[Overview]

The **key steps** of this problem can be stated like this. First, find some general English corpora which possibly contains plentiful heteronyms for the test of the program. Then, find which word is heteronym. Finally, tag the heteronyms with proper pronunciation and sort them in appropriate order.

The most experimental part in solving the problem was tagging heteronyms, and I used an **approach using POS and synonyms**. First, some heteronyms are different in POS, and some are not. For the heteronyms different in POS, such as PROduce and proDUCE, it is easy to find the POS and match with the pronunciation. Heteronyms same in POS such as tear(N) is more complicated. Using external English dictionary, I could find heteronyms with two or more pronunciation, and also get some pronunciation-synonym pairs for each word. Using these synonyms and a sample corpus in nltk, I could find the context of the synonym used. Compare the context of the heteronym and the synonyms, I could find out which context is the heteronym used for, so can match with the pronunciation.

[Detailed Approach]

First was **finding the heteronyms and getting the dictionary information**. It is impossible to search every words in dictionary and check whether it is a heteronym or not. Thus, using cmudict in nltk, words with two or more possible pronunciations were selected. Then, using Google dictionary API, those words were searched and their POS, pronunciation, synonyms were stored in .json file for further use. https://googledictionaryapi.eu-gb.mybluemix.net is the one I used. Although it is not an official one, since it provides free requests, giving pronunciation matched with the POS and synonyms, I decided to use it. Also, while I was working on the project, I could find that the words where different stress pa such as PROduce/ proDUCE, OBject/obJECT, REcord/reCORD were not appearing as heteronyms in this dictionary. Thus, by using cmudict, I manually added these words in the one from Google dictionary. First, I got some two-syllable words which has two different stress patterns in cmudict. For further easiness of pronunciation annotation, I only came with words with 'two' possible pronunciations. Using wordnet, I checked if they have both noun and verb form, then annotate pronunciations as 'N' if the stress pattern is [1,0] or 'V' if the stress pattern is [0,1]. However, if Google dictionary already has those words, I followed the one of Google dictionary since it contains more information than the one I can get from cmudict and wordnet.

Next, **finding corpus** was also important. Here, I used two sources for the corpus. The first one was articles from a blog. https://equipsblog.wordpress.com/page/10/ contains articles of various topics including wordplays, hints on writers and writing, thoughts on nature, military/veterans, book reviews, and re-blogs on some of these same topics (from 'About' page). Thus, it can serve both as a general English corpus and a possible donor for some interesting sentences for tests. The second one is the full plays of William Shakespeare. He is well known as using puns in his script. I got the full scripts in http://shakespeare.mit.edu/. Both of them are opened to public, and since each of them are written by one person, there is almost no worry of duplicated sentences. For these reasons, those two were used as the corpus. Before using them, I put them in a json file, matching the titles with the texts. For clean results, a brief preprocessing was done. Here, noise characters like '\n', '\xa0' were deleted. The texts were tokenized by nltk sent_tokenize and word_tokenize, and POS-tagged by nltk pos_tag.

For the **heteronym tagging and sorting**, the approach briefly explained above was used. If the heteronym has only one pronunciation with the tagged POS, that word was annotated with that pronunciation. However, if the heteronym has more than one pronunciation with the tagged POS, context comparison was done. For **context comparison**, the counter named 'similarity' was used inside the code, in the function 'findsyn()'. For a given word and POS, in dictionary, I could get possible pronunciations and synonyms matched with those pronunciations. For each synonym, those words are searched in a 'standard corpus' (here, I used brown corpus as a standard corpus). Then, for each occurrence, the sentence and nearby words (up to 3) were stored. Then, for each synonym, similarity with 'the problem word' was measured. More same words in same relative position results in higher similarity. For example, if a word just before 'the problem word' is same as the word just before the 'synonym word', that occurrence counts for similarity. Each position was given a specific 'weight'. Also, to find out the overall topic of the sentence, the sentence similarity was also included in the similarity. If there are same words in two sentences, then the similarity gets high. Also, the frequency of synonyms used is different, the total count is normalized by dividing it with a denominator proportional to the number of occurrence. Here, some pronunciations were not containing synonyms. For these pronunciations, I gave a default similarity value 1. After evaluating the similarities, the word

will be annotated as the pronunciation with the 'maximum' similarity synonym.

