Reranking using Supervised Learning in a Question Answering System

Mattias Eklund D10
Lund Institute of Technology
mat10mek

Christian Frid D10
Lund Institute of Technology
ada10cfr
@student.lu.se

Adam Wamai Egesa D07
Lund Institute of Technology
ada07awa

Abstract

This work will present the result and findings of the project in Language Technology that were conducted by the authors of this paper. The project consisted of the implementation of a Question Answering System for Swedish using the Swedish version of Wikipedia as the knowledgebase. The system reranks results from a traditional passage retrieval of the top nouns using supervised learning. Testing of the system showed 69% of the questions could be answered with reranking improving the average correct answer’s rank noticeably. Improving the median from 43 to 34.

1 Introduction

This project’s main focus was to implement a system that would answer a question with an answer. The answers would then be analysed to see why that answer was given and how it could be improved.

This project was initially thought of as mainly educational, wherein the partitioners of the project would learn the concept of, and basic implementation of a question answer system, and if there were time left at the end of the project, understanding of the shortcomings of their project.

To give some context to the project, it is necessary to point out that this is by no means groundbreaking. Similar systems already exists, for instance IBM Watson, the computer that won a game of Jeopardy against skilled people. IBM Watson is a system that gives answers according to the rules of the game; given an answer, what was the question? IBM Watson can do this due to its huge knowledge base and capability to understand naturally constructed sentences.

Our system is similar to some extent and we are not using the same variety of data. Language technology and AI is more at a research state rather than in a market state right now. It needs more attention to make it further up the ladder and eventually into products. Hopefully in a smaller packaging than the IBM Watson since it is big as a car. We view this project as a contribution to the interest of language technology among researchers and as an attempt to spread awareness of something that might be a very handy technology in the future.

The structure of this paper is as follows; Substance, where the project will be described, Related work, where the project will be compared to related work in the field, and Conclusion, where the results of the project are summarised.

2 Substance

The project has been divided into three smaller parts to make it more manageable. The parts are the Preemptive work, Question, and Answer. The preemptive work consists of setting up the backbone of the system. The Question part of the project covers the interpretation of the question, and the Answer part covers the answer and answer validation.

2.1 Preemptive work

For this part of the project, multiple tools were required. The main tool here was Apache Lucene (Foundation, 2014a). Lucene provides a powerful indexing and search tool that can be integrated into Java programs. There is also a web based version of Lucene called Solr (Foundation, 2014b) that was considered for the project but was discarded due to lack of understanding of the tool and time constraints. Instead, Lucene core was integrated into a standalone program.

The backbone of the system consists of an indexed knowledgebase. The knowledgebase used for this project was a dump of the Swedish Wikipedia from third quarter of 2014 and was acquired from
Wikidumps. The wikidump consisted of a 1.3GB xml file. The file could of course be indexed as it were, but this would result in that all possible answers would be located in “one article”, the wikidump. To make the wikidump useable with the system it needed to be broken down into articles and all the xml tags had to be removed from the resulting text.

By parsing the xml file with a program called Wikiforia (Klang, 2014) the xml-tags were removed and the articles were separated into separate text files. The functionality and implementation of Wikiforia is not a part of this project and will not be covered by this paper.

The articles were now indexed by Lucene. The index gives a list of all words and in what articles they appear. This index would be used by the program later to look up articles that possibly would contain the answer to the question.

At this point the general backbone of the system was set up. It was now possible to search the indexed files for specific words and get a list of articles that contained the word.

2.2 Question

To be able to answer a question, one must understand what is being asked. For humans this might seem trivial, but how do one teach a machine to recognise a type of question?

There are a lot of different types of questions, e.g. what, where, and why. So for example if someone ask you:

**Who was the king of Sweden before Carl XVI Gustaf?**

you would realise that you are being asked for the name of a person that was the king of Sweden just before Carl XVI Gustaf, i.e. Gustaf VI Adolf. This however, is not obvious for a computer.

Most search engines will most likely answer with Carl XVI Gustaf and provide a link to the wikipedia page wherein you could find the answer to the original question through further reading. This is however not desirable. The question has to be understood for the program to be able to produce a correct answer. The system implemented during the project utilises LibShortText (Group, 2013) to analyse the question to try and deduce what is being asked. The program can at this point deduce if the question is about a person or a location. This is the question classification. This classification is then passed on to the next phase where possible answer candidates are located in the indexed files.

At this point the question is parsed by the Stockholm Tagger (Östling, 2013) to extract noun(s). These nouns becomes the keyword(s) that Lucene searches for in the indexed knowledgebase.

2.3 Answer

The answer stage of the program tries to find an answer with the constraints from the preceding phase, such as keywords and question classification.

Lucene searches the index for articles containing the keyword(s) and returns a list of articles. These articles may contain an answer of interest. Answers of interest are nouns, but not every noun is a correct answer. The program now ranks the nouns after a simple algorithm

\[ \forall n \in A, \sum o(n, a) * s(a) \]

where \( n \) is a noun,

\( A \) is set containing the articles from the Lucene,

\( o(n) \) is the number of occurrences of \( n \) in an article \( a \in A \),

and \( s \) is the Lucene score for an article \( a \).

