Passage retrieval in a question answering system

Juri Pyykkö
dt07jp0@student.lth.se

Rebecka Weegar
ada09rwe@student.lth.se

Abstract

In this document a system for passage retrieval in a questioning answering system is described, and the result of the passage retrieval is evaluated. The system is designed for Swedish. The corpus of questions and answers is based on the Swedish board game Kvitt eller dubbelt, and as source for passages, the articles of Swedish Wikipedia have been used. The tools that mainly have been utilised are Apache Lucene for indexing and searching and Stagger - The Stockholm Tagger for part of speech tagging.

1 Credits

This system has been developed during the course Intelligent systems at LTH as part of the project Hajen. The questions of Kvitt eller dubbelt have been transcribed and classified in collaboration with Cristopher Käck och Robin Leussier under the supervision of Pierre Nugues.

2 Introduction

This project was designed as part of a question answering system such as the well known IBM Watson, which has served as an inspiration as well. The purpose of the system is to be able to answer questions of the Kvitt eller dubbelt game, which is a Swedish board game, and therefore it is required that the system supports the Swedish language. Beside the questions themselves, the cards of Kvitt eller dubbelt are also labeled with question categories and subcategories, which can be of use in finding the answer.

A schematic of the complete system is shown in Figure 1. However, this project was done in isolation from the question processing part and therefore required some adjustments to function, these are described further on. The question processor will not be discussed to any greater extent in this paper, only enough to explain its role in the system.

3 Question Processing

3.1 General

Given a question, the question processor should formulate an appropriate search query for the passage retriever. It should also determine the question category. Finally, the processor should also be able to determine the expected word class/type of the answer, which is used in the answer extractor.

3.2 Classification

To be able to classify the question and answer types, various machine learning techniques can be used. To this end, a training set was constructed in collaboration with the other participants in project Hajen who were responsible for the question classifier.

3.3 Training Set

164
Djur och natur
-
I terrängen
5000
Är en myr ett fuktigt eller torrt område i naturen?
Det är ett fuktigt område
(våtmark)
location

Table 1: Card example.

The questions were transcribed according to the example shown in Table 1. For each questions, all the fields except the last one were copied from its card. While the last field contains the answer class, which was determined manually. In order

from top to bottom in table 1, the different fields are described as follows:

- Card number - identifies the card.
- Category - specifies the general theme of the question.
- Difficulty - can be either '-' for normal or '*' for easy.
- Subcategory - like category but more specific.
- Points - only used in the actual game of Kvitt eller dubbelt.
- Question - restricted to the above mentioned categories.
- Answer - may consist of one word, several words or numbers.
- Answer explanation - holds any additional information or alternate phrasing of the answer, this was not available for most of the questions.
- Answer class - can be one of the following: action, binary, description, entity, human, location, numeric.

4 Passage Retrieval

4.1 General

The passage retriever's task is to reduce the amount of information into a manageable size for the answer extractor. When provided with a search query, the retriever proceeds to search for passages of text that are relevant to the query, and return these ordered by similarity (see Section 4.3.3). As mentioned, the question processor was not part of this project, therefore the search query was not constructed with keywords as shown in Figure 1, instead it was formed directly from the training set. As the training set contained the predetermined question subcategories for each question, these were used as additional keywords, i.e. the final query consisted of the entire question in its original form and the subcategory.

The passage retriever was built upon the existing functionality that Lucene Core provides. Lucene Core is an open source, Java-based project, offering advanced indexing and search capabilities as well as other utilities.

4.2 Wikipedia

The Wikmedia foundation regularly publishes updated data dumps or snapshots of the entire Swedish Wikipedia (and other languages) [4], which made a good source of information for the passage retriever. The texts within were full of XML, which could possibly have interfered with the passage retrieval. By using a certain Python script [5], all the XML was successfully removed, leaving all text unmodified.

