Question Answering

M. Tech. Dissertation

by

Deepa Paranjpe
Roll No: 02305016

under the guidance of

Prof. Soumen Chakrabarti

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai
July 13, 2004

Acknowledgement

July 13, 2004

I wish to thank **Prof. Soumen Chakrabarti** for the excellent technical advice and timely guidance that I received from him while working on the project. I would also like to thank **Prof. Pushpak Bhattacharya** for his constant encouragement and valuable feedback on the report and the system. I wish to convey my gratitute to **Ganesh Ramakrishnan**, whom I worked with during the entire project, for the numerous fruitful discussions we had. Also, a thanks to **Sujatha** for her feedback on this report and on the code.

Deepa Paranjpe
02305016

Contents

Abstract

A Question Answering (QA) system seeks to find an answer to a natural language question from a corpus of information. The current QA culture is one that involves intense human efforts towards the design and maintenance of question taxonomies, customized knowledge bases, rule sets and highly tuned ranking functions. We have designed and implemented a QA system that does not conform to this culture of doing QA. Our QA system "gets better with practice" by applying machine learning techniques to logs of past questions and their answers (QA pairs). The key technique is to recover from the question, fragments of what might have been posed as a structured query, had a suitable schema been available. The approach hinges on the idea of getting system-independent clues, that make one passage answer a question. These tasks, which are performed *offline*, involve the various learning modules that learn the method to extract the different fragments from the question from past logs of QA pairs. These clues are put to use during the *online tasks* of passage ranking and re-ranking. We get an accuracy of over 70% with the system that we have built. However, in our current setup, the ranking step is purely based on keyword search with a standard search engine such as Lucene. This leads to loss of recall during the ranking step. Our initial experiments were based on using the Text Retrieval Conference (TREC) data as the corpus. We suggest techniques that involve query expansion and transformations to improve the recall at the ranking step. We also make an attempt to use the World Wide Web (WWW) as a supplementary corpus in our experiments and find that several questions which were left unanswered with the TREC corpus, are answered in this new setup. We also suggest improvements to the re-ranking step in the form of improved NE tagging and sophisticated similarity measures suited for the QA task. The fact that we could use our approach without major changes for a different corpus with an improved ranking function and an improved re-ranking algorithm, confirms our belief of having provided a general recipe for the task of Question Answering.

Chapter 1

Introduction

A Question Answering (QA) system seeks to find a *perfect* answer to a natural language question from a collection of documents (called a corpus). The need for such systems arises since there is no easy way of quickly and reliably accessing the huge amount of *information* contained in the corpora at hand. While search engines and topic directories have made important strides in recent years, the problem of efficiently locating information is far from solved. Search engines and topic directories rely on *keyword search* for satisfying the users' information needs. However, it is important to realize that the corpus contains more information than what can be extracted using mere *syntactic* matching. QA systems extract significantly more *information* from a corpus by going beyond mere keyword search. Furthermore, literature suggests that it is more natural for humans to question a corpus using a natural language as done in Question Answering than querying it using a list of keywords, as is the case with search engines. Although the benefits of doing Question Answering on a corpus are evidently enormous, there has not been much success towards this end. Search technology has evolved but the problem of exactly locating the desired information still remains unsolved.

1.1 The Problem

The Question Answering problem can be defined as follows: Given a corpus and a question in natural language, a QA system is required to return either the answer or a snippet containing the answer where the answer is pin-pointed. A *factoid* QA system returns answer snippets for only *fact-based* questions *i.e.,* questions whose answers are only facts. Hence, a factoid QA system does not answer questions such as *Why did Sonia Gandhi decline the position of the country's prime minister ?* or *What is the meaning of life ?*

A factoid QA system is presented with a natural language question such as *What is the name of the highest mountain in Africa?* To a question such as this, it returns an answer phrase, **Kilimanjaro** or a snippet that contains the answer phrase. For the above question, a QA system should return a snippet such as ... *Mount* **Kilimanjaro***, at 19,342 feet, is Africa's highest mountain, and is 5,000 feet* ... . Most of the QA systems, however, prefer to return a ranked list of passages that are most likely to contain the answer, with the answer marked out in each of the passages. [32] conducted a user study to investigate this area and discovered that, overall, users prefer paragraph-sized chunks of text over just an exact phrase as the answer to their questions. Furthermore, users generally prefer answers embedded in context, regardless of the perceived reliability of the source documents.

1.2 Going beyond Keyword Search

Traditional IR techniques and search tasks return a ranked list of documents containing the keywords provided by the user as the *query*. The shift from keyword search to Question Answering results in a shift in focus from mere *document retrieval* to *information retrieval*. Unlike search engines which essentially do a keyword search, QA systems need to bridge the *lexical* and *semantic chasm* to be able to retrieve the correct answer phrase or the snippet containing the answer phrase. The following examples elucidate these two problems of lexical and semantic chasm.

1. Bridging the lexical chasm: Questions often contain terms that are different but related to terms in the answer. Consider the following question-answer pair,

   Question: *Who painted Olympia?*
   Answer: *Manet, who, after all, created Olympia, gets no credit.*

   This is an example displaying the problem of lexical chasm. Here, the question term "painted" is related to "created" in the answer. A simple keyword-based matching may fail to retrieve this answer. We call this vocabulary mismatch as *lexical chasm*. For successfully finding the answer to a question it is imperative to bridge the lexical chasm.

2. Bridging the semantic chasm: Successfully closing the lexical gap between question and answer terms is not sufficient for QA. The question-answer pair considered below is quite illustrative of this.

   Question: *Who killed Brutus?*
   Non-Answer: *Brutus killed Caesar*

   In the question, "Brutus" and "kill" are related by *obj* relationship, *obj* ("kill", "Brutus") i.e., "Brutus" is the *object* of "kill". On the other hand, in the non-answer, the relation is *agt* ("kill", "Brutus") i.e. "Brutus" is the *agent* of "kill". Thus, the two sentences, i.e. the question and the non-answer are semantically incoherent. To get the correct answer, we need the question and the answer to be semantically coherent. This is what we term as the semantic chasm which needs to be bridged.

The issues involved in Question Answering span a wider range of difficulties compared to those that are involved in search. The difficulty with QA systems is not just the complexity of the algorithms but extensive inter-play between the phases of processing of the unstructured corpus, the ranking algorithm and knowledge bases (lexicons, dictionaries, fact books). On the whole, if targeted for a massive and domain-independent corpus like the web, Question Answering is a very difficult problem. However, the advantages that Question Answering has over keyword-based search, motivate several research groups to venture into QA and it has thus become a widely researched area. Significant efforts are being put to develop QA systems that scale to the web and use linguistic techniques to help the systems go beyond keyword based search.

1.3 Our Aim

Our aim is to build a factoid QA system that takes as input a natural language question and returns a ranked list of snippets where the answer is pin-pointed. The corpus used is the one supplied by the *Text Retrieval Conference (TREC)* [1] for the QA track. We do not want our system to conform to the existing QA culture, where QA systems are substantial team efforts, involving the design and maintenance of question taxonomies, knowledge-bases and scoring heuristics. Current QA systems that are amongst the high-fliers at competitions such as TREC, involve intricate processing and complex algorithms, which are difficult to reproduce, to report accurate answers. These systems bank on the wisdom derived by QA experts from the corpus and the task is embedded in hand-written rules. There is no clear demarcation between knowledge and algorithms and the entire system lacks a clean, structured approach for doing the task. Our aim is to break this culture by providing a general recipe for doing QA in a *principled* manner. Our goal is to introduce a culture where QA systems are reproducible and the QA task is more structured. We wish to provide system-independent properties that make one passage better answer a question than another. We want our system to learn from already existing Question-Answer pairs (QA pairs) and thus have a system that "gets better with practice" by applying machine learning techniques to the QA pairs. A trainable QA system also eases the task of adapting to a new genre of corpus and even new languages. QA pairs can be acquired for training from a number of sources. FAQs abound on the Web; indeed, there is even a markup language, QAML3 , to represent FAQs. Search engines validate any new ranking tweak by regression tests across large query-response collections. Audit trails of paid services (like Google Answers[2]) can provide a wealth of QA-pair sources. QA systems are evaluated using a metric called the *mean reciprocal rank (MRR)*. Refer to Appendix B.1 for details about the metric. We aim to have a high value for MRR so that most of the questions get their answers in the first few snippets returned by our system.

1.4 Organization of the report

Chapter 2 describes our approach of using Machine Learning (ML)-based techniques for doing QA. Chapter 3 describes the key techniques that we use in our approach. In Chapter 4, the experimental evaluations of the system are provided. Chapter 5 provides an in-depth analysis and diagnosis of the experimental evaluations. Chapter 6 contains solutions to some problems in our system and refinements to some of our existing ideas. It also contains preliminary experiments using the refined setup. We present our conclusions in Chapter 7 [3] .

[1] `http://trec.nist.gov`

[2] `http://answers.google.com/answers`

[3] Typesetting rules : We use double-quotes for words and phrases that are picked directly from questions or corpus sentences. We use italics for questions and the first reference of key terms. We use bold fonts for words that are emphasized.

Chapter 2

Our Approach

Most of the QA systems use a general architecture for performing the Question Answering task. In this setup, the corpus is segmented into documents or passages of some suitable size. A Named Entity (NE) tagger marks people, places, organizations, etc., and the corpus with annotations is indexed by an IR system. A taxonomy of (hundreds of) question types is built by hand, and question classifiers designed to label questions with types. Based on the label, a new question is first transformed into a keyword query for the IR system. Responding passages are re-ranked using a variety of strategies, usually involving some natural language analysis. The approach that we use for doing Question Answering hinges on the goal of making QA systems reproducible and the QA task more structured. Instead of providing an **operational** treatment about *how* questions and passages are processed, *how* they are matched and scored, and *how* external knowledge bases (question taxonomy, dictionary, and thesauri) are exploited, we provide a more **declarative** treatment in order to provide system-independent properties that make one passage better answer a question than another. Our systems' tasks can be identified as those that are performed *offline* and those that are performed *online*. Here online and offline refers to whether the task is performed at *query time* or not. The offline tasks further involve two kinds of jobs. One is the *data preparation* task and the other is the training of the several learning modules using the QA pairs. On the other hand, the online tasks involve the two steps of ranking and re-ranking of the snippets most likely to contain the answer. We begin by explaining the key techniques used in our approach.

2.1 Key Techniques

The ideas of using a noisy simulation model for the Question Answering task, modeling the question as a group of fragments and learning to identify these groups from QA pairs, form the key techniques of our approach for doing QA. Each of these techniques are briefly described below and some of the sub-tasks are re-visited in Chapter 3.

2.1.1 The noisy simulation model

If all natural language texts were replaced by relational or semi-structured databases, and people asked questions in structured query languages with precise semantics, there would be no need for QA systems. Even if the corpus is structured, precise queries are hard on the searcher because s/he does not want to master the schema. In the RDBMS and XML communities, this gap has led to recent search systems where the user interacts with the system without knowing the schema. But the fact that there *is* a schema helps making some sense of a keyword query. E.g., the query `avg(salary (near physics))` submitted to a university database with a suitable schema is easily seen to ask for the average salary of people associated (closely) with the Physics department. Here `physics` is a constant in a select clause which will match the value of a column in a tuple (which we therefore call a **selector**), `salary` is the name of a column (which we therefore call an **answer type** or **atype** for short), and `avg` is an aggregator. Because the user isn't saying as much as in a query given in SQL (structured query language), some ambiguity may result; e.g., it is unclear if the query `professor (near stanford) (near cyclotron)` is supposed to bring up particle Physicists who graduated from Stanford or teach there. But this is still likely to remain much more focused than unstructured text search at large.

In QA, neither the searcher nor the database/corpus is aware of a schema. At best, there are third-party knowledge bases (such as WordNet [35] or CYC[1]) which can provide schema fragments (e.g., a bear is a kind of primate; a knob is a part of a door). Even so, we believe that query processing in a QA system is basically a **noisy simulation** of the structured counterpart. Given a suitable schema, most TREC QA questions can be cast into a `select...where...` form, where the blanks can be filled by analyzing the question. Thus, it is believed that the question is a noisy simulation of a `select...where...` query in a SQL kind-of a language. A question such as

Tokyo is the capital of which country?

gets simulated as an SQL query such as

```
select country from world(country, capital, ...)
  where capital = 'Tokyo'
```

2.1.2 Modeling the question

A question consists of words that are important for locating an answer and some words that are important for determining the *context* in which the expected answer must lie. The former consists of cue-phrases and/or other words in the question. The context is provided by question words that occur with little or no change in the answer passage. Thus, a question contains categories of words as given below.

