UiO : Department of Informatics
University of Oslo

Error Analysis in Open-Domain Question Answering Systems

Endre Aalrust Kristoffersen
Master's Thesis Spring 2015

Error Analysis in Open-Domain Question Answering Systems

Endre Aalrust Kristoffersen

15th May 2015

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Problem statement

With the explosive growth of available information in recent years, especially on the Internet, humans need new tools to help them sift through collections of information and find what they are looking for. The most common way people find information is with search queries, which contain some key words or phrases related to the information they want to find. For example, if someone wanted to find the name of the capital of Sweden, the search query would likely be «Sweden capital».

However, «Sweden capital» is a phrase meant only for the search engine. A human talking to another human would never utter this phrase if they wanted information. Rather, the query would be posed in natural language, for example «What is the capital of Sweden?».

In addition, the result from a search engine is usually a collection of documents, from which the user must find the answer themselves, whereas an answer from a human would be concise and to the point. Question answering systems attempt to provide an additional layer between the user and the search engine which can search through a specialised document collection or other type of semi-structured knowledge base. They are capable of transforming a question into a search query but also of taking the information from the search engine and narrow the collection of documents down to a short answer, in order to allow users to ask questions in natural language and receive an answer in natural language.

Modern question answering systems combine many different techniques and algorithms, which interact with each other in an attempt to find a short and concise answer. Because each technique only contributes a part of the answer, getting an overview of how well suited each technique is to help answering a question can be very hard.

The most common way of evaluating the performance of a question answering system is by overall performance, meaning that we only evaluate by the quality of the answer. This is the easiest evaluation metric to understand, and the only metric an end user is likely to care about.

Another, more detailed evaluation method is error analysis. Rather than evaluating only the end result – the answer – error analysis evaluates each individual part of the system to gain a deeper understanding of how each part of the system performs, and where each error first occurs.

Because modern question answering systems are so complicated, error analysis is typically a process that takes a lot of time for developers creating a question answering system.

In this thesis, we will construct a question answering system and analyse it in great detail in order to better understand how errors are found. We will then attempt to partially automate the error analysis, in order to facilitate better and faster development for question answering systems.

We will also, as a part of the construction of the question answering system, quantitatively analyse a question set that has, to the best of our knowledge, not been used in previous research. This data set is free, easy to obtain and contains questions that present a mix of different difficulties, making it well suited for new question answering systems.

The aim of this thesis is to examine the error analysis process and examine whether, for open-domain question answering systems, it is possible to automate error analysis, and whether automating error analysis is beneficial to developing the question answering systems, as evaluated either by time spent or by how detailed our understanding becomes.

1.1 Results

The new contributions in this thesis are two-fold. First, we statistically analyse a data set with question-answer pairs, and refine this data set to remove multiple occurrences of identical questions. We also evaluate the difficulty and diversity of the remainder of the questions.

Second, we perform a detailed error analysis on the techniques a typical question answering system uses. We will show how the analysis is performed and what data we can retrieve from these, and how the methods used in the analysis can be automated for a more efficient development. We will then validate these methods by going through a development cycle with and without automated analysis methods and show that automatic error analysis noticeably improves development time, and that the level of details can be at least as good as what is feasible to obtain with manual error analysis.

1.2 Thesis outline

Our thesis consists of three major parts, each containing two or more chapters. The first part gives an introduction to the thesis and the research question, as well as a background to the field of question answering. The second part details how we constructed a question answering system, based on standard current techniques and tools. This is also where we introduce the data set we used in this thesis, and show the analysis and refinement we did on this. The third part contains the remainder of the novel work in the field, with an in-depth reflection on error analysis and how this is performed, as well as the results we obtained from our work.

We will provide a background to the field of question answering, both historical and contemporary, in Chapter 2. We will also use this chapter

to discuss the linguistic methods that are most commonly part of question answering systems, different data sets used in question answering research tasks and how output from question answering systems can be evaluated. The chapter rounds off with an introduction to error analysis and how this can be performed.

The data set used in this thesis has not been used in any other published academic studies, to the best of our knowledge. Because of this, we have spent some time looking at the properties of the data set, analysing the data set statistically and refining the data set to be better suited to our needs. Our analysis and refinement of this data set are presented in Chapter 3.

In order to facilitate an analysis of a question answering system, we constructed a simple question answering system we could analyse and improve on. We will describe the methods and algorithms used in the system in Chapter 4, before we present the novel work in the field of error analysis in Chapters 5, 6 and 7.

We begin Chapter 5 with a discussion of what makes an error an error, and how to classify them. We then go into a detailed discussion of the errors we found in an initial error analysis of our system, and the first development cycle where we improved the system to reduce the number of errors. Errors found in the error analysis have been categorised, and each category will be described. We also examine the errors in one error subcategory in more detail.

The methods used in the error analysis are presented in Chapter 6. We also use this chapter to discuss how error analysis can be automated, the specific implementation details that went into the automation of our error analysis and compare the results of the automatic error analysis with the error analysis from Chapter 5. Chapter 6 ends with a discussion of how data sets can be annotated to improve error analysis, and how we can expand on error categories for features that we have not yet introduced in our question answering system.

Chapter 7 shows the second development cycle, where we further improved our system assisted by the methods in Chapter 6. We begin the chapter with a further refinement of the data set as a preparatory step towards a more useful error analysis, before we introduce a handful of new features into our system and evaluate their usefulness. Before we end the chapter with a summary of which features to include in the final system version, we reflect on the impact of the automatic error analysis on the development cycle.

We will end the thesis with some conclusions on our work and outline possible fields for further study in Chapter 8.

Chapter 2

Background

In this chapter, we consider the task and history of question answering. We give an introduction to the current state of question answering and the architecture of typical modern question answering systems. We also briefly discuss linguistic methods, data sets and different ways of evaluating performance in question answering. We end the chapter with an introduction to error analysis and introduce the problem we will attempt to solve in this thesis.

2.1 A brief history of question answering

For as long as humans have written information down we have also needed efficient ways to find the information when we need it. With the introduction of the Internet the amount of accessible information has increased exponentially, and has long since grown past the point where it is possible to read even a fraction of the available information, even if you spent your entire lifetime. Organising and searching through this information has become a large industry. Users interact with search engines through keywords. If you wanted to find information on ducks, you would go to your search engine of choice and simply type «ducks». If you wanted to know something more specific, you would add keywords – «ducks weight» – until you got the desired results. Even if our search for «ducks weight» returns dozens of documents about how much ducks weigh, we still need to look for the specific answer ourselves.

Interacting with search engines is different from how most people interact with other people. Our search keywords «ducks weight» probably stems from a question: «How much do ducks weigh?» With simple questions like these we will likely get some good results from most search engines, but the results are documents, and again we still have to find the answer ourselves. Intuitively we would like to get the specific answer when we ask a specific question, not a large set of documents.

Search engines seek to solve a problem called information retrieval (IR). Our more natural approach to retrieving information – ask a normal question and get an answer in return – is called question answering (QA).

A question answering system is a system which has the following task:

Given a question in natural language[1], return a correct answer, usually also in natural language. Question answering can be thought of as a specialised extension of information retrieval, which has finding relevant documents as its task, but leaves the problem of actually finding the answer in the document to the user.

Both question answering systems and the more general information retrieval systems operate against a collection of information. This can be a set of documents, a book, a database, or something else. Getting information from this collection is an information retrieval task. Question answering systems expand on this task in two ways: First, they allow questions to be posed in natural language instead of as a specialised query, and second, they aim to find the exact answer or at least a relatively short passage which contains the answer, instead of returning a whole, potentially large, document.

Most question answering systems are constructed to answer factoid questions. These are questions that ask for information that is known and relatively simple to summarise, such as «Where was Benjamin Franklin born?» or «What is the capital of Mongolia?». Non-factoid questions usually ask for an opinion or a reason, such as for example «What caused World War 1?» or «How do you change the oil filter of a car?».

Question answering systems can be either tailored to a specific subject, called a domain, for example finding restaurants or providing information from technical manuals, or they can be completely unrestricted and suited to answer all kinds of questions. The former is called «closed-domain» and the latter «open-domain». It is easier to get closed-domain systems to function well because we, at least to a certain degree, can anticipate the kind of questions users will want to ask, as well as the kind of information they will want to access and thus the correct answers. Open domain systems, on the other hand, need to be able to answer any kind of question, and we can only assume that the user input is a question for which it is possible to find an answer.

The earliest question answering systems were essentially a front end to database systems, and very specific to the subjects for which they answered questions – they were closed-domain systems. These were eventually abandoned because they relied heavily on mapping user questions to database queries, and these mappings had to be written manually, thus quickly becoming time-consuming. In addition, ordinary people outside of companies did not have access to large data sets, while people who did have access to large data sets did not have sufficient interest in accessing the data themselves (Webber and Webb, 2010).

In more recent days the Internet has provided access to vast data sets for ordinary people, which has led to a renaissance in question answering technology. Modern open-domain question answering systems require relatively large data sets to train on and search for information in, because

[1]The term «natural language» is used to differentiate the languages used in human communication – they have been formed naturally over time – from programming languages and other constructed languages.

the questions that can be asked are usually picked from quite diverse sets.

Two important conferences have been central in improving and innovating question answering systems since 1999. CLEF (Conference and Labs of the Evaluation Forum)[2] has run question answering problems in the years 2003–2013 under the names Multiple Language QA Main Task, ResPubliQA and QA4MRE (Question Answering for Machine Reading Evaluation). The «Machine Reading» in QA4MRE is a task of properly understanding a small number of texts, as opposed to getting evidence from many texts at once.

TREC (Text REtrieval Conference)[3] had a dedicated question answering track in the years 1999–2007. This track was designed to move the focus from the document retrieval part of question answering to the supporting methods for getting information from the retrieved documents (Voorhees, 2001).

IBM got some attention in 2011 for their question answering system named Watson, which performed better than top human contestants on the game show Jeopardy! (Ferrucci et al., 2010). Watson incorporates «more than 100 different techniques» throughout the pipeline.

Search engines have also begun to find answers to certain questions. Google Search answers simple conversion questions such as «How many miles are there in a kilometre?». Some also return a summary of facts gathered from Wikipedia and other sources for subjects that are well-known. A search for «Abraham Lincoln» will return such a summary, while the question «When was Abraham Lincoln born?» is treated like a normal query as of this writing.

2.2 Modern question answering systems

Most question answering system are abstractly similar to each other in the way they treat queries and data. Figure 2.1 shows a simple diagram of the work flow in a typical question answering system. First, it takes a question and analyses it. We can analyse questions for semantic information to be used for comparisons later, and we also tokenise and normalise the question so we can use it in information retrieval. We typically assign a set of answer types, and try and assign this question to an appropriate answer type.

The types can be as fine-grained as we like. For example, we can have a fairly general answer type like «location» or we can split it up into several other subtypes, such as «city», «country» and so on. The answer type is used later in the system, especially when we perform answer extraction. Some systems keep semantic or grammatical information from the question analysis as well.

With the exceptions of research tasks where the focus is not on the whole task of question answering, but rather on a smaller part, we need to perform information retrieval to find relevant documents. In the cases

[2] http://www.clef-initiative.eu/
[3] http://trec.nist.gov/

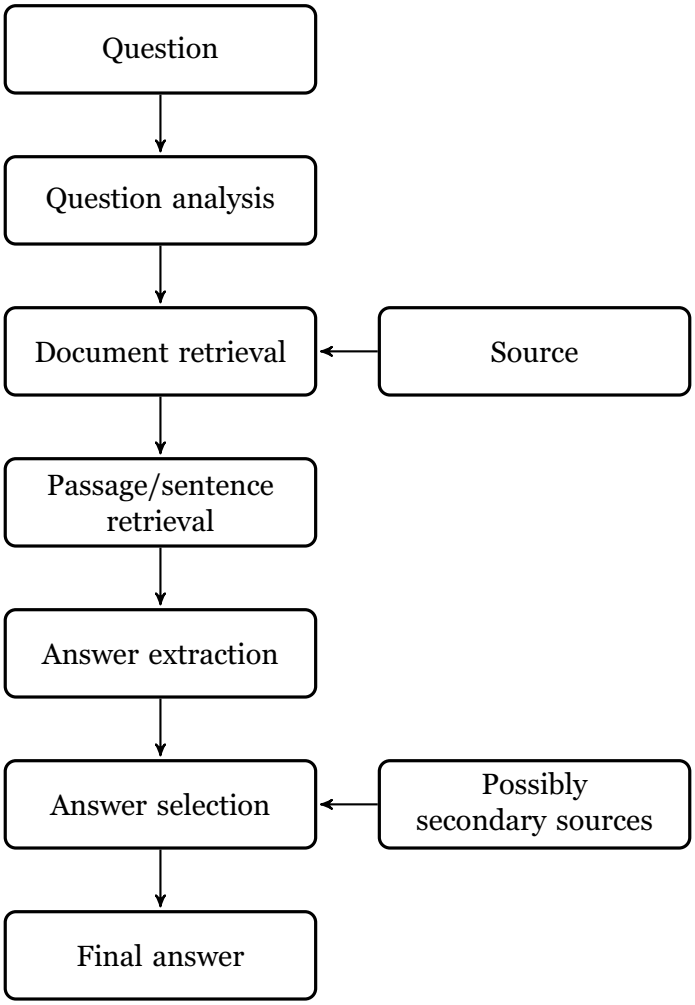

Figure 2.1: A typical question answering system.

where a document is not provided, the next step is to get one from our collection, using the information we got from our question analysis. This step is mostly like a traditional information retrieval problem, in that we have formed a query about what we want to get information on.

When we have one or more documents ready we are ready to find answers. The answer extraction step is probably the most complicated one, and can differ wildly from system to system, but always incorporate different techniques tailored to finding answer candidates. The thing they have in common is that they take information from the question and the answer type and attempt to find the pieces of information from the document that seem most likely to answer the question.

These techniques are usually combined so that each contribute to finding answer candidates. It is very common to create several answer candidates even though ultimately we only want to return a single answer. This is because we later might want to further evaluate and possibly discard answer candidates if we judge them to be too unlikely. We might also want to present more than one answer to the user and let them determine which

answer is most helpful.

The answer candidates are passed on to an answer selection step. In this part of our system we consider more carefully our answer candidates and select the one we think is best. For example, this is where we could discard all answers that are outside of the scope of the potential answer – if the question is «How many people live in Oslo?» we could program our system to find a range for how many people live in cities and discard answer candidates that are either not numeric or way too high or too low.

Another step that is sometimes performed during answer selection is to get information from secondary sources. If we know that the answer is present in one document we can consult other sources in order to validate our answers. If two or more answer candidates seem equally likely this is a good way to determine which is correct.

Depending on what kind of task we want to solve, we may return one or more answer candidates as our final answer. If we have more than one answer they can be presented with weights, in a particular order according to which answer we think is more likely or, if we do not care about the order of the answers, in a random order.

All these components and subtasks in question answering need to interact with each other. In natural language processing, where the problems are usually quite complex, the most common approach to solving problems is breaking the problem into smaller pieces and having many smaller programs that each solve a piece. In these cases we need a «software manager», often called a middleware, that tells the different programs when they should start working.

Middleware is responsible for keeping track of the different subsystems. It makes sure that systems wait until they have all the data they need from other systems earlier in the pipeline, and serves as an interface between systems if their data is not directly compatible.

2.3 Information retrieval

Information retrieval is a very important part of all question answering systems. Generally when we talk about information retrieval we talk about the task of returning any kind of data, including video and audio, that is relevant to a query. In question answering, however, we are mainly interested in retrieving text, and from here on out we will only consider the kinds of information retrieval that retrieve text. As used in question answering systems, information retrieval is commonly thought of as two subtasks: document retrieval and passage retrieval.

Information retrieval begins with a query. This is based on, but usually not the same as the question the system was given, meaning that the question needs to be processed in order to create the query. For example, it is common to remove the question word («when», «where» etc.). We might also want to remove stop words such as determiners, auxiliaries, prepositions and so on. If the available collection is small, it might be worthwhile to create several queries with different wordings, for example

by changing words with synonyms.

Document retrieval is the process of finding documents that contain information about the query. Documents are retrieved from a collection of documents. Different systems usually have access to different collections, and there are also different kinds of collections:

- Some systems are allowed to access all information they are able to find on the Internet.
- Some systems have a local and manually defined set of documents.
- Some systems are somewhere in between, so that they have a local set of documents that is updated with new documents regularly.

Each approach has advantages and disadvantages. Systems that can access everything on the Internet can answer any kind of question, but the massive amount of data available can make the system slow and inaccurate. A system like this also has the disadvantage that information on the Internet can be wrong. Additionally, some systems have limited network access and are therefore unable to use the Internet as a collection. Having a local set of documents makes the system faster, and also allows the developers to carefully choose which documents to preserve. On the other hand, these systems are not able to find new information if they encounter a question about something they do not have information on. Hybrid systems have a local set of documents about subjects that have been judged to be important, and are allowed to retrieve information from the Internet if they encounter a new type of question.

Document retrieval can be done using many different methods, from simple Boolean matching which finds documents that have all the terms in the query) to more sophisticated approaches with term weights and proximity of terms. One fairly simple system is the open-source Apache Lucene[4], which mainly uses Boolean queries and tf-idf (term frequency-inverse document frequency) weighing.

Classic information retrieval is usually content with returning a set of relevant documents, because the goal is to retrieve relevant information, and not a specific, short answer. In question answering, however, we also need to perform passage retrieval.

Passage retrieval is performed on the documents returned by document retrieval, so these two processes are closely connected. The underlying assumption in document retrieval is that some documents in a collection are more relevant than others, and the same assumption lies beneath passage retrieval – we assume that some passages in the document are more relevant than others.

A passage can be defined as any part of the document, depending on the needs of the system. It is often defined as a sentence, but can also be any number of sentences, a whole paragraph or a specific number of words or characters regardless of sentence structure.

[4]http://lucene.apache.org/

It is possible to use the same methods in passage retrieval as one would do in document retrieval, but the best systems use other and more specialised methods. Generally we divide passage retrieval methods in two types: the first type comprises bag-of-words methods, such as tf-idf (Jones, 1972) and BM25 (Sparck Jones et al., 2000), that do not care about the structure in the passages. In contrast, the other type cares about the order of the words. N-gram matching is one example of this type (Buscaldi et al., 2010; Soriano et al., 2005).

2.4 Typical linguistic methods in question answering

Linguistic analysis as performed on a sentence can be broken into subtasks.

We chose a sentence from our data set as an example:

Example 1. *Turtles lay eggs, like other reptiles, which are slightly soft and leathery.*

The words and punctuation marks in sentences are normally turned into tokens, so that they can be more easily processed later. Each token is usually normalised, for example by stemming or turning all characters lower case. The rest of our examples assume that the sentence has been tokenised. In this instance, this simply means that the combined word and punctuation as in for example «eggs,» has been turned into two tokens, «eggs» and «,», while words without punctuation attached to them are turned into a token without processing.

The stemmed form of our example sentence is

Example 2. *Turtl lay egg , like other reptil , which are slightli soft and leatheri .*

We sometimes want to remove common words – usually called stop words – from sentences as well, so that it is easier to search through them for information later. Stop words are words that are so common that we can not use them for determining what a document is about.

With stop words removed, our original example sentence turns into

Example 3. *Turtles lay eggs , like reptiles , slightly soft leathery .*

Different processing layers in question answering need different forms of the sentences. Information retrieval is commonly done with stemmed word forms, while linguistic processing, such as part-of-speech tagging and dependency parsing, need an unstemmed sentence in order to function. The final answer should also be in natural, unstemmed text.

Dependency parsing maps the syntactic relations between words in a sentence. Words must first be tagged with their word classes tell us what kind of word it is. Figure 2.2 shows a dependency graph of our example sentence, with these tags[5].

[5]The word tags have been generated with the Stanford Tagger, the dependency graph with MaltParser. We discuss these tools in Section 4.7.

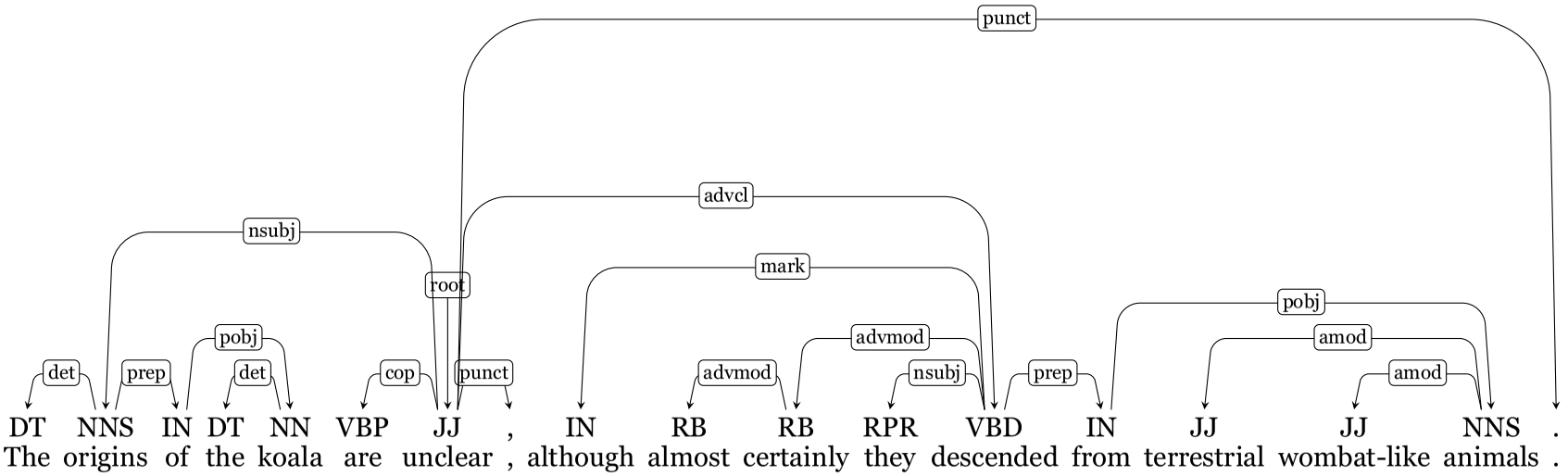

Figure 2.2: Example dependency graph, with words tagged.

