Word frequency effects in sound change as a consequence of perceptual asymmetries: An exemplar-based model

Code Documentation

Simon Todd, Janet B. Pierrehumbert, Jennifer Hay

Contents

D1 Files

We have provided the code used to run the model. The files included are as follows:

`exemplar_model.py` Used to run the model (see Section D2).

`exemplar_plotter.py` Used to plot model runs (see Section D3).

`combine_dataframes.py` Used to combine summaries from many runs (see Section D4).

`settings.json` Used to change the model settings (see Section D5). The provided file contains the settings used for the final runs of the model in the paper.

`initial_data.json` The initial conditions used to start the model (see Section D6). The provided file contains the initial conditions used for two-category model runs in Section 5 of the paper.

`initial_data_1cat.json` The initial conditions used for single-category model runs in Section 4 of the paper.

`initial_data_2xtypes.json` The initial conditions for a system with twice as many types and exemplars as the one used in the paper, as reported in Section S3.2 of the Supplementary Materials.

`initial_data_biastogether.json` The initial conditions for a system with categories biased together, as reported in Section S3.3.1 of the Supplementary Materials.

`initial_data_minpairs-10pc.json` The initial conditions used for a system where 10% of types are in a minimal pair relation, as reported in Sections S3.4.2–S3.4.3 of the Supplementary Materials. To use this file, also set `"shuffleFrequencies":true` in `settings.json`.

`initial_data_minpairs-all.json` The initial conditions used for a system where all types are in a minimal pair relation, as reported in Section S3.4.4 of the Supplementary Materials. To use this file, also set `"shuffleFrequencies":true` in `settings.json`.

`exemplar_analysistools.py` Tools used to analyze model properties (used internally).

`exemplar_iotools.py` Tools used for reading and writing files (used internally).

`exemplar_representations.py` Code for representations in the model (used internally).

`submit-moab.sh` A sample bash script to submit runs in parallel to a cluster with Moab workload management (see Section D2.1).

`submit-slurm.sh` A sample bash script to submit runs in parallel to a cluster with Slurm workload management (see Section D2.1).

The `.py` files in the distribution should all be kept in the same directory. The `.json` files may be placed in other directories, but the path to those directories will have to be provided in the appropriate places in the settings file / from the command line.

The code was written in Python 2.7.11, utilizing the following packages that are not part of the Python Standard Library: `pandas` (0.18.0); `numpy` (1.10.4); `matplotlib` (1.5.1); `scipy` (0.17.1). We cannot guarantee that the code will work with older versions of these packages.

The code has been written to be compatible with Python 2.6+ and Python 3+, with the help of methods from the `six` and `builtins` packages, which should be part of any standard up-to-date Python installation. If you experience unexplained errors when you run the code, please try installing a new version of Python.

For users who do not already have a Python installation, we recommend Enthought Canopy, which comes pre-installed with the packages required for running the model and associated tools. Canopy can be freely downloaded from `https://store.enthought.com/downloads/`.

D2 Running the model

To run the model, open the command prompt (Windows) or terminal (macOS/Linux) and navigate to the directory where the code is located. Type

```
python exemplar_model.py
```

to run the model with the default settings.

The basic options for running the model are as follows (defaults indicated in parentheses):

`--settings-path (settings.json)` The path to the settings file (see Section D5).

`--logs-path (Logs)` The path to the directory where the summary logs will be saved (see Section D2.2).

`--dump-exemplars` If provided, this option causes the exemplar space to be saved as the model is run, allowing it to be plotted later (see Section D3).

`--dumps-path (Dumps)` The path to the directory where the exemplar space will be saved, if `--dump-exemplars` is provided.

To provide a value for the option, type the option name and the value after the call to run the model, as in the following example:

```
python exemplar_model.py --dump-exemplars --dumps-path NewDumps
```

D2.1 Running the model in parallel

Several advanced options allow different model runs to be split up for parallel processing. The runs are split across pools (corresponding to nodes in a cluster), and each pool is handled by multiple processes (corresponding to processors on a node). The advanced options are as follows (defaults indicated in parentheses):

`--pools (1)` The number of pools to split the runs across.

`--processes-per-pool (1)` The number of processes to launch per pool.