Each answer score is then multiplied by the probability that the given answer is correct. This is determined by first predicting the answer classification for the given question using LibShortText. Each answer’s classification, given by Stagger, is then evaluated against the predicted classification given through the LibShortText model. If the answer’s classification is found to be the same as the predicted classification the answer score is multiplied with the probability that it is correct. While if they are found to be different the answer score is multiplied with the probability that it is incorrect. The probability that the answer score is correct is statically determined to be 0.74 and 0.26 that it is incorrect. This is derived from testing the classifier. Can be further noted that any answer that did not receive a classification from Stagger is treated as an incorrect prediction. In the GUI the answers are presented with the highest score at the top and the lowest scores at the bottom. High score correlates to the most likely answer and vice versa.
2.4 Results and Evaluation

In our implementation we use a model for identifying whether the question means we are looking for a person, location, concept, et cetera. These we call answer classifications. Taking 90% of the corpus to train the model and the remaining 10% to test it yielded an accuracy of 74%. Meaning that roughly every fourth question is thought to be classified incorrectly. The corpus in total amounting to 2310 questions with answers and answer classifications.

The system was tested using half of the corpus used for testing. i.e. 5% of the total corpus. 115 questions were thus tested against the system. For the passage retrieval the top 50 articles were retrieved with the nouns ranked without the reranking step comparing answer classifications outlining the baseline that was measured with the effect of reranking the answers then measured after it. An answer was further deemed to be correct if the lemma of the found answer was in the words of the correct answer (if there were more than one word). No consideration was shown towards any false positives this might have created.

Of the 115 questions tested, no answer was found in 35. Meaning that the top 50 articles were not judged to contain the answer in those 35. This further means that an answer was found for 69.6% of the questions. For the 80 questions that did have an answer the ranking was improved slightly through the reranking step. The baseline having an MRR of 0.106, mean of 329 and median of 43. The reranking improving this slightly to an MRR of 0.138, mean of 320 and finally a median of 34.

It was further observed that 30% of the nouns tagged by Stagger were successful. 70% were hence unsuccessful ending up not being tagged at all and thus in this system instead always considered incorrectly predicted and thus multiplied with the lesser value of 0.26 instead.

It further took a full minute on average (61 seconds) to test each question on a 2011 MacBook Air. Varying in execution time from 4 to 900 seconds. It OS did however also not likely give it priority at all times as other applications were being used at the same time.

3 Related Work

3.1 Answer type classification

In A Probabilistic Answer Type Model (Pinchak and Lin, 2006) an unsupervised model was used to classify answer types, specifically a probabilistic answer type model is used. Using an unsupervised model is here found to have “the added benefit of being easily adapted to different domains and corpora which a list of explicit possible answer types may be difficult to enumerate and/or identify”. i.e. one conclusion was that data could some in cases be more easily gathered using their unsupervised learning.

(Pinchak and Lin, 2006) further puts this in the context that previous systems using supervised learning and manually constructed rules struggle with finding appropriate predefined answer types. Stating that there are always answer types fall outside the scope for any of predefined answer types. While the problem may be mediated by having very general answer types this in term decreases its actual worth to the system as this also increases the amount of incorrect candidates. Instead choosing very specific answer types further enhancing the problem with an increased difficulty to tag the answer types correctly.

There have also been attention to more specific question answering systems, such as looking at improving answer typing for how-questions (C. and S., 2007).

3.2 Swedish Watson

The currently (at the date of this document) most advanced and best question answering is Watson by IBM (Ferrucci, 2012). Watson is however only tailored for the English language whilst the work in this document has focused on Swedish that suffers from dramatically less available resources and research. Where this project and others (Pyykkö et al., 2104) aims to achieve the success that Watson has achieved by essentially making a Swedish version.

In J. Pyykkö, R. Weegar, P. Nugues, 2014 (Pyykkö et al., 2104) a passage retrieval in a question answering system is implemented using the same corpus from the board game Kvitt eller dubbelt as was used in this project. In addition to also using an edition of the Swedish version of Wikipedia as the knowledgebase. Further also using Lucene
to index the knowledgebase as well as passage retrieval. Unlike the project for this paper, J. Pyykkö, R. Weegar, P. Nugues, 2014, further segmented each article into paragraphs. They then goes on to show that the answer was present in the top 300 paragraphs returned through passage retrieval of the questions contained in the previously mentioned corpus. Specifically up to 75% of the time for named entities, 91% for one-word answers. Showing the strong use for Wikipedia as a knowledgebase for the given corpus but that for named entity especially the solution was lacking. Marcus Klang and Pierre Nugues (Klang and Nugues, 2014) looks into named entity disambiguation for Swedish in general.

