NION's Presidential Ngram Viewer

Aaron Huffman Egan McComb Richard Perez

20 March, 2012

Contents

1 Functionality

1.1 Obtaining and Creating the Database

Our application stores a local, processed database of all the presidential state of the union speeches, their years, and the name of the president that gave them. This database can be loaded with `load('data.mat')`, and it introduces the arrays `names`, `years`, and `speeches`. To obtain this database, we call the argumentless, void `get_db()` function that calls the other functions of the database module, namely `parse_db()` and `create_db()`. For more details on the database module, refer to B.2.2 p.7. In normal use, it should never be necessary to regenerate the database.

1.2 Filtering Database by Year or President

Our application provides two ways to obtain a subset of the speech database, which is to filter by year or president. To do the first, we call the function `filter_by_year()`, taking two arguments: a row vector of years and the `years` array from the database. It returns the appropriate speech indices. To do the second, we call the function `filter_by_pres()`, taking two arguments: a row vector of presidential indices, and the `names` array from the database. Presidential indices are the index of unique presidents that gave speeches. The function returns the appropriate speech indices, as well as the number of speeches given by each president indexed, and the names thereof. For more information on the filtering module, refer to B.1.1 p.5.

1.3 Data Mining

Our application allows an arbitrary subset of speeches to be queried, each with an arbitrary set of regular expressions. To obtain the frequencies of matches to each regular expression in each of the speeches desired, we call the function `get_freqs()`, which is itself a frontend to the data backend function `regexp_freq()`. This former function takes three arguments to determine its calls to the latter: the speeches to query (probably a subset of the `speeches` array produced with the indices returned by the filtering module), a vertical cell array of regular expressions, and an array of Boolean values to determine whether each regular expression is treated case sensitively (0) or not (1). The function returns a numeric array with one record per speech, one field per regular expression.

1.4 Plotting

Our application employs one of two plotting procedures, depending on whether the database was filtered by year or president. For these respective states, it uses `plot_bar_year()` or `plot_bar_president`, respectively. Both take the data returned from the data backend module and plot multiple bar chart series on

appropriate axes with a legend. The latter applies one of two collation methods to combine data for each president: average or sum. For more information on the plotting module, refer to B.1.2 p.5.

1.5 Loading Files

Due to the generality of our filtering, data mining, and plotting modules, we can accept arbitrary user input via the importation of files from anywhere in the directory tree. For both filtering by year and president, the user can load plaintext files to construct the filter. These are done with unique, properly sorted records, with one year or presidential index per line, respectively. Our I/O module does do some basic checking and does pass the records through `unique()`, but take care in the construction of these files. An arbitrary length list of regular expressions can also by loaded from a plaintext file.

2 Frontend

We provide two frontends to our application, a graphical user interface, and a command line interface.

2.1 Graphical User Interface

The graphical user interface implements access to the full array of the functionality described in the previous section, *with nothing hard-coded*. It provides the option to set one, two, or three regular expressions, or to load an arbitrary list of them from a file. Each of these can be set to be case insensitive, to be treated as a phrase (*i.e.* a sequence of words separated by arbitrary white space), and to be treated as a literal string (*i.e.* all regular expression metacharacters are escaped). Furthermore, it provides options to set filters by year or by president, either via ranges, or via the importation of files of arbitrary lists.

With these parameters set, the graphical user interface allows the user to plot the mined data or save it as a `*.mat` file. By default, it is set to search all the presidents case sensitively. It also allows the user to update the database should she want to do that, and has quick access to regular expression documentation. All parameter and file input errors are handled so the user should not be interrupted by crashes, and the program will detect a missing database file and recreate it. A status line keeps the user informed of any errors or warnings, and pending operations. Since we make use of MATLAB's customary figure window, the user has access to the full array of plot tools offered, including importantly the ability to zoom plots and save them as images.