1. Category of words that **must** be present in the answer passage and which determine the context in which the expected answer must lie, called the *selectors*
2. Category of words that provide the *type* of the answer it is looking for, called the *atype*
3. Category of words that are *attributes* of the answer phrase

For a question,

Which American party did President Roosevelt represent ?

[1] `http://www.cyc.com/`

the selectors are “President” and “Roosevelt”. The word “party” forms the atype for this question and “American” is its attribute.

We can try to fragment the question words into these categories. The idea is to recover these fragments from the question. One fragment comprises selectors: tokens that are likely to appear (almost) unchanged in an answer passage. The other fragment contains question tokens which give clues about the answer type, and are expected to be replaced in the answer passage by tokens having the desired answer type. Selectors are like *where-clauses* which fill-up the second blank in the `select...where...` query. The answer types (atypes) are like column names in relational queries. We learn from QA pairs how to recover these fragments from the question. 2.2 gives a pictorial view of the model of the question.

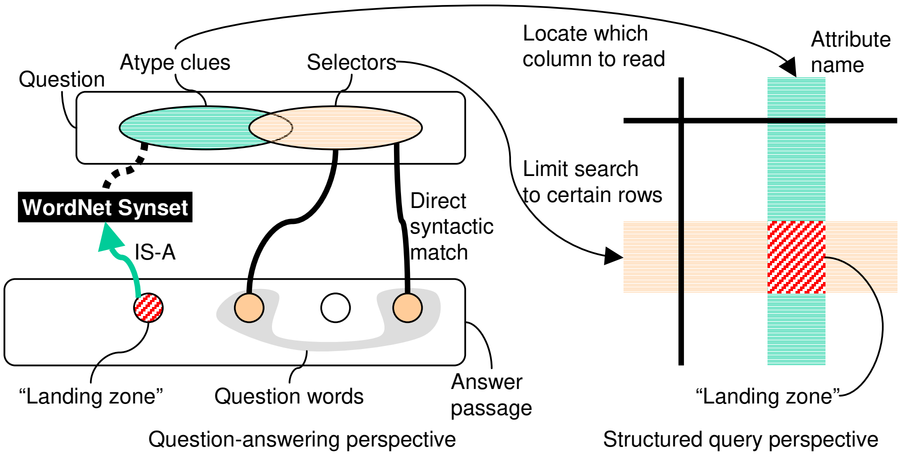

Figure 2.1: Modeling the question and correspondence with the structured query scenario.

For the question, *How many dogs pull the sledge in a Iditarod?*

the type of the answer expected (which is 11) is a quantity or measure. Thus the atype for this question is *quantity* or *measure*. “Iditarod” and “dog” are words that we expect to be present in the answer passage and hence they act as selectors.

The question *Which was the first Russian astronaut to do a spacewalk?*

seeks to find the name of an astronaut who was a Russian and who first did a spacewalk. Here, the word “astronaut” gives us the *clue* about the type of the answer to be present in the answer passage. It acts as the name of the column used in the query `select...where...`. On the other hand, “Russian” is an attribute of “astronaut”. Also, we would expect the word “Russian” to be present in the answer passage, perhaps as a variant in the form of say “Soviet”.

2.1.3 Question fragments

A question provides the clue for answer type, called the atype or the answer-type clue. The fragment of question words that provides the atype clue could be either the *wh-word* itself or some other set of words called the *focus words*. The focus words can have attributes associated with it and these attributes form the attributes of the answer in the passage. The fragment of question words which occurs with little no change are the selectors. We describe each of these fragments in the sections below.

2.1.3.1 Selectors

The second blank in `select...where...` is the "where clause" in SQL parlance which restricts the set of rows which will be considered by the query plan as shown in Figure 2.2. A relational database can guarantee a complete answer even as it prunes tuples not satisfying the where clause right off the bat. In QA (or IR) we may not be that lucky, but we can still hope to identify some words in the question which are very likely to appear in any passage which answers the question. We call these words **selectors**. Thus, selectors are the question words which occur unchanged/minimally changed in the answer phrase. E.g., any passage that answers the question

> Tokyo is the capital of which country?

is very likely to contain the word *Tokyo* in it. It becomes the selector of the question. The notion of a selector should probably be *soft*: in the question above, *capital* might also be a selector, but we are not as sure as with *Tokyo*; perhaps a passage writer might use "center of governance" with a small probability. *Country* is still softer: it is fairly likely that *nation* may be used instead.

2.1.3.2 Atype Clues

In most cases, an answer has an associated notion of a *type* such as person, animal, time, length, place, proper name, cardinal number, or money amount. In a structured query scenario, the first blank in `select...where...` would be used to specify the desired **answer type** (hereafter, **atype**) precisely, because the schema defines a universe of answer types. Several QA systems like to assume that a answer type catalog is available, in fact, if it is not, they build such catalogs with great care [19, 22] and classify each question into an atype. Excepting *list* questions like *What articles were found in the convict's handbag?*, most questions have a single atype. As per our noisy simulation framework, the atype clue in a question is a noisy specification of an attribute name in an unknown schema. The *atype clue* for a question is obtained either through the *cue phrase* of the question or the *focus word(s).* We use WordNet synsets, associated with either the cue phrase or the focus words, to represent the atype of a question. In case of focus words, it is easy to associate the synsets (the synsets will correspond to the words directly). However, questions where the cue phrase gives the atype, identifying the *representative Wordnet synsets* is not very straightforward.

2.1.3.3 Question Cue Words

Every question has what is called as the question **cue word**. Generally the head of the question, such as *what, which, who, when* and *why* , forms the question cue. However, it is not mandatory for the question cue to be present at the head of the question. It could be present anywhere in the question. Cue phrases such as *who, when, where, how_many, etc.* provide the atype clue for the question. For the question, *How many dogs pull the sledge in the Iditarod ?*

the cue phrase *how_many* provides the type of the answer that a numerical count or quantity. We term *what* and *which* as *under-constrained*. The atype for a question with these cue words cannot be obtained just from these cue words.

2.1.3.4 Focus Words

The need for **focus words** arises mainly for the questions that contain *what* or *which* as the cue words. For such questions the wh-word does not provide the necessary atype-clue. We identify the *focus* of the question and use the identified focus words to provide the atype clue. The next chapter contains a detailed treatment of focus words.

2.2 Offline Tasks

The tasks that are carried out *not* at query-time but *before* are called the **offline tasks**. There are two main tasks that are performed offline. One of them involves pre-processing of a raw corpus to make it *questionable*. It also involves extraction of training data for the various learning tasks. The other offline task involves the training of the various learning modules that recover the question fragments. These learning modules are put to use at query time *i.e.* during the online tasks, to identify the fragments in the question fired by the user.

2.2.1 Data Preparation

We used data from the TREC QA track because it offers not only a standard corpus and a set of questions, but also answer passages for each question. TREC QA made a major change in the corpus between 2001 and 2002. To study if our observations and approaches are sufficiently general, we initially focused on two years: 2000 and 2002. The data preparation steps involve tokenization, sentencification, POS tagging and NE tagging on the raw corpus provided by TREC. We use off-the-shelf available tokenizer, sentencifier, POS (part-of-speech) tagger and NE (named-entity) tagger. Nevertheless, we briefly explain each of these tasks. Recently, we have used machine learning techniques for learning to tag text with named identities.

2.2.1.1 Tokenization

Before any syntactic analysis of the corpus is performed, words must be isolated from the original stream of characters. The isolation of word-like units from a text is called **tokenization**. The results of this tokenization are two types of tokens: one type corresponding to units whose character structure is recognizable, such as punctuation, numbers, dates, etc.; the other type being units which will undergo a morphological analysis. In order for sentences to be isolated, tokens in the text should be isolated.

2.2.1.2 Sentencification

Sentences must be isolated in free-flowing text since most grammars describe sentences. **Sentencification** is a non-trivial task because even though sentences end in punctuations, most of them (specially the period) are extremely ambiguous. The use of regular expression grammars with some amount of look ahead, an abbreviation list and perhaps a list of exception rules rely on brittle parameters such as capitalization, number of spaces following a sentence delimiter etc. This can be avoided using machine learning techniques like Neural Networks and Maximum entropy approaches.

2.2.1.3 POS tagging

The sentencified corpus is then annotated with POS tags. There are generally pre-defined tags that are used. Some POS taggers use machine learning techniques for doing the job.

2.2.1.4 NE tagging

Named Entity recognition is a task in which person names, location names, organization names, monetary amounts, time and percentage expressions are recognized in a text document. This task has a major role to play in QA. Identifying entities such as time, monetary amounts and percentage is easier as the expressions for these are fairly predictable. Hence they can be processed most efficiently with finite state methods. But person names, location names and organization names are much more difficult to recognize and moreover, they are ambiguous. Also the list of persons, organizations is too numerous to include in dictionaries, changing constantly, can appear in many variant forms, and maybe abbreviated. In section 3.4, a learning setup for Named Entity tagging is presented.

2.2.1.5 Putting it together

The documents were first sentencified using the sentencifier provided by GATE [12]. We now use an *in-house* sentencifier[2] for the task. Following Tellex et al. [49], we picked sliding windows of three sentences as passages. For 2000, we got 978953 documents, 14721729 passages, and 693 questions. For 2002, we got 1031824 documents, 23266499 passages, and 500 questions. The passages were indexed by Lucene [4]. For each question, TREC provides a set of document identifiers which answer it, a regular expression which the participant has to match to score, and sometimes, a snippet from the document that contains the answer. From these, it was reasonably simple to assemble QA pairs (where the answer was marked in context). Previously, the questions and passages were tokenized using GATE. GATE also includes a part-of-speech (POS) tagger which tags each token with one of about 36 standard POS tags used in the PENN treebank corpus [33],[Brown corpus [3]]. For performance issues, we stopped using GATE and used QTag [34] for POS tagging and an *in-house* NE tagger[4].

2.2.2 Using Machine Learning

Logs of past QA pairs are used to *get better* at identifying the various question fragments mentioned above. Every learning task is a three-step process. The first one involves identifying the features suitable for the particular learning problem. Feature identification is the most crucial step in any Machine Learning problem. After carefully choosing the features, the training instances in the corresponding *feature space* are built using the QA pair logs. Finally, a classifier

[2] `http://laiirs3.cse.iitb.ac.in:8080/servlet/SentencifyGUI`

[3] `http://www.scs.leeds.ac.uk/amalgam/tagsets/brown.html`

[4] `http://laiirs3.cse.iitb.ac.in:8080/servlet/PassageTaggerGUI`

is trained using the built instances. We have built learners for identifying selectors and focus words in a question. We use a statistical approach for identifying the mapping between question cue phrases and the corresponding *representative WordNet synsets*. Instead of using an off-the-shelf available NE tagger, we have built a learner that learns to tag a piece of text with named entities. The re-ranking algorithm is also built as a learning mechanism where the features are identified by coming up with general properties that make one passage answer a question better than the others. All the learning modules are briefly mentioned below. Chapter 3 contains a detailed description of the feature set used and the instances built for each of these modules.

2.2.2.1 Selector Learning

Determining selectors is a fundamental aspect of translating our information needs into queries suited for search engines. A typical user is fairly sure that some words must appear in the answer passage, which we term as the selectors. Our goal is to mimic this process to be able to generate a keyword query from a question. With this idea in mind, we construct the feature set for the selector learning task. The details are provided in section 3.1.

2.2.2.2 Focus Word Learning

The focus words in the question are the ones that get connected to the answer in the passage. In order to capture this idea, we identify features for the focus word learning task using a concept of *focus* of a natural language unit (phrase or sentence). We give a detailed treatment of focus words and the learning setup for the same in section 3.2.1.