Jakob Grundström and Pierre Nugues (Grundström and Nugues, 2014) uses the same knowledgebase, corpus, indexing and passage retrieval as (Pyykkö et al., 2104) with the shifted focus to show reranking of answer candidates. With a focus on one-word answers. The main feature used for reranking was lexical answer typing but this was combined with other features and ultimately compared to a baseline solution using only the score given by Lucene and the frequency of each noun. Where significantly better results were shown. Improving both the mean and median rank. Specifically the median rank improved from 21 to 10.

4 Conclusion

Our QA system does not seem to measure up to previous work quite yet but does point to the same conclusions. Supervised learning can be used to noticeably increase the ranking of answers through an extra reranking step by predicting the answer classification given a question.

QA systems in general are alternatives of asking a computer for knowledge and is a much more natural one at that. As mentioned before, searching for specific words is what the common user is most familiar with but our system is easier to learn for technically unexperienced people. We believe that when systems like ours gets perfected it will replace some of the functions of a search engine.

Our program is able to learn what it should be looking for when presented with a question and then look up the best answer quickly through its knowledge base. The results (answers) of the question is given in classic search engine style with the top result at the top and the least likely result at the bottom.

As mentioned there is however room for a lot more improvements to this project.

The accuracy of the model was evaluated to be 74% but could easily be more precisely evaluated through a 10-fold cross-validation. Simply shifting the part that is already being extracted. The accuracy could regardless still likely be noticeably increased. The corpus used for training is only a little more than 2000 entries when there are well over millions of different questions that can be asked by humans. While LibShortText has support for a corpus of millions of entries as well. The difficulty lies however in gathering a data of that size given the very specific answer classifications that need to be given by some human for that to happen. For that reason alone it might be prudent to explore unsupervised learning techniques more. It is however possible that noticeable improvements can be made by simply changing the supervised learning model or tweaking it. Very little time was ultimately spent exploring different models. We are currently using linear regression as a model.

For pure speed improvements we could tag everything with Stagger prior to passage retrieval. Rather than tagging it in real-time as each passage retrieval is executed. As over 95% of the execution time was Stagger doing tagging it would increase the speed dramatically. It would also take some time to do it. Tagging an article took on average around 60 seconds on a 2011 Macbook Air with about 1.3 million articles it would thus take around 900 days. The variance on the execution time was however huge and could easily be cut down by a factor of five to ten or more. Another consideration is however that tagging increases the storage requirements by a factor of around five to ten times, when simply outputting the tagged file form Stagger. This could however likely also be optimised as most of the Stagger output is not used.

Stagger only tagged about 30% of the nouns found. Improving this to the 90+% that has previously been achieved on different corpuses by Stagger would thus likely substantially increase the results as the reranking is dependant on the correct answer actually being tagged by Stagger. It was however not measured how many of the correct answers
that were not tagged by Stagger, meaning it is unknown how big of an impact this had. Considering the high amounts of failure it is however quite likely that it had a very negative impact on the performance of the reranking.

Using paragraphs instead of articles could likely improve the speed of the system as it becomes more granular picking out less that needs to be tagged by Stagger. Though it at worst could suffer from disconnecting paragraphs from articles.

Though LibShortText does calculate probabilities for each class given a prediction we only got the top prediction each time. This was mainly due to time constraints coupled with an inexperience with LibShortText and LibSVM. Future work should amend this shortcoming and use the actual prediction for each class instead of merely hard coding a static prediction for the predicted class and the inverse for anything that does not fit. This would likely improve the performance substantially depending on how well Stagger is tagging the correct answers. Handling all answer candidates that do not have a classification would regardless still be an issue and future work would need to develop a new strategy for how to combat this.

When using the program it can sometimes misinterpret the question and give some ridiculous answers. When asking the question “Where is London located?” you don’t want the program to answer “London” but something different. Therefore we could simply filter out any answer that is similar to the relevant content in questions such as the one proposed. Another thing that could improve performance would be intentional positive weighting to answer classifications such as “person” when the question asked contained the word “Who”, “location” for “Where” etcetera. Stagger could here be used to get the lemma of a word to be sure that the correct word is removed. There are however edge cases where the answer is contained in the question. A multiplied weight to its score might therefore be more appropriate, to not completely rule all the edge cases out.

When programs like this are more advanced (and in a sense “perfected”) we can truly start to consider the implications of AI and how interactions with computers and computer programs can be more like interactions with human beings. We are pretty far from that but when one starts to imagine, ideas emerge and possibilities can be considered more seriously.

Acknowledgements

The project was undertaken in the course Projects in Language Technology with course code EDAN70 at Lund Institute of Technology, Sweden. Supervisor for the course and the project was Pierre Nugues (pierre.nugues@cs.lth.se). Credit also goes to Marcus Klang (Marcus.Klang@cs.lth.se) for his software Wikiforia, which was used to parse the Wikipedia dump to plain text.

References


David Angelo Ferrucci. 2012. Introduction to “this is watson”. ibm journal of research and development.


Christopher Pinchak and Dekang Lin. 2006. A probabilistic answer type model. in acl.
