Lab 2. Report

Implementation

This program adopts three approaches (including two language models, i.e. unigram and bigram language model) to choose the item in the question from the optional items.

During the pre-processing, the data would be read from the files and removed the punctuation by applying regular expression then processed to be list.

Unigram

In the unigram language model, each term would be counted by using the counter function from the collection then stored into a numpy array. An index dictionary would be constructed (i.e. the key is the term, the value is the index which locates its value in array). In the prediction process, the probability of each optional item would be calculated. The item which has the highest probability would be chosen as the answer.

Bigram

In bigram language model, based on the index dictionary in unigram language model, a sparse matrix would be constructed. In prediction process, bigram language model has two choice that is whether apply the add-1 smoothing. All optional terms would be filled in each question sentence, then the probability of whole sentence would be computed. High probability one for each sentence would be chosen as the answer. If the two optional sentences have zero probability, the answer will be set as NULL. If the two optional sentences have the same probability, the answer will be set as HALF which means that is half corrected whichever would be chosen.

Evaluation

The accuracy of each approach would be calculated and shown in the table below.

Unigram	**Bigram**	**Bigram add one smoothing**
0.6	0.7	0.9

It is absolutely reasonable and satisfied for expectancy. Unigram has the lowest accuracy because only using the single term to decide the answer is arbitrary and regardless of the order of the term in the sentence. The answers are shown in the table below.

Answers of unigram model									
whether	through	peace	court	allowed	check	hear	serial	choose	cell

For the bigram language model, the accuracy has been improved. The answers are shown in the table below.

Answers of bigram model									
whether	through	piece	court	allowed	check	hear	_NULL_	_NULL_	_NULL_

Here three answers are shown as "_NULL_", that means during the probability of the whole sentence, some terms never appeared in the data, so it has zero probability. During the computation, it is controversial that whether the probability of whole sentence should be computed. In this case, if only the terms which are next to answer location need to be computed, the accuracy would be 0.8. However, if all the terms in the sentence are computed, the accuracy would be 0.7. The reason of why computing less terms cause higher accuracy is that some terms in the sentence never appeared in the dataset so that cause the zero probability for sentence then the result would "_NULL_".

However, when it comes to bigram model add-1 smoothing approach, the accuracy are improved. The answers are shown as below.

Answers of bigram model add 1 smoothing									
whether	through	piece	court	allowed	check	hear	cereal	choose	sell

Using this approach avoids zero probability for sentence so that get the better result comparing the result of only bigram.

Bonus

Sparse matrix has been implemented in the program. According to the API of scipy, the coo_matrix (a sparse matrix in coordinate format) is efficient for constructing (about 7s) and for assessing the data in coo_matrix, the coo_matrix need to be transferred to the dok_matrix (Dictionary of keys based sparse matrix) because the dok_matrix has high efficiency on accessing data.

Using sys.getsizeof() function calculate and comparing the memory obtained in three different data structure (coo_matrix, dok_matrix, 2d-array) and in different accuracy data type (Float32 and Float64), obviously, 2d-array obtains the largest memory and coo_matrix obtains the smallest memory, but the difference between the memory of two data type is enormous. The only reason for that is coo_matrix data type is pointer. For two types of data, 2D array in Float64 take about two times memory than that in Float32. The comparison data is shown in the table below.

coo_matrix	**dok_matrix**	**2D array**
	Float32	
56	167,772,280	184,685,062,612
	Float64	
56	167,772,280	369,370,125,112