2.2.2.3 Discovering wh-word to WordNet synset mappings

Many QA systems use question classification as the first step and identify the *kind* of question it is. Depending on the identified kind they identify the kind of answer, the system should be looking out for. We do not use any explicit question classification since that would act contradictory to our principle of a *clean recipe* for doing QA. Question classification involves use of hand-tuned heuristics for identifying the type of question and expected answer. As mentioned previously, the representative WordNet synsets cannot be obtained straightforwardly for the question cue words (the wh-words). WordNet does not contain an entry for wh-words but we want the atype obtained from the wh-word to be represented as a WordNet synset. The process of identifying the atype clue is automated by *mapping* the wh-word to WordNet synset (concept). The WordNet synset identifies the concept under which the answer should subsume. A statistical method is presented in 3.2.3 to discover the mapping from QA pairs.

2.2.2.4 Learning to tag with Named Entities

The available NE taggers are not very accurate. It is difficult to customize them for the QA task (with the hope of improving their accuracy) since they heavily rely on handcrafted linguistic resources such as context free grammars, regular expressions, lists of trigger words, gazetteers or dictionaries. Substantial human effort is required to customize such a tagger for our task because that would involve adding entries to gazetteers and rules to grammars. We propose a method for learning the rules for NE tagging from a pre-tagged text. The tag-set is provided at the time of learning and any change in the tag-set (due to change in domain or any other reason) involves re-training our learner with training data that is tagged with the new tag-set. The entire learning setup is described in section 3.4.

2.3 Online Tasks

We use a two step process for providing a ranked list of passages with the most likely answer phrase marked in each passage. The first step being the *ranking* step and the second the *re-ranking* step.

2.3.1 Ranking

The first step is to retrieve a set of k passages that are related to the question and *pertain* to the answer. With this aim, we fire a query formed using the question to a search engine such as Lucene[4] and retain the top k ranked passages. There are various ways in which we can query Lucene. We tried using the selectors as query as well as the entire question as query to Lucene. The hits returned by Lucene do not change significantly with these two variations. This is because, Lucene uses TFIDF scores to rank the passages. So, even if we query it using the entire question, it suppresses the commonly occurring words since they have a high IDF. The selectors happen to be the words which occur less frequently, hence are given importance in the Lucene scoring formula. Refer A.3 for the details of Lucene indexing, searching and ranking mechanisms.

2.3.2 Re-ranking

We assume that the top k passages returned by Lucene in the ranking step contain the passages that contain the answer. k typically is 100 or greater so we need to re-rank the passages such that answer passages are pulled up and are present in the top 5 passages. We hypothesize that the answer passage, amongst the ranked passages will be the one which:

- Has *relatively* high similarity between some tokens in the passage(the answer tokens) with the identified atype clue in the question.
- Has *relatively* high linear proximity between these identified answer tokens and the identified selectors in the question
- Contains tokens that exactly match the identified selectors
- Is *relatively* highly ranked by Lucene

We learn the relative importance of these different features using the QA pairs log. We build instances using these features and build a regressor that regresses over these instances. This regressor is then used to re-rank the passages obtained in the ranking step for a new question.

2.4 The Complete System

The complete system is as shown in Figure 2.2. The offline tasks involve all the learning modules and these modules are put to use at query time. This is clearly depicted in the figure.

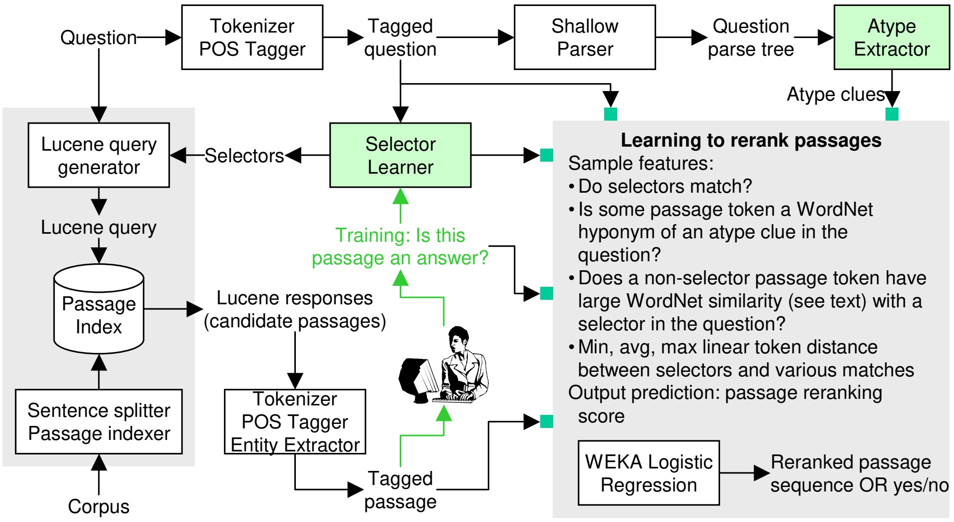

Figure 2.2: The Complete System using our approach

The key techniques are re-visited in the next chapter, where details of the learning tasks and the ranking and re-ranking steps are provided.

Chapter 3

Detailed Description of the system

Our approach of using Machine Learning techniques for the QA task hinges on the use of several key techniques. Chapter 2 provided an overview of each of them. These key techniques are re-visited and details for each of them are provided in this chapter .

3.1 Learning to identify Selectors

Selectors are defined as words in the question that must occur in the answer passage with little or no change. If a keyword query is to be constructed from the question, the selectors can act as the keywords in that query. We learn the general properties of words in a question that classify them as selectors. We use a learning setup wherein, the TREC questions tagged with selectors are used for generating the instances used for training. There are two kinds of features in the feature set of this learning problem. One, which are local to the question and the others, which are global and are defined over the document collection. An instance is created for every token in the question. The local features in the feature set include the POS of the token, POS of the neighbouring tokens, a boolean feature indicating whether the token is a focus word and a boolean feature indicating whether the token starts with an uppercase character. The global features include the fraction of documents which contain the word, number of *senses* the word has in isolation and number of words that describe that sense in the QA pairs. The steps given in Figure 3.1 are used to build the classifier.

Input: A collection of questions tagged with selectors

- For every question q in the collection do
  - For every token (word) in the question do
    1. Get the POS of this token
    2. Get the POS of the neighbouring tokens at a distance of $+window$ and $-window$
    3. Identify if the token's first character is uppercase
    4. Identify if the token is a focus word
    5. Get the fraction of documents which contain the word (some version of IDF), or whether this is a known stopword
    6. Get the number of senses the word has in isolation (available from WordNet)
    7. For a given sense described by this word, get the number of words that describe that sense in the QA pairs
    8. If the token is tagged as a selector, set label as $+1$ else -1
  - Form a training instance using the items obtained above
- Train Weka's DecisionTree classifier (J48) using the training instances

Output: A decision tree that can identify the selectors in a question

Figure 3.1: The selector learning procedure

3.2 Getting the Atype Clue

An atype (answer type) clue is obtained from the question either through the cue phrase or the focus words present in the question. The atype of a question, represented by WordNet synsets, is the one that gets *transformed* to the answer, again represented by WordNet synsets, in the passage. For questions which contain *what* or *which* as the cue-phrases, the atype is obtained using what we call as the focus of the question. The atype for such questions is identified by the WordNet synsets corresponding to these focus words. For some questions, the cue phrase is sufficient to identify the atype. However, it is not straightforward to identify the WordNet synset that represents the atype it this case.

We first motivate the idea of focus of a natural language sentence followed by the learning setup used for identifying the focus words in a question. Following that, we present a statistical method for mapping cue phrases to WordNet synsets.

3.2.1 What's the focus?

A declarative natural language sentence can be viewed to revolve around one or more important *concept(s)* or *entity(ies)*. A simple sentence will mostly contain only one key entity. This concept or entity is called the *topic* of the sentence and the words that represent this entity as the *topic words*. The topic word(s) is(are) generally supplemented by one or more *attributes* which enhance or modify the key concept represented by them. For example, consider the declarative statement *The Himalayas are the pride of India*. Here, "Himalayas" is the topic of the sentence. The entire sentence is about one concept of the Himalayas. It has no attributes. For a sentence such as *Catch-22 was one of the top best-sellers*, “Catch-22” forms the topic. For the sentence *The broad road ends after a kilometer*, “road” is the topic word with “broad” being the attribute of the topic word. For a question, the key concept which the question seeks in the answer is called the *focus* and the words that comprise of this focus in the question are called the *focus words*. This concept of focus is also applicable to well-formed phrases. For example, a phrase such as *Golden bracelet with ruby and pearl work* has “bracelet” as the focus word and “golden” and “ruby and pearl work” act as its attributes.

If the question is not simple, there will be several key concepts which can form the focus of the question. The focus of a question has helped us with our Question Answering task. Also, if the question is a simple sentence, there is no ambiguity while identifying the focus words.

3.2.2 Atype clue using the focus words

For the task of Question Answering, one of the most important subtasks is to identify the entity the question seeks for, as the answer, in the answer passage. For example, in the question *Which U.S. President is buried in Washington D.C.?*, we are looking for a passage that contains the name of a President, who was an American and who is buried in Washington D.C. . We define the focus of a question by extending the idea of the focus of a natural language sentence or phrase. In the above question, “President” forms the focus of the question and the words “U.S.” and “buried in Washington D.C.” act as modifiers or attributes of it.

It is easy to see that for the *what* and *which* type of questions, the focus words identified for the question serve to provide the atype clue for the question. Hence, this idea of focus words is applicable only to the what and which type of questions. For the other questions, the wh-word itself gives the atype clue for the question.

3.2.2.1 Learning to identify the focus

We give a general recipe of identifying the focus of a declarative sentence. This recipe is used for identifying the focus of a question. A learning setup is used wherein the TREC questions tagged with focus words are used for generating the instances used for training. A classifier is trained using these instances. This classifier is then used to identify the focus of the question. The parse of the sentence/question is used for identifying the instances required to train the classifier. Two kinds of instances and hence two kinds of feature sets are used. In one setup, we build an instance for every token in the question while in the other setup an instance is created for every phrase identified in the parse of the question. The parser mentioned in [28] is used for getting the various features of the instances. Refer to A.1 for details of Natural Language Parsing and the Lexicalized parser that we have used.

3.2.2.1.1. Using phrases as instances

In this setup, an instance is created for every phrase in the parse of the tree generated for the question. The parser API returns a parse tree for a piece of text. One can get the various phrases at the different heights in the parse tree. For example, for a question such as

Who is Tom Cruise married to?

the parser returns a tree of depth 4. The tree can be accessed as follows,

```
At depth : 0 (tokens in the question)
Phrases  : Who, is, Tom, Cruise, married, to, ?

At depth : 1 (tokens with POS)
Phrases  : (WP Who), (VBZ is), (NNP Tom), (NNP Cruise), (JJ married), (TO to), (. ?)

At depth : 2
Phrases  : (WHNP (WP Who)), (NP (NNP Tom) (NNP Cruise)), (PP (TO to))

At depth : 3
Phrases  : (ADJP (JJ married) (PP (TO to)))

At depth : 4
Phrases  : (SQ (VBZ is) (NP (NNP Tom) (NNP Cruise)) (ADJP (JJ married) (PP (TO to))))
```

```
                Corresponding parse tree :

                           SQ
                        /     \
                       /        \
                      /         ADJP
                     /          /  \
          WHNP      /  NP      /    PP
           |       /  /   \   /      |
           WP    VBZ NNP  NNP JJ      TO
           |      |   |   |    |       |
           Who   is Tom Cruise married to
```

Figure 3.2: Parse tree using the Lexicalized parser

The features used for this learning setup are the following.

1. Depth of the phrase. As shown in 3.2, it is obtained using the height of the parse tree at which the phrase is obtained.

2. POS of the head of the phrase.

3. Type of the phrase. The phrase type for the phrase *(VBZ is)* is *VBZ* while for *(JJ married)* it is *JJ*. The phrase type for *(ADJP (JJ married) (PP (TO to)))* is *ADJP*.

4. Phrase type of the phrase that lies to the left of this phrase. For the phrase, *(NP (NNP Tom) (NNP Cruise))* in 3.2, the phrase that lies to the left of this phrase is *(VBZ is)*.

5. Phrase type of the phrase that lies to the right of this phrase. For the phrase, *(NP (NNP Tom) (NNP Cruise))* in 3.2, the phrase that lies to the right of this phrase is *(ADJP (JJ married) (PP (TO to)))*.

The steps mentioned in Figure 3.3 are used to build a classifier that identifies focus words in sentences (questions) or phrases.