4.3 Lucene

4.3.1 Indexing

Searching the Wikipedia texts in their original format would be inefficient, indexing the texts first circumvents the need to do so. By Indexing the texts they are divided into smaller segments/units (called documents in Lucene), after which all words from the original texts are mapped to sets of new documents where they now occur. During this process Lucene also collects information about the terms and documents, which it then compiles into statistical data, needed at query time.

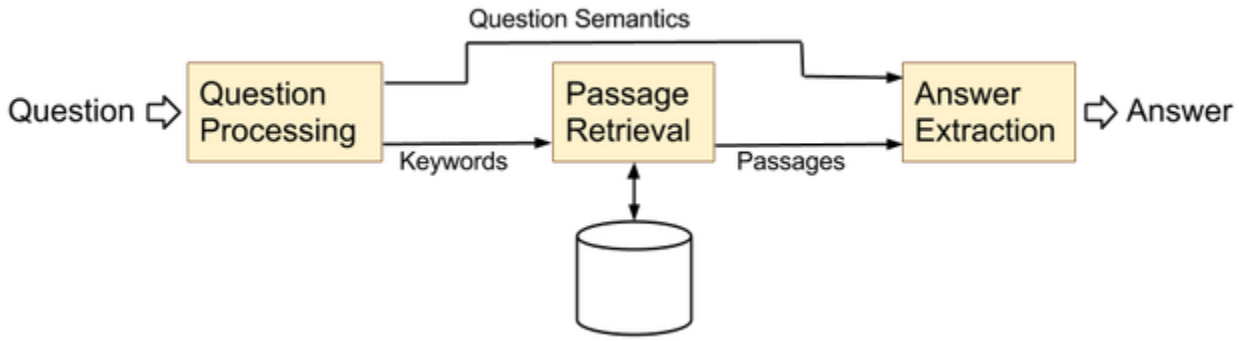

Figure 1: Overview

As mentioned, the unit of indexing and search is called a document in Lucene, the programmer decides what data and how the data is recorded into these. Since paragraphs were of the main concern in this case, it was decided that each document should contain one paragraph each. Throughout this paper, the words document, passage and paragraph are used interchangeably.

4.3.2 Searching

The searcher evaluates the similarities, if any, between the search query and each of the documents in the database, and then returns the hits ordered by similarity. A closer match between the search query and a document will result in a higher similarity score.

4.3.3 Similarity

Lucene uses a combination of the boolean model and the vector space model of information retrieval, and provides several implementations of different ranking functions. The custom ranking function that Lucene uses by default is depicted in Equation 1. The function gives the similarity score of a document d for the query q, where q can be a query of several terms/words t. The different components of the function are described as follows:

- coord(q,d) - accounts for the Boolean Model mentioned earlier, and gives a score based on how many of the query terms were found in the document.
- queryNorm(q) - allows for comparison of scores between different queries.
- tf(t,d) - the term frequency.
- idf(t) - the inverse document frequency.
- t.getBoost() - a factor that enables manual boosting (increase relevancy) of specific terms in the query.
- norm(t,d) - accounts for the varying lengths of all the documents scored. This component also includes boosting capabilities, but in this case for the different fields of a Lucene document.

The implementation of tf(t,d) and idf(t) may vary among different ranking functions. Lucenes default ranking function implements these as in Equation 2.

$$tf(t,d) = f(q,d) \qquad idf(t) = 1 + log(\frac{N}{df(t)}) \tag{2}$$

where f(q,d) is the actual frequency of a term in a certain document, N is the number of documents in the collection and df(t) is the number of documents that contain the term.

4.3.4 Analyzer

The Lucene package includes a wide array of different analyzers to accommodate for word stemming and stop words in different languages. The analyzers are used both during indexing and search, and need to be the same in both cases, i.e. an index created using the Swedish analyzer must be searched using the Swedish analyzer. In short, to achieve support for Swedish language, the Swedish analyzer was used in every instance throughout the project.

5 Answer Extraction

The answer extractor processes the information retrieved by the passage retriever, and attempts to find candidate answers within. A candidate answer is a word which is of the same type that the answer is expected to have. The answer extractor is also responsible for estimating the likelihood of each candidate answer being the correct answer.

