Universität Konstanz

Internship Project

Bigrams Generator and Frequency Calculator of English

Lingzhi Li
lingzhi.li@uni-konstanz.de

Overview

Aim: To create a database of bigrams with frequency which can be used for further linguistic studies.

Reason: Bigram frequency might have impact on word recognition and pronunciation.(Sauval&Chetail:2017)

Tasks:

- ✓ Scape bigrams
- ✓ Exclude space lines, punctuation, stop words
- ✓ Identify part of speech features
- ✓ Sort the list and calculate the frequencies
- ✓ Store the data and export to a database

Background

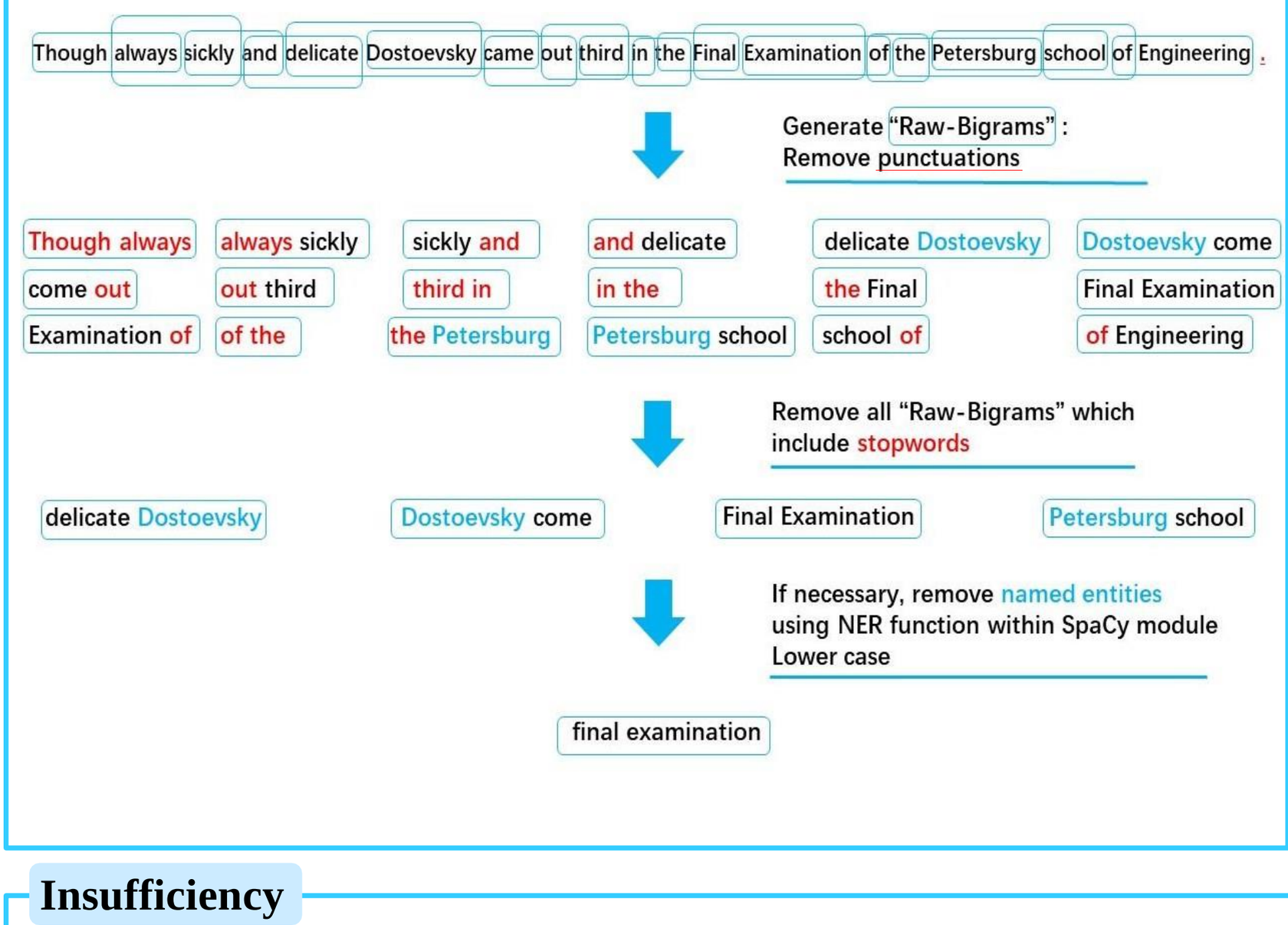
The original pipeline was designed as:

Text Cleaning & Normalization → **Bigram Recognition** → **Frequency Calculation** → **Data Exportation**

Text Cleaning and text normalization can be realized within NLTK functions. However, the bigram tagger from NLTK does not help with bigram identification at all.

Since an efficient and easy function for bigram recognition was not found. It needs to be built privately in order to cover specific tasks mentioned above.

Workflow



Extras

System I/O: File input output across different file systems via *os* module of python.
Besides, the program provides both local and online options to input the text.

GUI: graphic user interface to provide a user friendly program and enhance the use experience via python *tkinter* module.

Named Entity Recognition (NER): Thanks to the NER system in *spaCy*, we include a NER option in the program in case the user requires results without any named entities.
However NER system is case sensitive, so the case lowering process has to be executed later in pipeline.

Frequency Calculation:
I create my own functions upon *pandas* module to calculate occurrences of each bigrams and of each POS tag group.

Insufficiency

POS Tagger:
The accuracy of *spaCy* tagger is more accurate, because it uses dependency parsing based on the context. However, it becomes tricky when punctuations, stop words and named entities were removed. In this case, *NLTK* tagger was applied in this project, but it cannot recognized correct POS tag of the word under certain circumstances such as one word owns several POS tags at the same time.

Code Structure and efficiency:
In this project, it consists a lot of redundant programming which could definitely be improved. To sketch a more clear structure before coding is more efficient than fill the holes afterwards.

Accuracy:
The result of bigram extraction is not accurate enough. However, on top of it, it is still possible to find the real bigrams with high occurrence in the data.

If you are intersted, you can find this program under: github.com/wisd0mbalala/Bigram_crawler