We will use the example question «From which species does the koala descend?» to illustrate some common techniques for finding answer candidates. We will assume that document and passage retrieval narrowed our search down to this one sentence.[6]

One possibility is to look for words with tags that match the expected question type. The process of finding the expected answer type is called question typing, and can be implemented in many different ways. One simple and efficient way is to create a set of hand-crafted rules, such as «If the sentence starts with «what», assign answer type «noun»». More fine-grained answer types can be created, depending on the need of the system. Since our example question contains the question word «which» we assume that it asks for an object, and that it is likely that the answer is a noun. From looking at our tags, our answer candidates are «origins», «koala» and «animals». If the question had asked for a person or organisation it would be more appropriate to tag the words with a named entity tagger, which tag words with word classes depending on whether it refers to a person, organisation and so on, and look for words that have been tagged as names.

We could also create an answer prototype, by assuming that the answer is to be found in the form «koala descends from *», where * means any sequence of words, and see if we find this particular sequence of words in the sentence. In this case it does not find an answer candidate. We can however improve on this. We find the phrase «descended from» and retrieve everything until the next noun we find, which would get us the answer candidate «terrestrial wombat-like animals».

Answer validation, where we check the answer candidates to find the most likely one, is another step that can be solved in different ways. One way is to establish a key word – the most significant word in the question – from the question and look for word proximity.

Proximity is the distance between two words in the sentence. For example, the words «certainly» and «terrestrial» both have a proximity of 2 to the word «descended» in our example sentence.

In our example question the word «which» is a question word, and «from», «does» and «the» are so common that they are unlikely to be the most significant. We are left with «species» and «descend», and only the latter occurs in the text (though the grammatical tense is different). Thus, we are left with «descend» as the key word. When we simply looked for word types we found «origins», «koala» and «animals» as possible answer candidates, and rank them in order of proximity to the key word from our question, and are left with «animals» as the most likely answer candidate.

Another method in answer validation is to search for documents that support the answer, and use a cumulative formula to find which answer is most supported. In our example, we would try to find documents that supported that koalas descended from origins, koala and animals respectively.

As in information retrieval it might be useful to use synonyms when we search for answer candidates. A search for synonyms and words with

[6] In practise this is unlikely, but it makes visualising the example easier.

similar meaning, for example in WordNet (Fellbaum, 1998), would return most of the possible alternative words, which we can use to get more answer candidates.

Depending on how we defined passages in passage retrieval, it might be necessary to resolve cross-sentence references. Consider the following sentences:

Example 4. *The scientific name of the koala's genus, Phascolarctos, is derived from Greek phaskolos "pouch" and arktos "bear". Its species name, cinereus, is Latin and means "ash-coloured".*

If we consider the second sentence alone, it is difficult to understand exactly what the word «Its» refers to. It points back to the previous sentence, to the word «koala». Resolving this reference is necessary if we want to answer the question «Which species name means ash-coloured?» – if we do not, we might end with the unsatisfactory answer «Its».

2.5 Data sets

After the Internet became popular the number and quality of data that can be used for question answering has increased enormously. Some are generated specifically for this task, such as TREC's and CLEF's data sets for their question answering related tracks. Others, such as the J! Archive[7] were created for another purpose but can be used for question answering systems.

Several web sites offer a service where users can ask and answer questions, such as for example Yahoo! Answers[8] and answers.com[9]. Yahoo! has released fairly large data sets based on Yahoo! Answers with questions, corresponding answers, the best answer as marked by the asker and some other metadata[10]. Stack Overflow[11] has done the same thing with their data[12], which is specific to questions about programming.

For a data set to be usable in a question answering system, it strictly speaking only needs to contain questions. However, due to the current restrictions in what is possibly to do with question answering systems, it is not possible to use the data set if the questions are too difficult to answer. Stack Overflow questions, for example, are often long and include code samples to illustrate the problem the asker wants a solution for. They are also significantly harder than factoid questions because answers usually need to be explained in detail in order to be accepted.

It is useful but not necessary for a data set to include one or more suggested answers for each question. If suggested answers are not

[7] A fan-created archive of Jeopardy! games and players: http://www.j-archive.com/

[8] https://answers.yahoo.com/

[9] http://wiki.answers.com/

[10] http://webscope.sandbox.yahoo.com/catalog.php?datatype=l

[11] http://stackexchange.com/

[12] http://blog.stackoverflow.com/2009/06/stack-overflow-creative-commons-data-dump/

provided, researchers need to either evaluate system answers manually, which takes valuable time, or pay someone to do it, which becomes expensive.

The set might also include a set of documents in which we know that we should be able to find the answer. This is especially true for tasks that want to move the focus away from the document retrieval parts of question answering, for example the question answering tasks from TREC and CLEF.

2.6 Evaluation

Evaluating question answering systems is largely a matter of determining whether the system is capable of finding the correct answer. Of course, exactly how one goes about defining whether the correct answer has been found is up for debate. Question answering systems are built to answer questions for humans, so the best metric we have for evaluating whether an answer is correct is to ask a human whether they agree. In research and development this often becomes impractical – we would have to ask again every time the system came up with a different answer.

If we have the question

Example 5. *«What is the capital of Sweden?»,*

some possible answers are

Example 6. *«Stockholm»*

Example 7. *«The capital of Sweden is Stockholm»*

Example 8. *«Stockholm is the capital of Sweden»*

and so on. If we choose Example 6 as the gold answer – the suggested correct answer – to this question, a simple and strict form of evaluation that checked whether the answer exactly matched a suggested gold answer would judge the other two answers to be incorrect, while most humans would also accept the other two.

As we can see, how we evaluate our system answers can have a large impact on how we perceive the accuracy of our system.

Intuitively we would prefer answers that are concise and do not contain unnecessary information, particularly if that information is irrelevant to the question. For example, if we have the question «How old is Bill Clinton?» the answer «68 years» is usually considered better than «Bill Clinton is 68 years old» because it is shorter, and better than «Clinton, 68, has retired from politics» because that contains information that is irrelevant to the question.

Different methods for evaluation have been proposed. There is some debate on which features should be present in a good answer, other than that the answer should be correct. Some systems place a high value on creating syntactic sentences, others allow systems to return short passages

and mainly care about the relevant information being present. There is also some variation on how many answers a system should be allowed to return. Some researchers place a high value on systems that are able to not answer a question if they are uncertain, while others think it is equally bad to return an incorrect answer and no answer.

Two organisations that have been central in providing an environment in which systems can be compared to each other and get a measure of how they perform objectively are CLEF and TREC. TREC had its last run of question answering evaluations in 2007, and CLEF still has runs for evaluating performance within question answering systems. Typically, tests such as these provide a large volume of texts and quite a lot of questions, and the task is to find the answer to the questions within the texts. They are usually open-domain problems.

The main task of TREC's question answering tracks allowed systems to return up to either 50 or 250 characters and evaluated whether the answer was located in those characters. Whether the question was answered correctly was judged manually by a human.

CLEF's QA4MRE tasks provide question candidates and ask systems to choose one. This is called answer validation and is a variation or subtype of question answering. QA4MRE also allows systems to not return an answer to a question, judging that it is better to not return an answer than to return an answer that is incorrect.

Breck et al. (2000) attempt to automate the process of evaluating individual system answers by creating a list of potential gold answers manually, normalising both the gold and system answers and find that recall is a good way of evaluating whether they are similar.

Recall is the fraction of possible correct items we managed to get with our system, or more formally:

$$recall = \frac{TP}{TP + FN} \tag{2.1}$$

Where TP is the number of gold answers that we manage to retrieve and FN is the number of gold answers that we do not manage to retrieve. In this context it is used to see how many of the words in the system answer are present in the gold answer.

If the recall is high enough they assume that the system answer is similar enough (they suggest a threshold of 0.5 for recall). However, since they worked on the TREC tasks and the gold answers could not be more than 50 bytes (and their system answers were not allowed to be more than 50 bytes), they note that a more refined process is necessary for tasks where the answers can have variable lengths.

In addition to this, there are also different metrics with which we can rate the overall performance of a system, not just for each question. The simplest is to look at each question and give it a score of 1 if it is judged as correct and 0 otherwise, then calculate the mean value as the overall score.

If the system is allowed to return more than one answer to each question, it is common to have the system rank the answers according to likelihood, and adjust the score according to how high the correct answer

was ranked. TREC, as mentioned, allowed systems to return answers of either 50 or 250 bytes, and used human assessors to decide whether the returned answer actually contained correct information (Voorhees et al., 1999). The systems were allowed to return up to 5 different answers, ranked by how likely the system evaluated each answer to be. Each question got a score depending on the rank of the highest ranked correct answer:

$$Reciprocal\ rank = \frac{1}{n} \tag{2.2}$$

were n is the rank of the correct answer. If no correct answers were returned, the answer got a score of 0. More than one correct answer did not have an impact on the score of the answer. They then measured the overall system performance by Mean Reciprocal Rank (MRR).

Some argue that it is better not to answer than to answer incorrectly. Watson, for example, is very focused on only answering questions when it is very certain that it has the correct answer, since the rules of Jeopardy! penalise incorrect answers.

Another example of a measure where incorrect answers are considered worse than not returning an answer at all is QA4MRE's c@1, defined as:

$$c@1 = \frac{1}{n}(n_R + n_U \frac{n_R}{n}) \tag{2.3}$$

where n_R is the number of correctly answered questions, n_U the number of unanswered questions and n is the total number of questions (Penas and Rodrigo, 2011). With this evaluation metric the score of each unanswered question depends on how many correct answers the system returns. In other words, systems that answer that they do not know the answer to any questions will get the same score as a system that answers all questions incorrectly.

We will return to a more detailed discussion of evaluation, both of answers and of system performance, in Section 5.1.

2.7 Error analysis

Question answering is far from being a solved problem. This is because of a combination of the wide variety of questions one might be expected to solve, the difficult task of finding the necessary information in a large collection, as well as how many questions can have more than a single answer. Questions can be simple and request factual information, but we would also like to answer more complex questions («Why did the Allies win World War 2?»). Some questions are simple enough to answer, but are context specific, such as «Who is the president of the United States?», where the answer changes depending on when we ask the question.

In order to be able to answer the great variety of questions, question answering systems typically get fairly complex, incorporating a wide range of natural language modules in addition to the modules used by normal information retrieval systems. The combination of these two features –

a difficult problem and a complex system to solve the problem – means that it can be difficult to properly understand which parts of the system are responsible when a wrong answer is obtained.

It is important to understand not only which module fails but also how that module interacts with the rest of our system. An error that occurs in a module early in the pipeline, or in a module which many other modules depend on, will have ramifications for the other modules in the system. Errors causing errors in a chain reaction is often called «error propagation».

When we have run a system on a set of data and not obtained the answer we wanted, error analysis is the process in which we look through all the modules that might have led to this error and attempt to pinpoint exactly what went wrong, as well as how they may be fixed. In addition, error analysis may also give us insight in which modules in the system are responsible for the largest part of all errors. Error analysis is often performed manually, and is often time consuming because the systems are often complex.

Some attempts have been done at reducing the time needed for error analysis: Brill et al. (2002) simply remove non-essential components from their system and look at the impact of the removal. Ittycheriah et al. (2001) look at how many errors each component in their system makes, and perform a more detailed analysis of the document retrieval module. Moldovan et al. (2003) go further into detail on their system and looks at the kinds of errors each module can produce, as well as the quantity of each kind.

Though all these (and more) have performed detailed error analysis, none of them have tried to automate the process. Moldovan et al. (2003) perform a statistical analysis on performance by question stems («Where», «What», etc.) and answer type but do not analyse their performance further.

In this thesis, we will aim to create a simple question answering system in order to illuminate some of the problematic issues with error analysis. One of the largest issues with error analysis is that it is very time consuming. We will attempt to analyse how error analysis is performed manually, with the aim of aiding and possibly automating the process.

The performance of error analysis can be measured in several different ways. Perhaps the most interesting is how much time researchers have to spend in order to find where in the system the problem is located. We are also interested in generating error reports that show the problem in as much detail as possible, so it is easier to fix the error. For example, an error report that tells us that there is a problem with the document retrieval module is less useful than an error report that tells us that the document retrieval module was unable to retrieve a relevant document, or that it was able to retrieve a relevant document but did not rank it high enough to have an impact on the answer.

Part II

Building a question answering system

Chapter 3

Data set

In this chapter we will describe in detail the data set we chose for our question answering system. We will perform some basic statistical analysis on this data set in order to motivate some of the design choices we made when we designed our system, and attempt to estimate how well it is possible for a theoretical question answering system that always performs perfectly to perform on this data set.

This data set has to our knowledge not been used in any previous studies, so we will spend some time on detailing some problematic issues with this data set, as well as how we dealt with these problems.

3.1 Data set properties

Our data set[1] was made by students who took undergraduate natural language processing courses taught by Noah Smith at Carnegie Mellon and Rebecca Hwa at the University of Pittsburgh during Spring 2008, Spring 2009, and Spring 2010 (Smith et al., 2008). It had several properties that make it especially attractive.

First, it was freely available and of a decent size. A suggested answer was supplied with each question, as well as a WikiPedia article the question was generated from, and in which the answer could be found. Both these properties were good because they allowed us to focus on other parts of the question answering problem, rather than on document retrieval.

Because we have suggested answers we did not need to manually create a set of correct answers, and because the documents were provided we could skip the document retrieval part of the system in favour of focusing on everything else.

It also had the attractive feature of not being too hard – the answers were to a large degree possible to find in the provided documents directly, without a need for in-depth linguistic analysis. Most established question sets, such as the sets from TREC and QA4MRE, are both licensed and significantly harder than the set we chose.

The questions were generated automatically from Wikipedia articles by

[1]This data set can be found at http://www.ark.cs.cmu.edu/question answering-data/ .

computer systems the students made. The Wikipedia articles are diverse enough that we can call the data set an open-domain data set[2]. Each system was created independently by different groups of students. The questions were evaluated manually by other students and marked with an answer and a difficulty rating.

Each question came with the following information:

1. The name of the Wikipedia article from which the questions initially came.
2. The question.
3. A proposed answer.
4. A difficulty rating for the question as given by the system writer.
5. A difficulty rating assigned by the individual who evaluated and answered the question.
6. The location of the provided relevant document.

We did not use the difficulty ratings in our system, instead choosing to answer all questions equally. We also did not use the name of the Wikipedia article.

Since there were several systems which worked on the same files for source material, quite a lot of the questions were identical. In most of these cases all the provided answers were correct. In some cases one or more of the answers are incorrect:

Question In what years did Avogadro stop teaching at Turin University?
Answer 1 1853
Answer 2 1823

In addition, the answers (provided by humans) were sometimes not identical:

Question What does the word duck mean?
Answer 1 It is the common name for a number of species in the Anatidae family of birds.
Answer 2 to bend down low as if to get under something

This would likely lead to skewed results according to the relative difficulty of the questions, since our question answering system would provide the same answer each time a question was repeated.

To resolve the problems with identical questions we decided that we needed to remove duplicate questions. Due to the sheer size of the question sets, rather than go through all the questions manually we opted to keep the first instance of each question regardless of its answer, and remove all further identical instances of the question. If the same question appeared in more than one year, one instance of the question was kept for each year it

[2]Appendix A has an overview over all the categories.

was present. After most of these problematic questions had been removed, the three data sets – one for each year – had been reduced as shown in Table 3.1. The details for each set can be seen in Appendix A.

A small minority of questions were impossible to answer. Some of them simply did not have the answer available in the source file, or it was impossible to understand what the question asked for:

Question What do economy and law have in common?
Suggested answer (not sure how to answer this)

We accepted that it would be too time-consuming to manually go through every single question to remove these, and simply note that they exist in the data set.

Year	Set	Size
2008	Training	551
	Dev1	90
	Dev2	91
	Dev3	96
	Test	91
	Total	919
	Original total	1715
2009	Training	304
	Dev1	49
	Dev2	47
	Dev3	48
	Test	51
	Total	499
	Original total	826
2010	Training	475
	Dev1	79
	Dev2	79
	Dev3	78
	Test	81
	Total	792
	Original total	1459

Table 3.1: Summary of our data sets.

We then divided the data from each year by category into training, development and test sets. The training sets consisted of roughly 60% of all the questions. We had three different development sets from each year, which held roughly 10% of the questions each. Finally, a test set from each year was held aside for the final tests. This set also held roughly 10%. Table 3.1 shows a summary of the number of question-answer pairs in the different sets.

3.2 Statistical analysis

Without looking too much at specific individual questions, we wanted to perform some statistical analysis on our data set before we started designing our system. Our first observation was that a lot of questions had either «yes» or «no» as the suggested answer. We counted the number of questions that had either «yes» or «no» as the suggested answer, removing punctuation but not counting answers that had more words, such as for example «Yes, diglossia is a common feature in mainland China and Taiwan.»

We can see in Table 3.2 that for some sets a majority of the questions had either «yes» or «no» as an answer, and that this kind of question was the most prevalent in all sets. This suggests that a system that performs well on these kinds of questions, but not on any other kind, will still get a high accuracy.

Year	Set	Questions	Yes/No	%
2008	Dev1	90	46	51.1%
	Dev2	91	48	52.7%
	Dev3	96	45	46.9%
	Test	91	33	36.3%
	Training	551	237	43.0%
	Total	919	409	44.5%
2009	Dev1	49	25	51.0%
	Dev2	47	26	55.3%
	Dev3	48	23	47.9%
	Test	51	26	51.0%
	Training	304	150	49.3%
	Total	499	250	50.1%
2010	Dev1	79	29	36.7%
	Dev2	79	24	30.4%
	Dev3	78	33	41.8%
	Test	81	31	38.3%
	Training	475	161	33.9%
	Total	792	278	35.1%
	Overall	2210	937	42.4%

Table 3.2: Number of yes/no questions compared to the total number of questions across all sets.

We knew from looking at the data sets that some of the suggested gold answers were impossible to find in the supplied documents. On the other hand, the assumption that for each question it is possible to find an answer in the text is very useful when we build our system. If this assumption does not hold true we need to look for secondary sources in order to answer our questions.

In order to justify this assumption we needed to know how many of the suggested gold answers were present in the documents. We disregarded

all «yes/no» questions and looked at the remaining gold answers. For each gold answer we looked for an exactly matching substring of the corresponding document. If such a substring existed we knew that it was possible to find the answer.

Our criteria for determining whether the gold answer is present is obviously fairly strict. On the other hand, the results in Table 3.3 look promising. In most of the sets at least half of the gold answers have exact matches in the documents. Remember that these are only exact matches for the suggested answer, and small deviations means that there will be no match.

A more detailed analysis which could take small variations such as word forms and word order into account would probably give more matches, but would be outside of the scope of what we want to establish here.

Year	Set	Questions	Answer is substring	%
2008	Dev1	49	28	57.1%
	Dev2	53	25	47.2%
	Dev3	56	25	44.6%
	Test	65	38	58.5%
	Training	344	163	47.4%
	Total	567	279	49.2%
2009	Dev1	28	15	53.6%
	Dev2	21	12	57.1%
	Dev3	36	10	27.8%
	Test	25	15	60.0%
	Training	170	87	51.2%
	Total	280	139	49.6%
2010	Dev1	58	22	37.9%
	Dev2	59	29	49.1%
	Dev3	54	15	27.8%
	Test	50	24	48.0%
	Training	345	142	41.2%
	Total	566	232	40.9%
	overall	1413	650	46.0%

Table 3.3: Number of gold answers where the answer is a substring of the document (disregarding yes/no questions)

We can now make some estimates about how well we should be able to perform on our data set. We can imagine a theoretical question answering system that does not support secondary sources, but can find all «yes/no» answers and all answers that are directly present in the provided document.

If this theoretical question answering system was able to answer all these questions correctly, we can summarise Tables 3.2 and 3.3 and get the theoretical, perfect results in Table 3.4.

We will motivate the purpose of building a question answering system for this thesis in the beginning of the next chapter. We need this system to perform reasonably well if the results and analyses we perform are to

Year	Set	Questions	Correct	Accuracy
2008	Dev1	90	74	82.2%
	Dev2	91	73	80.2%
	Dev3	96	70	72.9%
	Test	91	71	78.0%
	Training	551	400	72.6%
	Total	919	688	74.9%
2009	Dev1	49	40	81.6%
	Dev2	47	38	80.8%
	Dev3	48	33	68.6%
	Test	51	41	80.4%
	Training	304	237	78.0%
	Total	499	389	78.0%
2010	Dev1	79	51	64.6%
	Dev2	79	53	67.1%
	Dev3	78	48	61.5%
	Test	81	55	68.0%
	Training	475	303	63.8%
	Total	792	510	64.4%
	Overall	2210	1587	71.8%

Table 3.4: Accuracy of a theoretical question answering system on our data set.

be meaningful. Because our data set has not been used in any previous studies we will need to establish an upper bound so we know what we can reasonably aim for. The results in Table 3.4 will serve as this upper bound.

Chapter 4

System architecture and implementation

In this chapter, we will describe the tools and design choices that went into our question answering system. The purpose of this system was not to be cutting-edge, state-of-the-art question answering technology, but rather to construct a system that would aid us in understanding the errors question answering systems make and how to detect and fix them.

The platform for our system was the Natural Language Toolkit (Bird et al., 2009), commonly referred to as NLTK[1]. All the tools we use are either implemented by us in Python or imported through an NLTK interface.

After a discussion of how we stored and normalised documents in our system, we will establish two baselines: a majority baseline based on the most common answer in our data set and a baseline constructed with a very simple question answering system. We will then go on to describe the system itself.

4.1 Data types

For both questions and documents we are mainly interested in sentences. We store each sentence separately. Sentences are stored sorted in lists, and each list represents a document. Each sentence is stored both as the original text and as a normalised version. We also store versions of the sentences that have been annotated in different ways, such as with part-of-speech or named entity tags.

Questions are stored in the same way. Suggested gold answers are not stored in this way, because we only use the text representation of the answer, and mostly only at the end of the system when we check whether the system answer corresponds to the gold answer.

