Question Answering System - Hypothesis Generation and Reranking of Answers

Jakob Grundström
π07, Lund Institute of Technology, Sweden
jakob.grundstrom@gmail.com
Supervised by Pierre Nugues

Abstract

This report describes a simple question answering system for Swedish. The system includes question analysis, hypothesis generation and reranking of answers.

The state of the art, question answering system IBM Watson served as inspiration. However, instead of *Jeopardy!* questions this system was trained and evaluated on questions from Swedish television quiz show *Kvitt eller Dubbelt - Tiotusenkronorsfrågan*.

A HTML dump of Swedish Wikipedia was used as knowledge base. Paragraph retrieval from Swedish Wikipedia gives acceptable coverage of answers when targeting *Kvitt eller Dubbelt* questions, especially for single-word answer questions.

The hypothesis generation module reproduces the results of [J.Pyykkö and R.Weegar2013] and expands on it. The question analysis part, developed in collaboration with *Anders Tilly*, performs lexical answer type prediction. Sorting answer candidates according to occurrence in the most relevant paragraphs gave a baseline ranking that performed better than expected. The reranker make use of information from the previous stages to estimate the correctness of the generated answer candidates. The correctness estimate is then used to re-weight the baseline ranking.

A 5-fold cross-validation showed improved ranking performance compared to baseline.

1 Introduction

This report describes a simple question answering system for Swedish. The system includes modules for question analysis, hypothesis generation and reranking of answers.

The state of the art, *Jeopardy!* playing question answering system IBM Watson [Gondek et al.2012] served as main source of inspiration. However, instead of *Jeopardy!* questions this system was trained and evaluated on questions originally from the 1950-60s Swedish television quiz show *Kvitt eller Dubbelt - Tiotusenkronorsfrågan* [Kvi2013]. The questions have been transcribed into *RDF* graph data locally at Lund Institute of Technology by previous students.

The free text resource Swedish Wikipedia [Swe2008] was used as knowledge base. As investigated by [J.Pyykkö and R.Weegar2013] on the same corpus of questions, paragraph retrieval from Swedish Wikipedia gives good coverage of answers, especially for single answer questions. For the purpose of answering *Kvitt eller Dubbelt* questions using Swedish Wikipedia was deemed to be the best starting point.

The system aims at answering questions of type and complexity similar to the *Kvitt eller Dubbelt* questions.

The hypothesis generation module reproduces the results of [J.Pyykkö and R.Weegar2013] and expands on it. The question analysis part, developed in collaboration with *Anders Tilly*, performs lexical answer type prediction and is responsible for producing a search query from the question text. The reranker module make use of information from the other modules to estimate the correctness of the generated answer candidates and thus create a ranking.

2 Background

As mentioned above the main source of inspiration for this project is IBM Watson. In 2011 IBM Watson gained fame by beating the two best human champions in a real-time two-game Jeopardy! competition [Ferrucci2012]. IBM Watson sets a

high standard and is, at the time of writing, considered to be the state of the art in question answering.

The IBM Watson system includes, in addition to its question answering core and the game play implementation, human-computer interaction components such as: speech recognition, language interpretation, answer formulation and speech synthesis. In this project the focus is on the question answering core.

The Watson system takes full advantage of the Jeopardy! format of questions. For instance in finding the *focus* (the part of the question that refers to the answer) and the *LAT* (the lexical answer type, which indicates the class of the answer) in the question analysis. Its question answering architecture *DeepQA* is heavily modularized and allows for multiple implementations that can produce alternative results. This creates alternative paths through its pipeline, paths that independent and can be pursued in parallel [Ferrucci2012]. Many candidate answers can be proposed using several different search strategies [Chu-Carroll et al.2012], for example: document search, passage search, lookup in structured databases with extracted facts (triples) or using knowledge graphs such as IBM's frame database *Prismatic* [Fan et al.2012]. This also allows for pursuing different interpretations of the question and category. More and more features and evidence is gathered and analyzed for each answer before the final reranking and narrowing down the answer candidates. Watson can then use the estimated confidence to decide on its Jeopardy game strategy, e.g. whether or not to answer a question or how much to bet.