2.2 Command Line Interface

Included are two frontend functions, `get_plot()` and `get_data()`. The former function produces a plot and is called as follows:

```
get_plot(FILTERMETHOD,FILTER,P_BOOL,REGEXPS,C_BOOL)
```

It filters the speeches via `FILTERMETHOD` — `'y'` or `'p'` for years or presidents, respectively — with filter vector `FILTER`. The function uses `P_BOOL` when filtering by president to determine whether averages or sums will be plotted. It then applies the regular expressions in the vertical cell array `REGEXPS`, with each expression being matched either case sensitively (0) or insensitively (1) as determined by the corresponding element in the Boolean vector `C_BOOL`. The latter function is called in the same manner as the former, but returns an array of frequencies from the data backend module rather than plotting them.

A Division of Labor

The division of labor breaks down as follows; bold indicates main contributor(s):

Functionality Planning: Aaron Huffman, Egan McComb, and Richard Perez.

Code Architecture: Aaron Huffman, **Egan McComb**, and Richard Perez.

Database Format: Aaron Huffman and Egan McComb.

Personnel Control: Aaron Huffman and **Egan McComb**.

Code Control: Egan McComb.

Database Backend: Aaron Huffman and **Egan McComb**.

Data Backend: Egan McComb and **Richard Perez**.

Filter Backend: Aaron Huffman and Egan McComb.

Plot Backend: Aaron Huffman.

GUI Frontend: Aaron Huffman, **Egan McComb**, and **Richard Perez**.

GUI Backend: Aaron Huffman, **Egan McComb**, and Richard Perez.

Exception Handling: Aaron Huffman and **Egan McComb**.

TUI Frontend: Egan McComb.

Documentation: Aaron Huffman, **Egan McComb**, and Richard Perez.

Video: Aaron Huffman, Egan McComb, and **Richard Perez**.

B Individual Contributions

B.1 Contributions by Aaron Huffman

B.1.1 Filtering Module

The filtering module's primary job is to handle user-defined filtering parameters and convert those parameters into a vector of relevant speech indices. Consistent with two possible filtering paths, by year or by president, the filtering module also collects other useful information needed by other modules in the program. The module consists of two functions, operating independently, which are called based on filtering parameters.

`filter_by_year()` The `filter_by_year()` function finds speech indices from year-based inputs. This function, as its name suggests, iterates through input years, finds the indices of those elements in the `data.mat` vector `years`, and returns those indices. It also checks for null or invalid inputs and returns appropriate errors if necessary.

`filter_by_president()` The `filter_by_pres()` function finds speech indices from president-based inputs. This function processes a vector of (chronologically sorted) president numbers (1 being George Washington, and so on) and outputs relevant speech indices. Other optional outputs include the number of speeches each president within the parameters gave, as well as a character array of all the relevant president names. These outputs are required by the collate pres() function in the plotting module (see B.1.2 p.5).

B.1.2 Plotting Module

The plotting module includes three functions designed to tailor and display regular expression frequency results using the user-specified method. These three functions constitute two filtering types, by year or by president, with `collate_pres()` functioning as an auxiliary component of `plot_bar_pres()`.

`plot_bar_year()` The `plot_bar_year()` function is explicitly designed to process filtering inputs wherein relevant data must be displayed graphically as individual speeches. It requires input of a frequency array as created by `get_freqs()`, along with the filtered speech indices, the search parameters, and the `years` array from `data.mat`. It then plots these data as bars on a single graph, with the largest bars plotted first to avoid obscuring smaller bars behind. Values of the x-axis are set as years within range, and duplicates are given relevant suffixes. A legend is included featuring the search parameters processed as regular expression, with color being assigned for each string.