[1] http://www.nltk.org/

4.2 Normalisation

All our questions, documents and to a certain degree answers were normalised by our system. As we have mentioned in Section 2.4, normalisation is commonly performed on text that we want to use in passage retrieval, but we can not use normalised text for most of the other linguistic methods we use in our system, because we lose information when we normalise text. Therefore, for all natural language we process in our system we kept both the original sentence and normalised version. The original sentence was needed for linguistic analysis, and the normalised version was used in passage retrieval.

Documents were split into sentences. Questions and documents were tokenised using the tokeniser included in NLTK and stemmed using the Porter stemmer (Porter, 1980). Hyphenated words were split into separate tokens.

We chose not to remove any stop words in the initial system version because we were worried about ranking relevant sentences too low if words were removed from both the documents and the questions. We will introduce removal of stop words as a feature in Section 7.3.2, and show that this was an unfounded concern.

Another normalisation step was to turn numerals, such as «three», into digits, such as «3». We performed this process on the documents and the questions so that they would correspond more closely. We also performed the process on the gold answers, so that the answer matcher would find matches even if the gold answer was written as a numeral. We found the basis for the actual code for converting at StackOverflow[2], and modified it to suit our needs.

The main reason for this conversion was that we observed that some of the expected answer types were numbers, and that turning numerals into digits would make the process of finding these easier. Of course, as in all cases of normalisation, we lose information when we turn numerals into digits. For example, in dates we almost always write numbers as digits, while low numbers in other contexts are usually written out as a numeral. Normalising the sentence «For December ten turkeys were bought» could introduce an ambiguity that was not there before. Luckily, this kind of sentence never occurred in our set of documents.

Sometimes the «and» between two numerals means that they are two different numbers rather than parts of the same number – «one hundred and ten» can mean either «110» or «100 and 10». This phenomenon also never occurred in our set of documents.

It should be noted here that we could have opted for the reverse solution – turning digits into numerals – but we elected not to for two reasons. The first and main reason was that we found that turning numerals into digits was simply easier to implement. The other, happily, was that date information was preserved this way. Our named entity tagger which for

[2] http://stackoverflow.com/questions/493174/is-there-a-way-to-convert-number-words-to-integers-python

a large part was responsible for recognising dates as well would have problems identifying «one thousand seven hundred seventy six» as a year, or «twenty six October» as a date.

4.3 Majority baseline

From looking at our data we know that the most common answer, regardless of question, in all of our data sets, is simply «yes». From this we can establish a very naive baseline system which only answers «yes» to every question. This system would perform as shown in Table 4.1. We have here assumed that we evaluate by the strict metric described in Section 4.8, meaning that the gold answer would also have to be exactly «yes» if the system answer is to be evaluated as correct.

Year	Set	No. of questions	Correct answers	Accuracy
2008	Training	551	169	30.6%
	Dev1	90	36	40.0%
	Dev2	91	31	34.0%
	Dev3	96	36	37.5%
	Average			32.9%
2009	Training	304	114	37.5%
	Dev1	49	16	32.6%
	Dev2	47	21	44.6%
	Dev3	48	8	16.6%
	Average			35.5%
2010	Training	475	107	22.5%
	Dev1	79	17	21.5%
	Dev2	79	15	19.0%
	Dev3	78	21	27.0%
	Average			22.5%
Overall	Average			29.7%

Table 4.1: Correctly answered questions by the majority baseline

The purpose of this baseline is to understand what results we can get without even trying simple methods. A system that performs worse than this baseline can not be said to perform in a satisfactory manner.

We have a question analysis step in our system that assigns answer types to questions. At this point it is natural to look at how well our system classifies answer types for yes/no questions.

The data set does not provide a gold standard for answer types, but for yes/no questions it is simple to construct one, since the yes/no questions should presumably always have either «yes» or «no» as the answer. We show the precision and recall of our answer type detector in Table 4.2.

In this table we have counted all answers that are either exactly «yes» or «no» or have a qualifier, such as «Yes, Volta was born in Como, Italy and was taught in the public schools there.», as «yes/no» gold answers. We did

not count answers that simply repeat the statement in the question, such as:

Question Is Nairobi not the capital as well as largest city of Kenya?
Gold answer Nairobi is the capital and largest city of Kenya

Because we only want to get a feel of how well we classify yes/no questions we do not need the exact numbers. Statistically, these kinds of answers were uncommon and should have little impact on this analysis.

Year	Set	Gold	System	Precision	Recall
2008	Training	241	243	97.1%	97.9%
	Dev1	46	45	100.0%	97.8%
	Dev2	48	50	94.0%	97.9%
	Dev3	46	51	88.2%	97.8%
	Total	381	396	95.9%	97.9%
2009	Training	150	145	98.6%	95.3%
	Dev1	25	25	96.0%	96.0%
	Dev2	27	25	92.0%	85.1%
	Dev3	21	22	90.9%	95.2%
	Total	223	217	97.7%	93.3%
2010	Training	189	202	85.6%	91.5%
	Dev1	34	38	89.4%	100.0%
	Dev2	29	28	89.2%	86.2%
	Dev3	42	42	95.2%	95.2%
	Total	294	310	87.7%	92.5%
	Overall	898	923	93.3%	95.2%

Table 4.2: Precision and recall of the question typer on yes/no questions.

4.4 Simple baseline

A question answering system that answers «yes» to every question is neither interesting nor useful. We constructed a small, simple question answering system in order to create a baseline that was actually able to answer other questions as well, which could be used for further analysis.

The details of the system will be described in the following sections, but we will provide an overview here: The questions are analysed to find expected answer types. We retrieve sentences with a frequency-based approach, which weighs uncommon terms in the document higher than common terms. Answer candidates are mostly found by word type according to the expected answer type, and the answer candidates that occurs most often in the retrieved sentences is chosen as the most likely candidate. We use a very strict evaluation technique to produce the results in Table 4.3.

The results are consistently worse than in the majority baseline, so we already know that this system does not perform well enough to be deemed

Year	Set	Questions	Correct answers	%
2008	Training	551	110	20.0%
	Dev1	90	26	28.9%
	Dev2	91	19	20.9%
	Dev3	96	17	17.7%
	Average			20.1%
2009	Training	304	88	29.0%
	Dev1	49	14	28.6%
	Dev2	47	8	17.0%
	Dev3	48	7	14.6%
	Average			26.1%
2010	Training	475	84	17.7%
	Dev1	79	11	13.9%
	Dev2	79	15	19.0%
	Dev3	78	17	21.8%
	Average			17.9%
Overall	Average			20.9%

Table 4.3: Correctly answered questions with the simple baseline.

acceptable. On the other hand, this approach at least has the chance of correctly answering questions where the answer is not a single «yes». It is also a system we can analyse for errors in a meaningful way.

4.5 Question analysis

In Figure 4.1, we repeat Figure 2.1 as a reminder of the pipeline of a typical question answering system. We will walk through the design of our question answering system, which will correspond closely to the question answering system shown in this figure.

Questions are read from a text file, generated from the data set as described in Chapter 3.

We extracted the expected answer type from each question with a simple pattern matching algorithm with hand-written rules. The rules look for matches in regular expressions, and return the first found match among the rules. We have included the exact rules in Appendix B.

Our answer types correspond closely to what kind of question word is used, and are named after the question words «when», «what», «who», «why», «where» and «how». In addition there are two extra answer types: one for yes/no questions («Are large pythons potential prey for leopards?») and one for questions that ask for a number («How many counties is Romania divided into?»).

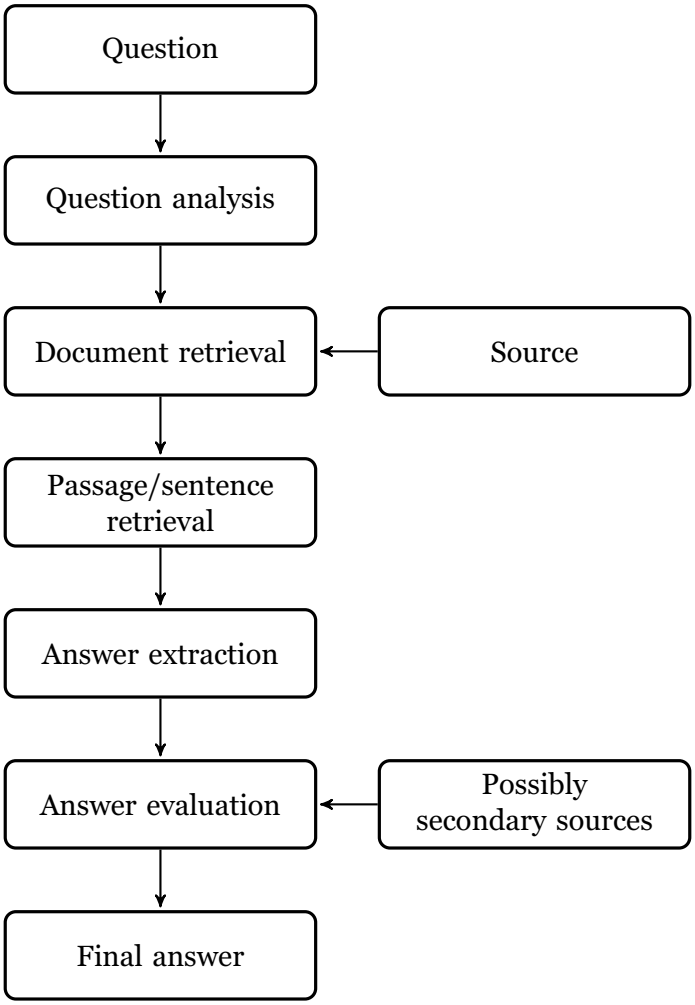

Figure 4.1: A typical question answering system, repeated from Figure 2.1.

4.6 Sentence retrieval

Document retrieval was not necessary in our system, because we knew for each question which document the answer should be located in.

In our system we decided to define a passage as a full sentence. Answers rarely needed to include more than one sentence, and we decided that full sentences made more sense than an arbitrary number of words or characters.

Our initial attempt at sentence retrieval used a frequency-based algorithm for retrieving relevant sentences. Each sentence received a score:

$$frequency\_score = \sum_{t \in q} \log \frac{f(t,s)}{f(t,d)} \tag{4.1}$$

Where $t \in q$ is the terms in the query (the question), $f(t,s)$ is how many times the word occurred in the sentence and $f(t,d)$ is how often it occurred in the document.

The sentences are returned with a score as defined in Equation 4.1, and ranked according to the score. Sentences with a score of 0 are never returned.

We will implement and discuss two new sentence retrieval algorithms in Section 5.4.3, and experiment further with sentence retrieval in Section 7.3.

4.7 Answer extraction

We attempted several different strategies for retrieving answer candidates. The simplest versions used word tags. We chose to use the Stanford Part-Of-Speech tagger (Toutanova and Manning, 2000; Toutanova et al., 2003) for tagging parts of speech and the Stanford Named Entity Recognizer (Finkel et al., 2005) for named entity recognition. Both these taggers are integrated with NLTK. The named entity recogniser has the added benefit that it is able to recognise dates as well as named entities.

Recall that we have an expected answer type from analysing the question, and a set of sentences from the sentence retrieval step. From the expected answer type we can make an assumption about the expected word type of the answer. The simplest algorithms we implemented for finding answer candidates used this assumption. Table 4.4 shows the kinds of word tags we looked for for each kind of answer type.

Nouns and verbs are located by part-of-speech tags from the Stanford tagger. Location, name and time are tags from the Stanford Named Entity Tagger. Numbers are located simply by the property of being represented with digits in the document (remember that we transformed written words into digits when we normalised our documents).

Words that occur in more than one sentence accumulate scores depending on the score of the sentences they occur in. This is a naive approach to evidence-based answer extraction, where the system attempts to get evidence for each of the answer candidates in order to establish the most likely.

«Why» answers usually ask for a reason. Because the answers to these questions are usually complex we want to return a full phrase every time. The naive approach to this is to use the relevance score from sentence retrieval and return the most likely sentence.

«Yes/no» questions got special treatment – because the only possible answers are «yes» and «no» there is no point in attempting to find answer candidates. Instead we used a purely pattern-based approach and looked for negations.

Example 9. *Is the leopard smaller than the other members of Panthera?*

For example, for Example 9 we want to find a sentence with patterns similar to «the leopard is <not> smaller than the other members of Panthera». If the «not» is present, we answer with a «no», if it is not present we answer «yes». Statistically the most common answer to yes/no

Expected answer type	Word type
What	Noun
How	Verb
Where	Location
Who	Person name
When	Time
Number	Number
Why	Special case
Yes/No	Special case

Table 4.4: Answer types and word types

questions in our data set is «yes», so we choose this as the default if we are unable to find a matching pattern.

Though our first, simple system only returned one-word answers we should ideally be able to return phrases. Phrases in this context means any sequence of words that is longer than one word. Because question answering ideally should return answers as natural language we need some way to extract these phrases from sentences. We will use dependency parser for this[3].

Our chosen dependency parser was MaltParser (Nivre et al., 2007). NLTK has an interface to this parser, and we were already familiar with how it worked.

MaltParser requires part-of-speech tags to function. The part-of-speech tags must conform to the Penn Treebank (Marcus et al., 1993) style of tags. We obtained these from the Stanford Part-of-Speech tagger, trained on the sections 0-18 of the Wall Street Journal section of the Penn Treebank. MaltParser itself was trained on sections 2-21 of the same part of the Penn Treebank extended with about 4000 questions from the QuestionBank (Judge et al., 2006). It outputs Stanford typed dependencies (De Marneffe and Manning, 2008).

One approach to getting phrases, which we will discuss in Section 7.4.2, is to look for subtrees of dependency trees where the head word in the tree had the required word type as specified by the answer type.

4.8 Answer evaluation

Already at our first look through the data it was obvious that looking through each answer manually every time we performed a test would be too time consuming to be practically feasible. Our first automatic evaluation technique was to look for exact matches to the suggested gold answers. Both the suggested answer and the gold answer were normalised by turning words into numbers as described in Section 4.2.

[3]Dependency parsers can be used in many other parts of a question answering system as well. Part of our motivation for implementing dependency parsing was that we wanted to compare dependency parses of sentences to dependency parses of questions. However, we did not have the time to do this.

This method was very good at evaluating yes/no answers, which is the majority of our data set as shown in Table 3.2. It also performed reasonably well on questions that asked for numbers or years.

We will present one alternative method for automatic evaluation in Section 5.5.7, and another alternative method in Section 7.4.1.

Part III

Error analysis

Chapter 5

Manual development cycle

In this chapter, we will first discuss an error taxonomy in order to better be able to discuss different kinds of errors, and what exactly an error is as opposed to something that is correct. We will briefly discuss different types of error analysis, before we go into the details of the error analysis we performed on our simple baseline system from the previous chapter. This chapter will show the results and findings that we obtained from our error analysis, and we will discuss the specific methods we used in more detail in the next chapter.

5.1 Error taxonomy

In order to talk about errors we need to establish more formally exactly what we mean by the term «error». Most people probably have some notion of what an error is, but because we spend so much time on discussing errors in the following chapters we will need to define the term more formally.

We have used the term «module» to denote a subpart of our question answering system. Each module takes one input and returns zero, one or more outputs[1]. In the following discussion we will use the term «module output» as whatever the module we discuss returns and passes on in the pipeline. An important note is that the final answer is also a module output – it is simply the output of the final module in the pipeline. Whether a system output is incorrect is also, as we have discussed in Section 2.6, sometimes not entirely clear, and can depend on how the output is evaluated.

Consider this seemingly simple example question:

Question How many people live in New York City?

If we wanted to be pedantic we could answer that it is impossible to give an exact number because people move in and out of the city all the time. We could also be obtuse on purpose and answer «a great many». Both these answers are technically correct but do not provide the information whoever asked the question wanted, and we can safely assume that if a

[1] Zero outputs is unfortunate, but possible.

question answering system returned these answers they would be perceived as incorrect.

What is the correct answer? According to Wikipedia the number of people living in New York City was 8 405 837 at March 27th, 2014, so we will use this as the perfect, ideal answer in this example. Other answers would probably also be acceptable. We could round the number to the nearest million («about 8 million») or place it in a reasonable interval («between 8 and 9 million»), and for some people these would be acceptable answers while to others they would be too imprecise.

As we can tell, an answer can be either correct or incorrect, but it can also be somewhere in between: partly correct, mostly correct, correct but not expressed precisely enough for the asker, and so on. Additionally, we must keep in mind that how the answer is evaluated depends on the context. This is important to us because many question answering systems are evaluated not by whether they answer the question perfectly, but by how well the question was answered. In other words, a question answering system does not have to answer a question perfectly, it just has to answer the question well enough. We can imagine a threshold that needs to be met – an answer that meets the threshold is marked as good enough or correct, an answer that does not meet the threshold is marked as incorrect.

The same principle holds true for the individual modules in a question answering system. Each module only has to perform well enough for the next module to do its job. For example, the answer selection module only needs to get one correct answer candidate from the answer extraction module in order to do its job. Further incorrect answer candidates from the answer extraction module detract from its output, but it will still be good enough. If, on the other hand, no correct answer candidates were retrieved, it would be impossible for the answer selection module to select the correct answer.

Similarly the answer extraction module should need only one passage containing the correct answer from the passage retrieval module. It is likely that the passage retriever has retrieved passages that do not help us find the answer, but the system should still be able to extract at least one correct answer candidate if at least one passage is relevant to the correct answer.

Thus, we have the following, somewhat informally defined metric for determining whether a module output is good enough: Is it possible for the next module to produce good enough output? Or, in a broader perspective: Will the output from this module be responsible for an overall failure in the system? A big part of error analysis is to identify the earliest module that caused the error. Because question answering systems can be fairly complicated, this can be difficult.

Because each module depends on the previous modules it is necessary that the input – the output from the previous module – is good enough. However, there are usually several possible inputs that will allow the current module to produce good enough output for the next module. This means that it is practically not feasible to define a set of «correct» outputs that we expect the module to produce if it performs well, simply because there are too many of them.

Note that this does not mean that it is impossible to know something about what we expect the output to contain. For some modules this is trivial, for example for the question analysis module, which is responsible for returning exactly one of a predefined set of possible expected answer types. For document and passage retrieval we can sometimes define what we want the modules to produce: if we know that there is a finite number of relevant passages in our set of documents, we could define a correct output from a passage retriever to be any output that contains at least one of these passages. In most cases, however, we do not know when we start the error analysis process how many, if any, passages contain the answer to the question.

More generally, in order to determine whether something is incorrect we need to have a notion of what properties the «perfect» answer is supposed to have. Without the context of the correct answer we have no metric with which to compare the output we want to evaluate. We do not always have to know the exact answer to a question to know whether a system answer is incorrect. Let us imagine that a question answering system answered our example question incorrectly:

Question How many people live in New York City?
System answer 1

If we had a human assessor evaluate this answer they would not have to know the exact answer in order to immediately dismiss the system answer as incorrect. For the human the system answer is obviously wrong, because most people know that New York City is a large city.

This principle also holds true between individual modules in a question answering system. For example, we could have a very simple pattern matching algorithm for determining the expected answer type which only looked at the first word of the example question. In the case of our current example it would be the word «how», and we could reasonably expect that this question asks for the manner of something, but if a human looked at the question this expected answer type would immediately stand out as incorrect.

If we do have knowledge of a suggested gold answer, however, we have yet another approach to finding errors. Looking only at the suggested gold answer, a number, we can see that an expected answer type asking for manner is incorrect. For our data set we already have access to suggested gold answers because they were part of the data set, and if we did not have access to them it would be possible to construct them manually.

This property of the data set is very useful, not least because it allows us to automatically evaluate the system answers, but also because it gives us another angle from which we can evaluate the output of each module. For example, we can look at the output from the passage retriever and see if at least one passage contains the suggested gold answer, as we did when we established the upper bound for this data set. If at least one passage contains the suggested gold answer we know that it is possible for the passage retriever to return an ideal output. This kind of evaluation would

be a lot more time consuming, if not outright impossible, if we only had access to the questions.

5.2 Error analysis

With the above reflections in mind, let us turn to error analysis[2]. In order to improve our system we need to know what kinds of errors the system makes. Error analysis is the process of finding this information.

There are different approaches to error analysis, depending on how much detail we want to have on each error. Broadly we can define the different methods as either statistical or manual. Statistical error analysis gives us a broad overview of subsets of questions and answers, and how well the system performs on these. If we want an overview over what kind of questions we want to focus on this is a good way of analysing errors. For example, Table 5.2 shows a statistical analysis of errors categorised by question type. When we perform statistical analysis we have to know the categories we want to look for before we start analysing.

When we perform manual error analysis, on the other hand, we construct the categories of errors as we perform the analysis. We perform manual error analysis when we want to know the exact details of what went wrong in our system. Manual error analysis is much more time consuming than statistical error analysis, but absolutely necessary if we want to understand what went wrong and how to fix the problems. In manual error analysis we go through the output of each module in an attempt to discover where the error originated.

Examples of categories when we perform statistical error analysis can be, for example, «by expected answer type» and «by length of question», and must be determined before we start the analysis. When we analyse errors manually the categories emerge more dynamically as we try to find errors that have something in common. Examples of categories can be «caused by passage retriever» or «unable to find relevant passage». It is up to whoever performs the error analysis to determine what categories make sense for this particular error analysis.

Error analysis in general is typically a combination of both statistical and manual methods. We often start with statistical error analysis and move to manual error analysis when we have a notion of what to look for. Statistical error analysis is usually fairly fast, both in collecting the data we want to get and in analysing it. Manual error analysis, on the other hand, can take a very long time. We will discuss our attempts to reduce the time spent on manual error analysis in Chapters 6 and 7.

[2]Error analysis is also called performance analysis. We prefer the term error analysis in this thesis because we want to focus on how to find and remove errors, while improved performance is a byproduct.

5.3 Analysing our baseline system

In this section we will go into detail on what error categories we found. We started with an initial manual error analysis, described in this section, and from this basis determined that we needed a more detailed error analysis on some questions, which we will describe in Section 5.5.

