Question Answering Report

1. Overall QA approach description

To construct the QA system, we started by creating a rather dumb system which randomly choose the answers from file. The evaluation performance of the baseline system shows a quite low MRR value of 0.006.
For part 2, we used the method of Sentence Retrieval + NP extracting + Semantic Class Checking.
We used passage (sentence) retrieval for getting candidate answers. We walked through all the documents to get these sentences instead of just taking top 5 or 10 documents because it's very likely the answer are not in the top 5 or 10 documents. PR allows us to get access to every possible candidate.
The NP are used with the help with Stanford POS library. We extracted noun phrases of the question and check every candidates to calculate the overlapped words number with noun phrases in the question.
We remove sentences that does not contain overlapped NP from candidate polls because those sentences has much smaller probability of containing the answers. We also used NP when we made our candidate sentence into less than 10-words answers. We only kept noun phrases (actually NER) as answers for that most who, when, where questions have NP as answers.
We also used semantic class checking to distinguish who, where and when question. We used Stanford NER library to give tags of PERSON, LOCATION, TIME and others to each word or phrase. For who question, we give answers tagged as "PERSON"; For when question, we give answers tagged as "TIME" or "DATE"; For where question, we give answers tagged as "LOCATION". This will help us to remove wrong type answers from candidates' pool and tried to give most suitable answers for each type of question.
Our final system shows an MRR of 0.176, which means the combination of these methods works well to construct our QA system.

2. QA system description

Three main referenced source and existed package are used in the system as follows:
1. English_Function_Words_Set Package
Include: EnglishAuxiliaryVerbs.txt, EnglishConjunctions.txt, EnglishDeterminers.txt, EnglishPrepositions.txt, EnglishPronouns.txt, EnglishQuantifiers.txt
Source: http://www2.fs.u-bunkyo.ac.jp/~gilner/wordlists.html#functionwords
2. stanford-ner-2015-12-09
Source: http://nlp.stanford.edu/software/CRF-NER.html
3. stanford-postagger-2015-12-09
Source: http://nlp.stanford.edu/software/tagger.html

We basically use the English_Function_Words_Set Package to remove function words in the questions for better performance in calculating the overlap score during the process of obtaining answer, the stanford-ner-2015-12-09 is implemented for named entity recognition and stanford-postagger-2015-12-09 for extracting noun phrase.

Code description:

There are 6 java files in our QA system which are Input.java, Ner.java, Postagger.java, Question.java, Sentence.java and Simpler.java. Among them, you can simply write the answers by running the main function in Simpler.java (after ensuring that the paths at the top of each file are correct).

To take a brief look at each java file, the Input.java file mainly deals with reading and processing the question.txt and writing out the answers to specific path. The Ner.java interfaces with the Stanford NER library and contributes to named entity recognition (NER) extraction. As for our classification of the questions, we classify them into four types of questions which are When, Where, Who and O (Others) questions to improve the specificity of the answers. The Postagger.java is used for extracting noun phrase in the question. Question.java and Sentence.java include most of the member fields we would like to use in the QA system which can be divided into two main groups, questions with related answers and vector features for sentences. The purpose of the Question.java is for labeling questions and add answers to each question object followed specific restrictions. Sentence.java is more complicated as it contains more processing. As discussed later, we mainly give scores to judge the quality of the answers based on overlapped words. The Sentence.java implements both the extraction of named nouns and words and the detailed calculation of the sentence vector features such as cosine distance (we tried SVM but have not done). The methods written in the Simpler.java (our main java file) are to process the whole pipeline of getting answers.

Code hierarchy:

We use two different ways to introduce the process of obtaining 5 highest scores among the answers, which are the call hierarchy of Simpler.java and the float chart. As mentioned before, the Simpler.java is the main java file for our QA system. It mainly contains the functions of counting the overlapping score of each sentence and pick the suitable and 5 answers of best performance through the process of picking up candidates, removing wrong types of answers and changing into shorter format.