This was all done for English. Is it possible to successfully use similar techniques for a Swedish question answering system?

In our project however search strategies are restricted to document search over paragraphs and the design of our system roughly corresponds to a simplified single path through the Watson pipeline.

3 Question Corpus

Thanks to previous students a corpus of Swedish questions from the quiz show *Kvitt eller Dubbelt - Tiotusenkronorsfrågan* was available locally at Lund Institute of Technology.

The question corpus contains 2310 questions. Out of these 1683 questions are single-word answer questions and most of the answers are nouns. Because of the *Kvitt eller Dubbelt* game play the questions are organized on 385 named cards. The cards are divided into 7 categories with 6 questions of different value on each card. Each question have a *question text*, an *answer*, optionally complemented with an *alternative answer*, and is annotated with one of the 9 *answer categories* listed in table 1.

In table 2 three example questions is shown. Note that some fields are sparse and that the categorization is partly incomplete. The answer category for question 3 should, for instance, probably be marked as a location. The question text in all the examples are single sentence and so are most of the question texts. However, question texts ranging from single noun phrases to multiple sentences occasionally occurs in the corpus.

Table 1

Answer Categories
misc
abrev
action
desc
description
entity
human
location
numeric

Since our objective is not to implement the game play but simply question answering, we only make use of the question text, answer, alternative answer and answer category fields. [1]

4 System Description

In the following subsections the system is described.

4.1 Overview

Figure 1 illustrates the path from question text to list of ranked candidate answers.

- Input to the system is a full text question.

[1] Card category and card name were only used experimentally in the question analysis part when formulating queries, however in the end only the question text was used to build queries.

Table 2

Field	1	2	3
value	250	5000	5000
text	I vilken stad sitter Sveriges regering?	Vem är kapten på rymdskeppet Millenium Falcon?	I vilket land är Rom huvudstad?
answer	Stockholm	Han Solo	Italien
alt answer	-	-	-
answer category	location	human	-
answer subcategory	-	-	-

- The **Question analysis** module predicts answer type using a trained answer type classifier and is responsible for building a search query.

- The search query is used by the **hypothesis generation** to find the most relevant paragraphs from the indexed free text resources, in this case Swedish Wikipedia. Candidates are generated from the retrieved paragraphs. Associated information that might be useful in later stages (Part-Of-Speech, dependency graphs, search score, etc.) stored for each candidate.

- Candidates are **merged** on equal lemma (normalized word form), the candidate features are merged with independent strategies. Furthermore, statistics such as occurrence and word frequency is computed.

- The question object, predicted answer types and corresponding estimated probabilities, the merged candidates with their features and statistics are sent to the **reranker** module and is used by a trained reranker model to create the final ranking.

- Output from the system is a list of answer candidates ranked by an estimated confidence.

4.2 Question Analysis

The question analysis module take the question text as input and is responsible for building a query for searching paragraphs. For query building only a few approaches were tested. An experimental test using the card category (i.e. the question topic), the answer type and the question text as query showed only a small gain in the number answerable questions and a drop in the rank of the first found correct answer, when compared to only using the question text as query. A question is considered *answerable* if the answer is found among the generated candidates.

Possible improvements or other approaches includes: generate queries consisting of extracted keywords only, create a better topic categorization and train a model for predicting, use more fine-grained answer types, use more than one query to generate candidates.

The query is sent to the hypothesis generation module.

In collaboration with *Anders Tilly* a *LibShortText* [Yu et al.2013] logistic regression model predicting the answer type from question text was trained and integrated into the question answering system. Training data could be produced using answer types and question text available from the question corpus.

LibShortText is a library for short-text classification. It basically implements document search (short-text as documents) using the vector space model. LibShortText uses the bag-of-word model to generate future vectors. We used features represented as TF-IDF (term frequency - inverse document frequency).

LibShortText provides pre-processing in the form of tokenization, stemming and stop-word removal. It should be noted that the stemming and stop-word removal is for English and thus of no use for this project. In our system we instead implemented stemming using Stagger [Östling2013], and thus added the possibility to use normalized question text for both training and predicting the answer type.