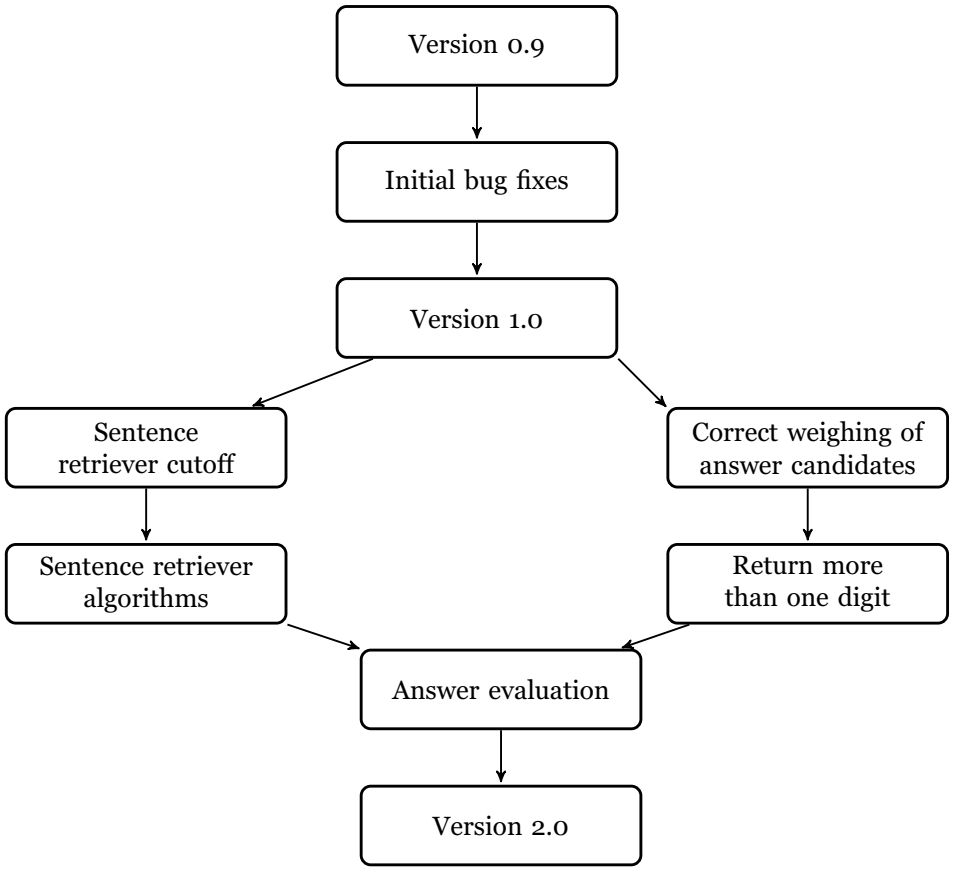

Figure 5.1: The workflow of our manual system development cycle.

Figure 5.1 shows a visual representation of the workflow in our development cycle. After a short round of bug fixes, we had a version we were happy with, and which we used for the remainder of our development cycle. The last improvement on the system was a new method for evaluating whether an answer was correct.

5.3.1 Initial bug fixes

In the following subsections we will assign categories to errors in the baseline system, and attempt to improve on the baseline system.

In the first round of error analysis, we found a handful of errors that were so obvious and easy to remove that we simply removed them and ran the system again. The results from this version of the system, let us call it version 0.9, are not interesting enough that we will use space on describing

them in detail, but because bugs are almost bound to happen in any kind of system engineering we will broadly describe what they were.

First, in this version of our software we had a module responsible for normalising numerals into digits. This part of the system also introduced a small problem because it turned the word «and» into a «0» – which is excellent when we want to translate «one hundred and twelve» into «112» (not «100 and 12»), but not when the module translates «cats and dogs» into «cats 0 dogs».

We also observed and fixed that the system originally would do part-of-speech and named entity tagging on partly normalised sentences. These methods assume that they will process normal natural language, so normalised sentences made them perform badly. This in turn caused problems when we extracted answer candidates by word types or named entity tags.

The remainder of this chapter describes variations on the fixed baseline system, without these errors, and we will call this version version 1.0.

5.3.2 Error analysis on the baseline system

Because we did not want to overfit our system to the data we could not perform detailed error analysis on all the results from the simple baseline. For our first error analysis we looked at only the dev1 sets from the three years. For each question that we were unable to get the correct answer for, we looked at the outputs of the different parts of the system and noted down where the error first originated.

Out of a total of 218 questions over all three sets we had 164 incorrect answers. An overview of the performance is shown in Table 5.1.

Year	Questions	Accuracy (%)
2008	90	21 (23.4%)
2009	49	14 (28.6%)
2010	79	19 (24.0%)
Total	218	54 (24.8%)

Table 5.1: Accuracy across all three years of dev1 sets.

Statistical analysis of results by expected answer types, shown in Table 5.2, suggests that our system performs comparatively well, though worse than the majority baseline from Table 4.1, on yes/no questions but badly on everything else. That the system performed better on yes/no questions than the other question types is by itself not surprising, since we know that yes/no questions are easier to answer due to the fact that they only have two possible answers, but it is still a useful result to have because it tells us that we do not have to look too closely at these kinds of questions but can focus on the other, more complicated questions.

Manual error analysis on our system yielded six different categories, as detailed in Table 5.3. To give a better understanding of how we came to these categories we will give some examples of each kind of error. The

Year	Question type	Total	Correct
2008	What	24	1
	Who	4	0
	When	2	0
	Number	5	1
	Yes/no	45	19
	How	2	0
	Where	6	0
	Why	2	0
2009	What	8	0
	Who	0	0
	When	3	1
	Number	7	0
	Yes/no	25	13
	How	1	0
	Where	5	0
	Why	0	0
2010	What	17	0
	Who	4	0
	When	7	0
	Number	8	0
	Yes/no	38	19
	How	3	0
	Where	2	0
	Why	0	0

Table 5.2: Summary of the different expected answer types according to our question typer, and performance of simple baseline on these.

sentence retriever error category has more than one cause, and we will review sentence retriever errors in Section 5.4

All these errors, except the two categories «System answer different from gold answer» and «Cross-sentence reference», are tightly connected to a module. «Sentence retriever» errors are tied to our sentence retriever. «Wrong answer retrieved from sentence» and «Negation detector» errors are tied to answer extraction, and «Question typer» errors to our question analysis module.

5.3.3 Wrong answer retrieved from sentence

Sometimes the correct sentence was returned, but the answer extraction and answer selection modules were unable to determine the correct answer:

Question What may a leopard be mistaken for?

Sentence 1 Johns Hopkins University Press 1999 ISBN 0 8018 5789 9 One of many spotted cats, a leopard may be mistaken for a cheetah or a jaguar.

Category	Number of errors
Sentence retriever	92
Wrong answer retrieved from sentence	29
Negation detector	17
Question typer	15
System answer different from gold answer	8
Cross-sentence reference	3
Total	134

Table 5.3: Types and number of errors according to manual error analysis on development set 1 across all three years.

Sentence 2 Males are considerably larger than females and weigh 37 to 90 kg compared to 28 to 60 kg for females.

Sentence 3 They are circular in East Africa but tend to be square in southern Africa.

Sentence 4 In Antiquity, it was believed that a leopard was a hybrid between a lion and a panther, as is reflected in its name, a Greek compound word derived from léon ("lion") and párdos ("male panther"), the latter related to Sanskrit prdaku ("snake, tiger, panther").

Sentence 5 leopard The leopard (Panthera pardus) is an Old World mammal of the Felidae family and the smallest of the four 'big cats' of the genus Panthera, along with the tiger, lion, and jaguar.

Sentence 6 The leopard consumes virtually any animal it can catch and ranges from rainforest to desert.

Sentence 7 A panther can be any of several species of large felid; in North America, the term refers to cougars, in South America, jaguars, and elsewhere, leopards.

Sentence 8 The leopard is an agile and graceful predator.

Sentence 9 The leopard has rosettes rather than cheetah's simple spots, but they lack internal spots, unlike the jaguar.

System answer East

Gold answer A cheetah or a jaguar

In this particular instance the error is due to an error in the weighing algorithm, which was supposed to give higher weights to answers that occurred in more than one sentence, but was implemented in a way that gave a penalty to answers that occurred multiple times. Another type of error in the same category was when two or more answers were given the same score, in which case we chose one at random. The randomly chosen answer could equally likely be incorrect as correct. We will analyse this category in closer detail in Section 5.5.

5.3.4 Negation detector

Our system detects negations simply by looking at whether any of the retrieved sentences have the word «not» in them. This is obviously not the

best way of detecting negations, since it is very sensitive to false positives. Recall that this analysis is performed on the performance of the system version 1.0, which retrieves up to ten sentences and therefore is very likely to introduce a false positive. As we already know the majority of questions are yes/no type questions, so all improvements on these kinds of questions are going to significantly affect the overall accuracy.

To have a measure of how many false positives the negation detector caused we let the system run one more time with the same settings, but answering only «yes» to all yes/no questions. Table 5.4 shows the impact. The high number of answers that are «yes» (see Table 4.1) is obviously a property of our particular data set and should not be seen as reflecting all possible data sets. Answering «yes» to all the yes/no questions is obviously not the way to solve this error, we only perform this statistical analysis to show how important it is to solve it.

Set	Baseline	Answering yes to all yes/no questions
2008	28.9%	44.4%
2009	28.6%	38.8%
2010	13.9%	30.4%

Table 5.4: Impact of answering yes to all the yes/no questions

5.3.5 Question typer

The «question typer» category is not limited to errors that are directly caused by the question typer. Assigning incorrect question types are not always the cause of an error – for example, a location is often also a noun, so a question that should be classified as a «where» question but is classified as a «what» question can still be answered correctly by our system. Looking only at the questions and expected answer types, we get the summary as shown in Table 5.5. Note how there are 16 errors in this Table, but only 15 errors categorised as a «question typer» error in Table 5.3.

This category also includes errors caused by an incorrect assumption by our part about the expected answer type, such as:

Question Where do leopards often hide their kills?
System answer Panthera
Gold answer in dense vegetation

Where-questions are assumed to have a geographical location as the answer, and as we can see this is not correct in all cases. In this case, the system mistook «Panthera» for a likely location (it is actually the Latin name for leopards).

One question is classified as a yes/no question, but is in fact a question that gives us two alternatives and asks us to choose one:

Question Did the golden age of xylophones come before or after the first usage of the European-derived orchestral?

Set	Questions	Incorrect answer type
2008	90	2 (2.2%)
2009	49	6 (12.2%)
2010	79	8 (10.1%)
Overall	218	16 (7.3%)

Table 5.5: Summary of incorrect answer types over all dev1 sets.

System answer YES
Gold answer after

5.3.6 System answer different from gold answer

There are some questions where the system answer is correct, just different from the gold answer:

Question Is Central Park adjacent to Uhuru Park?
System answer YES
Gold answer Central Park is adjacent to Uhuru Park.

In these cases it is also extremely difficult to get the exact same answer as the gold answer. For all intents and purposes the system answer is correct, but it is marked as wrong because of the way we judge correctness in our questions. We will show a more refined approach to judging answers in Section 5.5.7.

5.3.7 Cross-sentence reference

Cross-sentence references are errors where the system retrieved and identifies the correct sentence but the answer itself needs to be extracted from another sentence because the verbatim answer is not present. An example of this is:

Question Who spent the next 10 years painting for Look magazine?
Relevant sentence He spent the next 10 years painting for Look magazine, where his work depicted his interests in civil rights, poverty and space exploration.
System answer NULL
Gold answer Norman Rockwell.

In this example we would not be able to extract the gold answer because the the gold answer is not present – we would need information from another sentence in order to figure out what the word «he» refers to.

5.4 Improving sentence retrieval

One module that stands out as particularly problematic is the sentence retriever. This is mostly due to the fact that the sentence retriever has

many different outcomes. More simply put, we could argue that the task of sentence retrieval in our system is more difficult than question typing, which is the only task (other than normalisation) performed before sentence retrieval. It is also due to the fact that the rest of the modules depend on the output of the sentence retriever in order to do their job.

5.4.1 Sentence retriever unable to retrieve a relevant sentence

Sometimes the sentence retriever was unable to return a relevant sentence, for example for this question:

Question What is a hybrid animal resulting from a union between a leopard and a puma?

Sentence 1 The leopard consumes virtually any animal it can catch and ranges from rainforest to desert.

Sentence 2 Once distributed across southern Eurasia and Africa, from Korea to South Africa and Spain, it has disappeared from much of its former range and now chiefly occurs in subsaharan Africa.

Sentence 3 Johns Hopkins University Press 1999 ISBN 0 8018 5789 9 One of many spotted cats, a leopard may be mistaken for a cheetah or a jaguar.

Sentence 4 leopard The leopard (Panthera pardus) is an Old World mammal of the Felidae family and the smallest of the four 'big cats' of the genus Panthera, along with the tiger, lion, and jaguar.

Sentence 5 Physically, the spotted cat most closely resembles the jaguar, although it is of lighter build.

Sentence 6 In Antiquity, it was believed that a leopard was a hybrid between a lion and a panther, as is reflected in its name, a Greek compound word derived from léon ("lion") and párdos ("male panther"), the latter related to Sanskrit prdaku ("snake, tiger, panther").

Sentence 7 The generic component of its modern scientific designation, Panthera pardus, is derived from Latin via Greek

Sentence 8 However, it is believed instead to derive from an Indo-Iranian word meaning "whitish-yellow, pale"; in Sanskrit, this word's reflex was pandara, from which was derived pundarika ("tiger", among other things), then borrowed into Greek.

Sentence 9 The leopard is an agile and graceful predator.

As an error category this is perhaps not particularly interesting, but it is very important. Our sentence retriever had two problems: first, it was sometimes unable to retrieve a relevant sentence, as in this example. Our solution was to experiment with different retrieval algorithms. Second, if it was able to retrieve a relevant sentence it also retrieved many irrelevant sentences. The next two sections describe our attempts to solve these problems.

5.4.2 Sentence retriever cutoff

Passage retrieval is an important part of question answering systems. In our question answering system passages were defined as a single sentence, so passage retrieval is synonymous with sentence retrieval. If we are unable to retrieve a relevant sentence the remainder of the system will be completely unable to answer the question. In order to better evaluate the other modules of the system, which are perhaps more interesting to evaluate in the scope of this thesis, we need to make sure that the sentence retriever performs as well as it can. The following two subsections will detail some error analysis and analysis of the sentence retriever.

The baseline system ranks all the sentences in the documents according to a frequency-based measure and tries to figure out the answer from the up to 10 sentences with the highest scores[3]. During manual error analysis we looked at the sentences that were retrieved for each question and realised that for each question, there was usually only one sentence that actually contained the answer to the question. The rest of the sentences had an impact on performance, but they were merely noise and if they helped the accuracy of the system it was an accidental property of the document set, not something we had planned for.

On one hand we need to return enough sentences that we are certain that the sentence that answers the question is present. On the other hand, any sentence that we include when we search for answers that is not relevant is actually noise.

The number 10 seems like a suboptimal choice. Ideally we would like to only return the one sentence that is relevant, but this would be extremely hard to do automatically. Thus we need to aim for the next best thing, to return enough sentences that we can be reasonably certain that we return the relevant sentence, and no more than that.

In order to make an informed decision on what number of sentences to return as the default, we decided to allow the sentence retriever to retrieve 1 to 10 sentences, with the remainder of the system unchanged, and have a look at how the number of relevant sentences we retrieved had an impact on our results.

We can see a general trend in all the Figures 5.2, 5.3 and 5.4. Fetching a single relevant sentence is not ideal, but neither is 10 – and as we add more sentences the accuracy goes down. Intuitively this makes sense, since adding more sentences slightly increases the chance that we introduce the «relevant» sentence, but greatly increases the chance that the sentence we introduce is simply noise that will not further support the correct answer.

The ideal number of relevant sentences to get, just from looking at the data in the figures, is a matter of interpretation. It seems that for most questions the ideal number is one. On the other hand, there is a small peak for some sets at five sentences as well, though that peak is only in a few

[3]Sometimes, as in the example in the previous section, the sentence retriever returned less than 10 sentences because it was unable to find that many sentences that it judged to be relevant. The sentence retriever was strictly speaking allowed to return up to 10 sentences, we did not force it to return exactly 10.

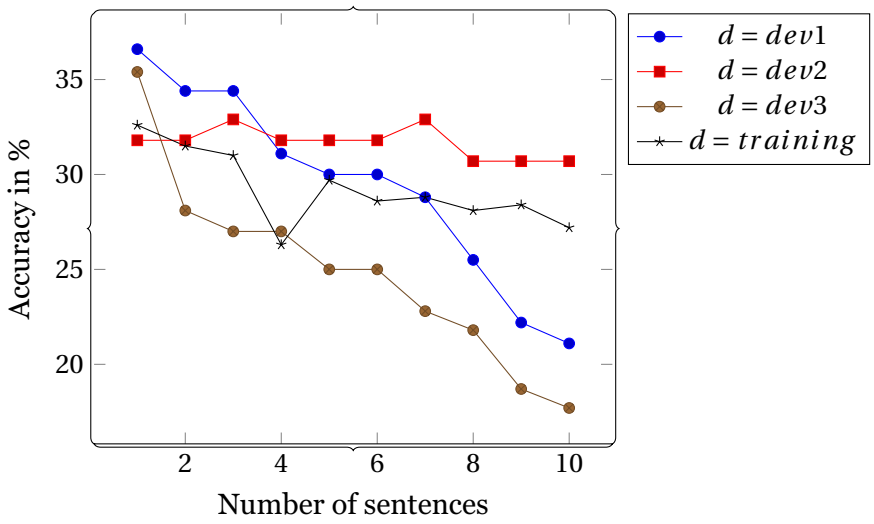

Figure 5.2: Impact of retrieving n sentences on accuracy for the question sets from 2008.

cases higher than the accuracy with one sentence.

We theorise that the questions that are answered correctly with one retrieved sentence do not necessarily overlap with the questions that are answered correctly with five retrieved sentences. We will return to this subject in Chapter 7, and for now choose to return five sentences in order to increase the chance of returning the correct sentence.

5.4.3 Sentence retrieval algorithms

We experimented with three different algorithms for sentence retrieval. The first is the frequency-based algorithm from the baseline version of the system as described in Section 4.6. The other two are based on respectively tf-idf and n-gram matching, and we will briefly describe how these work before we go on to analyse their performance. Because this analysis is closely connected the the sentence retriever, which we made some discoveries about in Section 5.4.2, the following subsection will describe a system version that is identical to the system version 1.0, except that we have built upon the work in the previous subsection and run the experiments so that each algorithm returns five sentences, not ten. The remainder of the error analysis is performed on the baseline system without modifications.

The first new sentence retriever algorithm we experimented with was based on tf-idf (Manning et al., 2008) with the question as the search term. We usually define tf-idf is the product of two numbers, term frequency (tf) and inverse document frequency (idf). It is a scoring algorithm that aims to give a higher weight to rare terms in a document on the assumption that rare terms are more meaningful.

It is possible to define tf in different ways, we chose to implement it in a way that simply counted how often a word (the term) occurred in the

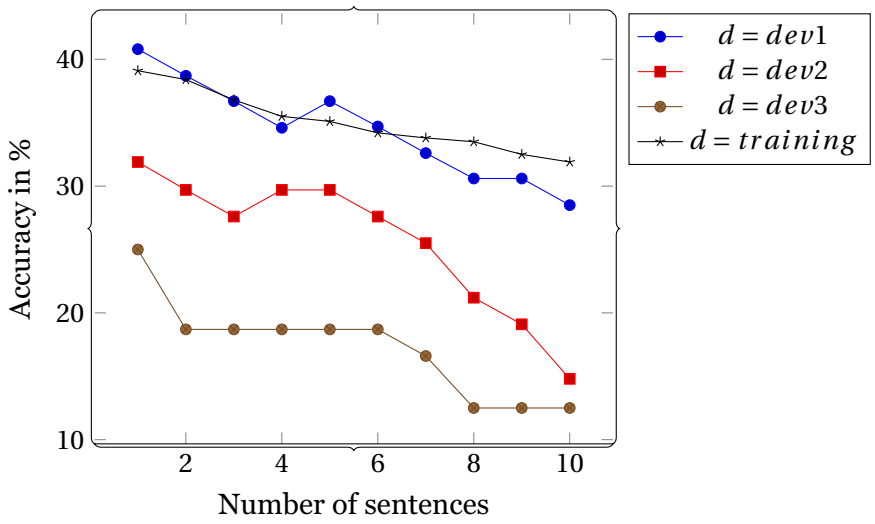

Figure 5.3: Impact of retrieving n sentences on accuracy for the question sets from 2009.

sentence. Because we used tf-idf for sentence retrieval instead of document retrieval we defined idf as

$$idf = \sum_{t \in q} \frac{f(t,s)}{\log(\frac{N}{f(t \in d)})} \tag{5.1}$$

Where $t \in q$ is the words in the question, $f(t,s)$ is the number of times the word occurs in the sentence, N is the number of sentences in the document and $f(t \in d)$ is the number of sentences the word occurs in.

The other sentence retrieval algorithm we implemented was based on finding n-grams that were the same in the question and in the sentences. An n-gram is a sequence of n words, so for example in this sentence:

Example 10. *«This is a sentence»*

there are two 3-grams: «This is a» and «is a sentence».

We extracted all n-grams for $n \geq 3$ from the question and searched through the document for exact matches.

If an n-gram in the question occurs many times in the sentences a very large number of sentences will be returned. Handling additional sentences is fairly expensive, runtime-wise, in our system, so we would like to not return too many. We ignore n-grams where n is less than 3, on the assumption that 2- and 1-grams are so common that they would render the method useless.

This means that we want to return at least one sentence (without at least one sentence we have nothing to find answer candidates from) but not too many.