Input: A collection of questions tagged with focus words

- For every question q in the collection do
  - Obtain a parse of q
  - For every phrase returned by the parser do
    1. Get the depth of the phrase. As mentioned above, it is obtained using the height of the parse tree at which the phrase is obtained.
    2. Get the POS of the head of the phrase.
    3. Get the type of the phrase.
    4. Get the phrase type of the phrase that lies to the left of this phrase
    5. Get the phrase type of the phrase that lies to the right of this phrase
  - Form a training instance using the items obtained above
- Train Weka's DecisionTree classifier (J48) using the training instances

Output: A decision tree that can identify the focus words in a question

Figure 3.3: Generating phrase instances for focus word learner

3.2.2.1.2. Using tokens as instances

We used another learning setup for building the focus word learner. In this setup, an instance is created for every word of the question and then a classifier is built with this set of instances. The features used are as follows.

1. Is the word under consideration, the head of a phrase(0/1)

2. The tightly enclosing phrase to which this word belongs to. In a phrase such as *(ADJP (JJ married) (PP (TO to)))* as shown in 3.2, the tightly enclosing phrase for "to" is *PP* while for "married" it is *ADJP*.

3. POS of this token

4. Phrase that lies to the left of the phrase identified in 2. For the instance corresponding to "to", this will be *ADJP i.e.*, the tightly enclosing phrase for the word on the left.

5. Phrase that lies to the right of the phrase identified in 2. For the instance corresponding to "married", this will be *PP i.e.*, the tightly enclosing phrase for the word on the left.

We use the steps mentioned in Figure 3.4 to build a classifier for identifying focus words in natural language sentences (questions) or phrases.

Input: A collection of questions tagged with focus words

- For every question q in the collection
  - Parse q
  - For every token in the question do
    1. Is the word under consideration, the head of a phrase(0/1)
    2. Get the tightly enclosing phrase to which this word belongs to
    3. Get the POS of this token
    4. Get the Phrase that lies to the left of the phrase identified in 2
    5. Get the Phrase that lies to the right of the phrase identified in 2
  - Form a training instance using the items obtained above
- Train Weka's DecisionTree classifier (J48) using the training instances

Output: A decision tree that can identify the focus words in a question

Figure 3.4: Generating token instances for focus word learner

3.2.3 Atype clue through wh-words

For many question types such as those containing *wh-words* such as who, where, when, how_many, *etc.*, the wh-word itself provides the atype clue. For example, a question containing *who* as the wh-word gives the atype clue as a *person* whereas a question containing *when* gives the atype as *time* or *day* or *year*. The atype clue however, needs to be represented by WordNet synsets. Since WordNet does not have an entry for the wh-words, synset identification is not straightward in this case of obtaining the atype through the wh-words. We use a statistical method for discovering the mappings between wh-words and the corresponding WordNet synsets from QA pairs. Using these mappings we can get the WordNet synset corresponding to a wh-word and thus obtain the atype clue for a question.

3.2.3.1 Use of question prefix trees

We try to discover the mapping between the WordNet synsets and the longest possible prefix in the question starting at the question cue. To be able to do so, we use question prefix trees. A given question is matched to a path in the prefix tree of maximum depth. The question prefix trees are as shown in Figure 3.5.

```
how                     When            What
 |--- many                |--- did        |--- city
 |      |--- times        |--- does       |--- country
 |      |--- acres        |--- do         |--- achievement
 |      |--- countries
 |
 |--- much
 |      |--- time
 |      |--- amount
 |
 |--- often
 |
 |--- far
 |
 |--- deep
```

Figure 3.5: Questions as prefix trees

3.2.3.2 Algorithm

We use the method present in Figure 3.6 to discover the mapping between the wh-words and the most related WordNet sysnets. We discover a mapping between the prefix trees and the WordNet synsets. We fix a set of roots for the prefix trees, viz. the "wh" words: "when", "what", "how", "where", "which", "who", "whom", "whose", "why", "name", and "define" and then use the steps in given in 3.6, to populate the counters corresponding to every node in the prefix tree.

Significance tests are used for identifying the mappings that stand above a *background distribution*. Refer B.2 for details about significance tests. In other words, using these tests, we can identify the significant mappings obtained and discard the unimportant ones. Except for *which* and *what* type of questions, the WordNet synsets obtained using the mapping from the question cue and the WordNet synset provides the atype clue for the answer. We discovered the significant mappings shown in 3.7.

Training: Given a QA pair (q, a),

1. Connect tokens a to WordNet synsets by lookups (more than one if ambiguous)
2. Walk from these synsets up hypernym edges and collect all ancestor synsets in a set (S_a)
3. Locate the wh-word in the question
4. Walk up to k tokens right of the wh-word
5. Build a prefix tree of depth up to k using these tokens
6. In each node of the prefix tree, maintain a map from synsets to counters
7. Find the path in the prefix tree corresponding to the k-prefix of question q starting at the wh-word
8. For each node on the prefix path increment counters for all synsets in S_a

Deployment:

1. Given a test question, locate its wh-word and walk to the right, trying to find the longest possible matched path in the prefix tree
2. At the last node to be matched, collect the surviving synsets

Figure 3.6: Discovering wh-word to WordNet synset mappings

```
How => {abstraction#n#6}
   fast => {magnitude_relation#n#1,rate#n#2}
   much => {fundamental_quantity#n#1, time_period#n#1}
   many => {measure#n#3, definite_quantity#n#1}

What => {region#n#3, entity#n#1,living_thing#n#1, casual_agent#n#1}
    city => {area#n#1}
    achievement => {physical_phenomenon#n#1}
    university => {geographical_area#n#1}

Where => {region#n#3}
Who => {person#n#1}
When => {calendar_month#n#1, happening#n#1, accident#n#2}
```

Figure 3.7: Sample frequent connections from *wh-words* to WordNet synsets. The wordnet synsets as shown are given in the form synset#part_of_speech#sense.

We indicate the synsets as *synset_name#POS#sense_number*. For example, as seen in 3.7, the question cue *who* gets mapped to *person#n#1*, the question cue *when* to *calendar_month#n#1*, *happening#n#1*. These synsets can be used during the re-ranking phase. The maximum value of similarity (using one of the proposed similarity measures) between this synset and the synsets corresponding to every word in the answer passage is used as a feature while re-ranking.

3.3 Connecting the atype with the answer token(s)

As mentioned in Chapter 2, a passage is very likely to contain the answer if some of its tokens have high *similarity* with the atype clue identified in the question. The similarity measure used for the task of connecting the atype of the question with the answer tokens plays a vital role for bridging the lexical chasm. To this end, we go beyond cosine-based similarity measures *i.e.* beyond mere term overlap and extensively use WordNet for defining the our similarity measures.

3.3.1 Similarity Measures

In order to identify the similarity between the question words with the answer words, an ordinary syntactic match does not suffice the purpose. To that effect, we explored a few well-known WordNet similarity measures.

Hyper-path: Given are two WordNet synsets corresponding to the two words, the similarity between whom we wish to identify. We do the following tests : Is there a hypernym (IS-A) path from one syset to another. The relation could be asymmetric *i.e.* the lengths of the *matching paths* on two Hypernymy paths corresponding to the two synsets might differ. The similarity between the two words is inversely proportional to the minimum of lengths of the matching paths.

Approximate Lin similarity [31]: WordNet's hypernym relation reveals that *Cat* and *dog* are very similar: the path from *cat* to *entity* (a root noun) overlaps almost entirely with the path from *dog* to *entity*. Suppose we associate with each synset s a set of synset IDs of all ancestors, A_s. Then one variant of Lin similarity between synsets s_1 and s_2 is $|A_{s_1} \cap A_{s_2}|/|A_{s_1} \cup A_{s_2}|$, called the Jaccard overlap.

Least Common Ancestor: Given two WordNet synsets corresponding to two words, we identify the synset that is a common ancestor to both the sysnets. We identify the common ancestor on the Hypernymy path. The similarity between the two synsets is inversely proportional to the distance between the common ancestor and the each of the word synsets.

3.3.2 Similarity measure to connect atype with the answer

For identifying the similarity between the atype of the question (represented by a WordNet synset) with synsets corresponding to the tokens in the answer passage, we use the customized similarity measure called the HyperPath. The similarity measure is based on the hypernymy relation given by WordNet. Let t be the atype of the question, and a a candidate answer token(s). Let t mean the synset in WordNet corresponding to t, and likewise with a. The simplest notion of the strength of a WordNet-based connection between t and a, is whether a is a hyponym descendant of t: a 0/1 feature. We found some benefit in refining this boolean feature to a continuous feature to encode some idea of the significance of the connecting path. Simple graph measures, like the number of links, will not suffice. As one example, a three-hop path from *entity* (the most general noun) to *artifact* is not as strong a connection as the three-hop path between *mammal* and *elephant*. This intuition is captured nicely by measuring the overlap of nodes on the path from each of t and a to all the noun roots. The HyperPath similarity (mentioned above) between two WordNet synsets captures this intuition. We compute the HyperPath similarity between t and a as given in Figure 3.8.

1. Unless t is a hypernym ancestor of a,
   the feature value is zero

2. Otherwise, we collect the sets of hypernym synsets of t and a;
   let us call them H_t and H_a

3. Compute the Jaccard overlap $\frac{|Ht \cap Ha|}{|Ht \cup Ha|}$.

Figure 3.8: Computing similarity between atype and answer tokens

3.4 Learning to tag with Named Entities

Named Entity tagging is a very appropriate task where the use of Machine Learning techniques can be applied. This is because, in the absence of such techniques, a lot of effort goes into building lexicons and rules for doing the NE tagging. We construct a learning setup wherein every sentence from a piece of text is parsed and there is an instance for every phrase in the parse. A classifier is trained using these instances which can then be used for tagging a text with named entities. The procedure of building the classifier is as given in Figure 3.9.

Input: Text marked with NE tags

- For every sentence s in the text do
  - Parse s
  - For every phrase of the parse, identify
    1. Whether the first word in the phrase starts with a cap letter
    2. Whether the first word in the phrase is all caps
    3. Whether the phrase contains any numeric quantity
    4. Whether the phrase contains any alpha-numeric quantity
    5. Whether the phrase contains any single char quantity
    6. The POS tag of each entity in the phrase : one feature for every word
    7. The POS of the last word on the phrase at -1 distance from the current phrase
    8. The POS of the first word on the phrase at $+1$ distance from the current phrase
    9. Whether there is punctuation before this phrase or whether sentence boundary lies before this phrase : currently being implemented as just looking at whether it is the start of a sentence
    10. Type of the phrase
    11. how much percent of the tagged chunk is formed by this phrase: on a scale from 0 to 1
  - Form a training instance using the items obtained above
- Train a DecisionTree classifier using the training instances

Output: A decision tree that can tag with named entities

Figure 3.9: Building an NE learner

3.5 Re-ranking

The re-ranking step of the two-step approach used for the Question Answering task was briefly described in Chapter 2. In the re-ranking step, we modify the ranks given to the top passage hits by the traditional search engine (Lucene in our case). The properties that make one passage answer a question better are identified. Instead of using pre-defined settings for these properties, we use Machine Learning on QA pair logs and learn to re-rank passages. The details of our approach are given below.

3.5.1 Learning to score passages

Given the top responses R_q from the passage index as judged by a vector-space search engine given a query q, the scenario is easily framed as a supervised learning or classification problem. Each (q, r) pair, for $r \in R_q$, forms an instance. The label is $+1$ if r contains an answer to q and -1 otherwise. During training, labels are directly available from QA-pairs. During testing, a suitable classifier can assign hard ± 1 labels to each $r \in R_q$. One approach is to simply discard all response passages where our classifier guesses label -1. In that case, we should seek a classifier that will eliminate as many negatives (non-answers) as it can, while ensuring that the precious positive(s) are not lost. The surviving passages can be output in the keyword response order.

The classifier assigns to each response r a score which is directly related to some belief that r does contain an answer to q (i.e. has label $+1$). The ranking given by keyword search is abandoned, and all passages in R_q are returned in order of the classifier-assigned score. However, keyword search engine's ranking is not completely discarded. The rank given by the search engine is encoded as a feature for the classifier.

