hyvä syyl toivonu kaik asiakirj koskev elinkeinol uudistuks tois sanoe myösk entin

6. `speech-collapse-speaker-year.py`

Description: Collapse speech data to speaker-year level. All speeches given during the year are combined under column `all_speech_clean` in the output csv's.

Input:

- `build/output/speeches/linked/speeches-[YEAR].csv`
- `build/output/mps-ministers.csv`

Output:

- `build/output/speeches/collapsed/speaker-year-[YEAR]-fi`: columns `year`, `speaker_id`, `all_speech_clean`

From speeches to bag-of-words

7. `bow-dictionary-yearly.py`

Description: Reads in yearly speech csv's and extracts unique bigrams, their count and the count of speakers using them.

Input:

- `build/output/speeches/collapsed/speaker-year-[YEAR]-fi.csv`

Output:

- `build/output/bow/term_list-[YEAR].pickle`: a list of unique bigrams used during the year
- `build/output/bow/counter-[YEAR].pickle`: how many times a phrase was used during the year
- `build/output/bow/doc_freq-[YEAR].pickle`: how many speakers used a bigram at least once during the year

8. `bow-dictionary-combine.py`

Description: Creates the bigram dictionary. Dictionary consists of all bigrams that survive the thresholds set to term and document frequencies.

Input:

- `build/output/bow/counter-[YEAR].pickle`: how many times a phrase was used during the year
- `build/output/bow/doc_freq-[YEAR].pickle`: how many speakers used a bigram at least once during the year
- Input parameters hardcoded in the script:
  - `dfmin` = 10 # Document frequency minimum, i.e. nr speaker-years for which phrase count > 1
  - `tfmin` = 100 # Term frequency overall minimum
  - `yeartfmin` = 10 # Term frequency minimum for at least one year

Output:

- `build/output/bow/dictionary-[FIRSTYEAR]-[LASTYEAR]-tf-[tfmin]-df-[dfmin]-ytf-[yeartfmin].pickle`

9. `bow-speaker-year.py`

Description: For the bigrams that pass the minimum thresholds, count the usage on speaker-year level.

Input:

- `build/output/bow/dictionary-[FIRSTYEAR]-[LASTYEAR]-tf-[tfmin]-df-[dfmin]-ytf-[yeartfmin].pickle`
- `build/output/speeches/collapsed/speaker-year-[YEAR]-fi.csv`

Output:

- `analysis/input/bow-[FIRSTYEAR]-[LASTYEAR]-tf-[tfmin]-df-[dfmin]-ytf-[yeartfmin].csv`