The n-gram approach in contrast with frequency count and tf-idf is less sensitive to noise from common words, but is also more strict. We may not be able to return any sentences if one of the words in the n-gram is a synonym for the word we wanted to match, or simply misspelled. In

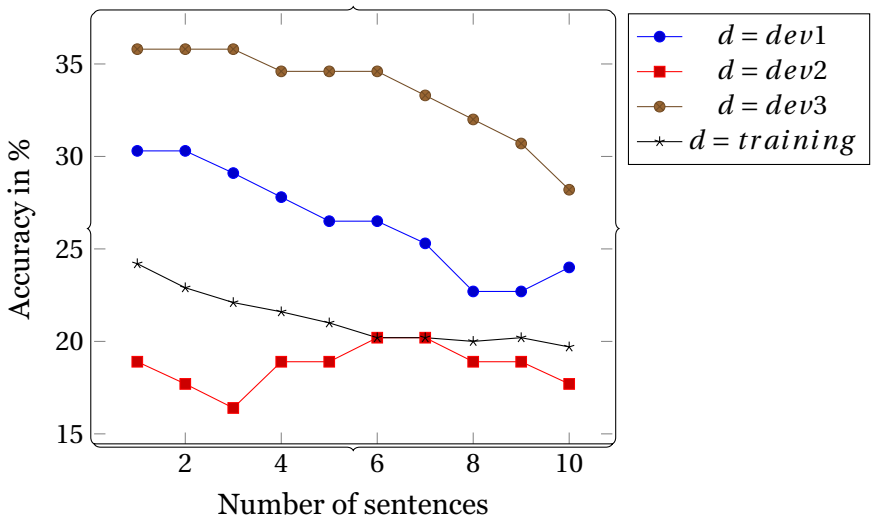

Figure 5.4: Impact of retrieving n sentences on accuracy for the question sets from 2010.

Year	Frequency	+n-grams	tf-idf	+n-grams
2008	35.5%	43.3%	40.0%	43.3%
2009	44.9%	42.9%	46.9%	46.9%
2010	25.3%	34.2%	32.9%	34.2%

Table 5.6: Accuracy on dev1 sets across all years, with and without n-gram sentence retriever

addition, this algorithm returns way too many sentences for the result to be useful if one n-gram in the question occurs many times in the text. On the other hand, n-grams take word proximity – how close to each other the words we are looking for are in the sentence – into account, while frequency count and tf-idf completely ignore this feature.

We did some analysis on the methods in order to establish whether they were beneficial to our system. Table 5.6 show the impact on accuracy for this system. For the n-gram method, we allowed the system to fall back to one of the other methods if the n-gram method returned zero or too many (more than five) sentences. We will experiment on mixing these approaches in Chapter 7.

Across all sets, n-gram matching with tf-idf as the fallback algorithm performed best. Additionally, we observed that using n-gram matching improved run times, because we did not need to calculate how common each word in the search term (the question) is in the document before we started searching for relevant sentences.

If tf-idf was used, the score is the tf-idf score. If an appropriate number of n-gram matches were found so this method was what returned sentences, each score is equal to the size of the n-gram squared. Sentences with a score of 0 were never returned.

Category	Size
Word incorrectly penalised	7
Only first number retrieved	6
Wrong word chosen at random	4
Retrieving correct answer is too hard	4
Tagger makes a mistake	3
Correct sentence ranked too low	3
Only one word of gold answer retrieved	2
Total	29

Table 5.7: Categories of errors we made when the correct sentence was retrieved

5.5 In-depth error analysis

The most interesting error category in several ways was the one where we managed to retrieve the correct sentence with our sentence retriever, but still got an incorrect answer. First, they are the errors that are closest to an actual, correct answer, and second, they are diverse and highlight some further problems both earlier and later in the pipeline. Therefore we look particularly closely at these errors.

We performed another manual error analysis on the questions that produced these errors, and broke them down into subcategories. We have summarised these subcategories of errors in Table 5.7. Except for the two largest subcategories, which are implementation errors in the code rather than design choices, the subcategories are rather similar in size. All of the errors seem like they can be removed, except the category where the answers were simply too hard to get. In the following paragraphs, we will walk through all sub-categories in turn, give examples, and discuss possible system improvements.

5.5.1 Word incorrectly penalised

Words that occurred in more than one retrieved sentence were incorrectly penalised in the first iteration of our system, instead of ranked higher as we intended. Our example in the previous section was this:

Question What may a leopard be mistaken for?

Sentence 1 Johns Hopkins University Press 1999 ISBN 0 8018 5789 9 One of many spotted cats, a leopard may be mistaken for a cheetah or a jaguar.

Sentence 2 Males are considerably larger than females and weigh 37 to 90 kg compared to 28 to 60 kg for females.

Sentence 3 They are circular in East Africa but tend to be square in southern Africa.

Sentence 4 In Antiquity, it was believed that a leopard was a hybrid between a lion and a panther, as is reflected in its name, a Greek compound word derived from léon ("lion") and párdos ("male panther"),

the latter related to Sanskrit prdaku ("snake, tiger, panther").

Sentence 5 leopard The leopard (Panthera pardus) is an Old World mammal of the Felidae family and the smallest of the four 'big cats' of the genus Panthera, along with the tiger, lion, and jaguar.

Sentence 6 The leopard consumes virtually any animal it can catch and ranges from rainforest to desert.

Sentence 7 A panther can be any of several species of large felid; in North America, the term refers to cougars, in South America, jaguars, and elsewhere, leopards.

Sentence 8 The leopard is an agile and graceful predator.

Sentence 9 The leopard has rosettes rather than cheetah's simple spots, but they lack internal spots, unlike the jaguar.

System answer East

Gold answer A cheetah or a jaguar

Both «cheetah» and «jaguar» occurred in more than one sentence and were therefore (incorrectly) penalised. «East», on the other hand, occurred in only one sentence and was therefore ranked highest. Because this was an implementation error, it was fairly simple to solve by implementing the correct ranking algorithm.

5.5.2 Only first number retrieved

The «only first number retrieved» error subcategory applies to a certain type of questions, in the subcategory where the expected answer type is a number. An error in the answer retrieval module made the system only return the first digit in these answers, so any number that had more than one digit was automatically returned incorrectly. This error was simple to remove once we were aware of it. An example is:

Question How many letters are in the basic Latin alphabet?

System answer 2

Gold answer 26.

5.5.3 Wrong word chosen at random

We have already discussed the error subcategory where the wrong word was chosen at random in Section 5.3.3. In these cases we returned several answer candidates, and the answer validation module ranked both the correct and some incorrect answer candidates equally. In these cases we did not have any way of finding the best candidate, and we were forced to return one at random. An example of this is the following:

Question Where is the leopard distributed?

Most relevant sentence Once distributed across southern Eurasia and Africa, from Korea to South Africa and Spain, it has disappeared from much of its former range and now chiefly occurs in subsaharan Africa.

System answer Korea

Gold answer southern Eurasia and Africa

Here the system correctly tries to find a location, assigns each item that occurs the same number of times in the sentence an equal weight and picks «Korea» at random. It is very difficult to deal with this kind of error. One approach, if the different answer candidates are close enough to each other, is to return a longer passage with all the different answer candidates, so that we could be certain that the returned answer actually answers the question. We will explore phrase retrieval in Section 7.4.2. On the other hand, this approach would make the system answer longer and contain more irrelevant information, which is hardly ideal. In this particular instance we would have to return more or less the whole sentence because the different answer candidates are so far from each other. Another approach, which we described in Section 2.4, is to use proximity to a key word to find the most likely candidate.

5.5.4 Retrieving correct answer is too hard

For some questions, retrieving the correct answer was simply too hard for our system. Even if all our modules did their jobs perfectly the system would not be able to return the correct answer. Some questions require a level of analysis that our system will probably never be able to achieve.

Question When did Yuan Shikai die?
Relevant sentence Yuan gradually consolidated power and became by 1915 the new emperor but died less than a year into his reign.
System answer 1915
Gold answer 1916

Getting the system to add one to the year because of the phrase «less than a year into» would require a level of engineering that is outside the scope of this project.

This is also the case in the following example:

Question Was the xylophone associated with the folk music of the United States by the 19th century?
Relevant sentence The xylophone, which had been known in Europe since the Middle Ages, was by the 19th Century associated largely with the folk music of Eastern Europe, notably Poland and Eastern Germany.
System answer YES
Gold answer No.

Resolving this would require the system to know whether «Eastern Europe, notably Poland and Eastern Germany» are parts of «the United States», which is a trivial task for humans but significantly harder for computers.

5.5.5 Tagger makes a mistake

Sentences were tagged with a part-of-speech tagger and with a named entity recogniser. In same cases, these were not able to correctly tag the sentence, which could lead to errors when we performed answer candidate extraction. For example, for this question we found a sentence containing the correct answer, but the named entity recogniser did not tag «Beiping» as a location:

Question In 1949, where did Communist forces enter without a fight?
Relevant sentence Mao Zedong proclaiming the establishment of the People Republic of China in 1949 A man stands before a column of tanks which were sent to Tiananmen Square earlier to suppress the Tiananmen Square protests of 1989 On 31 January 1949, during the Chinese Civil War, Communist forces entered Beiping without a fight.[4]
System answer China
Gold answer Beiping

5.5.6 Correct sentence ranked too low

The subcategory labelled «correct sentence ranked too low» is hopefully self-explanatory. The system returned up to 10 sentences, and if the system answer was selected from a sentence ranked higher than the correct sentence we counted it as an error in this subcategory.

5.5.7 Only one word of gold answer retrieved

For some questions, we managed to return one word of the suggested gold answer, but not all. An example of an error in this subcategory is:

Question Who wrote the first novel in Finnish?
System answer Aleksis
Gold answer Aleksis Kivi

Here we have returned the first name of the author, but not the whole name. Whether this answer should be accepted as correct is up for debate. Ideally, though, we would like to return the whole answer and not just one word.

This subcategory highlights two different problems with this version of our system. The first problem was the fact that for most answer types we retrieved only one-word answers, but the suggested answers were more than one word. The exception was «why», which accounted for only two out of all the questions we tested on in our error analysis. If we had been

[4] As an aside, this sentence looks strange because it consists of three sentences. The first two sentences are apparently attached to an image in the Wikipedia article and have nothing to do with the answer, while the last («On 31 January 1949 ...») is the actual sentence. This happened because the source file was formatted in a way that made our sentence segmentation algorithms make errors. We could have attempted to fix this problem, but it turned out to be quite difficult. We will simply have to accept it as one of the problems with this data set.

able to retrieve the correct sentences more often this type of error would probably grow to be a lot larger. Table 5.8 shows how many of the gold answers we could potentially answer correctly with the current approach. We will return to returning more than one word in Section 7.4.2.

Year	Set	Questions	One-word gold answers
2008	Training	551 (237)	318 (81)
	Dev1	90 (46)	57 (11)
	Dev2	91 (48)	57 (9)
	Dev3	96 (45)	57 (12)
2009	Training	304 (149)	200 (51)
	Dev1	49 (25)	36 (11)
	Dev2	47 (26)	38 (12)
	Dev3	48 (21)	26 (5)
2010	Training	475 (161)	209 (48)
	Dev1	79 (29)	41 (12)
	Dev2	79 (24)	36 (12)
	Dev3	78 (33)	41 (8)

Table 5.8: Number of gold answers that are only one word, across all sets. The numbers in parentheses are for the sets with yes/no questions removed.

The second problem was that the way we evaluated system answers was very strict, requiring the system to return answers that were exactly identical to the suggested gold answers.

As our system started performing better we realised that we needed a better way to evaluate answers. Breck et al. (2000) use an automatic evaluation method that allows them to do manual evaluation only once, when they construct suggested gold answers. They compare their system answers to the gold answers and calculate the recall based on word similarity, but their gold answers have a fixed length, and they specifically note that a different approach must be used when gold answers have variable lengths. With this in mind, we noted which answers we as humans would mark as correct and recorded precision, recall and F-scores.

We analysed the 218 questions in the dev1 sets across all years and observed that the system answers we judged to be correct had some common properties, and they all met at least one of these criteria:

1. They match exactly with the gold answers
2. The recall was greater than or equal to 0.5
3. The precision was exactly 1
4. The system answer is «yes» or «no» and the gold answer begins with the same, but has a qualifying statement.

The latter criterion is met 55 times across all sets (not just the dev1 sets). An example of an answer that has this latter property is:

Question Is the octopus a cephalopod?
Gold answer Yes, the octopus is a cephalopod.

Based on this, we made our system first evaluate questions based on exact matches, as in criterion 1, and then give us a list of answers that met criteria 2-4 so we could evaluate these answers manually.

Compared to the two evaluation methods we have considered so far – fully automatic and fully manual – this semi-automatic evaluation method is a decent middle-way. Manual evaluation would be too time-consuming, and as we have seen fully-automatic evaluation is not suitable for our set of gold answers. Out of the 218 question-answer pairs in the dev1 sets, 18 answers had to be evaluated manually with this method. This is a significant improvement time-wise from manual evaluation, and a fair improvement accuracy-wise from the automatic evaluation method.

5.6 A new system version

After we found and to a large degree removed the errors we have discussed in this chapter, we considered the first development set to be largely exhausted. At this point we incorporated all the improvements we discovered in this chapter into out question answering system, and in Chapter 7 we will use this new version of the system, version 2.0, as a basis for further improvements and analysis.

The differences between version 1.0 and 2.0 are as follows:

1. We retrieve up to 5 sentences we believe are relevant instead of 10.
2. We use an n-gram-based method for sentence retrieval, falling back to tf-idf. instead of using the frequency-based method described in Section 4.6.
3. We convert numerals to digits in the normalisation step.
4. We return numbers with more than one digit.
5. Sentences are ranked correctly.
6. Answer candidates occurring in more than one sentence are no longer penalised.

We have so far not done anything to improve the system handling of negations, but as we will see this category has been reduced as a result of the sentence retriever retrieving fewer sentences. We have also not handled cross-sentence references, but this category is small and likely not worth the work it would take to improve upon.

Before we move on to discussing the methods we used in error analysis, we will finish this chapter with a summary of the improvement in accuracy on the first development set.

We can observe the impact of our problem fixes in Tables 5.9. There is a significant improvement! While we still get more than half our answers

Year	Questions	Version 1.0	Version 2.0
2008	90	21 (23.3%)	44 (48.8%)
2009	49	14 (28.6%)	24 (48.9%)
2010	79	19 (24.0%)	33 (41.7%)
Average	218	54 (24.8%)	101 (46.3%)

Table 5.9: Difference in accuracy on dev1 sets, before and after first round of error analysis and fixing.

incorrect, an improvement of almost 25 percent is encouraging. We have also beaten our majority baseline from Table 4.1.

Category	Before	After
Sentence retriever	92	33
Wrong answer retrieved from sentences	29	48
Negation detector	17	4
Question typer	15	11
System answer different from gold answer	8	16
Cross-sentence reference	3	5
Total	164	117

Table 5.10: Errors, before and after manual development cycle.

In Table 5.10 we can see that the improvements we introduced to the sentence retriever have helped quite a lot. Some of these have been moved to the category under, «Wrong answer retrieved from sentence». The detailed analysis of this category can be seen in Table 5.11, where the largest subcategory is «Wrong word chosen at random». This suggests that in addition to improving the accuracy of the system, we have also improved the system in that we have moved the errors closer to the end of the pipeline. This in turn means that the number of potential errors caused by these question-answer pairs have been reduced, and we are thus closer to a solution for the answers that are incorrect.

Category	Before	After
Word incorrectly penalised	7	0
Only first number retrieved	6	0
Wrong word chosen at random	4	20
Retrieving correct answer is too hard	4	8
Tagger makes a mistake	3	6
Correct sentence ranked too low	3	6
Only one word of gold answer retrieved	2	8
Total	29	48

Table 5.11: Detailed errors, before and after manual development cycle.

We also see a shift in the kinds of errors the system makes. Words that are incorrectly penalised and bugs where only the first number is retrieved are gone, and the number of sentences from which we get one correct word

increases. The same happens to the subcategory of errors where we get a wrong word by random. These errors are in a way better than the errors we had before we fixed our system, because they are at least closer to the correct answer.

Chapter 6

Automating error analysis

In this chapter we will discuss error analysis methods and how they can be automated. In the previous chapter, we detailed the results of a manual error analysis, without much reflection on the methods themselves. This chapter will begin with a description of how a manual error analysis is performed and what kinds of results we wish to obtain with it, before we move on to investigate how the methods used in error analysis can be automated.

We will also compare the results from a manual error analysis with an automatic error analysis on the data set and final system version from the previous chapter.

Finally, we will draw some conclusions on where our system can be further improved, based on the two error analyses.

6.1 Methods for error analysis

Error analysis has several requirements that must be met in order to be useful to researchers and system developers:

1. The error analysis must reveal new information about how the system functions and where the errors occur.

2. The information revealed must be correct, so that we do not base new improvements on false information.

3. The error analysis must not take too long.

The first and second requirements overlap a lot. The first requirement says that for each analysis of an error we would like to get a better understanding of that error, not just confirm the information that we already have. The second requirement is usually assumed to be met, but most error analysis, particularly manual error analysis, is vulnerable to human errors and misunderstanding of what the data shows, and we need to be aware of this requirement.

We intuitively assume that spending more time on error analysis will help us meet both the first and the second requirements. However, we

can usually not spend too much time on each kind of error, because time spent on error analysis is time that we can not spend on system improvement, experiments and so on. This is where the third requirement comes in. Spending more time means that we find better and more accurate information, and therefore helps us meet the first and second requirement, while removing us from the third requirement.

A balance must be found between all the requirements, and what the best balance is will vary between systems and who performs the error analysis.

As an example, let us consider a case from our own system. For a word to be retrieved as the most likely answer candidate for a «what» question with a one-word gold answer, it must be the word that has the highest score among words that are marked as nouns, across all retrieved sentences. We would like the system to be able to retrieve that word, and not any other words. If the system does not retrieve the correct word, we need to perform error analysis.

In order to get a full understanding of why a word is chosen, we need to make a list of all the words that are tagged as nouns in all the retrieved sentences, sum up the score for each word, sort the list by score and then choose the highest ranked words, and check if the chosen word is the gold answer. If the chosen word is not the gold answer, we then need to locate the gold answer in the list of answer candidates and determine why it was not ranked higher.

All of this is certainly possible to do for a human, but if it has to be done more than once it is not practically feasible. This means that when error analysis is performed manually, the error analysis in our example meets the first requirement very well, but if we have to perform the analysis many times we fail to meet the third requirement.

If we wanted to meet the third requirement better, we could do so at the cost of not meeting the first requirement to the same degree. For example, a human could say «I see that the answer is present in this set of sentences, but is not retrieved», which would take a significantly shorter time than the full analysis and still provide useful information.

An automatic analysis of the same error, on the other hand, can quickly determine whether the answer has the expected tag, and if so where in the ranked list of answer candidates it is located. This means that the automatic error analysis (in this instance) can find the same errors, with a high degree of detail, but significantly faster than we could reasonably expect from manual error analysis.

As we can see, how much detail we can expect to retrieve from error analysis differs between manual and automatic error analysis. Manual error analysis has the advantage that humans are better at seeing subtle differences, as well as the advantage that analysers can come up with new categories when necessary. Automatic analysis, on the other hand, can be better at analysing and counting different but similar instances of errors quickly.

6.2 Automating error analysis methods

As we mentioned in Section 5.2, there are, broadly speaking, two kinds of error analysis: Manual and statistical. If we want to automate our manual error analysis methods, we need to convert them to statistical methods.

Statistical error analysis methods count the number of occurrences of something. This «something» needs to be known before we begin the automatic error analysis. This is a very different approach from manual error analysis, where we start with a set of uncategorised errors and categorise them as we perform the analysis.

In other words: in manual error analysis we start with no pre-defined categories, and make categories as we perform the analysis. In statistical error analysis, we start with a full set of categories and assign each error to one category (or several, depending on how we choose to perform the analysis), but we are unable to make new categories as the analysis proceeds.

Let us return to our definition of an error. In Section 5.1 we discussed that in order to judge an output as incorrect, we often do not strictly speaking need to know exactly what the expected output is. Rather, we can say something about the expected properties of the output and see whether it conforms to these expectations.

In the example in the previous section, we had the expectation that the answer to a «what» question should be a noun. If the question is a «who» question, we have the expectation that the answer should be a person. If the output from a module does not meet this expectation – for example, if no sentences from the sentence retriever contain a word tagged as a person – we can assume that we have found an error.

At this point we have enough information to construct a statistical error analysis method: For each «who» question, we check each sentence we retrieve from the sentence retriever. If no sentences contain a set of words marked as a person, we can count this as an error in the category «retrieved sentences do not contain a person».

Because some errors are similar, and are caused by the same module in our system, we can either generalise or specialise the categories before we perform an analysis. For example, the category we just constructed likely has counterparts in sentences that do not contain a date for «when» questions, sentences that do not contain a location for «where» questions and so on. We can either create a specialised category for each of these errors, or we can create a more general category that collects all these errors.

Either way, these categories need to be defined beforehand, and how much detail we get from automatic error analysis depends on how detailed the error categories are.

6.3 Implementation and categories in automatic error analysis

We attempted to automatically perform an error analysis that could be compared with the error analysis in the previous chapter. The implementation of the automatic error analysis was fairly straightforward. For each module, we decided which properties of the expected output we could expect to be able to find automatically. We then implemented some helper methods to our system which compared the actual output to the properties of the expected output, and raised an alarm if they did not correspond.

For each question-answer pair only one alarm could be raised, meaning that a pair that generated an error in all modules would still only count as one error. We chose this approach mainly because an error in an early module usually means that the following modules will also cause an error, so counting several errors for each pair would likely misrepresent the performance of the later modules in our system. For each category we retrieve some information based on the gold answer, make an assumption about the expected properties of the module we evaluate and compare the output from a system module with the expected properties.

An overview of the different error categories can be found in Table 6.2, but we would like to begin with an overview of the thought process that went into the construction and implementation of detecting each error category.

For the question typer, we expected that for number questions the gold answer should be a number (and nothing else), and for yes/no questions the gold answer should start with either yes or no. We could have checked for word tags for the other expected answer types, but knew from the manual error analysis that these errors were rare and more often caused by a mistake in the tagging process than by an incorrect assumption about the word tag.

Errors in the sentence retriever were easier to find. Because the answer extraction is performed on the retrieved sentences, we know that the gold answer has to be present in at least one of the retrieved sentences. The set of sentences with the gold answer present were called the «gold sentences». Because the retrieved answer is in most cases retrieved from the highest ranked sentence, we also included another error category for when the highest ranked sentence was not one of the gold sentences.

For question types that were not «number» or «yes/no», we expected the gold answer to have either a part of speech tag or a named entity tag depending on the question type. If the retrieved sentences did not include the gold answer tagged (by the system) with the corresponding tag, we assume that it is an error caused by an incorrect tag. If the gold answer consists of more than one word, it is sufficient that one word in the gold answer has the expected tag.