The other features from (q, r) are encoded in the view of the following. If r is a positive instance, we expect, among some other things,

- All or most selectors to match between q and r
- A connection from q's atype clue through a WordNet synset to a hyponym embedded in the answer passage
- The expected answer tokens to be close, in terms of number of intervening tokens, to matched selector(s)

A complete specification is shown in Figure 3.10.

Either a generative or a discriminative classifier can be used to return a score. A generative classifier needs a parametric model of the distribution from which training data is sampled; training involves estimating the parameters of this model. For unstructured domains like text, which has many attributes and complex relationships between them, proposing an accurate generative model that captures their joint distribution accurately is nearly impossible. In contrast, discriminative classifiers do not assume a distribution. They simply optimize a parametric *separator* function between the two classes while avoiding over-fitting. In view of all this, a discriminative classifier is chosen for the task. Since the number of training instances is quite large, we used Logistic Regression (in WEKA) rather than kernel methods.

Input: QA-pair (q, r)

1. Map q to atype synset s_q
2. Remove from consideration all selectors of q that appear in r
3. For each WordNet-based similarity measure (see text) find that token t_r in r which has the largest similarity to s_q
4. Call t_r the landing zone

Output: A feature vector for (q, r) including the following attributes:

- Number of exact selector matches
- Similarity score (possibly 0/1) for each WordNet-based similarity specified
- Minimum, average, maximum token distance between a landing zone and a token
- Initial keyword search-based passage rank of r

Figure 3.10: Generating features from a QA pair.

The key technique of identifying the focus of *what* and *which* type of questions that provide the atype clue for those questions, was detailed in this chapter. For questions whose question-cue is other than *what* and *which*, we use the *wh-word* itself to get the atype clue. The statistical procedure for mapping the *wh-words* to *WordNet synsets* was outlined here. The effectiveness of the connection of these synsets or the synsets associated with the focus words with the answer tokens, hinges on the similarity measures used for the task. The customization of several similarity measures using WordNet were outlined here and also the measure that is best suited for the QA task was pointed out. Finally, we outlined the learning framework for re-ranking passages obtained after the ranking step.

Chapter 4

Experimental Evaluation

In this chapter, we present the experimental results for the various learning modules and the performance of the entire system. The experimental results mainly constitute the training, testing and cross-validation accuracies, precision, recall and the F1-scores. Before the focus learner was built, the atype clues were extracted from a shallow parse of the question by the Link Parser [48]. For the learning tasks, we used the J48 decision tree and the logistic regression packages in WEKA [51]. Our code is entirely in Java, except for call-outs to the Link Parser. For each year of TREC data, our total data preparation and training time on a dual 1.3GHz Xeon server was several hours, most of it spent in tagging top passages returned by Lucene. WEKA itself took only several minutes. Typically, each query could be answered in under 30 seconds. For each of the learners, we give the training and test accuracies, precision, recall and the F1-score. For a confusion matrix such as this

		Classified As	
		+1	−1
Actual	+1	n_{11}	n_{12}
Label	−1	n_{21}	n_{22}

Figure 4.1: The confusion matrix

We use the following for calculating the accuracies, precision, recall and the F1-score.

1. Accuracy(AC) $= \frac{n_{11}+n_{22}}{n_{11}+n_{12}+n_{21}+n_{22}}$
2. Precision(PR) $= \frac{n_{11}}{n_{11}+n_{21}}$
3. Recall(R) $= \frac{n_{11}}{n_{11}+n_{12}}$
4. F1-score $= \frac{2*R*PR}{(R+PR)}$

4.1 Focus Word Learner

As mentioned earlier, we use two kinds of instance vectors for the purpose of Focus Word Learning. One wherein we have an instance for every token in the question and another, wherein we have an instance for every phrase in the parse of the question. Since its a two class problem, we used Weka's *Logistic Regressor* to learn to identify the focus words. For getting the instances, we used the Lexicalized Parser for parsing the question.

It can be seen that in all of these learning problems, the accuracy of correctly classifying the negative examples is very high. This is because in all the setups, the number of negative examples is significantly high as compared to the positive examples.

4.1.1 Using tokens as instances

TREC 2000	TREC 2002
Accuracy = 87.66%	Accuracy = 88.85%
P = 0.67	P = 1.00
R = 0.276	R = 0.239
F1 = 0.39	F1 = 0.3857

TREC 2000:

	+1	−1
+1	161	422
−1	77	23196

TREC 2002:

	+1	−1
+1	64	203
−1	0	7997

Figure 4.2: Training accuracies for the focus word identifying classifier built using **tokens as instances**. Label +1 means the instance is a focus, −1 means it is not.

TREC 2000 trained on TREC 2002	TREC 2002 trained on TREC 2000
Accuracy = 87.66%	Accuracy = 88.85%
P = 0.705	P = 0.733
R = 0.205	R = 0.262
F1 = 0.317	F1 = 0.386

TREC 2000 trained on TREC 2002:

	+1	−1
+1	72	279
−1	30	2124

TREC 2002 trained on TREC 2000:

	+1	−1
+1	80	225
−1	29	1946

Figure 4.3: Test accuracies for the focus word identifying classifier built using **tokens as instances**. Label +1 means the instance is a focus, −1 means it is not.

4.1.2 Using phrases as instances

TREC 2000	TREC 2002
Accuracy = 81.58%	Accuracy = 91.72 %
P = 0.74	P = 0.38
R = 0.412	R = 0.618
F1 = 0.529	F1 = 0.47

	+1	−1
+1	260	370
−1	90	1675

	+1	−1
+1	196	121
−1	55	1756

Figure 4.4: The focus word identifying classifier built using **phrase as instances**. Label +1 means the instance is a phrase containing only the focus, −1 means it is not.

TREC 2000 trained on TREC 2002	TREC 2002 trained on TREC 2000
Accuracy = 79.62%	Accuracy = 90.17%
P = 0.764	P = 0.67
R = 0.325	R = 0.668
F1 = 0.456	F1 = 0.668

	+1	−1
+1	205	425
−1	63	1702

	+1	−1
+1	212	105
−1	104	1707

Figure 4.5: Test accuracies for the focus word identifying classifier built using **tokens as instances**. Label +1 means the instance is a focus, −1 means it is not.

4.2 Named Entity tagger

For this multi-class problem, we used Weka's *Decision Trees(J48)* as the classifier to learn to tag with NE tags. We used to following NE tags in the training set:

1. COUNT → a
2. DATE:DAY → b
3. DATE → c
4. YEAR → d
5. ORGANIZATION → e
6. FEMALE:PERSON → f
7. MALE:PERSON → g
8. PERSON → h

9. LOCATION → i

10. CURRENCY:MEASUREMENT → j

11. LENGTH:MEASUREMENT → k

12. MEASUREMENT → l

13. TITLE → m

14. TIME → n

Accuracy = 89.77%

	a	b	c	d	e	f	g	h	i	j	k	l	m	n
a	30549	280	0	704	28	1	18	69	221	2078	1324	669	11	88
b	1673	5460	0	91	258	167	255	158	2536	552	560	169	84	65
c	0	1	0	0	0	0	0	0	1	0	0	0	0	1
d	1302	48	0	3025	0	0	0	0	3	262	50	8	0	0
e	50	186	0	0	3402	194	781	1436	6662	4	6	7	166	6
f	0	145	0	0	492	1498	1370	739	2450	0	1	0	85	0
g	9	125	0	0	737	976	2480	883	3720	1	1	0	94	3
h	69	240	0	0	1310	607	1326	4555	5879	9	25	36	157	37
i	337	655	0	0	1582	679	824	1620	22881	25	40	34	348	109
j	2691	96	0	55	0	0	0	0	58	30699	684	296	12	5
k	2849	163	0	19	11	0	6	19	243	637	7761	400	12	30
l	1759	63	0	14	11	0	4	22	60	649	630	3456	1	20
m	64	171	0	0	691	221	483	659	3456	2	10	2	1149	3
n	133	74	0	0	26	11	38	29	174	35	39	13	12	890

Figure 4.6: Training accuracies for the NE tagging classifier using 15 NE tags. We use Weka's Decision Tree as the classifier.

4.3 Re-ranking

We use Weka's logistic regressor for the two class problem of learning whether a passage is an answer passage or not. The scores given by the regressor are used while sorting the re-ranked passages. Figure 4.7 gives the training accuracies of the regressor.

TREC 2000, 84.88%

	+1	−1
+1	3539	4837
−1	419	25855

TREC 2002, 89.46 %

	+1	−1
+1	559	2160
−1	44	18139

Figure 4.7: The re-ranking classifier can reject negative (non-answer) passages very effectively. Label +1 means the passage has an answer, −1 means it does not.

4.4 System Performance

Suppose $n_q \geq 1$ is the earliest rank of the passage at which the answer to question $q \in Q$ is found. Then MRR is defined as

$$(1/|Q|) \sum_q (1/n_q)$$

MRR is between 0 and 1, and a higher MRR is better. The mean reciprocal rank (MRR) of the system is above 0.7. Figure 4.8 shows that for many questions, the answer passage was missed during the ranking step. We call the sharp spike towards the end of the histogram as the tail of the histogram. We can see a large tail in the given histogram which indicates that the number of questions which were missed in the ranking step is quite large.

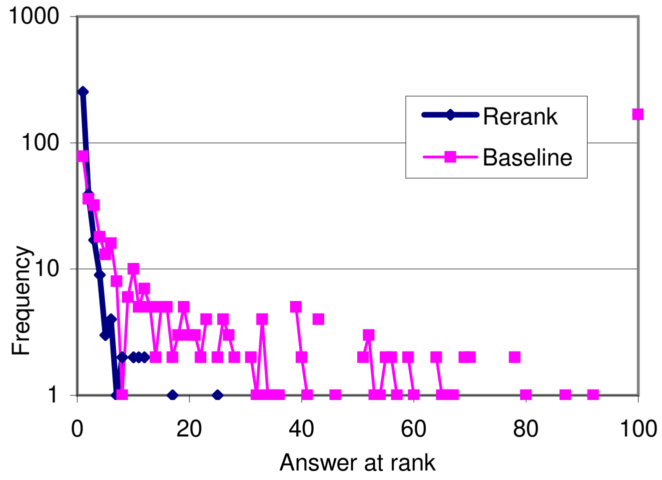

Figure 4.8: Re-ranking significantly boosts MRR. The x-axis is the rank at which the (first) answer is found, the y-axis counts questions for which the first answer was found at the given rank. Here we re-ranked only the top 100 Lucene-returned passages.

MRR@in which stands for MRR-at-infinity (refer B.1 for the definition) values obtained with various training and test datasets are given in 4.10. These MRR values are computed taking into account all the passages which contain the answer upto a rank of 100 (ideally should be infinity).

Trained on	Tested on	Re-ranked MRR@in
2002	2002	0.565
2000	2002	0.539
2000+2001	2002	0.534
2000+2001+1999	2002	0.541
2000	2000	0.710
2002	2000	0.705
2002+2001	2000	0.627
2002+2001+1999	2000	0.693

Figure 4.9: Various MRR@in values with different TREC datasets

Trained on	Tested on	Re-ranked MRR@5
2002	2002	0.553
2000	2000	0.706

Figure 4.10: MRR@5 values do not differ much from their MRR@in counterparts

Figure 4.9 gives the MRR@5 values for TREC 2000 and TREC 2002 trained and tested on the same corpus. For computing MRR@5, passages only upto rank 5 are considered. Passages that contain the answer but lie below rank 5 are discarded while computing MRR@5.

For each of the question types (depending on the cue phrases), the improvement via re-ranking

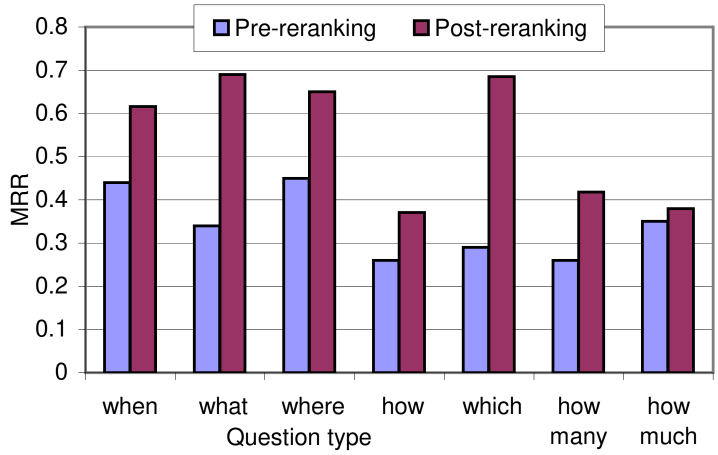

Figure 4.11: Sample MRR improvement via re-ranking separated into question categories.

was observed. The results are as tabulated in Figure 4.11.