`--pool-number (0)` The number of the current pool.

To run the model in parallel on a single computer, use the `--processes-per-pool` option. To run the model in parallel on a cluster, also use the `--pools` option and tell your workload manager to iterate through nodes providing different `--pool-number` options.

We have provided sample bash scripts to illustrate submission to clusters managed by Moab and Slurm. Note that these scripts assume that a compatible python module has already been loaded and may need to be updated to reflect specifics of your use, e.g. your account details for the cluster.

To submit to a cluster managed by Moab, enter in the terminal:

```
msub -t [0-<x>] submit-moab.sh
```

To submit to a cluster managed by Slurm, enter in the terminal:

```
sbatch -a 0-<x> submit-slurm.sh
```

In both cases, replace `<x>` with the number of pools, minus one. For the scripts we have provided, we set the number of pools to be the total number of parameter combinations multiplied by the number of repeats, divided by four. For the `settings.json` file we have provided, there are 90 total parameter combinations (18 sets of values for $(\sigma, \mu, \beta, \iota, \alpha, \delta)$ $\times$ 5 values for `deltaDiff`) and 10 repeats, yielding 225 ($90 \times 10/4$) pools; thus `<x>` in this case would be replaced with 224 ($225 - 1$).

D2.2 Model output

The basic output of the model is a .csv file containing the following summary information at regular iteration intervals for each category, both across the category as a whole and separately across low- and high-frequency types:

- centroid of the distribution;

- width (standard deviation) of the distribution;
- mode of the distribution;
- peak activation of the distribution;
- skewness of the distribution;
- kurtosis of the distribution.

Additionally, the model calculates the following summary information for every pair of categories in the input:

- width of the overlap between the two categories, measured as the span of the overlapping region;
- number of exemplars whose values fall in the overlapping region, as a proportion of the total number of exemplars in the two categories;
- area contained within the intersection of the two category distributions, as a proportion of the total area underneath either distribution.

The difference between the number of exemplars in the overlapping region and the area contained within the intersection of the two categories is that, for each point in the acoustic space, the number calculation counts all exemplars from both categories at that point, while the area calculation counts only the exemplars from the category with fewer exemplars at that point. This difference is accounted for in the way that proportions are taken, so that, in each case, a value of 1 would indicate that the categories are identical, while a value of 0 would indicate that they have no overlap whatsoever.

When there are multiple runs of the same model (using the setting `runsPerSetting`; see Section D5), they are all combined in a single .csv file, but repeated runs (using the setting `repeatsPerRun`; see Section D5) are stored in separate files, and runs of different models (i.e. with different parameters) are also stored in separate files. These separate files can be combined using `combine-dataframes.py`; see Section D4.

When the `--dump-exemplars` option is provided, at each iteration, each run of the model additionally outputs a log of all of the exemplars in the space at that time, following the format outlined in Section D6. These files are used to create plots of the distributions with `exemplar_plotter.py`, as outlined in Section D3.

D3 Plotting results

If the `--dump-exemplars` option was provided when the model was run, the output dumps of the exemplar space can be used to plot the exemplar distributions and activation fields (at regularly-spaced iterations). For this, run `exemplar_plotter.py`, which takes the following options (defaults indicated in parentheses):

`--settings-path (settings.json)` The path to the settings file originally used to run the model.

`--dumps-path (Dumps)` The path to the directory where the exemplar space was saved.

`--plots-path (Plots)` The path to the directory where the plots will be saved.

`--plot (Activations)` What you want to plot: `Activations` (the activation fields), `Distributions` (the raw exemplar distributions), or `Both`.

`--no-exemplars` By default, plots of the activation field overlay a rug of exemplars at the bottom. Providing this option removes that rug.

`--separate-runs` By default, all separate runs/repeats of the same model parameters are averaged and presented together in one plot. Providing this option instead generates a unique plot for each run/repeat.

`--plot-freqs` By default, the entirety of each category distribution is used for plots. Providing this option instead uses the low- and high-frequency sub-distributions.

`--ymax` The maximum value to be plotted on the y-axis, across all plots. By default, a unique value is chosen for each plot, which may make plot comparison difficult.