Note that for some question-answer pairs the error is caused by an incorrect assumption on our part, for example when we expected the gold

answer to be tagged as a person in this example:

Question Who did Wilson win in 1917?
Gold answer Irish Americans

We will address this subset of errors in the next chapter, in Section 7.2.

In the version of the system we finished with in the previous chapter, only one word is returned from the answer extraction module. In order to be judged as at least partially correct, the returned word has to be a substring of the gold answer. The answer extraction module returns a set of answer candidates ranked by score (determined by number of occurrences and the score of the sentences the occur in) and chooses the candidate with the highest score. We created three different error categories for the answer extraction module:

1. None of the answer candidates are substrings of the gold answer.
2. At least one of the answer candidates is a substring of the gold answer, but is not one of the most likely answer candidates.
3. At least one of the answer candidates is among the highest ranked sentences, but a different answer candidate is chosen at random.

All these errors are variations of problems in the answer extraction module.

The second to last error category collects answers that are substrings of (but not completely identical to) the gold answer, and the final error category collects questions that have not been found in any of the previous steps but still do not produce the correct answer.

It is important to remember that the system has an inherent robustness – each module only has to perform well enough, not perfectly – so it is possible to automatically find an expected error that turns out to not be significant enough to produce an incorrect answer. For example, as we will see in Section 6.6, for some questions there are many sentences that contain the gold answer, and it is sufficient to choose any one of them even though a human would possibly say that that sentence does not answer the question.

Despite the robustness of our system, we would still like it to perform as well as possible. In Section 6.6 we will perform some manual annotation of our data set in order to analyse how well certain modules perform.

6.4 Comparison of results from manual and automatic error analysis

In order to explore the usefulness of automatic error analysis, we performed automatic error analysis on the dev1 sets from the previous chapter.

We repeat the results from the manual error analysis, which we performed in the previous chapter, in Table 6.1, and show the results from the corresponding automatic error analysis in Table 6.2.

The categories chosen for our automatic error analysis are based on the findings in the manual error analysis, so it is not surprising that the

Error type	Size
Sentence retriever	33
Negation detector	4
Question typer	11
System answer different from gold answer	16
Cross-sentence reference	5
Wrong answer retrieved from sentences	48
- Wrong word chosen at random	*20*
- Retrieving correct answer is too hard	*8*
- Tagger makes a mistake	*6*
- Correct sentence ranked too low	*6*
- Only one word of gold answer retrieved	*8*
Total	117

Table 6.1: Error types found manually at the end of manual development cycle. Categories in italics are subcategories of «Wrong answer retrieved from sentences».

categories to a certain degree overlap. For most of the error categories in both analyses there is an underlying assumption that it is connected to one specific module, and that that module is where the error originates.

Many categories in the manual error analysis overlap with categories in the automatic analysis. The manual error category «Sentence retriever» is for example very similar both in intention and size to «No retrieved sentences ...» and «Highest ranked sentence ...». Similarly, two categories in the automatic error analysis, «Question type is number ...» and «Question type is yes/no ...» have to do with the question typer, and correspond closely to the «question typer» errors in the manual error analysis.

The error categories «Negation detector» and «System answer different from gold answer» overlap with the general error category «Answer marked as incorrect» in the automatic analysis. The same holds true for the questions that we marked manually as «Retrieving correct answer is too hard», but this also overlaps to a certain degree with the automatic category «No sentences in the document ...». Cross-sentence references is another example of a category that we were not able to search for and count automatically.

«Wrong answer retrieved from sentence» overlaps with «Gold answer must be chosen at random», «Gold answer was not scored highest» and «No answer candidates were found». In these cases the automatic error analysis is more detailed than the manual analysis.

The category for errors that stem from a lack of gold answers in the source document overlaps with several of the categories in the manual error analysis: some of them are placed in the category for questions that are too hard, and others are manually categorised as a sentence retriever error, because if no sentences contain the correct answer it is impossible to retrieve a correct sentence.

Error type	Size
No sentences in the document with an exact match to gold answer	26
Question type is number, gold answer is not	7
Question type is yes/no, gold answer is not	3
No retrieved sentences contain gold answer	26
Highest ranked sentence does not contain gold answer	6
No relevant sentences were found	0
No retrieved sentence contains gold answer with expected NER tag	0
No retrieved sentence contains gold answer with expected POS tag	4
Gold answer must be chosen at random	7
Gold answer was not scored highest	6
No answer candidates were found	9
System answer is substring of gold answer	15
Answer marked as incorrect	17
Total	126

Table 6.2: Error types found automatically, across dev1 sets.

We might say that the two different error analyses show us the same errors from different angles, but since the error categories do not overlap completely it can be difficult to compare them directly. Rather, we can consider both to be useful in different ways. Manual error analysis can show details that automatic error analysis can not, and automatic error analysis can count details that a human analyser could not reasonably be expected to do.

6.5 Analysis of results from the different error analyses

With two different error analyses of the same system on the same data set, it is probably safe to begin to draw some conclusions on how we might improve the system. Aside from the category of answers that we consider impossible to find, we find that the largest error categories in both the manual and the automatic error analysis are caused by the sentence retriever and the answer extraction module.

Both «No retrieved sentences contain gold answer» and «Highest ranked sentence does not contain gold answer» from the automatic analysis are caused by the sentence retriever. The manual analysis shows the same tendency – though the category named «Sentence retriever» likely contains some errors caused by a lack of gold answers in the document, the sentence retriever also causes errors in the «Correct sentence ranked too low» category.

For the answer extraction module we have more detailed information on what went wrong. First off, we know that the difference in total errors between the two analyses are caused by answers that are automatically marked as incorrect, but are actually correct when judged manually. In addition, the sum of errors in the categories «Gold answer must be chosen

at random», «Gold answer was not scored highest» and «No answer candidates were found» adds up to a relatively large number of errors caused by the answer extraction module. At the same time, they suggest that the reasons for the errors in this module can be varied, since we do not see that all the errors are due to, for example, that the correct answer candidate receives a score that is too low.

The category for systems answers that are substrings of gold answers is also connected to the answer extraction module and fairly large, and should not be too hard to improve on if we can make the system return more than one word.

6.6 Data set annotation

Automatic error analysis needs information about what we expect the output from each module to conform to in order to function. The properties that the output needs to conform to can often to a certain degree be generated automatically.

For example, we expect that the sentences from the sentence retriever should contain the gold answer, so if we have the gold answer we can, after the sentence retriever has retrieved sentences, automatically check whether the sentences contain the gold answer. However, it is often the case that the gold answer is present in more than one sentence. For example, 52 sentences in the document connected to this question-answer pair contain the phrase «Nikola Tesla»:

Question Who was born precisely at midnight during an electrical storm in the present-day Croatia?
Answer Nikola Tesla

In this document there is only one sentence that a human would retrieve if they were asked to find the sentence that answered the question, but because we only expect the sentence retriever to perform well enough and not perfectly, we can choose to not mark it down as an error if the answer extraction module is able to return the correct answer. This robustness in the system is a nice property that we could not have if we expected each module to perform perfectly.

On the other hand, we want each module to perform as well as possible in order to reduce the chances of an error in the system. Therefore, we could analyse each module individually and see how well they perform. If we know what the perfect answer is we could mark it down as a partial error, in order to keep track of how often the system performs perfectly.

In addition to automatically retrieving sentences that contain the gold answer and assume that the sentence retriever must return at least one of these sentences in order to perform well enough, we can manually annotate the data set with the perfect output[1]. In that case, we would need to go through the sentences that contain the gold answer and find the one that

[1]This use for annotated output is commonly called an oracle.

is related to the question. This means that we get even more accurate information on how well a module in our system performs.

Again, we need to consider the time aspect. If finding the «perfect» sentence takes much time and gives us little to no new information, the requirements to economic error analysis are not met.

This data can be used for two purposes. First, it can be used for evaluation: instead of making an assumption about what properties the module output should have, we can check the module output against the data we have found manually. What this means is that we can get the accuracy of manual error analysis, but perform automatic error analysis, though only for the question-answer sets we have annotated.

The second purpose of annotated data sets is to simulate system performance when one system module performs perfectly. In these cases we run the system as normal, but instead of running a module we replace the output from the module with the annotated data. For example, we can manually find the «best» sentence the sentence retriever can find and instead of running the sentence retriever simulate that it returns the sentence we have actually found manually.

To examine the usefulness of this method we will annotate the output of two modules in our system: the question typer and the sentence retriever.

6.6.1 Question typer

One module for which we can manually annotate our data set in order to make the system behave as though the module performed perfectly is the question typer. A manual error analysis on only the question typer reveals that the question typer performs quite well, with only six errors across all three dev1 sets. We show the errors in Table 6.3.

		System							
		What	Who	When	Number	Yes/no	How	Where	Why
Gold	What	41	0	0	0	0	0	0	0
	Who	0	7	0	0	0	0	0	0
	When	2	0	12	0	0	0	0	0
	Number	2	0	0	20	0	0	0	0
	Yes/no	1	1	0	0	108	0	0	0
	How	0	0	0	0	0	6	0	0
	Where	0	0	0	0	0	0	18	0
	Why	0	0	0	0	0	0	0	2

Table 6.3: Confusion matrix of question typer performance.

One observation we can make is that our question typer performs quite well. The greatest number of miscategorised questions end up as «what» questions. This is because our question typer defaults to «what» when it is unable to properly categorise the question.

Another observation is that the question typer is incorrect only 6 times across these sets. Compare this with the results we got in Table 6.2, where we got 10 question typer errors. The question typer errors in the automatic

error analysis are also connected to two specific question types, «number» and «yes/no», while in the manual error analysis they are slightly more spread out. This means that the question type errors in our automatic error analysis are not always caused by the question typer itself, but rather by the assumptions we make when we detect some question types.

The combination of Tables 6.2 and 6.3 show a more detailed representation of what errors are caused by the question typer – either directly because the question type is incorrect or because we make an assumption that is not 100% correct – than the manual error analysis could show by itself.

Error type	Normal	Annotated
No sentences in the document ...	2	2
Question type is number, gold answer is not	0	1
No retrieved sentences contain gold answer	1	1
No retrieved sentence ... with expected POS tag	2	0
No answer candidates were found	1	0
Total	6	4

Table 6.4: Automatic error analysis of the six questions with incorrect answer types. Categories with no occurrences in either run have been removed.

The automatic error analysis of the system performance on the six questions that get an incorrect question type is shown in Table 6.4. A manual annotation of the data set shows that improving the question typer does improve the overall performance of the system. As expected, the improvement is found in the category connected to an unexpected part of speech tag. This is because the question typer and the expected tag are closely connected. However, the data set is perhaps too small for these results to be of significant value going forwards – a consequence of the fact that our question typer performs quite well.

6.6.2 Sentence retriever

One module that we know is important is the sentence retriever. The reason for this is that while the sentence retriever performs the same operation regardless of the previous modules in the system (except if the question type is yes/no), the rest of the system is very dependant on how well the sentence retriever performs. Therefore, improving the system in the modules before the sentence retriever might not improve the final output of the system if the sentence retriever does not perform well enough.

At this point we have two sets of sentences that are going to be retrieved: one for the system and one for the error analysis. The first is the set of sentences from the system. These are the sentences the system retrieved, based on the information from the question.

The second set is the set of sentences where we know that the gold answer is present. These are the sentences we compare the system sentences with in our automatic error analysis. The usefulness of these

lie mainly in the fact that we know that if we do not retrieve any of these sentences, the system will not be able to answer the question correctly. These sentences are retrieved according to the answer, and are therefore only available for error analysis, not to the system itself.

A gold standard for either of these sets can be constructed manually. Further, the gold standard for either set is going to be the same. The ideal set is going to consist of the one sentence that we think is most likely to answer the question. If the gold sentences are replaced by this, we have a gold standard we can automatically compare the output from the sentence retriever to, and if we replace the output from the sentence retriever with this sentence we can simulate that the sentence retriever has performed as well as possible.

Annotating sentence retriever output turned out to be more difficult than question typer output. First off, as we have already mentioned, for some questions there were over 100 sentences that contained the gold answer, which made the process lengthy. Second, for some questions the gold answer was present in some sentences, but not in any sentences that directly answered the question. One example is cross-sentence references, which we made a separate category for in the manual error analysis and decided that would be too hard to solve in the scope of this project. Another example is questions that are too hard, such as this:

Question How many territories joined to form Romania?
Sentence As a nation-state, the country was formed by the merging of Moldavia and Wallachia in 1859 and it gained recognition of its independence in 1878.
Gold answer 2

112 sentences contained the number «2» without having anything to do with the question, and the correct sentence does not contain the number, the system would have to count «Moldavia and Wallachia» as two territories if it was to get the correct answer from the sentence.

For some questions more than one sentence contained the gold answer, and could also be chosen by a human as the correct sentence:

Question When did Beijing host the Olympic Games ?
Relevant sentence 1 The city hosted the 2008 Olympic Games.
Relevant sentence 2 On 13 July 2001, the International Olympic Committee selected Beijing as the host for the 2008 Summer Olympics.
Relevant sentence 3 Beijing hosted the 2008 Summer Olympics and the 2008 Summer Paralympics.
Gold answer 2008

With this in mind, we collected the 36 question-answer pairs from the dev1 sets where the gold answer was present in more than one sentence in the document. For each question, we found the sentence we thought was most appropriate when seen in context of the question. If no sentence was appropriate (for example if the question was too hard) we removed

the pair from the set. When more than one sentence was appropriate, we chose the shortest sentence on the assumption that shorter sentences would introduce less noise than long sentences. After these steps had been performed, we were left with 29 question-answer pairs where we knew what the «most» correct sentence was, as judged by us.

Error type	Normal	Gold	Retrieved
No sentences in the document with ...	0	0	0
Question type is number, gold answer is not	0	0	0
Question type is yes/no, gold answer is not	0	0	0
No retrieved sentences contain gold answer	8	12	0
Highest ranked sentence does not ...	1	0	0
No relevant sentences were found	0	0	0
No retrieved sentence ... w/ expected NER tag	0	0	0
No retrieved sentence ... w/ expected POS tag	1	1	2
Gold answer must be chosen at random	3	2	5
Gold answer was not scored highest	3	2	3
No answer candidates were found	3	3	4
System answer is substring of gold answer	3	3	3
Answer marked as incorrect	2	1	2
Total	24	24	19

Table 6.5: Automatic error analysis of results from three runs on a set of 29 question-answer pairs. «Normal» is with no annotation, «Gold» is with the gold sentences replaced with the annotated sentences and «Retrieved» is with the retrieved sentences replaced with the annotated sentences.

The results in Table 6.5 show that annotation is indeed helpful. (The error types here are shorter versions of the full descriptions in Table 6.2, shortened so the table would fit on the page.)

For the column labelled «Gold» we replaced the set of all gold sentences with the one sentence we judged to be the correct sentence, meaning that instead of a potentially large number of sentences that could match the retrieved sentences we had only one. Not surprisingly, this caused the evaluation to show more sentences that did not match the gold sentences, because the gold sentences was only this one sentence.

In the column labelled «Retrieved» we replaced the output from the sentence retriever with the one sentence we judged to be correct, meaning that we have simulated a system run where the sentence retriever always performs perfectly.

The point of annotating the most correct sentences and comparing the output from the sentence retriever to these was to see how well the sentence retriever performs. The slight increase in errors in the sentence retriever categories shows that the sentence retriever performs quite well on these sentences, with only two additional occurrences when the size of the set of correct sentences are decreased. That it has increased at all suggests that the system, as we have already mentioned, has an inherent robustness that allows it to return correct answers even if the sentence retriever does not perform optimally.

In the second run we have substituted the retrieved sentences with the annotated sentences, meaning that the system always «retrieves» exactly one correct sentence. We first of all see that the number of overall errors have been decreased by 5, and we also see that the errors shift to later error categories. These results are not surprising, and show that annotation of the data set can indeed shift the focus of the automatic error analysis away from a certain module and onto the following modules.

6.7 Expanding categories for automatic error analysis

When we improve and add new features to our system, we can assume that we will also introduce new errors as well. Because we need to know the error categories before we can perform automatic error analysis, we would ideally like to consider what they can be before we observe the new errors.

There are several ways to handle this: we can either compare the results of automatic error analysis before and after the new feature has been added, or we can anticipate what kinds of errors the new feature will introduce and attempt to count them. Another solution is to perform a manual error analysis on a small subset of the errors that are produced with the new version of the system and attempt to use the error types found in this analysis as a basis for categories for a new automatic error analysis.

In the next chapter we will introduce some new features to our system. Before we introduce each feature we will consider what kinds of errors the feature might introduce, and how we can detect these errors automatically.

Chapter 7

Semi-automatic development cycle

In this chapter, we will discuss a second development cycle, assisted by the automated methods we introduced in the previous chapter. Again, rather than constructing a cutting-edge question answering system the point of further development is to evaluate whether the techniques introduced in the previous chapter are useful in system development.

Rather than use full data sets, the data sets used in this development cycle were reduced to minimise the number of errors we could not do anything about, and the chapter will begin with a description of how this reduction was performed.

We will then introduce some new features to our system, based on the observations we have from the error analyses from the previous two chapters, and for each new feature analyse the system with automatic error analysis as well as its impact on overall accuracy.

Figure 7.1 shows the order in which we introduced which new features to our system. Each new feature will be described in detail in its own section. The modules we add the new features to are the sentence retriever, marked with yellow in the figure, and the answer extraction module, marked with green. For a reminder on which modules are present in our question answering system, please refer to Chapter 4 or Figure 2.1.

Because we are going to introduce phrase extraction to our answer extraction module, we need to reconfigure how we detect answers that are potentially but not necessarily correct. This reconfiguration will also be discussed.

We will reflect on how the automatic error analysis helped us in introducing these new features, before we finally will show the overall improvement on our system.

7.1 Data set processing

In the previous chapter we showed that the data set contains a large number of errors caused by a lack of gold answers in the documents. We classify these questions as impossible to answer, and performing error analysis on

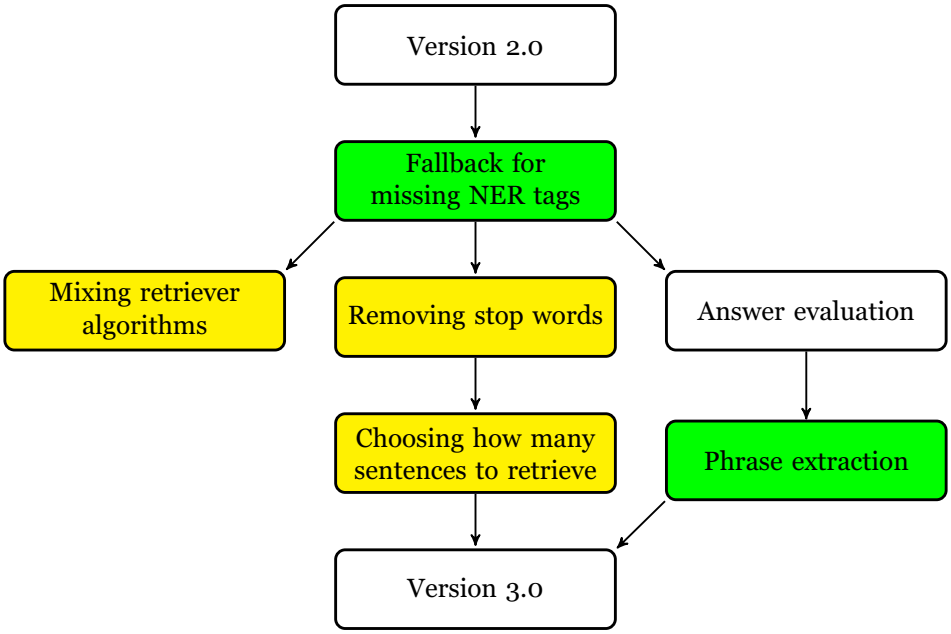

Figure 7.1: The workflow of our automatic system development cycle. Yellow boxes are new or changed features in the sentence retriever module, green boxes are new features in the answer extraction module.

the system when it attempts to answer these questions would probably not be particularly interesting. In addition, more question-answer pairs means that the system itself needs more time before a run is finished, which in turn means that our development takes longer because we have to wait for the system to finish. For these reasons, for the second development cycle we decided to remove question-answer pairs where the document did not include the gold answer.

Because we reduced the data sets, we used both the dev2 and dev3 sets from all years in the second development cycle so that the total number of question-answer pairs did not grow too small. When we only wanted to evaluate the impact a new feature had on the system performance we used the reduced data sets.

We could have gone through the data sets and manually removed pairs, but decided that since the goal was to reduce noise introduced by impossible questions, not removing them completely, removing question-answer pairs automatically would be good enough for our purposes. The removed question-answer pairs had to fit both of the following criteria:

1. The question should not be a yes/no question, as judged by the question typer in the system.
2. The suggested gold answer, exactly as written, should not be present in the provided document.

The rationale behind this reduction of our data set is that question-answer pairs that are impossible to answer for our system do not yield any

Set	Full size	Unanswerable	Reduced size
2008, dev2	91	10 (11.0%)	81 (89.0%)
2008, dev3	96	15 (15.6%)	81 (84.4%)
2009, dev2	47	7 (14.9%)	40 (85.1%)
2009, dev3	48	11 (22.9%)	37 (77.1%)
2010, dev2	79	17 (21.5%)	62 (78.5%)
2010, dev3	78	11 (14.1%)	67 (85.9%)
Total	439	71 (16.2%)	368 (83.8%)

Table 7.1: Size of dev2 and dev3 sets, for all years, before and after all unanswerable questions have been removed. Percentages are of size of the full sets.

useful information in error analysis, especially compared to all the other questions. We would rather focus our efforts on question-answer pairs that have the potential to produce useful information than pairs that do not.

Set	Full	Reduced
2008, dev2	44.0%	49.4%
2008, dev3	44.8%	51.6%
2009, dev2	44.7%	52.5%
2009, dev3	35.4%	43.2%
2010, dev2	32.9%	43.5%
2010, dev3	44.9%	55.2%
Total	41.5%	49.7%

Table 7.2: Accuracy for system version 2.0 on full and reduced data sets.