For the call hierarchy of the simpler.java, there are several steps to run:
The simpler.java starts by creating an ArrayList of Question objects. This invloves geting the questions.txt path, question text, and questionId.
For each question in the arraylist, there are a series of related answer documents. Simpler.java stores all the paths of answer documents for each question.
Then we step into the loop which contributes to find the 5 best answers for each question.
For each question, the loop creates a new Sentence object with its related question content and questionId. Each Sentence object extracts the nouns by calling the function of Sentence.extractNouns.
With the question's nouns, Simpler.java calls Simpler. getOverlappingSentences to calculate all the answer score. This is done using a non-normalized cosine distance (counting the number of overlapping words)
Then the Simpler. removeWrongTypeAnswers helps to remove wrong type of answers in the set based on the classification of question types which are WHEN, WHO, WHERE and OTHER questions. Specifically, we use Stanford's NER to classify words into possible answers to WHEN, WHO, and WHERE answer types. If a candidate answer does not contain the relevant NER type (i.e for a WHERE question, the response doesn't contain a LOCATION word) it is removed from the possible answers.
After calling the method of Simpler. extractWords which aims to extract all the non-function words into a bag of words style HashMap where (key=noun) and (value=number of occurances in the sentence), we use the method of Simpler. getOverlappingSentences for obtaining the best answers.

The method of Simpler. getshortAnswer helps with the modification of the previous answers by implementing the Stanford NER library to extract related named entities and remove duplicates. At last, Simpler.getFinalGuess will give each candidate final scores (overlapped words number + length of the candidate) and get the 5 answers of highest scores in the set. In the end, we call the method of Input.writeAnswers to write answers we get to the answer_path for the evaluation and stop the Simpler.java. The MRR value from evaluation is approximately 0.176. Meanwhile, our baseline which randomly choose the answers from random document file shows a MRR value of 0.006. We can see the advantages of our QA system.

The steps that our system takes to answer a question is like Fig1 below (details will be explained in part 3):

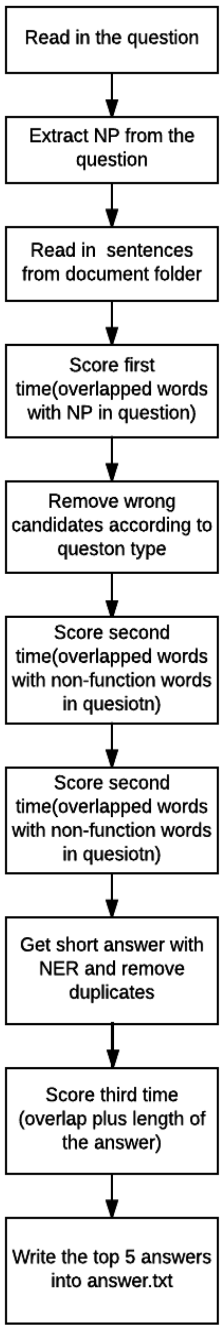
Read in the question
Extract NP from the question
Read in sentences from document folder
Score first time(overlapped words with NP in question)
Remove wrong candidates according to queston type
Score second time(overlapped words with non-function words in quesiotn)
Score second time(overlapped words with non-function words in quesiotn)
Get short answer with NER and remove duplicates
Score third time (overlap plus length of the answer)
Write the top 5 answers into answer.txt

Fig1 Flow chart of QA system

3. Walk-through of handling one question

For a specific question, for example, “Where is Belize located?” (ID: 89) The detailed walk through is like below:

1. Read in the question with the question text along with its ID number;
2. Extract all the NP in the question into a Hashmap<String, Integer>. Here there is only one NP in this hashmap: belize (change it to lowercase for later comparison);
3. According to the question ID, read all the sentences from the document folder (89) and only extract sentences that are the title or main text;
4. For each sentence, calculate the overlapped words between the sentence and the question NP hashmap. If the sentence has overlapped words with question, keep it as a candidate (For this question, there are 179 sentences containing word “belize”);
5. Score those candidate sentences with the overlapped words number and get top 50 (this parameter can be changed) sentences;
6. According to the question type, get rid of some answers that do not contain this type of answer. For this question, all the sentences that do not contain “LOCATION” will be removed from the candidate pool. After this, we now have 44 candidate sentences for question 89;
7. Score those candidates again. This time, we first remove all the function words of the question, then put remaining words into a Hashmap. Then we count the overlap words of the answers with this hashmap and use the overlap words number as the score to sort the candidate sentences. After that, we get top 20 answers (20 can be changed);
8. For the 20 candidates, extract NER from them according to question type. Get rid of duplicate words in this sentence. Also, if there are duplicates answers after extracting, just keep one of them. For example, "Belize is located in Central America" and "Belize is a country in Central America" for this question will get same short answers "Belize Central America" and we only need to keep one of them. We also restrict the length of each answer as 10 according to the instruction demands. Now we only have 10 candidates for question 89 after this step;
9. Then, we score the remaining candidates again. Their original score is the overlapped words number. But, as we can see, if the answer contains more words, the probability of having the right answer is bigger. So we try to score them with overlapped words number plus the length of the candidate answer (the weight of the length can be changed). Then we get top 5 guesses;
10. Finally, we write the top 5 guesses into answer.txt with the right format.

