**Chapter 4: Extracting and correcting the last name of the speakers**

Extracting the names of the speakers from the raw text file is a most important task. The name of the speaker is a critical meta variable for stratifying the text analysis. The textual content can be expected to vary with speakers or groups of speakers (such as speakers belonging to the same political party or coming from the same state). A stratified text analysis can tell us about important topics that are being raised by different groups of speakers.

Considerable effort went into formatting the raw text file to make the automatic extraction of speakers possible. Each speech started on a new line which allowed us to separate easily the corpus into individual speeches. Furthermore, each speech started with the word "Mr" as its first word and the speaker's last name as the second word. This structure helped us extract the speaker's last name automatically. Unfortunately, typos were made and numerous speaker names extracted from the second word of each speech were misspelled. But misspellings could be corrected by comparing the extracted (and possibly incorrect) second word with the correct last names of all speakers from the 39th Congress. Such a master list of all senators and house members was available to us. We also learned that the second word of a speech occasionally included terms such as "president", "secretary", "speaker", "clerk", "chairman", "presiding", "members", and "senators". These terms occurred when a speech was given by the president, the secretary, or the clerk of either the senate or the house. The above eight terms were added to the master list that was used for similarity matching. A similarity index of an extracted speaker's name with each of the terms on that master list was then computed, and the term on the master list that matched the extracted name most closely was substituted for the misspelled name –

provided that the similarity index exceeded a certain threshold. An extracted word for which the largest of its similarity scores was smaller than the threshold was labeled as "UNKNOWN".

For that we use the function stringsim() of the R package stringdist. We use the longest common substring (lcs) method of the function stringsim() when calculating the similarity index between two strings. Letters in the longest common substring of ordered letters are paired between the two strings, whereas the remaining letters that are not in the longest common substring are left unpaired. The number of unpaired letters is referred to as the lcs-distance. The similarity index between two strings is defined as one minus the ratio of the number of unpaired letters (the lcs-distance) and the number of total letters of the two strings. For example, the similarity index between “buchanan” and “buckland” is given by 1 - (6/16) = 0.625; the longest common substring of ordered letters is “bucan”, there are 6 unpaired letters (the letters h, a, n, k, l, d) and 16 letters in total.

For each misspelled name, the program goes over all terms in the master list and calculates a similarity index for each pair. The term in the master list with highest similarity index is picked as the closest match of the misspelled name. For information on this function, see <https://cran.r-project.org/web/packages/stringdist/stringdist.pdf> and

<https://github.com/cran/stringdist/blob/master/R/stringsim.R>

A threshold on the similarity index is used for correction. We found that matches for a similarity index above 70% provides a reliable correction. Thus, misspelled names with a similarity index above 70% are replaced by their closest name, while misspelled names with a similarity index below 70% are marked as “UNKNOWN”. The automatic correction was quite successful. Of the approximately 8,300 misspelled speaker names (among the almost 100,000 speakers), more than 5,500 could be corrected through such similarity matching. We could correct about 66 percent (5500/8300=0.663) of the misspelled names. The remaining 2,700 speakers (amounting to about 3 percent of all speakers) were marked as “UNKNOWN”.

We noticed yet another problem with the extracted last name. There were several instances of Senate and House members having the same last name: Johnson (Senator Reverdy Johnson, a conservative senator from Maryland, and Philip Johnson, a House representative from Pennsylvania), Henderson, Davis, Harris, Williams, Dixon, Morgan, Wright, Ross, Patterson, and Wilson (with two different house members James Wilson from Iowa and Stephen Wilson from Pennsylvania, and Senator Henry Wilson from Massachusetts). Speeches in our master text file were arranged chronologically for consecutive Senate and House sessions. This structure allowed us to automatically infer the Senate/House association of speakers with the same last name by counting the number of unambiguous senate members that spoke within a window of 30 prior and past speeches around the ambiguous last name in question. The last name was allocated to the Senate for high counts, and to the House for low counts. We experimented with the count cutoff and found that for an optimal cutoff very low misclassification errors were possible.

The three steps – (1) extracting the speaker's last name from the second word of the speech; (2) correcting the last name by comparing the extracted word with terms on the master list; and (3) determining the affiliation of Senate and House speakers with the same last name – resulted in a quite reliable meta information on the speaker of each given speech. Not all speakers could be identified (we found that roughly 3 percent of the speakers could not be identified) and we may have misclassified a very few Senate/House speakers with the same last name. Furthermore, our approach could not untangle speakers with the same last name in the Senate (there are two senators named Lane) and in the House (there are two representatives each named Washburn and Wilson). Despite these problems, our automatic extraction method performed quite well, and we thought that further refinement and computer programming wasn't worth the extra effort. Details of the R-code that made this extraction possible are shown on the website. This discussion and the code given on the website illustrate that in many text mining applications such as ours considerable programming may be needed to extract the relevant meta information.