From the decision values predicted by LibShortText when using a Logic Regression classifier the probabilities of each class could also be computed. This enables us to use more then one answer type.

The top two predicted answer types and their associated estimated probabilities are predicted by

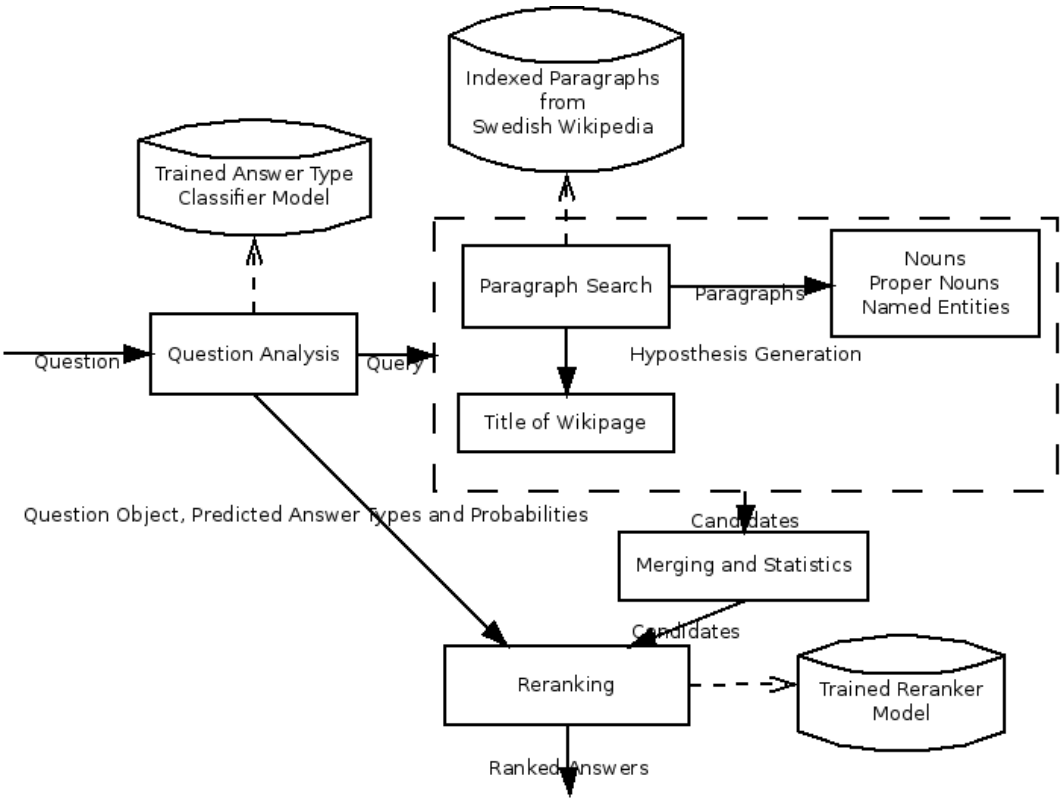

Figure 1: System Overview

the trained model and then stored in the question objects for later use in the reranking module.

4.3 Hypothesis Generation

The hypothesis generation component is implemented using paragraph-based search. Other units of text, e.g. article or sentence, could work as good. But paragraphs are assumed to give good performance because they contain strongly related information, whereas an article may contain unrelated information. Using single sentences might discard some useful information, such as when corefereneces span multiple sentences. Anyway, in a larger system multiple search strategies of different granularities and with distinct algorithms could be used in parallel, see 2.

To make searching the huge amounts of HTML data, in case of the Swedish Wikipedia about 10GB, feasible the paragraphs were extracted using *JSoup* [jso2013] and an index was created using the search engine library *Lucene* [Apa2013a] with html paragraphs as documents (as in the information retrieval sense). Swedish stemming and stop words (*Swedish Analyser*) were used both when indexing and when querying.