4. Analysis

4.1 Performance evaluation

The system has two major hyperparameters. The first hyperparameter is the number of sentences retained (per question) after the initial noun-based similarity check. This hyperparameter is an important elimination step, and two things are valuable to notice: firstly that it doesn’t make sense to have it set to fewer than 5 sentences, as we are allowed 5 answers, secondly that the larger this parameter is, the more unrelated sentences will be retained for each question. See the figure below for a graph of different values of this parameter on the development set. 25 was selected because it was one of the points that gave the highest performance, and yet allowed the most information to be retained for each question.

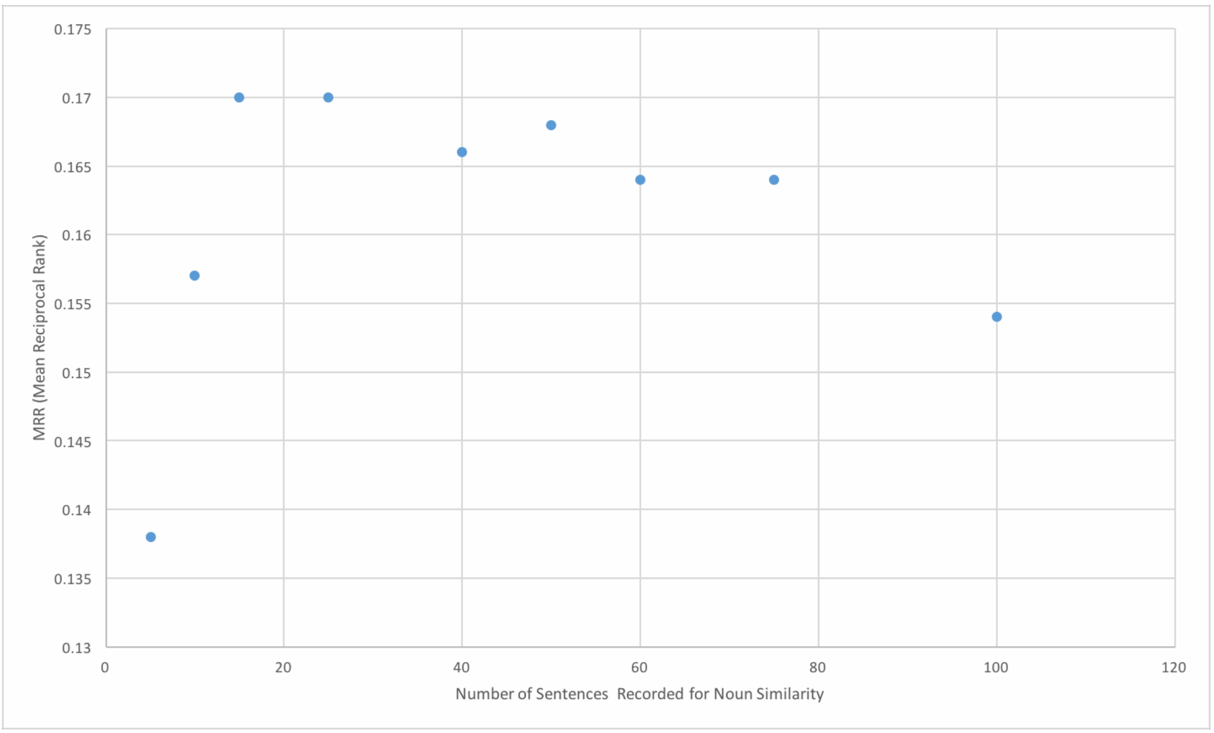

Fig2 Number of nouns recorded in the first pass vs MRR. Other hyperparameters was held constant. 25 was selected.