We see the reduction of the data set in Table 7.1. It is important to remember that removing impossible question-answer pairs is without a doubt going to improve the perceived accuracy of our system. We need to know the initial accuracy of the system so we have something to compare our system to when we introduce new features. For our system version 2.0, Table 7.2 shows the overall system accuracy before and after we reduced the data sets. New features have to be compared to the rightmost column in this table, not the results we obtained when we ran the system on the full sets.

It should be noted at this point that even though we have removed all the questions we judge to be impossible to answer, the corresponding category in the automatic error analysis is not reduced to 0, as we are going to see in the following sections. This is because we skipped yes/no questions as judged by the system when we reduced the data set, but for some yes/no questions the suggested answer does not contain either «yes» or «no», and is therefore assumed by the automatic error analysis to be impossible to answer. While we could have removed these questions as well, as long as we are aware of them and what properties they have, and because all the question-answer pairs have the same properties, we judged that the extra work needed was not worth it.

7.2 Falling back to nouns in case of lacking named entity tags

As we mentioned in the previous chapter, our system assumes that if the question type is «who», «where» or «when» we need to retrieve a word that is tagged with a corresponding named entity tag. As we have shown, this is not always a correct assumption, as for example in this example which we have shown before:

Question Who did Wilson win in 1917?
Gold answer Irish Americans

«Irish Americans» is not a person name and therefore will not be tagged as a person by our named entity tagger. Therefore, in the system version we had at the end of Chapter 5 – system version 2.0 – we would never be able to answer this question. For some question-answer pairs this faulty assumption caused our system to return no answer.

Therefore, the first improvement we introduced to our system was simply that the system was allowed to return a noun for «who», «where» and «when» questions if it was unable to retrieve a word with the expected named entity tag. Note that even though both this feature and phrase extraction, which we discuss later in this chapter, are features in the answer extraction module, they are not directly connected. In fact, the only question-answer pairs this feature does something for are questions that we did not retrieve any answers for in the previous system version, meaning that there is no way for this feature to lower the accuracy of the system.[1]

Because this feature was so obviously needed, could not adversely impact the system and was at the same time easy to implement, we implemented this feature before we added the other features in this chapter (refer to Figure 7.1), and use the results from the system including this feature as the baseline for further improvements.

Set	Baseline	With fallback
2008, dev2	49.4%	50.6%
2008, dev3	51.6%	51.6%
2009, dev2	52.5%	52.5%
2009, dev3	43.2%	43.2%
2010, dev2	43.5%	43.5%
2010, dev3	55.2%	55.2%
Total	49.7%	49.9%

Table 7.3: Total accuracy of our system, before and after fallback to nouns have been added.

We must mention at this point that for this and the following sections, the total count of errors differ slightly between the table showing accuracy

[1]This statement holds true because of the way we evaluate our answers – as we discussed in Chapter 2, for some systems no answer is considered a better outcome than an incorrect answer, but this is not the case for us.

and the table showing the results of the automatic error analysis. The table showing overall accuracy has been judged semi-manually by us after the criteria in Section 5.5.7. The table showing the results of the automatic error analysis shows all possible errors found, meaning that some errors in the table for automatic error analysis are not present in the table for overall accuracy. This has been done to keep the tables for accuracy comparable to previous, similar tables.

As expected, the impact on overall accuracy is minimal but positive, as we can see in Table 7.3. Perhaps more interesting is to see how many times this feature improved the output from the answer extraction module. This can be seen in Table 7.4, which shows the results from the automatic error analysis on the system after this feature had been added, and we can see that the category «No answer candidates were found» has been reduced slightly, pushing the errors down to the other subcategories of the answer extraction module.

Error type	Baseline	Fallback
No sentences in the document with ...	10	10
Question type is number, gold answer is not	6	6
Question type is yes/no, gold answer is not	7	7
No retrieved sentences contain gold answer	43	43
Highest ranked sentence does not ...	19	19
No relevant sentences were found	0	0
No retrieved sentence ... with expected NER tag	0	0
No retrieved sentence ... with expected POS tag	10	10
Gold answer must be chosen at random	13	13
Gold answer was not scored highest	16	16
No answer candidates were found	13	7
System answer is substring of gold answer	19	20
Gold answer is substring of system answer	2	2
Answer marked as incorrect	48	52
Total	206	205

Table 7.4: Automatic error analysis of system with and without fallback to nouns.

Note that in this automatic error analysis we have introduced a new error category, «Gold answer is substring of system answer», that was not present in the previous chapter. This is in preparation of a feature we are going to discuss in Section 7.4.2, when we find phrases instead of just single words. Until then, this category is going to contain a few instances of «why» questions, where we retrieve full sentences instead of words.

7.3 Improvements in sentence retriever

From the analysis in the previous chapter, we know that one of the modules that cause the largest number of errors is the sentence retriever. Therefore,

we decided to implement some additional features in order to improve on the performance of this module.

The features we considered were removal of stop words, combination of retrieval methods and finally a better approximation of how many sentences we want the sentence retriever to return, in order to minimise the noise we introduce for the answer extraction module. First, we will discuss some new evaluation methods for the sentence retriever.

7.3.1 Evaluation of sentence retriever

With the introduction of gold sentences – the set of sentences that contain the gold answer to a question – and automatic error analysis in the previous chapter, we have more than one way of evaluating our sentence retriever.

The metric we have used so far is how much impact a change has on the total number of correct answers. We can introduce two additional possible metrics:

1. We can see how many errors are detected to originate in the sentence retriever.

2. We can see for each question-answer pair how many sentences we have to retrieve in order to retrieve a gold sentence.

The first new metric is based on the automatic error analysis. It simply counts how many errors are detected in the error categories «No retrieved sentences contain gold answer», «Highest ranked sentence does not contain gold answer» and «No relevant sentences were found». These are the errors that are found to be directly caused by the sentence retriever.

This evaluation metric has the weakness that, because we only report the first error that occurs, it will not be able to detect problems in the sentence retriever if the question-answer pair we analyse has already been marked with an error.

The second metric is perhaps more accurate. What it tells us is how many sentences we need to retrieve in order to theoretically be able to answer the question. We can then collect these values and find how many sentences we have to retrieve to be theoretically able to answer a certain percentage of our questions. For example, if we say that we have to be able to find the answer to all questions, we need to take the worst case – the question where we have to retrieve the highest number of sentences before we find one that contains the gold answer – and apply that number to all the questions. This metric will be explained further and used in Section 7.3.4.

«Perfect» performance by the sentence retriever in this case is when the first sentence the sentence retriever gets is in the gold sentences, meaning that we only have to return one sentence.

7.3.2 Removing stop words

In Section 4.2 we mentioned that we did not remove stop words from the normalised sentences, because we feared that this would remove important

information from the sentences. However, because we at this point know that the sentence retriever is one of the largest contributors to the errors in our system, it is necessary to actually evaluate the effects of removal of stop words.

Rather than creating our own list of stop words we chose the list provided by the NLTK package. This list, compiled by Martin Porter in connection with his work on the Porter stemmer (Porter, 1980) includes 127 stemmed words, and can be seen in Appendix C.

Stop words were removed from the stemmed versions of the sentences, meaning that the only system module which should be directly affected by this change is the sentence retriever. For example, this change should not reduce the number of potential answer candidates directly, since the answer extraction works on unstemmed sentences. Of course, it is always possible that the performance of other modules are affected as a result of the new output from the sentence retriever.

Set	Baseline	Without stop words
2008, dev2	50.6%	50.6%
2008, dev3	51.6%	49.4%
2009, dev2	52.5%	47.5%
2009, dev3	43.2%	48.6%
2010, dev2	43.5%	43.5%
2010, dev3	55.2%	56.7%
Total	49.9%	49.7%

Table 7.5: Overall accuracy, before and after stop words have been removed.

We see the impact on overall accuracy in Table 7.5. Somewhat surprisingly, we can not immediately conclude from this table that removing stop words have a positive impact on our system. Luckily, we have two further evaluation metrics with which we can evaluate the performance. Let us look at the automatic error analysis of the system performance.

Table 7.6 shows the results of the automatic error analysis. This analysis shows that introducing removal of stop words has been beneficial for our sentence retriever, even though the system as a whole performs about as well as it has done before. The error category «No retrieved sentences contain gold answer» has been reduced significantly, but the errors have mostly just been moved to later parts in the system pipeline, especially to the other categories connected to the sentence retriever («Highest ranked sentence does not contain gold answer» and «No relevant sentences were found»).

This represents an improvement within the sentence retriever, because where we earlier had 43 cases where we had no chance of finding the correct answer («No retrieved sentences contain gold answer»), we now have 31. That the highest sentence does not contain the gold answer is a much smaller problem, and in many cases does not mean that the system is unable to find the answer.

Error type	Baseline	No stop words
No sentences in the document with ...	10	10
Question type is number, gold answer is not	6	6
Question type is yes/no, gold answer is not	7	7
No retrieved sentences contain gold answer	43	31
Highest ranked sentence does not ...	19	26
No relevant sentences were found	0	2
No retrieved sentence ... w/ expected NER tag	0	0
No retrieved sentence ... w/ expected POS tag	10	12
Gold answer must be chosen at random	13	15
Gold answer was not scored highest	16	14
No answer candidates were found	7	6
System answer is substring of gold answer	20	21
Gold answer is substring of system answer	2	1
Answer marked as incorrect	52	57
Total	205	208

Table 7.6: Automatic error analysis, before and after stop words have been removed.

7.3.3 Combining sentence retrieval methods

So far, only one sentence retriever algorithm has been used for retrieving each sentence. In some cases, if one sentence retriever algorithm is unable to produce a satisfactory output we can fall back to a more sturdy algorithm, but in these cases the output from the first sentence retriever is completely discarded (see Section 4.6 for the details).

Our n-gram based sentence retriever has been shown to improve the overall accuracy of our system, but has the weakness that for some questions, it will return many sentences with the same score, simply because they all have the same number and length of n-grams present in the retrieved sentences. For example, the smallest n-gram we expect to be meaningful is a 3-gram. For some questions, n-gram sentence retrieval finds many sentences with a 3-gram, and which subset of these sentences is retrieved is mostly random (recall that we ultimately retrieve up to a set number of sentences, not all sentences that match).

One solution to making the decision for which sentences to return in n-gram sentence retrieval less random is to give the sentences to one of the other retrieval algorithms and ranking the sentences by that measure as well. That way, the sentences are ranked instead of in a random order before we cut off the excess sentences and return what is left.

For example, if we have a document containing 300 sentences and the n-gram sentence retriever finds 50 sentences containing a 3-gram matching the question, the system at this point will return 5 of these 50 sentences at random. The feature we are about to implement will allow the system to give these 50 sentences as the input to one of the other retrieval algorithms, and allow the other retrieval algorithm to choose which 5 sentences to return, instead of picking 5 at random.

We have assumed that this combination of retrieval algorithms – first retrieving sentences with n-gram retrieval and then sorting them with another algorithm – is the ideal way of combining methods. This is mostly founded on our analysis in Section 5.4.3. Another possibility is to imagine that our primary algorithm, n-gram retrieval, performs badly, or at least significantly worse than another algorithm, on a certain question type and have the other retrieval algorithm handle all the questions of this type. Detecting this question type (or another property of each question) could then be treated as a machine learning problem, and determining which combination of sentence retrieval algorithms to use for a question would depend on which properties that question has.

While using machine learning to find how to combine algorithms could certainly be interesting, we found that we were limited by time and resources. Implementing the machine learning algorithm could potentially turn out to take a long time, and we judged that it would be outside the scope of this thesis.

As in the previous section we will evaluate the impact of this feature by overall accuracy and the output of automatic error analysis.

Set	Baseline	Mixed algorithms
2008, dev2	50.6%	48.1%
2008, dev3	51.6%	50.6%
2009, dev2	52.5%	50.0%
2009, dev3	43.2%	40.5%
2010, dev2	43.5%	45.2%
2010, dev3	55.2%	58.2%
Total	49.9%	49.1%%

Table 7.7: Overall accuracy, before and after mixed algorithms have been implemented.

The overall accuracy is seen in Table 7.7. For the 2008 and 2009 sets, the accuracy is reduced, while the accuracy on the 2010 sets has been improved. The overall accuracy has also been reduced.

The automatic error analysis shows more or less the same thing. The distribution of errors have shifted slightly to later in the system pipeline, but the difference is small. There is some evidence that the ranking of sentences has changed, since there is a difference in how the answer extraction module has retrieved answers, but other than that the introduction of this feature has apparently not changed the performance of our system in a major way.

The reduction in overall accuracy made us evaluate the addition of this feature to be detrimental to our system, and it is not part of the final system version.

Error type	Baseline	Mixed
No sentences in the document with ...	10	10
Question type is number, gold answer is not	6	6
Question type is yes/no, gold answer is not	7	7
No retrieved sentences contain gold answer	43	40
Highest ranked sentence does not ...	19	20
No relevant sentences were found	0	0
No retrieved sentence ... w/ expected NER tag	0	0
No retrieved sentence ... w/ expected POS tag	10	10
Gold answer must be chosen at random	13	14
Gold answer was not scored highest	16	16
No answer candidates were found	7	7
System answer is substring of gold answer	20	22
Gold answer is substring of system answer	2	2
Answer marked as incorrect	52	55
Total	205	209

Table 7.8: Automatic error analysis, before and after mixed methods have been implemented.

7.3.4 Number of relevant sentences

As we mentioned in the beginning of this section, we have a new way of evaluating the performance of sentence retrieval, which is directly connected to how many sentences we should retrieve.

In Section 5.4.2 we evaluated the impact of retrieving a different number of sentences on the overall accuracy. With the introduction of gold sentences in the previous chapter, we now can evaluate how many sentences we have to retrieve in order to retrieve at least one gold sentence. If we retrieve fewer than that many sentences, we will not be able to answer the question, and if we retrieve more sentences we increase the probability that we introduce noise to our system. Thus, the optimal number of sentences to retrieve is in most cases just enough to retrieve the first gold sentence, and no more.

When we look at Tables 7.9 and 7.10, we should keep in mind that the differences in the system features mean that a slightly different number of total sentences are retrieved. When we remove stop words the accuracy improves, but we also retrieve a slightly smaller number of sentences overall. However, the sentences that we do not retrieve are sentences that are would not help the system, being for example ranked as sentence number 130 and therefore scored too low to impact the choice of answer candidate.

Table 7.9 shows how certain we can be that we retrieved at least one sentence containing the gold answer when we retrieve 1 to 5 sentences. Recall that the point is not by itself to increase the percentage to the highest possible number – if that was what we wanted we would just retrieve all sentences – but rather to to find the point where we retrieve just enough sentences that we are able to answer most questions, which is a balance

Method	Set	1	2	3	4	5
Baseline	2008, dev2	50.0%	71.4%	71.4%	78.6%	82.1%
	2008, dev3	57.7%	73.1%	96.1%	96.1%	96.1%
	2009, dev2	66.7%	91.7%	91.7%	91.7%	91.7%
	2009, dev3	76.9%	84.6%	100.0%	100.0%	100.0%
	2010, dev2	44.4%	59.3%	66.7%	70.4%	70.4%
	2010, dev3	81.8%	90.9%	95.5%	95.5%	95.5%
	Overall	60.2%	77.2%	88.2%	90.2%	91.2%
Stop words	2008, dev2	68.0%	84.0%	88.0%	92.0%	96.0%
	2008, dev3	68.0%	80.0%	96.0%	100.0%	100.0%
	2009, dev2	81.8%	90.9%	90.9%	90.9%	90.9%
	2009, dev3	61.5%	92.3%	92.3%	92.3%	100.0%
	2010, dev2	44.4%	66.7%	85.2%	92.6%	92.6%
	2010, dev3	69.6%	73.9%	91.3%	91.3%	95.7%
	Overall	63.7%	79.0%	91.2%	95.2%	97.2%

Table 7.9: Percentage of sentences that include a gold sentence, for 1 to 5 retrieved sentences, with different features added to sentence retriever.

between retrieving enough sentences (increasing the percentages in Table 7.9) and not retrieving too many, because that would introduce noise for our answer extraction.

Table 7.10 shows, for the set of sentences that contain at least one sentence with a gold answer, the mean position of the highest ranked sentence containing the gold answer. For this table we removed all question-answer pairs where the highest gold sentence was ranked higher than 10 (with 1 being best), on the assumption that if we retrieve more than 10 sentences we would not be able to retrieve the answer from the set of retrieved sentences. This assumption is partially supported by our analysis in Section 5.4.2.

Because the mean position for the first gold sentence is between 1 and 2 for all sets of question-answer pairs, we choose 2 as the number of sentences to be retrieved in the final system version. This also means that we should be able to potentially answer at least 2/3 of all questions with question type other than yes/no.

By all metrics, removing stop words improves accuracy in sentence retrieval. We did find that we lost information, and that we were able to return sentences for slightly fewer question-answer pairs when stop words were removed, but we also found that we in those cases without exception were unable to rank the correct sentence high enough to impact our choice of answer candidate.

7.4 Improvements in answer extraction

The other module into which we introduced new features was the answer extraction module. The first feature, falling back to nouns when we miss an answer candidate with an expected named entity tag, has already been

Method	Set	Mean
Baseline	2008, dev2	1.69
	2008, dev3	1.84
	2009, dev2	1.27
	2009, dev3	1.38
	2010, dev2	1.85
	2010, dev3	1.19
	Overall	1.59
Stop words	2008, dev2	1.54
	2008, dev3	1.56
	2009, dev2	1.10
	2009, dev3	1.62
	2010, dev2	1.88
	2010, dev3	1.59
	Overall	1.60

Table 7.10: Mean position of first gold sentence in retrieved sentences, ignoring positions greater than 10.

discussed in Section 7.2. The other feature is to retrieve longer phrases, in the sense of «more than one word», instead of single words. In preparation of this feature, though, we need to have another look at how we find potentially correct answers. As we mentioned in Section 6.7, we also needed to introduce a new error category, which we already did in Section 7.2.

7.4.1 Evaluation

In the previous versions of the system described in this chapter we have used the method for finding potentially correct answers as described in Section 5.5.7. Initial attempts at introducing phrase extraction to our system showed that this method was not as well suited any longer, because retrieving more words means that while recall can go up, precision is in many cases going to fall significantly.

The previous method required recall to be at least 0.5 or precision to be exactly 1. In practise, this meant that if the word we retrieved was present in the gold answer the system answer could potentially be correct. When we attempted to introduce phrase extraction, however, we found that the criteria needed to change.

In the next section and the final system version, which we are going to call version 3.0, the criterion for being manually evaluated is going to be that the system answer must at least partially overlap with the gold answer. This means a slight increase in the number of answers we have to evaluate manually, while automatically evaluating all answers that are either definitely correct or definitely incorrect.

7.4.2 Phrase extraction

Our phrase extraction method builds upon the previous answer extraction method. The previous answer extraction module retrieved a single word that was deemed to be the most likely answer candidate.

In our phrase extraction method, we take the retrieved sentences from the sentence retriever and the word we found in answer extraction. We then find the most likely sentence that also contained that word. From that sentence, we find the longest phrase (as measured by number of words) that has that word as its head. We chose to return the longest phrase because we care less about returning short answers than we do about returning correct answers. Our system is already unable to return answers that are longer than one sentence.

Let us consider an example to show how the method works:

Question What is a person that makes or repairs violins called ?
Relevant sentence A person who makes or repairs violins is called a luthier, or simply a violin maker.

The parse tree for the relevant sentence can be seen in Figure 7.2.

The answer extraction module finds that the word «luthier», marked with a green box in Figure 7.2, is the most likely answer candidate. This word is the head of several phrases, the longest of which is marked with a yellow box (the other phrases are subphrases of this phrase). The longest phrase is the phrase we return, meaning that where the previous answer extraction module would return only the word in the green box, we now return the whole phrase in the yellow box.

For some questions the answer was previously correct, but could be improved on. One example is when we are asked for a person and the system is able to find the correct name, but could previously return only one of the names:

Question Who wrote the first novel in Finnish?
One word Aleksis
Phrase Aleksis Kivi

For many questions the answer was previously incorrect or not accurate enough, but with the introduction of phrase retrieval the answer was expanded enough to be judged as correct:

Question James Watt's improvements of what were fundamental to the changes wrought by the Industrial Revolution?
One word engine
Phrase The steam engine

In this example the single word «engine» is part of the correct answer, but not precise enough to be judged as correct. Finally, we had some questions where the answer was either expanded so that it became too long to be precise enough, or simply added information that made the answer ungrammatical or otherwise impossible to understand:

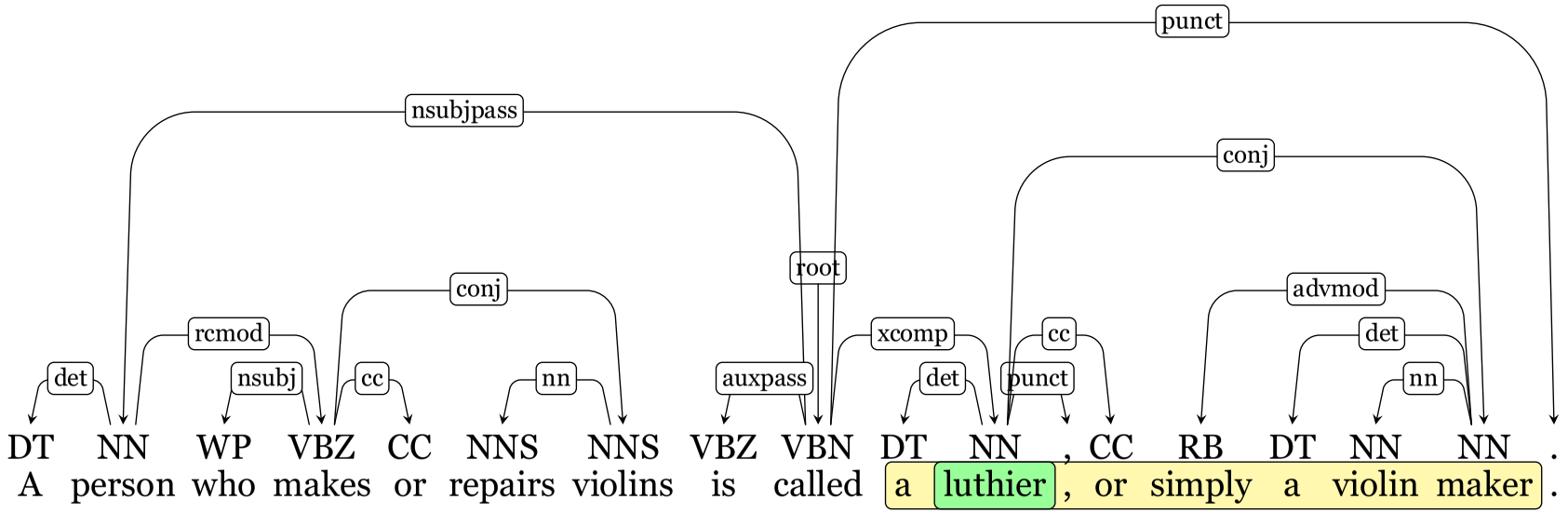

Figure 7.2: A person who makes or repairs violins is called a luthier , or simply a violin maker .