The basics of Lucene scoring can be found in [Apa2013b] but a summary of the details will be given here. Lucene combines the Boolean model of information retrieval with the vector space model. Work is saved by only applying the vector space model for document scoring if the documents are first found by the Boolean model. By default the Lucene vector space model use TF-IDF weights. This means a vector $V(d)$ representing a document d from the set of all documents D and with entries corresponding to terms t, can be written as:

$$V(d) = (v_{t,d}) \tag{1}$$

$$v_{t,d} = tf_{t,d} idf_{t,d} \tag{2}$$

$$tf_{t,d} = (\frac{count_d(t)}{|d|})^{0.5} \tag{3}$$

$$idf_{t,d} = 1 + \log(\frac{|D|}{1 + |\{d \in D : t \in d\}|}) \tag{4}$$

where $v_{t,d}$ is an element of $V(d)$, $tf_{t,d}$ the *term-frequency* and $idf_{t,d}$ the *inverse document-frequency*.

The vector space model score of document d for query q is the *cosine similarity* of the TF-IDF

weighted query vectors $V(q)$ and $V(d)$.

$$sim(q,d) = \frac{V(q) \cdot V(d)}{|V(q)||V(d)|} \quad (5)$$

Lucene re-weights the cosine similarity for search quality, usability and efficiency. This leads to *Lucene's Conceptual scoring formula*:

$$score(q,d) = coord(q,d)qboost(q) \frac{V(q) \cdot V(d)}{|V(q)|} norm(d)dboost(d)$$

where

- $norm(d)$, normalizing V(d) to the unit vector removes all document length information. Lucene uses a custom document length normalization factor instead.
- $dboost(d)$, boost the weights on specific documents.
- $qboost(q)$, boost the weights on specific query terms
- $coord(q,d)$, reward for documents matching more query terms, which is usually larger when more terms are matched.

The hypothesis generation module retrieves the 60 most relevant paragraphs found by searching using the query from the question analysis. The restriction to 60 most relevant paragraphs is made to keep the system responsive enough for near real-time performance on an ordinary laptop and is arbitrary.

The free text of the retrieved paragraphs is processed by POS-tagger and NER (Named Entity Recognition) Stagger *Stagger* [Östling2013] and dependency parsed with *MaltParser* [Hall et al.2013]. The entity types recognized with Stagger is shown in table 3.

Table 3

Entity Types
person
place
inst
work
animal
product
myth
event
other

Nouns (NN), proper nouns (PM) and named entities are extracted as candidates. Named entities can consist of multiple words otherwise only single word answer candidates are supported. The candidate generation could be expanded to include whole noun phrases by applying a chunker on the sentences.

All information that might be useable for estimating the correctness of each extracted answer candidate is stored in the candidate objects, see figure 2. This includes: occurrence, frequency, named entity type, the sentence in which the candidate answer occurred in with POS and dependency graph, Lucene score, wikipedia article title, paragraph text.

Usually a lot of duplicate answer candidates is generated. That is why candidate merging is performed. Candidates are simply merged on equal lemma (case-insensitive). It is possible to use independent merge strategies for each feature. For occurrence accumulate could be appropriate. Keep most relevant named entity type according to Lucene score. Use *or* on boolean features to see if some event occurred at least once. Merging reduces the number of candidates significantly. Finally, hypothesis generation gives a list of candidates objects for reranking.

As a consequence of the paragraph search described above, candidates equaling the words from the queries often get both high frequency among the most relevant paragraphs and high lucene score. Usually, those answer candidates are incorrect. A simple stop-word filter removing those candidates has proved effective. Another approach that did not show as effective, was adding a boolean feature for if the word is contained in the question text or not.

4.4 Reranking

Simply ranking the generated answer candidates by the occurrence of the candidate answers in the retrieved 60 most relevant paragraphs proved surprisingly effective and was adopted as **baseline**.