Chapter 5

Analysis of the Experimental Results

In chapters 2 and 3, we described our approach of using Machine Learning based techniques to build a complete QA system. In Chapter 4, the accuracies and other performance numbers of the various learning modules and those of the entire system were given. When our system is trained using a sufficiently "good" training dataset, it gives an an MRR of about 0.7. That implies, that there are about 30% questions for which our system cannot retrieve an answer in the top 5–10 passages. In this chapter, we provide a failure analysis of our system. We give an exhaustive list of the questions that our system fails to answer. By failing, we mean that the system was unable to rank the answer passage in the top 5 or 10 output passages. We focus on the questions given in TREC 2000 and TREC 2002. There are various reasons for the answer not being provided by our system. Some questions fail during the ranking step while some during the re-ranking step. We enumerate the various probable reasons that cause our system to falter.

5.1 Performance Statistics

The statistics of the output of our system for the TREC 2000 question set are given in Figure 5.1.

```
Total questions : 693
Questions answered : 78.2% (542 out of 693)
Questions answered@Rank0 : 44.7% (310 out of 693)
Questions answeres@Rank0 by Lucene : 27.99% (194 out of 693)
Questions for which no answer passage
is retrieved in top 100 : 21.78% (151 out of 693)
```

Figure 5.1: TREC 2000 Statistics

The statistics of the output of our system for the TREC 2002 question set are given in Figure 5.2.

```
Total questions : 499
Questions answered : 64.7% (323 out of 499)
Questions answered@Rank0 : 30.86% (154 out of 499)
Questions answeres@Rank0 by Lucene : 15.89% (79 out of 499)
Questions for which no answer passage
is retrieved in top 100 : 35.27% (176 out of 499)
```

Figure 5.2: TREC 2002 Statistics

5.2 Diagnosing the ranking step

The histogram plotted using the output of the system for the TREC 2000 questions is as shown in Figure **??**. It can be seen that the number of questions for which our system could not return an answer in the top 100 passages, is quite large (for TREC 2000, 151 out of 693 and for TREC 2002, 176 out of 499).

Consider the following questions and the queries that were fired to Lucene. Lucene failed to retrieve the answer passage in the top "k" hits, where "k" is typically set between 100 through 300.

1. Where did bocci originate?
   Query: 1. Entire question, 2. bocci originate

2. What does laser stand for?
   Query: 1. Entire question, 2. laser stand for

3. Where does dew come from?
   Query: 1. Entire question, 2. dew come

4. Where did the ukulele originate?
   Query: 1. Entire question, 2. ukulele originate

5. What are birds descendents of?
   Query: 1. Entire question, 2. birds descendents

6. What does NAFTA stand for?
   Query: 1. Entire question 2. NAFTA stand for

7. What company sells the most greeting cards?
   Query: 1. Entire question, 2. company sells greeting cards

8. What is the name of the longest ruling dynasty of Japan?
   Query: 1. Entire question, 2. ruling dynasty Japan

For many questions provided in the TREC *test questions*, the corpus provided for a particular year does not contain the answer. For example, consider the question, *What state has the most Indians?* The regular expression provided by TREC for locating the answer itself fails. We assume that for such questions, the corpus does not contain the answer. It is difficult to identify such questions. Only a return of 0 hits for the entire question can indicate that the corpus does not contain any information relevant to the question. However, mostly the question contains atleast a few commonly occurring words hence for the entire question given as query, it rarely happens that Lucene returns 0 hits. It was easy in the case of the following questions to see that the required answers were not present in the corpus since Lucene returned 0 hits even when the entire question was given as query to Lucene.

1. What is a nematode?
   Query to Lucene: nematode
2. What is a caldera?
   Query to Lucene: caldera
3. What is a stratocaster?
   Query to Lucene: stratocaster
4. What is tyvek?
   Query to Lucene: tyvek
5. Where did bocci originate?
   Query to Lucene: 1. bocci 2. bocci originate

Lucene returns 0 hits mostly when the selectors imposed are strict and if they do not appear exactly as they appear in the query. For the above questions, if we query Lucene with the entire question, hits are returned but they are not about the required *entity* the question is about. It is not always that the corpus does not contain the required entities but due to some reasons (explained further in this chapter) the passage containing the answer is pushed below. Lucene fails to retrieve the answer passage in the top "k" passages even for questions whose answers are present in the corpus. We attribute this failure to the following reasons.

- The query fired to Lucene is too strict
- The query fired to Lucene is too general
- The proper nouns present in the query do not match the ones present in the answer passage
- The passages contain unresolved references due to the data preparation step of breaking the documents into smaller sized passages.
- Unable to bridge the lexical chasm: Question tokens other than proper nouns are present in some variant in the answer passage
- Unable to bridge the semantic chasm: The query lacks the atype clue and hence cannot retrieve passages that not only contain the *selectors* in the query but also the required entity that constitutes the answer.
- The question is ambiguous

We present each of these problems in detail in the following subsections.

5.2.1 Data preparation problems

For many questions, it happens that the corpus does not contain the answer in the right context. By corpus here we mean the processed corpus that is obtained after our data preparation step. In the data preparation steps, we use a sentencifier to sentencify the documents (Initially, we used GATE's sentencifier but now we use the sentencifier that we have built). We picked sliding windows of three sentences as passages. For 2000, we got 978953 documents, 14721729 passages, and 693 questions. For TREC 2002, we got 1031824 documents, 23266499 passages, and 500 questions. The passages were indexed by Lucene [4]. It can happen that a document is about a certain entity or fact but after breaking the document into passages, we lose this information. For example, a document can talk about *Abraham Lincoln* but the passages reference him only using pronouns or "the President" and so on. The answer passage present within the document cannot be retrieved if it does not contain the term *Lincoln*. For the question, *"Where does dew come from"*, the passage in which the answer is present does not contain the word "dew". And for the question *"What does NAFTA stand for"*, the answer passage does not contain the term NAFTA. During the ranking step, such passages (which actually contain the answer) get missed. However, the corresponding documents from where the passages are retrieved contains these keywords.

One crude way of getting over this problem is the following. During the data preparation step, for every document identify the terms which have high IDF *i.e.* terms that are important. Attach these terms to every passage that is formed from this document. Index these modified passages.

5.2.2 Proper nouns do not match

It is difficult to find the proper nouns occurring in questions to occur in the same form in the passages. So, when a query to Lucene contains proper nouns and none of the passages contain the same proper nouns but their variants, no hits are returned or the ones that are returned do not contain the answer.

Examples of such questions from the TREC 2000 set where Lucene returns 0 hits is :

1. Who was Whitcomb Judson?
   Query : "Whitcomb Judson"

2. Who owns the St. Louis Rams?
   Query : "St. Louis Rams"

3. Where did David Ogden Stiers get his undergraduate degree?
   Query : "David Ogden Stiers" undergraduate degree

5.2.3 Lexical chasm

Consider the question from the TREC 2000 test questions, *"Where did the ukulele originate?"*. If we give the question as the query or the filtered question *"ukulele originate"* as a query to Lucene, we do not get the answer in the top 100 passages returned by Lucene. However, if we query Lucene with only *"ukulele"*, then we get the following passages that contain the answer :
Rank 23: The **ukulele** , *introduced* from Portugal into the Hawaiian Islands about 1879 , was

first used in Canadian schools in the Maritime provinces about 20 years ago to teach music , he said .

Originate in not present in this passage and one can see that *originate* is transformed to *introduced* in this case. To pick this passage during the ranking step, one needs to bridge the *lexical chasm* and connect "originate" to "introduced".

5.2.4 Semantic chasm

Consider the question from the TREC 2000 test questions, *"Who was Darth Vader's son?"* For this question, Lucene retrieves passages that contain *Darth Vader* but is unable to identify the answer passage because it is unable to identify passages which contain the name of *Darth Vader's* son. Similarly for the questions,

1. What is the largest variety of cactus?
2. In the Bible, who was Jacob's mother?
3. What year did California become a territory?

the absence of any atype clue in query to Lucene, causes Lucene not to retrieve the answer passage in the relevant hits. The atype clue in one sense, helps us bridge the semantic chasm. Hence, the absence of any query terms containing the atype clues, leaves us with an unbridged semantic chasm.

5.2.5 Ambiguous questions

Some of the questions present in the test-sets are ambiguous. Examples of such questions are,

- When did Muhammad live?
  It is difficult to identify from the question, which Muhammad is it about.
- What are the opening words of the Declaration of Independence?
- What is the appropriate gift for a 10th anniversary?

5.2.6 Question-type not supported

Our system does not provide answers for questions that are non-factoid. Hence, it fails to answer the following questions,

1. How do you say "French fries" in French
2. How did Micky Mantle die
3. How do you say "house" in Spanish
4. How did Molly Shannon's mother die?

5.2.7 Query is too strict

The query chosen for firing to Lucene consists of only the words present in the question. For this reason, all the query tokens are *hard*, *i.e.* they need to syntactically match the passage tokens. We do not allow for any variants of the query terms in the passage. Also, presence of proper nouns and quoted strings make the query all the more strict. For such queries, the recall at the ranking step is quite low.

5.2.8 Query is too general

For some questions, almost all the words present in it are very common. Most of question terms have low IDF but they are not necessarily stop-words. It is observed that for such questions, Lucene fails to retrieve the answer passage in the top "k" passages. Examples of such questions are :

1. How much sleep should a child get at night?
2. How old must you be to become President of the United States?
   Lucene fails to get the passage which contains the answer because most of the returned passages containing the high IDF words, such as *President* and *United States* contain a context which is different from the answer context.
3. Who was the first president to speak on the radio?
4. What was the name of the first enclosed baseball park?

For such questions, it is important to identify the selectors to be used as the query to Lucene. If the entire question is fired to Lucene, more often than not, the answer passage does not turn up in the top "k" passages.

5.3 Diagnosing the re-ranking step

For some questions, it happens that the ranks given by Lucene are better than those obtained after the re-ranking is done. This happens if the question does not contain sufficient clues about the answer or if it has an atype that is difficult to obtain. Examples of such questions are :

1. Who invented the paper clip?
2. Who is the leader of India?
3. What is the population of Japan?
4. Where is the Danube?
5. Who made the first airplane that could fly?
6. How many years ago did the ship Titanic sink?

5.3.1 Difficult Atype for Questions

It is difficult to identify the atype of some questions. We get the atype clue for *what* and *which* questions through the focus words of the questions . However, for many questions, the focus words are not so straightforward to identify. For example, consider the question, *"What do ladybugs eat?"*. The focus of this question is food but our system cannot identify that. In such cases, the re-ranking mechanism will not be able to connect the focus (which is not correctly identified) and the answer tokens. Hence, our system will not be able to rank the correct answer passage appropriately. Here is a list of such questions.

1. What makes Black Hills South Dakota a tourist attraction?
2. What does the abbreviation OAS stand for?
3. What did Vasco da Gama discover?
4. What do penguins eat?
5. What do you call a group of geese
6. What are birds descendents of?
7. What does NAFTA stand for?
8. What do ladybugs eat?
9. What are John C. Calhoun and Henry Clay known as?
10. What did brontosauruses eat?
11. What do you call a baby sloth?

5.3.2 Failing to connect the atype with the answer

The similarity measures that we use for connecting the atype of the question with the answer tokens is constrained by WordNet coverage. If the words that we want to connect are not present in WordNet, our similarity measures will not remain effective. Consider, for example, the question, *"Who wrote 'Fiddler on the Roof?'"*. The answer for this question is *Sheldon Harnick*. If we do not use a NE tagger for tagging the corpus with named entities, the onus of this job of connecting such named entities with the atype is placed on WordNet. However, if *Sheldon Harnick* does not have an entry in WordNet then it cannot get connected with the atype of the question, *i.e. person*. It can also happen that the *HyperPath* similarity measure cannot connect related synsets because of the inherent shortcomings of the WordNet hypernymy relation.