The second hyperparameter is the number of sentences retained in the final pass, where similarity is evaluated based on full sentence text (including both noun and non-noun words). This parameter cannot be higher than the first parameter, and should be at least 5, as we are allowed 5 answers. The results of this parameter are shown below, the value of 15 was selected.

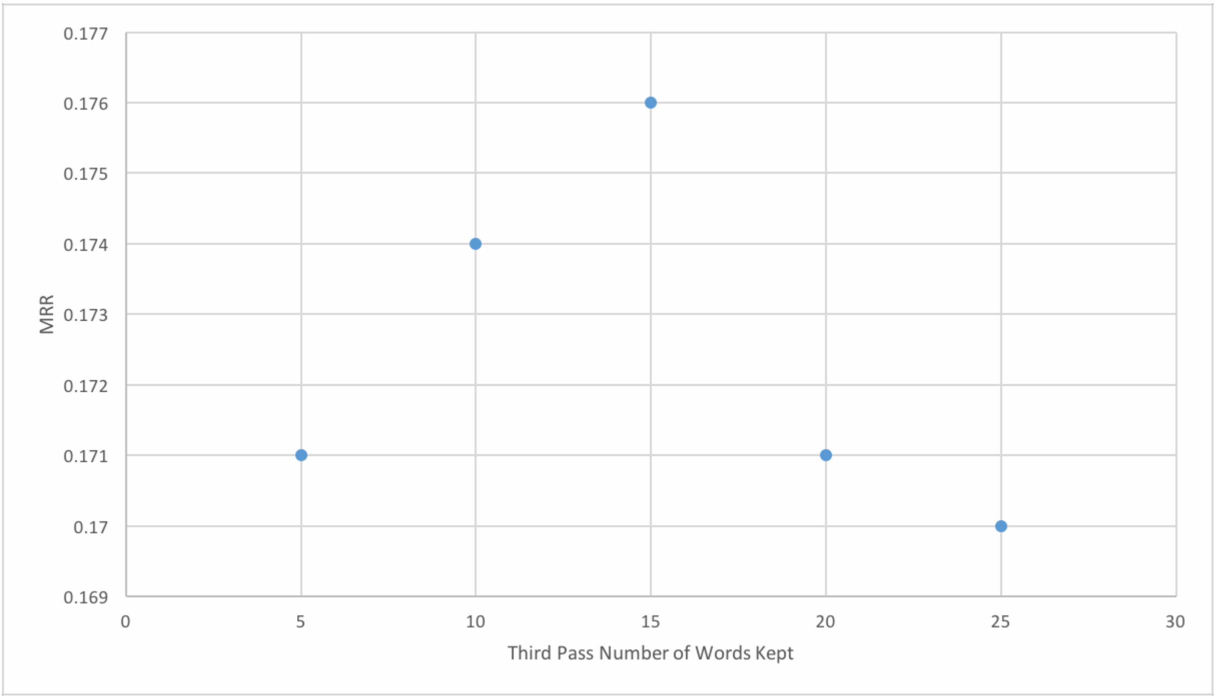

Fig3 Number of words kept in the final pass vs MRR. All other parameters help constant. 15 was selected.

The third hyperparameter is for the final weight, and helps to select sentences of different lengths. This parameter turns out to have had very little impact in the end, but was intended to compensate for our non-normalized similarity measure. The results of varying this parameter are shown below, 1.2

gave the highest value.