`plot_bar_pres()` The `plot_bar_pres()` function is designed to handle filtering inputs where outputs should be displayed by president. It requires input of a

frequency array, as created by `get_freqs()` and altered by `collate_pres()`, along with president indices, the search parameters, and the `names` array from `data.mat`. This data is plotted as bars, with the largest bars plotted first to avoid obscuring smaller bars. Values of the x-axis are set as individual presidents, and duplicate presidents — *i.e.* presidents that gave nonconsecutive speeches or different presidents with the same name — are given numeric tags as suffixes. A legend is included featuring the search parameters processed as regular expressions, with color being assigned for each string.

`collate_pres()` The `collate_pres()` function is designed to alter the frequency array from `get_freqs()` in such a way that all speeches for a given president and search parameter are averaged or summed, as per the selection of the user. The function requires input of the frequency array, the president indices input by the user, and the number of speeches each president in the range gave, all returned by `filter_by_pres()`. Additionally, it requires a Boolean argument specifying sums or averages. The output is another frequency array, which can be interpreted by `plot_bar_pres()`.

B.1.3 Graphical User Interface

Aaron Huffman played a central role in authoring the GUI backend, and is personally responsible for its core functionality. On startup, the GUI checks for a valid `data.mat` database, sets GUI strings according to the values within that database, and loads the GUI figure. At that point, the user is free to enter any combination of regular expressions, either from text box inputs or from a plaintext file. The user may also set the parameters for any number of regular expression handling functions, allowing case sensitivity, accurate phrase handling, or literally interpreted strings. Also determined from the GUI are broad filtering parameters (by year or by president) and then more precise filters, such as a single president, a range of presidents, a range of years, or any number of president indices or years from a plaintext file. Any plain text files containing a single column of years or president indices will be accepted; all others will result in an error displayed in the GUI itself. In the backend, these inputs determine flow underneath the "Plot" function and the "Save" function, wherein the search parameters are modified if necessary, and the filtering, searching, and plotting functions are applied if applicable. If the "Save" button is used, the frequency data is saved into a `*.mat` file of the user's choice, rather than plotted. The "Regex Help" button loads MATLAB's `regexp()` documentation. The "Update Database" button will run the database creation module and save the results in `data.mat`.

`regexp_asphrase()` The `regexp_asphrase()` function was developed to interpret user input(s) as connected and isolated phrases. To that end, it iterates through every populated input regular expression, adding whitespace metacharacters at the beginning and end of the string, and replacing all internal whitespace with a more accurate metacharacter equivalent.

B.2 Contributions by Egan McComb

B.2.1 General Contributions

Egan McComb devised the overall architecture of the application, and coördinated the work of all the group members to that end. He reviewed code for style and documentation consistency, as well as for optimization.

B.2.2 Database Module

The database module consists of five functions that produce a local mirror of select data held online at `http://www.presidency.ucsb.edu/sou.php`. Two of these functions were separated for modularity: `surlread()` and `detox()`. The other three functions perform separate stages of the procedure required to generate the local database.

`get_pids()` The first stage of the database creation procedure is to correlate the PHP numeric page identifiers with the year of the speech to which they refer. The website index uses these disordered numeric identifiers to link to the individual speeches. The `get_pids()` function begins by retrieving the index and discarding the data before the opening `"menu"` anchor. It then finds the starting indices and match string for every opening hyper-reference anchor that contains a page identifier. Since the matches are returned as a cell array, it is cast to a character array before it is iterated through. The length of each match is used to account for variation in the length of the page identifier, and index offsets are produced for the location of the desired data based thereon and the index of the appropriate matches therein. Having extracted every page identifier and its corresponding year, the output array is finally sorted by year before it is returned.

`parse_db()` The second stage of the database creation procedure is to retrieve the individual speeches and populate the records properly. Taking the array returned by `get_pids()` as its only input argument, it iterates through each page identifier, retrieving the data it identifies. It extracts the president's name from the `title` element via two regular expressions. The year is simply copied from the input array, and the speech is extracted from the `span` element with the `"displaytext"` class via two regular expressions. The speech is then filtered by the `detox()` function to be as clean as possible. Originally, the speeches were stored as vertically concatenated strings, but this proved to be inefficient for the data backend functions due to the filler spaces added, so cell arrays were used instead. Finally, the arrays containing the names, years, and speeches, are returned.