5.3.3 Inadequate Named Entity tagging

For many questions, we do not get to the answer passage because the Named Entity that the question is looking for isn't marked by the NE tagger. We aimed to connect the atype with the NE tag of the answer token. However, NE tagging is not exhaustive and hence most of the named entities are not tagged or they are inappropriately tagged. Most of the NE taggers depend on manually compiled lexicons and rule sets. No matter how exhaustive the lexicons and rule-sets are, there will always be examples for which the tagger fails to tag appropriately. Also, the set of tags is generally quite limited and very generic in most of the NE taggers.

For such cases, the atype doesn't get connected with the answer clues and it can happen that the passage doesn't get pulled up after re-ranking. Following are such examples.

1. Where does the vice president live when in office?
   Problem : "U.S. Naval Observatory" is not known to be a "place"
   Ans=Before approaching Dulles, the same two F-16s apparently flew in prohibited airspace over the U.S. Naval Observatory, the official residence of the vice president, Shumann said.

2. What is the name of the heroine in "Gone with the Wind"?
   Problem : Scarlett O'Hara is-a heroine
   Ans=Sanctimoniously yours, a fully clothed Dr. Laura Schlessinger.Dear editors: I am willing to leap into the void and write the second sequel to "Gone With the Wind." I see it shaping up as something you might find surprising: a bloody horror story scarier than "Creepshow," with Scarlett O'Hara as a cruel sadist and serial killer.

3. What group sang the song "Happy Together"?
   Problem : "The Turtles" is-a group
   Ans=The Turtles were another victim. When California teenagers Mark Volman and Howard Kaylan signed with White Whale records in the '60s, they apparently didn't read the fine print. The label not only owned all the masters to their songs (including "Happy Together"), it owned the band's name as well.

5.4 Summary of the analysis

To summarize, we present Table 5.1, where we have classified the questions where our system faltered.

5.5 Bounds on the MRR using this approach

The previous sections provided a detailed diagnosis of the output produced by our system. We also enumerated various reasons that could possibly cause our system to falter. We suggest ways to get around some of the problems in the next chapter. Before presenting the possible solutions, it is worthwhile to give a thought to a general upper bound on the accuracy of Question Answering systems. We tried to identify the general class of QA systems and in many ways our approach matches the simple two-stage procedure of first finding a set of potentially relevant passages and then "pin-pointing" the exact location of the answer by searching for an entity whose semantic type matches the question type. [50] provides empirical limits on the effectiveness of this approach. The upper bound established in [50] is at around 70% with perfect passage retrieval, named entity detection and question classification. The bound is established purely on the basis of empirical results. No formal methods for proving the same are presented therein.

Questions	Question Class
What does the abbreviation OAS stand for ? What does CNN stand for ? What does R B stand for ?	Abbreviation Questions
What does "E Pluribus Unum " mean ? What is written on the U.S. tomb of the unknown soldier ? Who was Whitcomb Judson?	Description Questions
What is the Keystone State? What is a caldera? What is a nematode?	Definition Questions
What do penguins eat? What do the French call the English Channel ? If something has petrified what has it turned into? What are the Black Hills known for ?	Unknown Atype
What is the chemical formula for sulphur dioxide ? What is the zip code for Fremont, CA? What is the zip code for Parsippany, NJ?	Unknown Answer format

Table 5.1: Question classes for which the system falters

Our approach relies on perfect passage retrieval and named entity detection. Although, our approach does more involved processing during the re-ranking phase of the algorithm, its degenerate case can match the two-stage procedure. It does no question classification per say but uses the question-answer pairs to learn the mappings between the question cue phrases and the WordNet synsets which in a way act the question categories. Also, we rely on the various learning modules for identifying the fragments in the question. The onus of the effectiveness of our re-ranking phase is also on the appropriateness of the training data. However, our experiments show that, no matter how much training data is provided, the learners tend to saturate after using just about 20–30% of the training data. This shows that we cannot learn too much from the training data using the feature set that we currently use. However, we focus more on providing a general recipe for doing QA and to this end, we provide a generic framework for doing the same. The feature set can be enhanced, the off-the-shelf obtained tools used as black-boxes can be replaced with more accurate modules and the ranking algorithms can be improved without changing anything in the setup.

We emphasize here that, while trying to overcome the short comings of our system, one should not lose sight of this *bound* and not expect exceptionally high MRRs values. In order to significantly boost the MRR values, the system will require complex processing within each sub-module and perhaps some "in-house" design of tools which are currently gotten off-the-shelf and are used as black-boxes. However, our claim of having provided an effective and generic recipe for doing QA stands unaltered.

Chapter 6

Future Directions

In Chapter 5, we presented the failure analysis of our QA system and its various performance statistics. While these statistics are vital, they conflate and obscure the performance of the individual components of the system and the difficulty of the task. If the score is low, we need to understand what went wrong and how to fix it. If the score is high, it is still important to understand why. With such understanding, one can hope to:

- improve the system performance,
- simplify the system (if one particular characteristic of the system is responsible for good performance and the other features are parasitic),
- predict how a system will perform on different types of questions and/or different document collections

With this in mind, we present solutions to some of the problems that were described in Chapter 5. We outline the improvements that we incorporated into the system during the ranking and the re-ranking step and present some preliminary experimental results.

6.1 Reducing the histogram tail during Ranking

As was pointed out in Chapter 5, we lose a lot in the ranking step. This happens either because the query that we fire is too restrictive or too general. The idea of selectors was introduced in the first place to obtain an appropriate query from the question to be fired to Lucene (or any search engine).

6.1.1 Refining the idea of Selectors

At the onset, we defined selectors as those words of the question that are important for determining the context in which the expected answer must lie. They mostly occur without minimum change in the answer passage. However, if we fire the selectors obtained with such a mindset, the query to the search engine becomes too restrictive. We propose the following refinements to our basic idea of selectors.

1. selectors can be hard or soft
2. hard selectors *may* need expansion based on the corpus.

3. soft selectors, on the other hand, require transformation based on lexicons like WordNet.

4. Let us take some sample questions *viz.* "*When was the first hair dryer made* ", and "*What business was the source of John D. Rockefeller ' s fortune?* ",

   - "hair dryer" is a hard selector. It perhaps needs no expansion. If it all it needs any expansion, this expansion should be dictated by the corpus being queried, for the purpose of restricting the query to improve precision. Many nouns should fall in the category of hard selector. This however does not go as a rule. "business" is a soft selector since it actually determines the category of the answer. So is "source" a soft selector. On the other hand, "John D. Rockefeller" is a *hard selector*. However, it could occur in the form of variants such as "John Rockefeller" or simply "Rockefeller". This transformation will also be eventually dictated by the corpus being queried. However, "hair dryer" may however appear as "dryer for hair" etc. Question is: how many such transformations should we account for ?
   - "made" is a soft selector. It may need a *transformation* to something like "invented" or "produced" or "introduced". Also, as mentioned above, "business" and "source" are soft selectors and may undergo variations such as "cause" *etc.* or may not even appear in the answer passage. "fortune" is less of a *soft selector*. "fortune" may get transformed to "wealth".
   - Verbs in general cannot be *hard selectors*.
   - Hard selectors mostly should undergo syntactic transformations whereas soft selectors might undergo semantic transformation. Also, these expansions, if done without user's sanction could prove dangerous.

6.1.2 Query Expansion

One way to improve the recall at the ranking step would be to increase the number of passages on which re-ranking is applied from 100 to a large number. After that the onus of getting the answer passage, lies on the re-ranking algorithm. However, this method is not very robust. And even a large number like 1000 might not be sufficient to get the answer passage. The re-ranking step might not be able to pull up a passage that is ranked very low by lucene to top positions. We need to make the query to Lucene more appropriate by applying transformations to it (expansion or replacement). The query should be less restrictive but not too elaborate. There are two important issues when we deal with query expansion/transformation.

1. What to expand/transform

2. How to expand/transform

We first consider expansion of question words. There are two ways to expand the query, one is to use corpus based knowledge and the other is to use some ontology like WordNet. A good expansion method would be to combine these two methods for every question.

6.1.2.1 Using the corpus

Consider that we have a set of documents obtained as a result of a query to a search engine like Lucene. We claim that all the terms present in all of the documents present in the result set are related to the query words in some form or the other. We ignore stop words and other very frequently occurring words. For all the other words, we create pairs where the query term is paired with the terms in the resultant documents. We keep track of the number of times each of the pairs occurs. We use significance tests to identify which of the pairs are significant. These can then be the likely candidates for expansion of the given query term. Following is the algorithm to identify the related terms for a given term in the query:

```
Fire a query q to Lucene
Let S be the set of the documents returned by Lucene
For every doc d in S
    For every word in w in d
        If (q,w) is present then
            count++ for that pair
        else
            Form a pair (q,w)
    end For
end For
Use significance tests to identify the significantly correlated words.
```

Figure 6.1: Identifying correlated words using the corpus

In order to reduce the response time while querying, we build a term-to-term index. Where every term is mapped to the words that are related to it in such a way that these terms can replace the given term. This index is undoubtedly huge but it saves a lot of querying time.

6.1.2.2 Using WordNet

For many questions, it so happens that the words in the question do not syntactically match any of the passage words. The answer is present in passages where none of the question words are present in the exact syntactic form. For such questions, we require words that can replace the question words and retrieve the passages containing the answer in the ranking step.

We use WordNet synonyms for the question words to form a query for lucene. The query will be of this form :

```
(question_word1 OR synonym1 OR synonym2 ..)
AND (question_word2 OR synonym1 OR synonym2 ..)
AND ...
```

The important issues in this expansion using WordNet are as follows :

1. Numerous senses for each word

2. Drifting from the original meaning of the word

3. Too many synonyms for many words

We could also use other WordNet relations like hypernymy, meronymy for query expansion. The caveat with query expansion is the limitations of WordNet and increase in recall at the cost of topic drift and thus a fall in precision.

6.1.3 Handling Proper nouns

Many proper nouns may not appear in the corpus in the same form as they appear in a given question. For example, a question may contain *John F. Kennedy* while the corpus addresses the same person as *J.F.Kennedy* or *John Kennedy.* It is not always possible to disambiguate the proper nouns present. For example, in the above case, if the former President is addressed as just *Kennedy*, then it is difficult to map the term to *John F. Kennedy.* If we look up WordNet for *Kennedy*, it gives the following,

```
Overview of noun Kennedy
The noun Kennedy has 2 senses (no senses from tagged texts)

1. Kennedy, Jack Kennedy, John Fitzgerald Kennedy, JFK,
President Kennedy, President John F. Kennedy -- (35th President
of the United States; established the Peace Corps; assassinated
in Dallas (1917-1963))
2. Kennedy, Kennedy International, Kennedy International Airport
-- (a large airport on Long Island east of New York City)
```

Thus, WordNet gives the the variants of some of the frequently occurring proper nouns. However, for cases which are not present in WordNet, we propose the *pendulum algorithm* proposed here. We use a heuristic for matching 2 proper nouns. If we have 2 proper nouns say $X\ Y\ Z$ and $A\ B\ C$. If they refer to the same person, X should match with A or if one of them is just a character, it should match the other's first character. Same goes with Y and B but Z and C should match completely. Instead of giving a boolean (0/1) value as output, we output the extent to which the proper nouns are similar.

This intuition is captured in this algorithm. The steps of the algorithm are given in Figure 6.2.

6.1.4 Going beyond the TREC corpus

For the purpose of improving the recall, it was mandatory to not restrict ourselves to just the corpus provided by TREC. To this effect, we made an attempt to use the World Wide Web corpus as a supplementary corpus to get answer passages for the questions. We used the API provided by the Google search engine [1] to extract passages from the web.

We have made the following observations while querying Google :

- Definition questions can be answered better using the web
- Questions about *"famous"* or *"known"* entities can be better answered "Googling" the web

[1] http://www.google.com/apis/

```
Let pn_1 and pn_2 be the two proper nouns to be matched
    l_1 := wordLen(pn_1), l_2 := wordLen(pn_2)
    start_1 := 0, end_1 := wordLen(pn_1)
    start_2 := 0, end_2 := wordLen(pn_2)
    match := 0, count := 1
    while start_1 < end_1 && start_2 < end_2 do
        begin
            if (word@start_1 == word@start_2 && word@end_1 == word@end_2) ||
            (word@start_1.charAt(0) == word@start_2.charAt(0)
            && word@end_1 == word@end_2)
                match = 1/count; return match
                else start_1 ++, start_2 ++, end_1 --, end_2 --
            endif
        end
        return 0
```

Figure 6.2: Pendulum algorithm for matching Proper nouns

- The context surrounding the answers for questions for which Google was able to retrieve in the answer in the few top documents, is more relevant than that obtained in TREC

To this end, we obtain hits using Google in the ranking step and use a re-ranker to rank the top “k” passages. “k” is typically set to 100. The re-ranker is trained using the past logs of QA pairs provided by TREC.