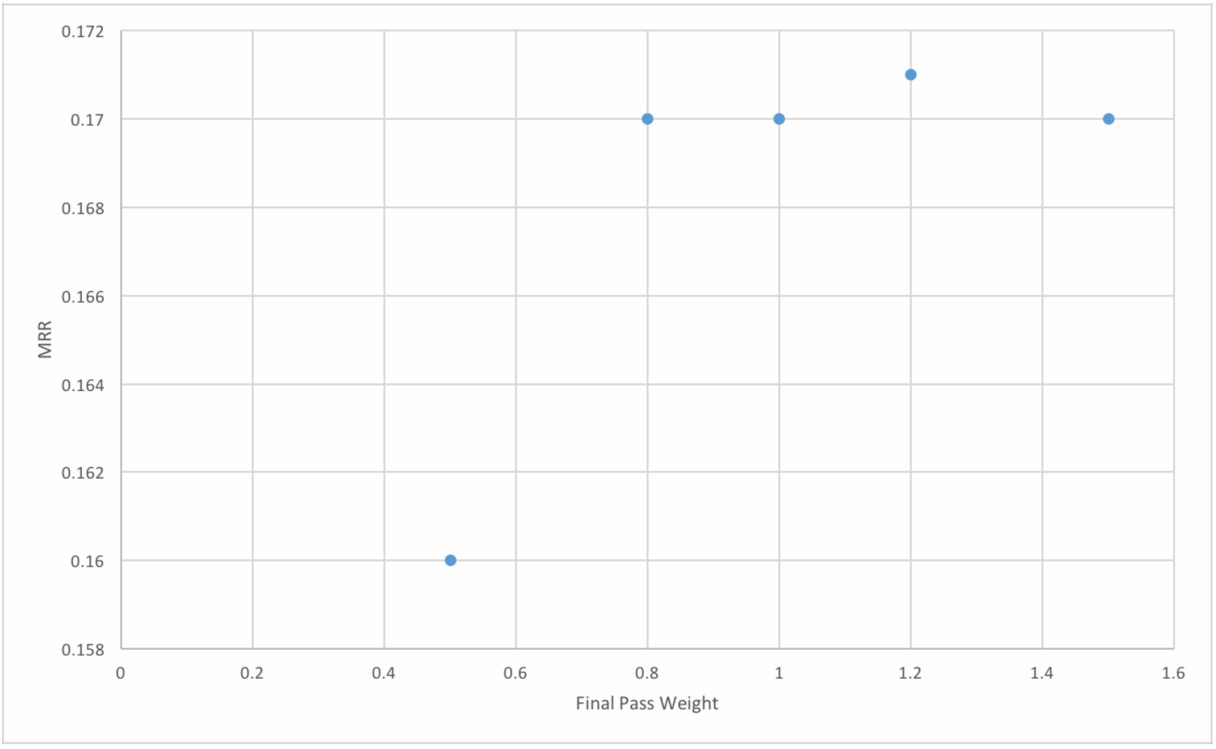

Fig4 Final Pass Weight vs MRR at constant values for other hyperparameters.

MRR = 0.176 is got with parameters value 25 (number of nouns recorded in the first pass), 15 (number of words kept in the final pass) and 1.0 (final pass weight).

Overall, our system's performance using these metrics was fairly good. However, many questions were left without responses, and the system would not be very good at real-world answering questions. For example, when asked "Where is Belize located?" the answers were :

"89 31 America Panama Belize El Salvador
89 33 Belize City CANA Mexico
89 29 Belize
89 47 Washington Belize Chicago
89 1 Guatemala Belize"

While the answers are present (Guatemala) the results are fairly unuseful, as you don't know which is the right answer of all of the words returned by our system.

4.2 Comparison with the baseline system

Our baseline system was randomly returning 10 words. This wasn't the best baseline so it only achieved 0.006 MRR. Therefore our solution is both smarter than our baseline, as it picks out relevant words, and it therefore performs better than the baseline system.

4.3 About our system

Our system worked well in the end. Each piece worked well individually. Going in, we assumed that the NER to decide on best answers based on question type would perform the job nearly perfectly, but we could not get a system that relied so heavily on NER and question type to achieve better than 0.1 MRR. Therefore the first pass to extract sentences with similar sentences was very important, as was the final pass to decide between all of the previously chosen answers using all words in the sentence.

The main weakness was that we didn't necessarily fill all 10 word slots for each answer. While this means that we put less text in the answers, which might make the answers more useful to a human, this means that we did not necessarily maximize our MRR. It is caused by us treating the whole sentence as an object and keep getting candidate sentences till we limit the answers’ length. At that time, we began to extract NER. If we extract all the noun phrases and retrieve list of NPs instead of whole sentences, we may be able to fill more slots of 10 words limitation and improve MRR.