The candidate answers were found by tagging the retrieved passages, using a Swedish part-of-speech tagger called Stagger [6]. These were then ranked according to occurrence (normalized form). This procedure was based on the assumption that the answer should occur more frequently than other words, since the passages were retrieved by searching for words more closely related to the answer.

$$score(q,d) = coord(q,d) \cdot queryNorm(q) \sum_{t \in q} [tf(t,d) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(t,d)] \tag{1}$$

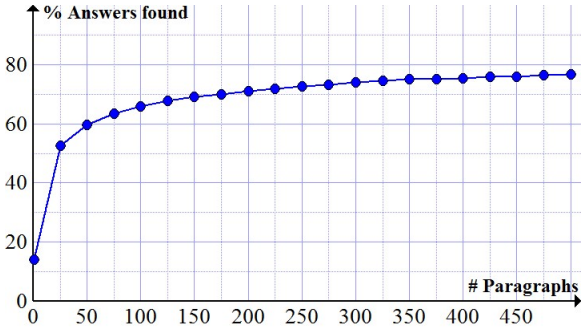

Figure 2: Percentage of answers found.

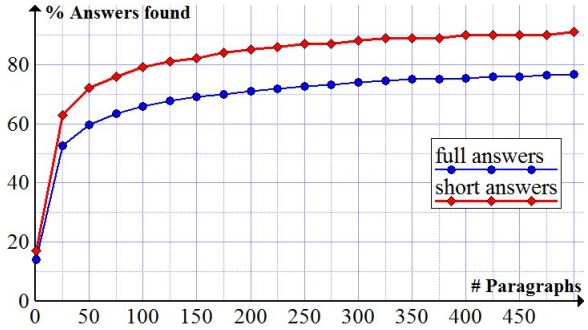

Figure 3: Percentage of long and short answers found.

6 Results and evaluation

In the first step we used Lucene, as described above, to extract paragraphs from Wikipedia. As queries we used the questions of ”Kvitt eller dubbelt” together with the subcategories of the question cards. For each of the questions we checked if its answer was present in the retrieved text. For this test we used a set of 1374 questions.

In figure 2 the results of this test is presented. It shows the percentage of answers that were found for different numbers of paragraphs extracted. This shows, that when looking only at the one paragraph with the highest similarity to the Lucene query, the answer to the original question was present in that paragraph for 14 percent of the questions. For 300 paragraphs, the answer was found for 74 percent of the questions, and for even larger amounts of text extracted not much improvement was found.

6.1 Matching answers

In this stage of our project, we used a very basic approach to determine if an answer was present in a text passage. We checked if the exact string of the answer was present in the text. An occurrence of a lowercase version of the string also counted as a match. No other normalization of the answers were used in this step.

This simple method for matching answers meant that some answers were very unlikely to be found. There are answers within the ”Kvitt eller dubbelt” game that are in the form of full sentences, and also answers which gives a list of alternative answers, and questions of the type: ”name one of the two”, were both possible answers are given but either would be correct. To see how big influence these types of answers had on our results, we did the same test again, but now only using questions were the answer consisted of only one word. In figure 3 you can see the frequencies for found answers. On average, the improvement is about 10-15 percentage points.

Even with this simplification, there are surely correct answers that are missed. This limited matching of answers means that our results describes a low limit on the actual occurrence of answers that would be considered correct during a game of ”Kvitt eller dubbelt”. In reality, the number of correct answers present could be higher.

6.2 Ceiling

One interesting thing to investigate is how many of the answers to the Kvitt eller dubbelt questions that are actually present within Wikipedia. This sets a limit (ceiling) to the results that are possible for us to find, even if we allowed for the whole of Wikipedia to be used as our retrieved passage.