Inspired by [Chu-Carroll et al.2012] we wanted to rerank (the process of creating a better ranking is called *Reranking*) the answer candidates using machine learning techniques on the the features and evidence gathered in previous modules. The

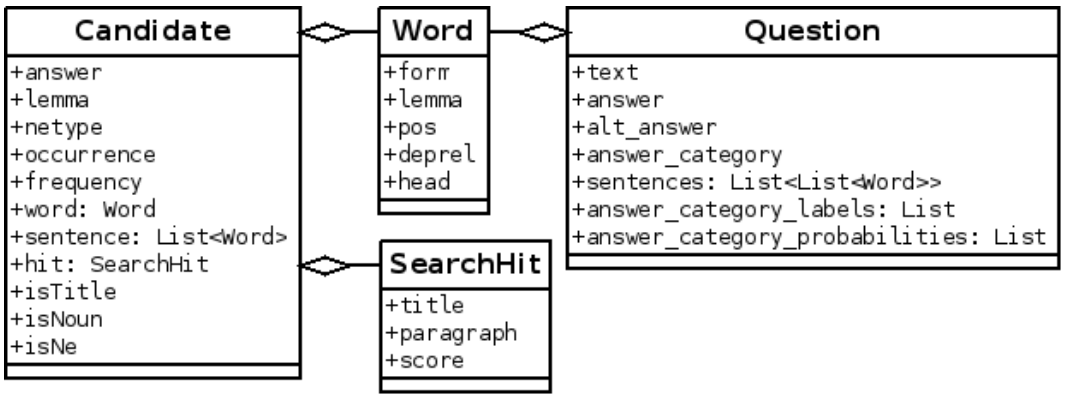

Figure 2: Candidate and Question UML

key idea is to use the question corpus to train a Logistic Regression model estimating the correctness of an answer candidate.

Features were extracted for each candidate-question pair. Labels were produced by comparing candidate answer to the known answer. A LibLinear [Fan et al.2008] Logistic Regression model was trained via the *Weka* machine learning framework [Hall et al.2009]. Due to the unbalanced nature of the data, only data from questions having a correct answer was used. Furthermore, the the number of examples of incorrect answers were restricted and the observations were chosen by uniform sampling. Nevertheless, the training data files ended up ended up with close to $500,000$ lines, each having 24 fields.

In addition to nominal, numerical or binary features shown in figure 2 or mentioned elsewhere, we used a bag of word of lexical features in an attempt to build a contextual model. The lexical features were extracted from both the question context and the answer candidate context. Triples, Subject-Verb-Object (SVO) frame projections, were extracted from the dependency graphs of the last question text sentence and from the paragraph sentences containing the candidate answers. Figure 3 shows an example dependency graph for a corpus question. The first word of the question sentence were added to the bag of words, as it is assumed to be important because it is usually is an interrogative (vem, var, när, etc.) and thus often say something about the answer type. For the lexical features a common vocabulary was built and only the most frequent words where kept as separate classes. Words not present in the vocabulary is represented with a missing value flag.

Table 4 gives an overview of the features.

Table 4

Type	Question	Candidate	Both
nominal	2 best answer types	netype	-
numerical	2 best probabilities	occurrence, frequency, lucene score, # words in answer	-
boolean	isMultiChoiceQuestion	isTitle, isNoun, isNE, isNumeric, *isDate*	isWordInQuestion
lexical	first, SVO	first, SVO	-

The predicted probabilities, or estimated correctness, from the model was then used to improve the ranking. Re-weighting the baseline score with the estimated correctness produced better results than ranking with it directly. The final reranker **MLReranker** re-weights the baseline score with both the Lucene score and the estimated correctness.

The results of the machine learning reranker was evaluated by comparing to baseline.

5 Results and Evaluation

The **correctness criteria** used for candidate answers is case insensitive equality on word form or on lemma for either the answer or the alternative answer given by the question. If one of the an-

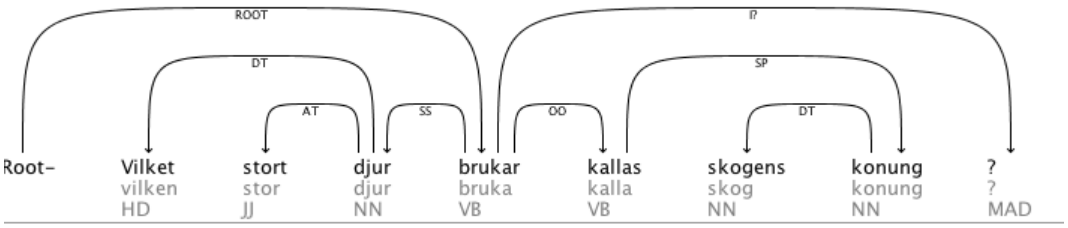

Figure 3: An example dependency graph for a question from the corpus. The graph shows both dependency relations, lemma and POS tags. In this case the SVO triple is *djur brukar kallas*.

swer candidates generated for a question fulfills the correctness criteria the question is considered **answerable**.

