Patent Data Cleaning

Shaheen Ahmed-Chowdhury

July 2019

1 Summary

Data input from GooglePats.csv file. Contained three columns, 'publication_number', 'publication_date' and 'text'. The first contained a patent number, second contained an eight digit date identifier, and the third contained a string of text. Each function created in the analysis is described below.

1.1 read_first_n_rows() and add_spelling_mistake_bool()

N rows are read into Python by `read_first_n_rows`. The `nltk.stem.wordnet` package contains the object `WordNetLemmatizer`. An instance of this is created. The text that is to be corrected is cast to a string, put into lower case and cleaned for special characters. Four corpora have been used from the `nltk.corpus` package, `words`, `stopwords`, `brown` and `wordset`. These are converted to Python sets and combined.

The data is placed in a dataframe. The column with text to be corrected is iterated over, with a boolean being set to `False` at the start of each iteration. This boolean describes if a spelling mistake has been detected. For each row in that column, each word is also iterated over. Each word is lemmatized by using the `WordNetLemmatizer` object. If the lemmatized form of the word is in our combined wordset, we do nothing and move on to the next word. If any lemmatized form of a word is not in our wordset, we set our above boolean to `True`, and then break out of the loop over words for that row. If the boolean is set to `True` for the row, then an entry of `True` is given for that row in another column, storing all results of the spell check.

Note, verbs such as "evaporating" are not detected, unless one includes a value of 'v' for a word-type parameter in the lemmatizer, therefore, in `add_spelling_mistake_bool` we test if either of two lemmatizers with the 'n' (noun) or 'v' word-type parameter respectively return a word in our wordset.

1.2 correct_spelling()

In the `spellchecker` package in Python, the `Spellchecker` object is imported. For each row, if a spelling mistake has been detected (as above), then the entry in the column containing the relevant text is split into its constituent words, and each is passed through the `Spellchecker.correction()` method. Each word in the text is then substituted inline for the results from this method, and the words joined back to reform the corrected text.

This new text is compared to the original text, and if any changes have been made, then a boolean is stored for the row as `True`. If no changes have been made, then the boolean is stored as `False`. The new form of the text is stored regardless.

1.3 run_error_detection_and_correction()

`add_spelling_mistake_bool` is ran on the original text, to see if there are any spelling mistakes, with a boolean being stored to detail if there are any. `correct_spelling` is then ran on the text, to store a corrected form of the text, and a boolean to say if any changes have been made. `add_spelling_mistake_bool` is then ran again on the corrected form of the text, to see if the string still contains any spelling mistakes (according to our combined corpus of course), and whether our corrections have resulted in any more 'clean' strings. A boolean, detailing if there are any mistakes, is again stored.

After this, one may expect that running `correct_spelling()` may be able to correct the text further. But in fact, it doesn't, as all possible corrections have already been made in the first iteration of the function. Thus our data cleaning process stops here.

1.4 Output files

Two files are currently output. The first is called 'patent_analysis.csv', and it outputs all text, corrected text and booleans mentioned above. The second is called 'erroneous_rows.csv', and it outputs all remaining rows with spelling mistakes, in the formal of the original input file.

1.5 extract_statistics()

`extract_statistics` groups the data by year, by shortening the eight digit year code (YYYYMMDD) to the first four digits (YYYY). The number of rows in each year are then summed, and combinations of the above mentioned booleans compared, to gain statistics on the performance of the correction procedure. Note that boolean values are summable in Python's `pandas` package, and we utilise this fact in `extract_statistics`.

1.6 plot_results()

`plot_results` allows the user to choose to view total values, or percentage values, when plotting the results of the data cleaning. Two separate figures can be saved in the folder that the user inputs to the function, by setting the `percentage_bool` boolean to `True` for percentages, or `False` for total values.

1.7 run_process()

This function encapsulates all of the above functions into one, which the user is able to call, to run the entire analysis. Four parameters are required, `filename`, `number_of_rows`, `edit_distance` and `percentage_bool`. `edit_distance` determines how many letters are allowed to be varied in each word's input to the `Spellchecker.correction()` method. The value is currently set to 1, as 2 takes too long due to a large combinatorial increase in potential spelling corrections.

1.8 test_word()

This small function can be used to check individual words in 'erroneous_rows.csv', or sanity check other words. Outputs the noun and verb-lemmatized forms, and a boolean indicating if the word is in our wordset.

1.9 Results

Results from the cleaning are displayed below.