We examined this limit for our different tests using the set of 1374 questions. We found that, when using questions with full answer (without explanation), the answer was found in Wikipedia for 1238 (90.1%) of the questions. For the 1013 of these questions that had one word answers the answer was present in Wikipedia for all but 13 questions, this means that the answer was found for 98.7% of these questions. 449 of the questions had the two properties that the answers consisted of only one word and that they were classified as entity. For these, 98.5% of the answers were found. An answer was considered present if the exact string of the answer was matched, or if a lower-case version of the string is matched. No other stemming or normalization was used.

The conclusion of this is that no matter how many paragraphs we would extract, the answers can not always be found. For questions with full answers this has a larger influence on the result.

About 10% of the full answers are not present in Wikipedia. We earlier made the assumption that one word answers would be easier to find, and the results for these answers (98.7% found) shows that that assumption was true. See Figure 4.

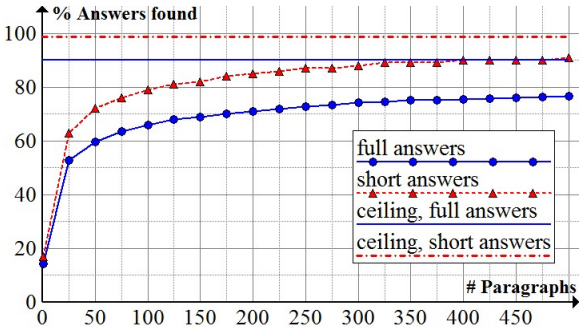

Figure 4: Ceiling of answers present in Wikipedia.

6.3 Generating and ranking candidates

The next step was to try to extract candidates for answers from the retrieved passages. As mentioned previously, the text was tagged with part of speech by Stagger. We limited our search and included only questions were the answer had been classified as entity. We also only used questions with one word answers to get around the problem of automatically matching candidates to the correct answers.

To match the type ”entity”, we considered only the words in the retrieved passages that were tagged as nouns as answer candidates. Stagger also normalized the words, and then the frequency was counted for each of them. Words with many occurrences in the retrieved passages were considered as possible answers.

At first we did a check to see how many correct answers this method could find for different numbers of passages retrieved. The results are shown in Figure 5. When using 300 paragraphs or more, about 75 percent of correct answers were found.

Here is an example that explains our method. First, the question *Vad kallas hundens ungar?* (What do you call a baby dog?) is turned into a Lucene query, and when extracting 150 paragraphs, 1783 words are returned. These words are then tagged, and all nouns are selected. The text, in this case, had 384 different nouns, which were ordered according to frequency.

Here, the correct (normalized) answer, *valp* is present 4 times and gets the rank of 8, it is the 8th most common noun in the retrieved text.

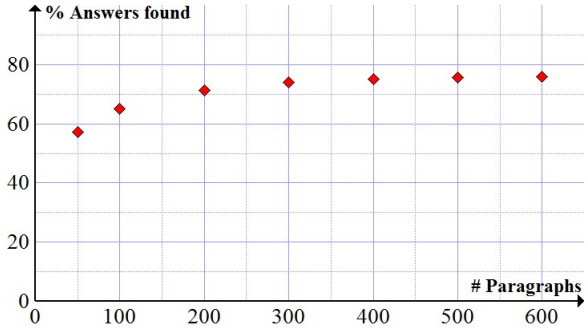

Figure 5: Found answers among tagged candidates.

This means that the original source of information, the articles of Swedish Wikipedia, which has a size of about 700 MB, is reduced to an ordered list of 384 words.

In Figure 6, the rank for the correct answers are shown. Ranking here is strictly based frequency for each answer, and a high frequency gives a low rank which means a likely answer.

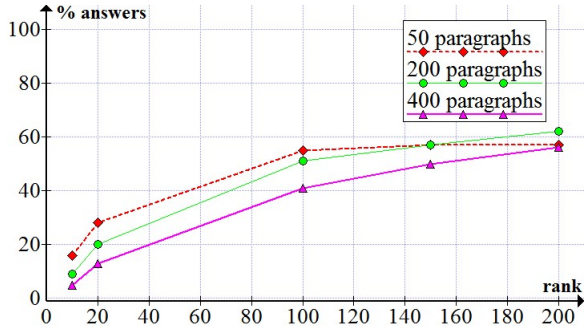

Figure 6: Ranking of answer candidates.