Question Who died at his house at Hampton Court on August 25, 1867?
One word Faraday
Phrase accessed June 2006 Faraday

Set	Baseline	With phrase extraction
2008, dev2	50.6%	48.1%
2008, dev3	51.6%	51.9%
2009, dev2	52.5%	52.5%
2009, dev3	43.2%	40.5%
2010, dev2	43.5%	46.8%
2010, dev3	55.2%	59.7%
Overall	49.9%	50.5%

Table 7.11: Overall accuracy, with and without phrase extraction.

As we can see from Table 7.11, phrase extraction does improve the overall accuracy of our system. It is interesting to note that accuracy for some sets decreases slightly, but the increase in the accuracy for the 2010 sets is large enough to positively impact the overall accuracy, showing that the introduction of phrase extraction is positive for our system.

Error type	Normal	
No sentences in the document with ...	10	10
Question type is number, gold answer is not	6	6
Question type is yes/no, gold answer is not	7	7
No retrieved sentences contain gold answer	43	40
Highest ranked sentence does not ...	19	20
No relevant sentences were found	0	0
No retrieved sentence ... with expected NER tag	0	0
No retrieved sentence ... with expected POS tag	10	10
Gold answer must be chosen at random	13	13
Gold answer was not scored highest	16	18
No answer candidates were found	7	7
System answer is substring of gold answer	20	16
Gold answer is substring of system answer	2	8
Answer marked as incorrect	52	58
Total	205	213

Table 7.12: Automatic error analysis of system, with and without phrase extraction.

The results from the automatic error analysis, in Table 7.12, show what we would expect: The category for gold answers that are substrings of system answers grown larger. The error category for when system answers are substrings of the gold answer has decreased in size and some errors have moved to «answer marked as incorrect», likely because the one word that was a substring has been expanded to a phrase that does not exactly match the gold answer.

We conclude that the addition of phrase extraction is useful. Though the overall number of errors that were found by the automatic error analysis has been increased, the increase in accuracy is good.

7.5 Impact of automatic error analysis on development cycle

Although we have no way of quantifying exactly the effects of automatic error analysis on the cost of system development and refinement, we definitely felt that the addition of automatic error analysis helped us evaluate and develop the system. The development cycle that was assisted by automatic error analysis was faster both in terms of how much time we had to spend developing, because development could target isolated, known problems in the system, and in particular in terms of how quickly we could evaluate whether a new feature was beneficial to our system.

It also allowed us to quickly evaluate which modules a feature had an impact on. For example, when we introduced removal of stop words in Section 7.3.2, the overall accuracy, which was the only evaluation metric we could extract automatically in our previous development cycle, did not show that the system improved significantly. Automatic error analysis, on the other hand, showed us without a doubt that removing stop words helped the sentence retriever module, which in turn opened up new possibilities for refinements in modules later in the system.

We could naturally have found the same numbers previously with repeated manual error analysis, but manually analysing all retrieved sentences for each question to see whether the gold answer was present, for example, would take a long time, much longer than for other modules. In this regard, automatic error analysis helped us more with analysing the sentence retriever than it would have done for the question typer, for example.

7.6 A new system version

The final system version included most of the features from this chapter, but not mixing of retrieval algorithms. This means that the difference from the system version 2.0 was:

1. The system could fall back to retrieving nouns if it did not find a word with the appropriate named entity tag for «who», «where» and «when» questions.
2. Stop words were removed in the normalisation step in order to help the sentence retriever module.
3. We retrieved only 2 sentences for each question, not 5
4. The way we evaluated system answers differed slightly, but not in a way that would give different results for the system version 2.0.

For comparison we repeat the numbers from Table 7.2, which showed the accuracy of the system version 2.0, and expand with the accuracy of the system version 3.0.

Table 7.13 shows the overall accuracy. We can safely claim that this is a substantial improvement. We will compare the results of this system version with the previous system versions on a held-out test set in the next chapter, to ascertain that we have not over-fitted our system to the data set we have worked with.

Set	v2.0, full	v3.0, full	v2.0, reduced	v3.0, reduced
2008, dev2	44.0%	49.5%	49.4%	54.3%
2008, dev3	44.8%	47.9%	51.6%	53.0%
2009, dev2	44.7%	46.8%	52.5%	55.0%
2009, dev3	35.4%	43.8%	43.2%	51.4%
2010, dev2	32.9%	39.2%	43.5%	45.1%
2010, dev3	44.9%	52.6%	55.2%	58.2%
Total	41.5%	46.9%	49.7%	53.0%

Table 7.13: Comparison of accuracy from system version 2.0 and 3.0, for reduced and full data sets.

Chapter 8

Conclusion

In this chapter, we will summarise and conclude on the results we have found in this thesis. These results consist of findings on the data set we used, which we hope can be used in other studies as well, the system results from our system as a way of showing that the system development was successful, and of a reflection on how useful the automation of the error analysis was.

We will end the chapter with some suggestions on fields that can provide interesting opportunities for future work.

8.1 Data set

We have examined a data set that has, to the best of our knowledge, not been used in other published academic studies. We have analysed the set both statistically and in detail. Our analysis of this set shows that the question set is fairly diverse in terms of different topics and difficulty.

The data set was fairly difficult for our simple question answering system. On the other hand, compared to many other data sets, some of which we discussed in Chapter 2, the answers were mainly actually quite easy. For the most part we are guaranteed to be able to find the suggested answer to the question in the provided document, and generally speaking a large proportion of the answers can be found without deep natural language processing or reasoning.

We believe that the data set is well suited for new and experimental question answering systems, such as ours. The relatively large number of yes/no questions lend themselves well to a system that could focus specifically on detecting negation in sentences or lack of evidence for a fact.

On the other hand, the data set does contain question-answer pairs that are problematic, either because they are not grammatically well-formed or because they are sometimes impossible to answer given the data in the document. We have outlined our methods for improving the quality of the data set, mainly by removing duplicate questions, but note that a more thorough refinement of the data set would without a doubt improve the quality further. We note the data set as a potential field of further research in Section 8.4.

8.2 System results

We have so far withheld a test set from our data set. In order to fairly evaluate whether our system actually has improved between versions, and that we have not simply trained our system to perform well on the specific portions of the data set that we have used in system development, we present the results from the three different system versions as well as from the majority baseline from Section 4.3 in Table 8.1 and the upper bound from Table 3.4.

Set	Majority	1.0	2.0	3.0	Upper bound
2008	23.3%	18.9%	33.3%	43.3%	78.0%
2009	37.3%	41.1%	37.3%	43.1%	80.4%
2010	30.0%	15.0%	37.5%	46.3%	68.0%
Overall	29.0%	22.6%	35.7%	44.3%	74.9%

Table 8.1: Comparison of the overall accuracy of the different system versions on the test set.

We can definitely see that the system improved as a result of our work. Much of that work consisted of seeing what the system does and does not do correctly, which is the main portion of the error analysis.

The 2009 set turned out to be somewhat atypical for our data set, with a larger portion (over 50%) of the questions being yes/no questions, which explains the comparatively small improvements on this set.

We see an increase from the majority baseline of almost 15% overall, or almost one and a half times as high accuracy. For the 2008 set we almost double the overall accuracy of our system compared with the majority baseline. The fact that the majority baseline is so high means that it was quite difficult to improve on, and we are happy with these results.

The overall accuracy also improves between the different system versions, with about a doubling in accuracy between version 1.0 and 2.0, not counting the 2009 set. Similarly, an increase in about 10% between version 2.0 and 3.0 means that the second development cycle was also beneficial to our system.

As we have already stated, the goal of developing the question answering system was not to attempt to meet the upper bound, but rather to improve on the system and gain a further understanding on how development and error analysis is performed. As such, we are not disappointed in the results compared to the upper bound.

The fact that the improvement is smaller was simply caused by the inevitable effect that the most obvious errors had been removed and the most obviously beneficial features added between the first and second system versions. At the same time, the development time needed was shorter due to the error analysis, but we also felt that we could allocate our time more to actual development and less to going through the output of the different modules to find different errors. As such, we can conclude that automating error analysis is beneficial and that improving methods used in

error analysis is going to be a useful contribution to question answering systems in general.

8.3 Error analysis

The methods used in error analysis have been discussed at length in Chapter 6. In automatic error analysis, we need to know what kinds of errors – the error categories – to consider before we start the analysis. We also need to know some properties of the expected output of each module, so we have a way of detecting errors. This means that we usually need to perform at least one manual error analysis first if we want the categories to be precise. Of course, if we only want an overview of which modules produce an error we can simply make assumptions about the properties of the output of each module and compare the system output to what we expected. In these cases the error categories are simply the same as the modules.

One problem with the automatic error analysis we have discussed is that because all the expected error categories need to be found before the analysis starts, the groundwork needs to be laid by by performing manual error analysis first, which constrains the usefulness of automatic error analysis if repeated testing is not needed.

We have also discussed annotation of the data set and expected output of each module. This annotation can take a long time for some modules, but improve the accuracy with which we can evaluate the output of that module. We can also use annotated output to run simulated versions of the system to see whether further errors occur later in the system pipeline.

We expect that the vast majority of systems go through several cycles of development. For these systems, we conjecture that automating error analysis without a doubt saves time for developers in the long run. How much time is spent developing and refining the methods for error analysis should in most cases be offset by only a single run with automatic error analysis, and all further test runs with the same error categories have a trivial time requirement for error analysis, because the time spent is spent analysing the results of the error analysis instead of on the analysis itself.

In our experience, automating the error analysis process was tremendously helpful in system development. A further refinement of the categories and how to find errors in each category would probably help development even more.

8.4 Future work

The data set used in this thesis has been analysed to a certain extent by us, but we definitely feel that it could be refined further. Many questions appear multiple times, sometimes with different answers. Our solution to this problem was to remove all but the first occurrence of each question, without regard for whether that question-answer pair was the best out

of the set. Spending more time on the refinement of the data set would probably help remove the question-answer pairs where the suggested answer is not neither supported by nor present in the document, as is the case for some answers.

As regards the error analysis part of the thesis, we feel that an interesting and natural next step could be to combine automatic and manual error analysis further. We could manually analyse the system performance on question-answer pairs that cause a certain type of error detected by automatic error analysis in an attempt to gain a deeper understanding of what caused the error. This could in turn be used to further improve the categories and methods for automatic error analysis.

For example, we found a quite large category of errors called «Answer marked as incorrect» that would probably be useful to take a closer look at. The less specific the category in the automatic error analysis is, the greater the potential for the category to be large, and the probability of manually finding further subcategories is going to rise.

Part IV

Appendices

Appendix A

Detailed tables of categories of questions

This appendix contains a full list of all categories in the data set we used in this thesis, as well as their size. Each category name corresponds to the name of a WikiPedia article from which the question-answer pair was generated. The data set itself is discussed in Chapter 3.

Capitalisation and underscores in category names have been preserved.

Category	Number of questions	% of full size
Canada	34	3.7%
turtle	38	4.1%
Qatar	25	2.7%
beetle	30	3.4%
Grover_Cleveland	25	2.7%
polar_bear	31	3.5%
duck	26	2.8%
Theodore_Roosevelt	18	2.1%
Anders_Celsius	16	1.7%
Ghana	17	1.8%
Ulysses_S._Grant	23	2.5%
Singapore	38	4.1%
elephant	27	2.9%
otter	38	4.1%
Indonesia	24	2.6%
Calvin_Coolidge	27	2.9%
John_Adams	29	3.2%
Gray_Wolf	18	2.0%
kangaroo	37	4.0%
Finland	27	2.9%
Nikola_Tesla	3	0.3%
Training	**551**	**60.0%**
leopard	32	3.5%
Romania	29	3.2%
Woodrow_Wilson	29	3.2%
Development 1	**90**	**9.9%**
James_Watt	7	0.8%
Abraham_Lincoln	27	2.9%
Gerald_Ford	28	3.0%
Millard_Fillmore	29	3.2%
Development 2	**91**	**9.9%**
Uruguay	37	4.0%
penguin	32	3.5%
Henri_Becquerel	13	1.4%
Amedeo_Avogadro	14	1.5%
Development 3	**96**	**10.4%**
Egypt	18	2.0%
Liechtenstein	34	3.7%
James_Monroe	38	4.1%
Test	**90**	**9.8%**
Total	**918**	**100.0%**

Table A.1: Categories and number of questions in 2008 data set.

Category	Number	% of full size
Alessandro_Volta	27	5.5%
Swan	18	3.6%
Fox	9	1.8%
Xylophone	9	1.8%
Charles-Augustin_de_Coulomb	28	5.7%
Anders_Celsius	19	3.8%
Tiger	9	1.8%
Santiago	12	2.4%
Swahili_language	4	0.8%
Flute	9	1.8%
Japanese_language	3	0.6%
Cymbal	17	3.4%
Giraffe	9	1.8%
French_language	10	2.0%
Nassau	8	1.6%
Otter	9	1.8%
Dhaka	8	1.6%
Isaac_Newton	9	1.8%
Ottawa	19	3.8%
Bee	9	1.8%
Nikola_Tesla	9	1.8%
Cello	18	3.6%
Chinese_language	5	1.0%
Amedeo_Avogadro	27	5.4%
Training	**304**	**61.0%**
Turtle	17	3.4%
Beijing	11	2.2%
German_language	13	2.6%
James_Watt	8	1.6%
Development 1	**49**	**9.8%**
Blaise_Pascal	9	1.8%
Henri_Becquerel	26	5.2%
English_language	12	2.4%
Development 2	**47**	**9.4%**
Arabic_language	8	1.6%
London	8	1.6%
Violin	9	1.8%
Copenhagen	8	1.6%
Lima	15	3.0%
Development 3	**48**	**9.6%**
Trumpet	9	1.8%
Lyre	7	1.4%
Michael_Faraday	34	6.8%
Test	**50**	**10.0%**
Total	**498**	**100.0%**

Table A.2: Categories and number of questions in 2009 data set.

Table A.3: Categories and number of questions in 2010 data set.

Category	**Number**	**%**
Vincent_van_Gogh	7	1.0%
Alessandro_Volta	27	3.4%
San_Francisco	27	3.4%
Jackson_Pollock	9	1.1%
Antwerp	9	1.1%
Lyre	9	1.1%
Cougar	9	1.1%
Koala	8	1.0%
Vietnamese_language	17	2.1%
Charles-Augustin_de_Coulomb	20	2.5%
James_Watt	9	1.1%
Piano	17	2.1%
Butterfly	9	1.1%
Guitar	18	2.8%
Michelangelo	7	0.9%
Swahili_language	9	1.1%
Saint_Petersburg	17	2.1%
Flute	17	2.1%
Cymbal	9	1.1%
Giant_Panda	18	2.3%
Berlin	18	2.3%
Taipei	9	1.1%
Jakarta	26	3.3%
Zebra	9	1.1%
Drum	23	2.9%
Dragonfly	3	0.4%
Isaac_Newton	17	2.1%
Malay_language	24	3.0%
Melbourne	9	1.1%
Eel	9	1.1%
Korean_language	23	2.9%
Chinese_language	18	2.3%
Portuguese_language	15	1.9%
Training	**475**	**60.0%**
Norman_Rockwell	10	1.3%
Pablo_Picasso	10	1.3%
Xylophone	9	1.1%
Swedish_language	8	1.0%
Octopus	16	2.0%
Nairobi	17	2.1%
Finnish_language	9	1.1%
Development 1	**79**	**9.9%**
Montreal	9	1.1%
Turkish_language	17	2.1%

TableA.3– *Continued on next page*

TableA.3– *Continued from previous page*

Category	**Number**	**% of full size**
Leonardo_da_Vinci	10	1.3%
Blaise_Pascal	16	2.0%
Kuala_Lumpur	18	2.3%
Henri_Becquerel	9	1.1%
Development 2	**79**	**9.9%**
Ant	26	3.3%
Arabic_language	16	2.0%
Lobster	9	1.1%
Pierre-Auguste_Renoir	8	1.0%
Violin	19	2.4%
Development 3	**78**	**9.8%**
Nikola_Tesla	18	2.3%
Trumpet	27	3.4%
Amedeo_Avogadro	18	2.3%
Michael_Faraday	17	2.1%
Test	**80**	**10.1%**
Total	**791**	**100.0%**

Appendix B

Question typer rules and categories

```
import re

def question_type(qtext):
   """Accept a question as qtext, return an answer type for
       this question"""
   qtext = " ".join(qtext)

   if
       re.search(r'\b\A(can|could|will|would|have|has|do|does|'+\
           'did|is|are|was|may|might)\s', qtext, re.I):
      return "YES/NO"
   elif re.search(r'\b\A(what|which)\s+(\w+)', qtext, re.I):
      nextword = re.search(r'\b\A(what|which)\s+(\w+)', qtext,
          re.I).group(2)
      if nextword == "year" or nextword == "date" or nextword
          == "day":
         return "WHEN"
      else:
         return "WHAT"
   elif re.search(r'\bwho\s', qtext, re.I):
      return "WHO"
   elif re.search(r'\bwhy\s', qtext, re.I):
      return "WHY"
   elif re.search(r'\bwhere\s', qtext, re.I):
      return "WHERE"
   elif re.search(r'\bhow\s', qtext, re.I):
      nextword = re.search(r'\b(how)\s(\w+)', qtext,
          re.I).group(2)
      if nextword == "many" or nextword == "much" or nextword
          == "long":
         return "NUMBER"
      else:
         return "HOW"
   elif re.search(r'\bwhen\s', qtext, re.I):
      return "WHEN"
   elif "in what year" in qtext or "in which year" in qtext:
      return "WHEN"
   else:
      return "WHAT"
```

Appendix C

List of stop words used in stop word removal

a, about, above, after, again, against, all, am, an, and, any, are, as, at, be, because, been, before, being, below, between, both, but, by, can, did, do, does, doing, don, down, during, each, few, for, from, further, had, has, have, having, he, her, here, hers, herself, him, himself, his, how, i, if, in, into, is, it, its, itself, just, me, more, most, my, myself, no, nor, not, now, of, off, on, once, only, or, other, our, ours, ourselves, out, over, own, s, same, she, should, so, some, such, t, than, that, the, their, theirs, them, themselves, then, there, these, they, this, those, through, to, too, under, until, up, very, was, we, were, what, when, where, which, while, who, whom, why, will, with, you, your, yours, yourself, yourselves

Bibliography

Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 2009.

Eric Breck, John D Burger, Lisa Ferro, Lynette Hirschman, David House, Marc Light, and Inderjeet Mani. How to evaluate your question answering system every day and still get real work done. *Proceedings 2nd International Conference on Language Resources and Evaluation LREC-2000*, pages 1495–1500, 2000.

Eric Brill, Susan Dumais, and Michele Banko. An analysis of the AskMSR question-answering system. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 257–264. Association for Computational Linguistics, 2002.

Davide Buscaldi, Paolo Rosso, José Manuel Gómez-Soriano, and Emilio Sanchis. Answering questions with an n-gram based passage retrieval engine. *Journal of Intelligent Information Systems*, 34(2):113–134, 2010.

Marie-Catherine De Marneffe and Christopher D Manning. Stanford typed dependencies manual. *URL http://nlp. stanford. edu/software/dependencies manual. pdf*, 2008.

Christiane Fellbaum. *WordNet*. Wiley Online Library, 1998.

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building Watson: An overview of the DeepQA project. *AI magazine*, 31(3):59–79, 2010.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

Abraham Ittycheriah, Martin Franz, Wei-Jing Zhu, Adwait Ratnaparkhi, and Richard J Mammone. Question answering using maximum entropy components. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics*

on Language technologies, pages 1–7. Association for Computational Linguistics, 2001.

Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

John Judge, Aoife Cahill, and Josef Van Genabith. Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 497–504. Association for Computational Linguistics, 2006.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, Cambridge, 2008.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.

Dan Moldovan, Marius Paşca, Sanda Harabagiu, and Mihai Surdeanu. Performance issues and error analysis in an open-domain question answering system. *ACM Transactions on Information Systems (TOIS)*, 21(2):133–154, 2003.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135, 2007.

Anselmo Penas and Alvaro Rodrigo. A simple measure to assess non-response. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1415–1424. Association for Computational Linguistics, 2011.

Martin F Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

Noah A Smith, Michael Heilman, and Rebecca Hwa. Question generation as a competitive undergraduate course project. In *Proceedings of the NSF Workshop on the Question Generation Shared Task and Evaluation Challenge*, 2008.

José Manuel Gómez Soriano, Manuel Montes y Gómez, Emilio Sanchis Arnal, and Paolo Rosso. A passage retrieval system for multilingual question answering. In *Text, Speech and Dialogue*, pages 443–450. Springer, 2005.

Karen Sparck Jones, Steve Walker, and Stephen E. Robertson. A probabilistic model of information retrieval: development and comparative experiments: Part 1. *Information Processing & Management*, 36(6):779–808, 2000.

Kristina Toutanova and Christopher D Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 63–70. Association for Computational Linguistics, 2000.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.

Ellen M Voorhees. The TREC question answering track. *Natural Language Engineering*, 7(04):361–378, 2001.

Ellen M Voorhees et al. The TREC-8 question answering track report. In *TREC*, volume 99, pages 77–82, 1999.

Bonnie Webber and Nick Webb. Question answering. In Alexander Clark, Chris Fox, and Shalom Lappin, editors, *The Handbook of Computational Linguistics and Natural Language Processing*, chapter 22, pages 630–654. Blackwell Publishing Ltd, 2010.