In order to evaluate the candidate generation we can look at the paragraph rank histogram normalized by the total number of questions and the cumulative distribution of that histogram, see 4. We can see that approximately 71 % of the questions are answerable if retrieving 500 paragraphs and considering all questions. If considering only the single-word answer questions up to 80 % of the questions are answerable.

As we restricted the number of paragraphs in the hypothesis generation to the 60 most relevant, we can at most expect that 57 % out of all questions is answerable and roughly 65 % of the single-word questions. [2]

To evaluate the reranker we look at the distributions of candidate ranks for the first found correct answer candidate, figure 5. The candidate ranks were computed using a 5-fold cross validation, i.e. dividing the question corpus in 5 parts, training the reranker model on 4 parts and testing it on 1 part. Then switching the part being the test set until all questions have been in tested. [3] The ranks for each fold were then combined in the same histogram. The upper plot shows the rank distribution of the baseline and the lower plot the rank distribution of the MLReranker. The figure shows that the distribution has shifted to the left in the lower plot, this means the rank of the first found correct answer candidate has decreased, which is good. Both the median and the mean has improved when using the MLReranker. The interpretation of the median is that: half the first found correct answer candidates are ranked better then 12 for the MLReranker and 21 for the baseline.

6 Conclusion

It was possible to find answers to a lot of question using primarily ordinary search engine technology. However, those results are like ”needles in a haystack” [Chu-Carroll et al.2012] and NLP techniques proved useful to improve the ranking of the candidates. The 5-fold cross-validation showed improved ranking performance. If someone implemented the *Kvitt eller Dubbelt* game, then maybe this system could at least beat a 4-year old child. But it is no where near becoming the next Jeopardy! champion.

Smaller gains in performance might gained form standardization of features, better handling sparse features, etc. There is probably more room for improvement in other places.

The *lexical answer type* is a very strong indicator of the correctness/incorrectness of a candidate answer. Using a more fine-grained answer type categorization might give significant improvements. Especially, if used together with better type resolution from the candidate answer side. For the MLReranker only named entities got a type, but at least Stagger's named entity types overlaps slightly (human,location) with the answer type of the questions. Using a knowledge graph (dbpedia, yago, etc.) the candidate answer could find its most popular equivalent representation through the ”isa”-relation.

Strategic verbs in the question texts can be identified, for example $XwroteY$, $X = authorOf(Y)$. Then use dbpedia or yago relations for those. From this we could find probable answer types and with a knowledge graph also

[2] The numbers differ slightly from [J.Pyykkö and R.Weegar2013], explanations for this can be that they used answer matching in free text as correctness criteria, i.e. not comparing to the extracted candidates, they used some subset of the questions. Anyway, all code is different and maybe not even the wikipedia dumps are the same.

[3] Note that the answer type prediction model was not part of the cross validation, i.e. it was trained using all corpus questions. The evaluation of the reranker can be considered optimistic or at least under the assumption that answer type prediction works ideally.

