20170396 예성현

The implementation is mainly divided into three steps; finding heteronyms from cmudict, getting the word meaning considering the context, mapping meaning to pronunciation using Wiktionary. The corpus used is 'The complete works of William Shakespeare', available at http://shakespeare.mit.edu/. I divided heteronyms into two groups, 'pattern-heteronym' that differ in position of accent and POS tag, such as 'produce', 'conduct' which pronounce as PROduce and CONduct for noun, proDUCE, conDUCT for verb. Other heteronyms are 'nonpattern-heteronym' and includes words such as 'wind', 'bass', and 'tear', which has somewhat more irregular changes in the pronunciation.

For pattern-heteronyms, the word has to have multiple pronunciation with different accent and different POS. If the set of the POS is (verb, noun), the instance that has stress on front should be noun and the other one be verb. The behavior should be same for (adjective, verb) case. This criterion retrieves 197 words. For nonpattern-heteronyms, only one element in the phoneme list should be different with same position. Then, because heteronyms have to have different meaning as well, it has to check the path similarity of all synsets with threshold of 0.07. This criterion retrieves 60 words. Because this criterion does not exactly narrow down much, further filtering is needed via Wiktionary. For 60 words, the program excludes ones that has single meaning or pronunciation in the dictionary. After filtering, the nonpattern-heteronyms consist of 15 words. The reason the first filtering is needed is because fetching information from Wiktionary takes much time. In other words, it does a candidate selection job.

For a given sentence, we have to look if there is a heteronym. If there is, tag the target word with POS tagger. For the pattern-heteronym group word, we can easily get the correct pronunciation only considering POS information. Therefore, we only consider nonpattern-heteronym group from now on. If the target word is a verb or adjective, chunker is used to distinguish if there is a NP tree right after the verb word. For example, if 'tear' is used for 'crying', it is likely that there would be no NP tagger. However, if it is used for 'ripping', it is likely that there would be NP tagger such as the clause 'tear the paper'. If the sentence structure is similar for example sentences of each synset examples, the synset is appended in a list. If the target word is a noun, pywsd is used to get the specific synset of a sentence.

Because we now know the specific synset for the target word, we can look into the definition of the synset. Then, compare it with the definition in Wiktionary and choose the one that has the most number of common words. Lastly, match the definition instance with pronunciation. If there is no synset from step 2, only POS information is considered, unfortunately.

The last step is to rank the output. I followed the rule in the HW document for 1,2. However, to get meaningful output, I thought that rule 3 should be to rank according to sentence that has more nonpattern-heteronym words because pattern ones are easy to distinguish. Because I divided heteronyms into 'pattern' and 'non-pattern' heteronyms in the first place, I think this ranking is more reasonable to rank tear+tear higher than PROduce+proDUCE.

Looking at the output, there are some issues to discuss. First, for pattern-heteronyms, the outputs are expressed in ARPABET format. There is blank pronunciation for 'torment', and this is due to incorrectness of POS tagger. The word is used as noun in the sentence, but the POS tagger see the word as adjective. Other than that, every output is correct with no false positive and negative. For nonpattern-heteronyms, let's first look at the 15 heteronyms in the txt file. There are some false negative cases such as not detecting the word 'tears' while it detects 'tear'. This is because for some reason, Wiktionary parser does not parse the IPA pronunciation for 'tears'. Also, there are some false positive case such as 'barrow'. This is because the program thinks words with different pronunciation in the UK and US as multiple pronunciation words. For improvement, the program must decide only one of those and make the other one invalid. Overall, 13 out of 15 heteronyms are classified correctly. Looking at the output file, nonpattern-heteronyms are expressed in IPA format. For heteronyms that are noun, there are some incorrect pronunciation such as 'wind' pronounced '/waɪnd/' when the correct one is '/ˈwɪnd/'. The reason for this wrong output is due to the incorrectness of pywsd tool. I tried to use some advanced tools from pywsd that uses max_similarity algorithm, but there was maximum word limit on these tools. Because the corpus is a literature work, the sentences are much longer than other genres, so it was not possible to use the tool. For verb, adjective heteronyms, the algorithm can be improved by not considering the example sentences from wordnet, but the sentences in Wiktionary because there were not many sentences in wordnet to decide precise synset. Also, when using Wiktionary sentences, it can directly map definition that the example is included in to pronunciation without having to compare the synset definition in wordnet and definition in Wiktionary. This would increase both the efficiency and correctness of the algorithm. Also, NP chunker might be not enough. There are some words such as 'bow' that is usually followed by a preposition such as 'to' when it is used as 'to bend as a gesture'. Considering this grammar rules as well would lead to correct tagging of pronunciation.