In **sorting**, four standards were used in total. First, the number of heteronyms (counter in the code) were the first standard. Then, the number of homonym pairs were evaluated by (num of pronunciations) – (num of words). More heteronyms, higher ranked, and more homonyms, also higher ranked. Then, the third standard is the number of POS, where small number of POS is in higher rank. The last standard, which is an additional one to problem documentation, is the number of words. Multiple same words with same meaning were ranked low.

[Limitations and Improvements]

There are some critical limitations.

First, in heteronym finding steps, there are problems in cmudict and also in the google dictionary. I already mentioned that google dictionary sometimes do not show a 'right' pronunciation. Although it is partly solved by using cmudict, some heteronyms like concrete(adj) and record does not appear in my dictionary. Although concrete(N, V, ADJ) appears in Google dictionary, since all three of them have same pronunciation there, it is not added in the dictionary. In further update by cmudict, since I just classified noun and verb form, the adjective form of concrete cannot be translated by the dictionary. In the case of record, when I updated the dictionary by cmudict, I just collected the words with two possible pronunciations where those two are different in stress for further matching between pronunciation and POS, I discarded 'record' since it has three possible pronunciations. Moreover, since some of the words do not provide synonyms, some synonym lists are empty so that I manually gave them default similarity value in my code. This was a problem especially in the case of 'bass'. The two well-known definitions, 'fish' and 'low voice', both appeared in the dictionary with POS 'N', but also both did not contain any synonyms. Thus, in every case, both of them gets the default similarity value. Thus my program cannot distinguish between those two. This will be partly solved by using holonyms. It might not be perfect, but if you use 'fish' rather than 'bass', than there will be a better result. Other relationships such as antonyms and meronyms can be used, but this will need more fine-tuning for better result.

Analyzing the results, I could find some interesting points. First, this approach works really bad in some 'literature' sentences. This is actually shown in the examples of Shakespeare's texts, where they use 'spoken form' language and some not very grammatical sentences. Since I am not using a lot of information about the context, those characteristics of Shakespeare texts make noise in the output. Difficulties in POS tagging in these texts is the additional reason for low accuracy (output 29). Also, I could find out that the word 'tears' is sometimes analyzed in wrong way when the word 'tear' is analyzed correctly (26-28). This seems to be happened by both the characteristics of the sentence (not perfectly grammatical, so hard to analyze), and the characteristic of the word (maybe synonyms can be used in similar word context such as determinants or prepositions). While some of the outputs were bad, some wordplays were analyzed correctly. For example, "Upon seeing the tear in the painting I shed a tear.", "The farm was used to produce produce.", "The dump was so full that it had to refuse more refuse.", and "The insurance was invalid for the invalid." (the latter two are not in the rank 30 output.)

Sometimes, in really short sentences, my approach did not work really well since it does not contain plenty of contextual information. My current approach works with context, so short sentences with very less context information will fail to be annotated properly.

Some improvements can be done. First, use all the synonyms of all the definitions in the case of having multiple definitions in single POS/pronunciation. Currently, I'm only using the first definition for the synonyms. However, if all of the definitions will be used, it will increase accuracy of the synonyms.

For the problem of 'literatures', it will be better to use different types of 'standard' corpus for each type of input corpus where brown corpus was used here for both test corpora. This is because the words people use in everyday life, web chats, and literature are all different.

Fine-tuning the weights and the default value is also needed. Here, since any kinds of machine learnings were avoided, I just manually put some weights, considering relative importance of the words nearby. However, if machine learning used in this part, finding the 'context' and returning a fine-tuned similarity value, then it will be much more accurate.

Some minor problems such as cannot detecting words outside of cmudict, errors in tokenization or pos tagging also appears. This will be solved by taking more time using better dictionaries, use better tokenizer or pos tagger such as the one in allennlp. Also, in context comparison, if the synonym consists of two words, then it will not be found in the standard corpus since the standard corpus is already word tokenized. It can be solved by changing code, but at the same time, it will make the program complicated.