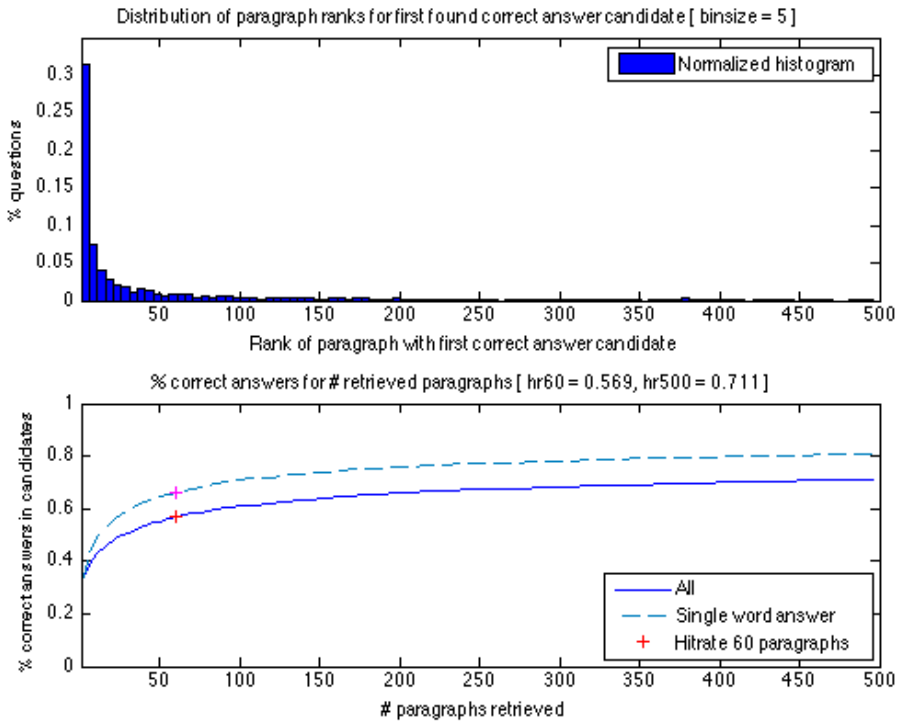

Figure 4: (u) Distribution of paragraph ranks for first found correct answer candidate. (l) Cumulative Distribution.

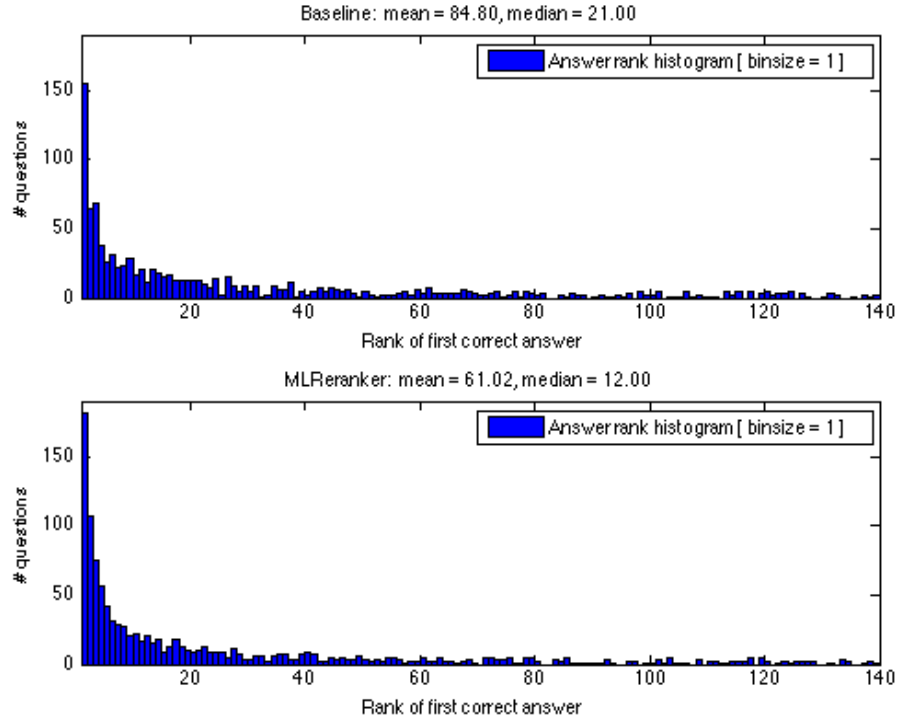

Figure 5: Distribution of candidate ranks for first found correct answer candidate. (u) Baseline (l) MLR-eranker

generate candidates.

One way to improve performance is to *specialize*. Specialize for example in location questions and use appropriate resources (geonames, etc.). Important question classes, for example always having the same answer type, that are easy to identify handled as special cases.