`create_db()` The third and final stage of the database creation procedure is to save the data into a `*.mat` file. Unfortunately, the sort implemented in `get_pids()` fails when two different presidents gave a speech in the same year.

In order to ameliorate this, `create_db()` implements another sort. It begins by finding the initial index of each set of duplicate years. Since only two presidents can give a speech in one year — barring some extreme turmoil or tragedy — the algorithm proceeds to check whether the second president in each set of duplicates is isolated: if it is, its corresponding record is switched with the previous one. Once the records have been properly sorted and transposed to proper orientation, the final database file is saved.

`surlread()` The `surlread()` function just calls MATLAB's `urlread()` function with some error catching.

`detox()` The `detox()` function applies some important filters to clean up the speech text. It replaces HTML markup with spaces, and replaces HTML escape sequences with their ASCII equivalents. Finally, it removes bracketed parentheticals, since these are not technically a part of a given speech.

B.2.3 I/O Module

Egan McComb coördinated the development of the I/O module in the graphical user interface backend.

`load_file()` The `load_file()` function handles the importation of a user's arbitrary length list of regular expressions. It is essentially a frontend to MATLAB's `textscan()` function, though it trims duplicate records and sorts.

B.2.4 Graphical User Interface

Egan McComb helped in the graphical and functional design of the graphical user interface, as well as in the assignment of handle tags. He devised the algorithm for retrieving and constructing an arbitrary list of regular expressions that can be both disabled and enabled. Additionally, he designed all the exception handling, including notably the handling of missing or improper manual and file inputs. Finally, he wrote a couple backend functions, detailed below.

`regexp_asliteral()` The `regexp_asliteral()` function escapes all the regular expression metacharacters, in case the user does not want to use regular expression syntax but rather prefers to search for literal strings including symbols like the period or the question mark.

`isyear()` The `isyear()` function tests an input year to see if it is a valid year in the `years` array. This function is used to check the validity of year range text input in the filtering section of the graphical user interface.

B.2.5 Command Line Interface

As a supplement to the graphical user interface, frontend functions for the entire library were written.

`get_plot()` The `get_plot()` function calls the filtering, data backend, and plotting modules to allow the user to plot desired data with one function invocation.

`get_data()` The `get_data()` function is equivalent to `get_plot()`, but rather than plotting, it returns an array of the data mined by the data backend in conjunction with the filtering module.

`get_db()` The `get_db()` function is a frontend to the database module and handles the creation of the database with one, argumentless call.

B.3 Contributions by Richard Perez

B.3.1 Data and Statistics Module

Richard Perez contributed by writing upper level and lower level functions of the data backend. His work was instrumental to the functionality of the program because the backend functions determine the frequency of matches of user provided regular expressions in each speech of the desired range of speeches.

`get_freqs()` The `get_freqs()` function takes a cell array of strings, a cell array of regular expressions, and an array of Boolean values for case sensitivity, and returns an array of frequency values for the matches of each regular expressions in each string by calling the `regexp_freq()` function. These frequency values are stored in an array, and from there, the plotting functions can extract and return the frequency values to the user in a graphical manner.

`regexp_freq()` The `regexp_freq()` function takes a string, a regular expression, and a Boolean value (either 0 for case sensitivity, or 1 for case insensitivity) and returns the number of times the regular expression is matched in the string.

B.3.2 Graphical User Interface

Richard Perez ensured that the GUI is as compact as possible by aligning and rearranging each feature thereof. He did this for both the prototype and final versions of the interface. Thus, the GUI looks neat in appearance, which is no small feat due to MATLAB's poor GUI design tools.

C Agreement

All team members have read the task summaries contained in this report and have been given an opportunity to comment.