For example, to create plots of activation fields for the frequency sub-distributions of exemplars, averaged over many model runs, and hide the exemplar rug, type the following at the command line:

```
python exemplar_plotter.py --no-exemplars
```

Note that creating plots in this way requires the original settings file still to exist. We recommend giving each settings file a unique name and/or putting it in its own directory with the output that it generated.

D4 Combining the results of many runs

The summary logs for repeated model runs (using the setting `repeatsPerRun`; see Section D5) are stored in separate files, as are the logs for runs of different models (i.e. with different parameters). These separate files can be combined using `combine-dataframes.py`, provided they are all in the same directory. There are two options, as follows (defaults indicated in parentheses):

`--dir (Logs)` The name of the directory where the summaries are contained.

`--outcsv (combined_summary.csv)` The name that the processed file will be saved under.

In processing, the .csv files in the provided directory are combined, and the unique[1] parameter values they were generated with are saved as columns in the .csv. After processing, all processed files are deleted, and the new summary .csv is saved in the same directory. This combined summary file can be imported into statistical analysis software for investigating the effects of different parameter settings.

Note that **all** .csv files in the directory will be processed; for this reason, please ensure that summaries for different investigations each are placed in their own, unique folder.

[1] If you want to include non-unique parameter values in the output .csv, those parameter values have to be provided as lists with single elements in the settings file when running the model (see Section D5).

D5 Changing model settings

The file `settings.json` contains the settings used in launching the model, which are as follows:

`dataPath` The path to the initial conditions file (see Section D6).

`iterations` The number of iterations to run the model for.

`dumpEvery` The number of iterations between successive logs of the model.

`runsPerSetting` The number of times to run the model with the same setting, within the same call to the code. Cannot be split across processes.

`repeatsPerRun` The number of times the model runs will be repeated, across different calls to the code. The logs for each repeated run will be marked separately. Repeated runs can be split across processes, but are slow to initialize; for this reason, it is efficient to balance `runsPerSetting` and `repeatsPerRun`.

`sigma` The initial width of categories.

`mu` The initial distance between categories.

`beta` The size of the bias per iteration.

`iota` The amount of noise added to productions.

`alpha` The size of the perceptual window for activating exemplars.

`delta` The discriminability threshold (for median-frequency types). When the discriminability threshold is a function of type frequency, this parameter sets the value of λ.

`deltaDiff` The difference between the discriminability threshold for high/low-frequency types and the discriminability threshold for median-frequency types (the threshold decreases linearly with frequency; for example, if `delta`=0.5 and `deltaDiff`=0.5, then the lowest-frequency types have discriminability threshold 1, the median-frequency types have discriminability threshold 0.5, and highest-frequency types have discriminability threshold 0). In the Supplementary Materials, we refer to this parameter as ϕ.

`tau` The typicality threshold (for median-frequency types). When the typicality threshold is a function of type frequency, this parameter sets the value of κ.

`tauDiff` The difference between the typicality threshold for high/low-frequency types and the typicality threshold for median-frequency types (the threshold increases linearly with frequency; for example, if `tau`=0.5 and `tauDiff`=0.5, then the lowest-frequency types have typicality threshold 0, the median-frequency types have typicality threshold 0.5, and highest-frequency types have typicality threshold 1). In the Supplementary Materials, we refer to this parameter as ψ.

`epsilon` The size of the entrenchment window.

`chi` The factor determining the extent to which nonwords compete with real words during the discriminability evaluation. Must be between 0 (no competition) and 1 (competition as expected based on category activation).

`spaceMin` The smallest possible value in the exemplar space.

`spaceMax` The largest possible value in the exemplar space.

`spacePoints` The number of discretized points to create in the exemplar space, linearly spaced between the minimum and maximum values.

`shuffleWords` A Boolean flag indicating whether exemplars should be shuffled across types in the same frequency bin on different runs. This should be `true` unless you want to investigate results that are particular to a given initial assignment of exemplars to types.

`shuffleFrequencies` A Boolean flag indicating whether frequencies (and corresponding sets of exemplars) should be shuffled across types in the same category on different runs. This should be `false` unless you want to explore the consequences of introducing minimal pairs, independent of the frequency ratios between types in minimal pair relations.

`freqBins` A dictionary specifying the type frequencies that belong in each frequency bin (used for shuffling exemplars and for generating sub-distribution summary statistics).