As mentioned in section 2, *more search strategies* could be used. Add document search, maybe with articles as documents. Add custom similarity measures, for instance distances to question words from the location of the answer candidate in the paragraph text, other information measures such as point-wise mutual information. Also use plain text passage search. Use anchor text and the title from the wikipedia articles as answer candidates. As in [Chu-Carroll et al.2012] we can find wikipedia documents with titles contained in the question text, *title in clue*. Use structured databases with SVO-triples and other relations to generate candidates. Create knowledge graphs such as *IBM's Prismatic* and compute aggregate statistics over frames extracted from free text to assist evidence gathering.

We could link to more *external resources* and web services, for example use freebase to infer answer candidate types and to reduce ambiguity or why not use google to generate candidates.

For questions such as *"Anja Pärson är uppvuxen i samma lilla fjällby som bl a Ingemar Stenmark. Vad heter den?"*, *coreference solving* for questions could lead to improved performance by both adding richer context to the question sentence and helping out in finding the lexical answer type.

Use *better context models* for question text and for candidate sentence. One approach for more robust context modeling is to use word clusters instead of lexical features [Pinchak2006], i.e. create a vector quantization of words based contextual features (lexical, POS, dependency grammar). A word cluster then represent words often appearing in similar contexts.

More features making use of both question and candidate context, for instance counting words occurring in both contexts.

References

[Apa2013a] 2013a. Apache lucene core. http://lucene.apache.org/core/.

[Apa2013b] 2013b. Tfidf similarity, lucene's conceptual scoring formula. http://lucene.apache.org/core/4_5_1/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html.

[Chu-Carroll et al.2012] J. Chu-Carroll, J. Fan, B.K. Boguraev, D. Carmel, D. Sheinwald, and C. Welty. 2012. Finding needles in the haystack: Search and candidate generation. *IBM Journal of Research and Development, Issue 3.4*.

[Fan et al.2008] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

[Fan et al.2012] J. Fan, A. Kalyanpur, D. C. Gondek, and D. A. Ferrucci. 2012. Automatic knowledge extraction from documents. *IBM Journal of Research and Development, Issue 3.4*.

[Ferrucci2012] D. A. Ferrucci. 2012. A framework for merging and ranking of answers in deepqa. *IBM Journal of Research and Development, Issue 3.4*.

[Gondek et al.2012] D.C. Gondek, A. Lally, A. Kalyanpur, J.W. Murdock, P.A. Duboue, L. Zhang, Y. Pan, Z.M. Qiu, and C Welty. 2012. A framework for merging and ranking of answers in deepqa. *IBM Journal of Research and Development, Issue 3.4*.

[Hall et al.2009] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The weka data mining software: An update. *SIGKDD Explorations, Volume 11, Issue 1. 2009.*, 11.

[Hall et al.2013] Johan Hall, Jens Nilsson, and Joakim Nivre. 2013. Maltparser. http://www.maltparser.org.

[J.Pyykkö and R.Weegar2013] J.Pyykkö and R.Weegar. 2013. Passage retrieval in a question answering system.

[jso2013] 2013. jsoup: Java html parser. http://jsoup.org.

[Kvi2013] 2013. Kvitt eller dubbelt - tiotusenkronorsfrågan. http://en.wikipedia.org/wiki/Kvitt_eller_dubbelt.

[Östling2013] Robert Östling. 2013. Stagger: an open-source part of speech tagger for swedish. *Northern European Journal of Language Technology*, 3:1–18.

[Pinchak2006] Christopher Pinchak. 2006. A probabilistic answer type model. *In EACL*.

[Swe2008] 2008. Swedish wikipedia static html dump. http://dumps.wikimedia.org/other/static_html_dumps/current/sv/wikipedia-sv-html.tar.7z.

[Yu et al.2013] H.-F. Yu, C.-H. Ho, Y.-C. Juan, and C.-J. Lin. 2013. Libshorttext: A library for short-text classification and analysis. http://www.csie.ntu.edu.tw/~cjlin/papers/libshorttext.pdf.