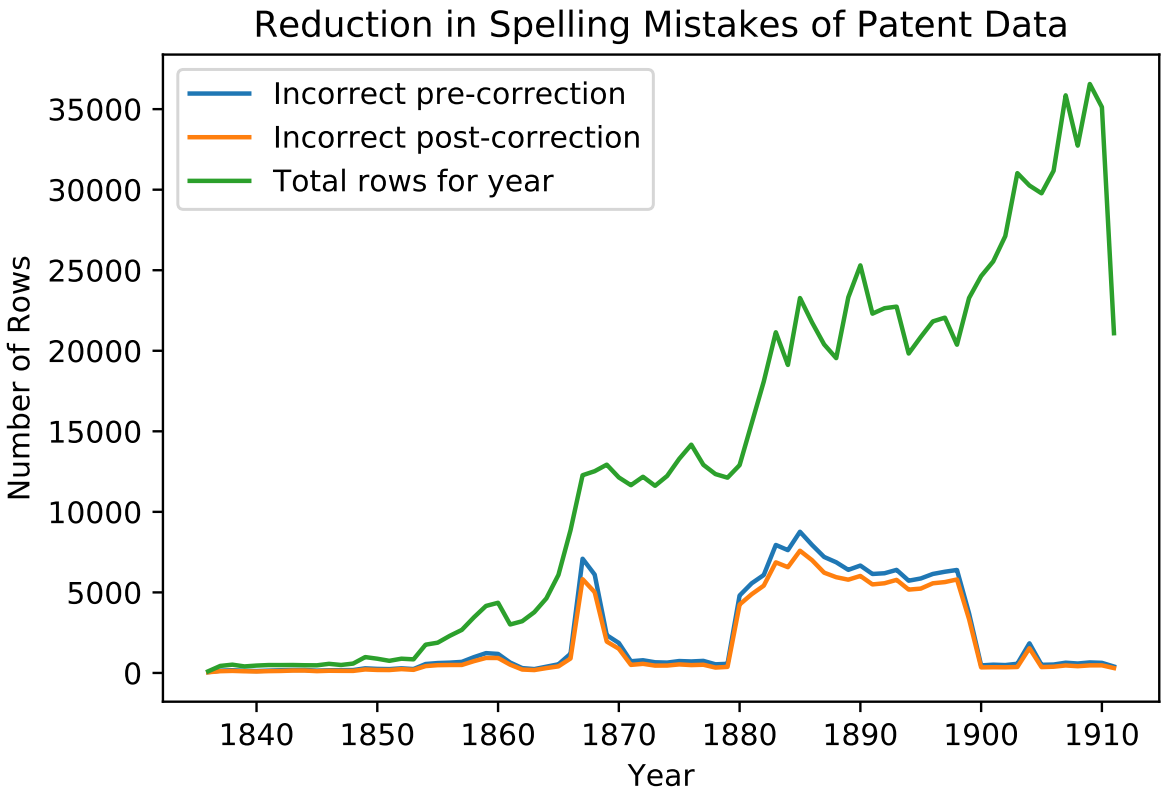

Figure 1: Total reduction in spelling mistakes.

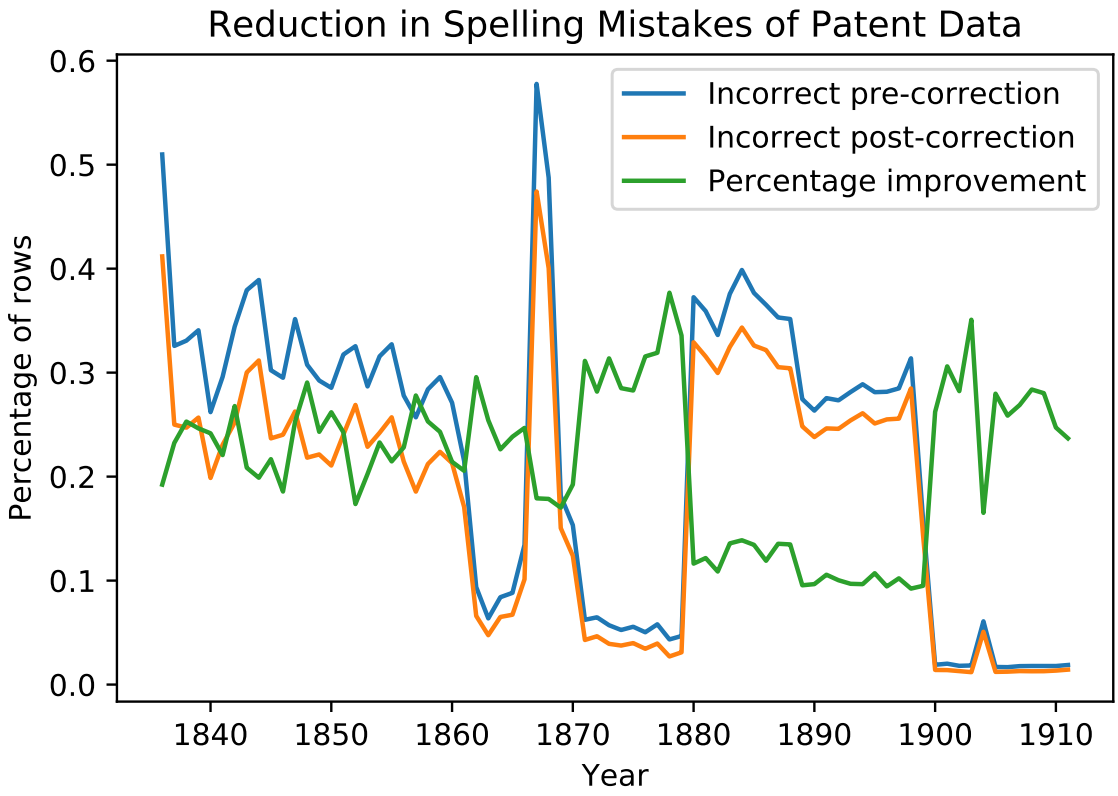

Figure 2: Percentage reduction in spelling mistakes