Figure 6 shows a sort of distribution for the ranking of candidate answers for different numbers of retrieved paragraphs. Each point in the diagram means that a certain percentage of the correct answers were given this rank or a better rank.

When using 50 paragraphs the answer was found for 57.2 percent of the questions and the answer was among among the top 100 nouns/candidates for 55 percent of the questions. If the search was extended to 400 paragraphs, the answer was found for 75.0 percent of the questions and the correct answer was among the top 100 nouns/candidates for 41 percent of the questions. There is a tradeoff between how many paragraphs that we retrieved and the ranks of the correct answers. The more paragraphs retrieved, the more likely it is that the correct answer is found, but the rank of the correct answer is also lower.

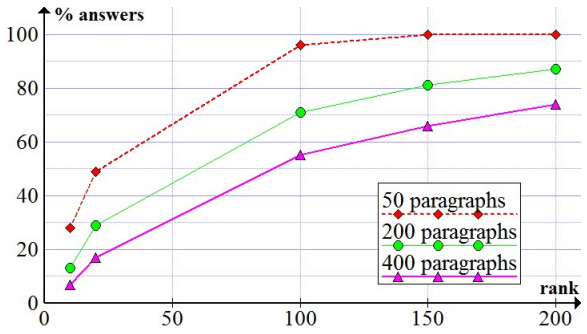

Figure 7: Ranking of answer candidates for found, correct answers.

The more text extracted in the first step, the more ”garbage” will be found. The extreme would be to rank all the nouns in the whole Wikipedia, then the answer would be present with high probability, but it would be more difficult to find it among all the extracted words.

This is more clearly visible in figure 7 where the rank is presented only for the questions where the correct answer was actually found. Here we can see that for 150 paragraphs, all answers are ranked 150 or lower, but for 400 paragraphs, only 66 percent of the answers are ranked 150 or less.

7 Conclusion

During this project we have managed to create a system that finds and ranks answer candidates for the questions of Kvitt eller dubbelt. For short answers, we managed to extract passages of text that contained the answer for 91 percent of the questions, using Lucene. And for short answers of type entity, we found and ranked that the correct answers for 75 percent of the questions. Ranking was done by using the frequency of the most common nouns in the retrieved text. Thus, the original source text of the whole of Swedish Wikipedia was reduced to a list of ordered words.

It would be possible and maybe even necessary if one was to develop the project any further, to consider the long answers as well. To accomodate for these one would need to devise some way of comparing and determining if the answer was present in a passage of text.

As shown, the answers to most of the questions were contained in the Wikipedia database, but to reach 100% the database would have to be extended with other sources as well.

We have tried some techniques for finding text passages and extracting answer candidates from them. In a full scale system, it would be necessary to improve on the accuracy of the extraction and ranking, and to take more word classes than nouns into account. This would have required more advanced techniques than the ones we used, one example would be to utilise named entity recognition.

References

[1] Murdock et al. (2012) *Textual Evidence Gathering and Analysis*, IBM Journal, volume 1, no. 3.

[2] Lin, X. Roth, D. (n.d.) *Learning Question Classifiers*.

[3] Apache Software Foundation. Apache Lucene. `http://lucene.apache.org/core/` [2013-06-04]

[4] Wikimedia Foundation. Wikipedia. `http://dumps.wikimedia.org/svwiki/latest/svwiki-latest-pages-articles.xml.bz2` [2013-06-04]

[5] University of Pisa. Multimedia Laboratory. Wikipedia Extractor. `http://medialab.di.unipi.it/wiki/Wikipedia_Extractor` [2013-05-23]

[6] Stockholm University. The Department of Linguistics. Stagger - The Stockholm Tagger. `http://www.ling.su.se/english/nlp/tools/stagger/stagger-the-stockholm-tagger-1.98986` [2013-05-22].