The file we provided with the distribution contains the settings we used for the final models in the paper. Note that, in all investigations in the paper, we kept `epsilon` and `tauDiff` both fixed at 0.

To perform runs for different values of a parameter, provide a list of values for the parameter. For example,

```
"sigma":  [0.6, 0.8, 1.0]
```

would perform runs for $\sigma \in \{0.6, 0.8, 1.0\}$.

When multiple different parameters are given multiple values in this way, all combinations of those parameter values are run. For example,

```
"sigma":  [0.6, 0.8, 1.0],
"iota":  [0.5, 0.6]
```

would perform runs for six pairs: $(\sigma, \iota) \in \{(0.6, 0.5), (0.8, 0.5), (1.0, 0.5), (0.6, 0.6), (0.8, 0.6), (1.0, 0.6)\}$.

To include only certain combinations of parameters, they can be specified jointly. For example,

```
"sigma,iota":  [[0.6, 0.5], [0.8, 0.6]]
```

would perform runs for only two pairs: $(\sigma, \iota) \in \{(0.6, 0.5), (0.8, 0.6)\}$.

The parameters that have multiple values are those that will be noted in the creation of summary logs. To also include parameters with only a single value, enclose that single value in square brackets. For example,

```
"tau":  [0.1]
```

would perform runs only with $\tau = 0.1$, but would create a column for `tau` populated by 0.1 when later running `combine_dataframes.py`.

D6 Initial conditions

The initial distribution of exemplars across types and categories is provided by `initial_data.json`. This file is formatted as a dictionary with category names as keys, mapping to category dictionaries. Each category dictionary has a `bias` key, which indicates how much relative bias the category should be subjected to (i.e. how many times β should be added to productions from this category), an `adjustment` key, which indicates how much relative adjustment the category should receive at initialization (i.e. how many times each exemplar in the category should have μ added to its value

when first loading the system), and a `words` key mapping to word type dictionaries. The keys of the word type dictionary are the names of word types, which map to dictionaries specifying the `frequency` of the type and list of `exemplars` (values) used to initialize the type (the number of entries in this exemplar list should be equal to the frequency of the type).

For example, in `initial_data.json` distributed with the code, there are two categories (outer keys): `Pushee` and `Pusher`. The `Pushee` is subject to no bias (`"bias": "None"`) and no initial adjustment (`"adjustment": 0`), and contains word types such as `Pushee_F9T5` and `Pushee_F2T8`. `Pushee_F9T5` has frequency 9 (`"frequency": 9`), and 9 initial exemplars, with values 0.1, -1.6, -0.2, 0.1, 0.1, 0.4, 0.2, 0.3, and -1.0.

Dumps of the exemplar space used to generate plots have the same format as this file. The `adjustment` field is set to 0 for every category during dumps, so that the dumped files can be re-used to initialize new runs exactly as-is, without interference from the parameter μ.

To properly utilize the parameters τ and μ (controlling initial category width and distance respectively), each initial distribution should have standard deviation 1 and centroid 0, and one of the categories should have an `adjustment` value of ± 1 (here, `Pusher` has `adjustment` -1, indicating that μ is subtracted from each exemplar's value at initialization).

The file `initial_data.json` distributed with the code contains the initial distributions used for the two-category simulations in the paper, while `initial_data_1cat.json` contains the initial distributions used for the single-category simulations in the paper. The other four files contain the initial distributions used for additional runs reported in Section S3 of the Supplementary Materials, where they differ from the initial distributions using in the paper. `initial_data_2xtypes.json` contains the initial distributions for a system with twice as many types and exemplars as the system in the paper (Section S3.2 of the Supplementary Materials). `initial_data_biastogether.json` contains the initial distributions for a system with two categories (the two from the paper) that are biased together (Section S3.3.1 of the Supplementary Materials). `initial_data_minpairs-10pc.json` contains the initial distributions for a system where 10% of types are in a relevant minimal pair relation (Sections S3.4.2–S3.4.3 of the Supplementary Materials). `initial_data_minpairs-10pc.json` contains the initial distributions for a system where all types are in a relevant minimal pair relation (Section S3.4.4 of the Supplementary Materials).