6.1.5 Backoff Model vs More recall

To improve the performance of our system during the ranking step, we need to retrieve passages containing the answer in the top “k” hits. The previous subsections suggest that the query that was provided in our initial setup to either Lucene or Google was quite primitive. We say it is primitive mainly for the reason that the first step towards passage retrieval is a *keyword based* search. There is no attempt at bridging the *lexical chasm* or the *semantic chasm*. We suggest the following variations to the query to be fired to the search engine.

- Entire question retaining the quoted portion
- Entire question relaxing the quoted portion
- Selectors
- Selectors, focus words
- Question tokens without stop words, cue words
- Hard selectors, soft selectors along with their synonyms as obtained from WordNet or the corpus
- Variants of proper nouns, either obtained from WordNet or using the pendulum algorithm 6.2
- Noun phrases (NP chunks) as obtained from the parse of the question

We can use these variants of the query in two ways. We can either use a backoff model, wherein we fire each variant (in some order) and get the hits. Using a criterion to see if there is a need for backoff, we decide to backoff or use the hits obtained. The criterion used is the number of hits obtained for the current query. We have observed that if the number of hits is very large, the chances of getting the answer passage in top hits is less. It indicates that the query is general and could not get quite pertinent information. On the other hand, very few hits indicate that the query is too specific and hence we need some relaxation for it. However, this criterion may not hold in some cases and the backoff model can thus fail.

Another way to use to these query variants is to use the hits obtained by each of the variants together. We fire to Lucene each of the above variants and obtained say around 100 hits. We do the same using Google. We then combine all the hits and obtain around 1000 or so hits and then perform re-ranking on them. This is guaranteed to improve the recall at the ranking step.

6.2 Improving the re-ranking

There are some questions for which the answer passage is better ranked by Lucene than our re-ranker. After re-ranking, the answer passage is pushed down. We presented some example questions where this happens and also pointed out the possible reasons why this happens in Chapter 5.3. To re-iterate, this could possibly happen because of the following reasons :

1. The features for the re-ranking classifier are not appropriate and exhaustive
2. The feature set is appropriate but the values for the features, while the instances are generated, is not very effective
3. The Named Entity tagging is inadequate

To combat the first problem, we propose an enhanced feature set where we add the following features to the already existing feature set.

1. The score (between 0 and 1) returned by the focus word identifier for the token(s) which has(ve) maximum similarity with the answer token(s). For question whose *wh-word* acts as the focus, this feature is set to 1.
2. The similarity of the atype with the NE tag of the answer token(s). For example, for a *who* question, if the answer token has a NE tag of *person*, this feature is set to 1.

To combat the second problem, we define enhanced and improved WordNet similarity measures for connecting the atype with the answer tokens. Following subsection gives a detailed information about the same.

6.2.1 Similarity Measures

In the previous chapter, we gave a brief account of the similarity measures that we used for connecting the atype in the question with the possible answer tokens. These similarity measures are effective but they are computationally expensive. We try to make the task of identifying the similarity between a pair of words as efficient as possible. In order to do that, we do a couple of tasks offline. Also, we use the WordNet descriptive glosses and define similarity measures that use the gloss overlap. We also build glosses using the hypernymy and meronymy relations given by WordNet.

The task is to find similarity between two words-senses. Each word-sense consists of an associated part of speech tag and a sense number. For words present in WordNet, the sense number corresponds to WordNet sense number. Words not from WordNet will default to having only one sense. How this is done will be described later in this section.

There are two parameters for finding similarity between a pair of words - the type of gloss used and the similarity measure used.

6.2.1.1 Descriptive glosses

A word, with its associated part of speech and an associated sense number, has a description.

Description for first noun sense of "Volcano" a fissure in the earth's crust (or in the surface of some other planet) through which molten lava and gases erupt

Figure 6.3: An example *descriptive-gloss* from WordNet

We call these descriptions as *descriptive glosses*. For word-senses picked up from WordNet, the WordNet glosses are the descriptive glosses. WordNet glosses also contain example usages of the word. We have dropped out examples from the descriptive glosses.

For other word-senses, the descriptions could come from glossaries (like glossaries of software terms), encyclopedias (for names of people, places, events, pacts *etc*), world fact books, abbreviation lists *etc*. Examples glosses picked up from above sources are listed below.

descriptive-gloss for "Tuareg" A white-skinned nomadic people of the Sahara who speak an Hamitic Berber **descriptive-gloss for "piccolo"** an instrument of the woodwind family. Most of these instruments were once of made of wood, and because they are played by blowing with air or wind, they are called woodwind. **descriptive-gloss for "DARPA"** The Defense Advanced Research Projects Agency is the central research and development organization for the Department of Defense in the U.S.

6.2.1.2 Hypernymy glosses

The gloss for a particular sense of a word could also describe what hierarchical categories it belongs to. For instance, the hierarchical categorization of the first noun sense of the word "Vesuvius" is

```
=> volcano
      => mountain, mount
          => natural elevation, elevation
              => geological formation, formation
                  => natural object
                      => object, physical object
                          => entity
```

Based on this hierarchical categorization of the first noun sense of "Vesuvius", we describe its *hypernymy-gloss* as the collection of all nodes in its hypernymy-path to the root - *viz.* "entity".

> **Hypernymy gloss for first noun sense of "Vesuvius"** {volcano}, {mountain, mount}, {natural elevation, elevation}, {geological formation, formation}, {natural object}, {object, physical object}, {entity}

Figure 6.4: First noun sense of Vesuvius

Whereas *descriptive-glosses* can be derived even for word-senses not present in WordNet, *hypernymy-glosses* require classification of word-senses into nodes into an ontological structure - like the hypernymy hierarchy of WordNet. This is not that easy to procure for words not present in WordNet. One possible solution, and the one that we actually resort to, is to find the named entity tag for a token (if one exists) and then map the tag to a node in WordNet. For example, the token "President Musharraf" is not present in WordNet. But this token can be tagged as a *PERSON* and *PERSON* could be mapped to a node in WordNet - *viz.* the first noun sense of "person" (person#n#1). Similarly, the token "26^{th} December 2003" has a DATE named-entity tag. DATE could be translated to the 7^{th} sense of the word "date" (date#n#7) in WordNet.

Thus, *hypernymy-glosses* of many words not present in WordNet (which currently default to having only one sense) could be evolved from their named-entity tags.

Another parameter for measuring similarity between two word-senses is the similarity metric. Given two word senses, once we pick their glosses, we need to find similarity between the word senses, based on the glosses choses. We next describe different similarity metrics.

6.2.2 Similarity metrics

The similarity between two word-senses, for a given gloss is found by tokenising each gloss, constructing vectors from the two tokenised glosses and finding similarity between the vectors.

Cosine similarity

One standard metric of similarity, as used in information retrieval, is the cosine-similarity. We find the cosine similarity between the *term frequency-inverse gloss frequency (tfigf)* vectors of the two glosses. The *inverse gloss frequency (igf)* of a token is the inverse of the number of glosses which contain that token and it captures the "commonness" of a particular token.

There have been fancier definitions of similarity in literature [31] which involve information theoretic measures of similarity between word-senses, based on the hypernymy path and DAG structure of WordNet. These methods are heavily dependent on frequencies of synsets in a sense-tagged corpus. The idea is that two word-senses are highly related if their subsuming synsets are highly information bearing - or in other words, have high information content. Information content is computed from a sense tagged corpus - word-senses with *high* frequencies of occurrence have *low* information content. This brings in the the problem of data sparsity - because sense-tagged corpora are very scarce and of short size. Their coverage of synsets is poor as well. Hence there is the danger of making the similarity measure biased toward the sense-tagged corpus.

Also, these methods are very slow and CPU intensive, since finding similarity between two word-senses at run time involves traversing the WordNet graph, in the direction of hypernymy links, up to the least common ancestor.

On the other hand, a cosine similarity on tfigf vectors built from *hypernymy-glosses*, gives a low similarity value between word-senses whose hypernymy-glosses overlap in very frequently occurring synsets relative to the synsets which are not common to their glosses. This is because *igf* implicitly captures the information content of a synset - the higher the igf - higher is the information content of a synset. The purpose served by a sense-tagged corpus is cumulatively served by the collection of hypernymy glosses of all the WordNet synsets. This method is also more reliable since the *igf* values come from WordNet which is very exhaustive, unlike sense tagged corpora (like SemCor) which will have bias and data-sparsity in terms of which words occur in the corpus and which sense is picked for a word. (The reader might want to note some work which has been done to illustrate that words can inherently have multiple senses in a given context).

The cosine similarity on tfigf vectors built from *descriptive glosses* is very much like the similarity found between document and query vectors, since the tokens in descriptive glosses are regular words. Cosine similarity is intuitively the most useful similarity measure on descriptive glosses since cosine similarity of *tfigf* vectors takes care of stop words and very non-informative words like "the" *etc.*

Jaccard similarity

Another metric of similarity is the *jaccard similarity*. Jaccard similarity between two sets of tokens (glosses) is computed as $\frac{|A \cup B|}{|A \cap B|}$. Here A and B are the two glosses.

Jaccard similarity is appealing only if the glosses used are *hypernymy-glosses*.

Asymmetric measures of similarity

The above two were symmetric measures of similarity. A third asymmetric similarity measure is one that takes a value of 0 if the intersection of the glosses of two word-senses is not equal to the gloss of one of the word-senses. Else, the similarity is equal to one of cosine or jaccard similarity measures. This means that there are actually two asymmetric similarity measures - one due to jaccard and the other due to cosine.

The asymmetric version of *jaccard similarity* was used for question answering. Symmetric measures can be misleading for question answering. This is because, in QA, we are looking for an answer that is *subsumed by* the *answer-type* token in the question **or** an answer that *subsumes* the *answer-type* token in the question.

Question: What state has the most Indians?
Answer : California , Atype token = state
Question: How much folic acid should an expectant mother get daily?
Answer : 400 micrograms (MEASURE/MICROGRAMS), Atype token = how much (QUANTITY)
Question: Where is Belize located?
Answer : Central America , Atype token = where (LOCATION)
Question: Who invented the paper clip? Answer : Norwegian Johan Varler (PERSON) , Atype token = who (PERSON)
Question: Name a flying mammal
Answer : bats , Atype token = mammal
Question: Where's Montenegro?
Answer : Yugoslavia , Atype token = where (LOCATION)
Question: When did the Vesuvius last erupt? Answer : 1944 (YEAR) , Atype token = when (PERIOD)
Question: Name one of the major gods of Hinduism Answer : Brahma , Atype token = god

Figure 6.5: Example QA pairs where the answer token is subsumed by the Atype token in the question

Question: What was the name of the famous battle in 1836 between Texas and Mexico?
Answer : Alamo , Atype token = battle
Reason for failure is that "battle" and "alamo" are both siblings of "group action" in WordNet
Question: What did Vasco da Gama discover ?
Answer : sea route , Atype token = Gama
Reason for failure is that wrong Atype was selected. This motivates the need for keeping in reserve one or more other Atype clues to fall back on

Figure 6.6: Example QA pairs where the asymmetric similarity fails

Hence, the measure of similarity we use between a candidate answer token Ans, and the given Atype word Q, is:

$$sim(Ans, Q) = assymJac(Ans, Q) + assymJac(Q, Ans). \tag{6.1}$$

It will be an interesting problem to determine which of the two components on R.H.S to pick, depending on other tokens in the question (*e.g.* clues like "kind" and "type").

6.2.3 Named Entity Tagging

As pointed out in chapter 5, one of the drawbacks of our current approach is insufficient named entity tagging. For example, in the answer passage for the question,

> What group sang the song "Happy Together"? namely,
>
> The Turtles were another victim. When California teenagers Mark Volman and Howard Kaylan signed with White Whale records in the '60s, they apparently didn't read the fine print. The label not only owned all the masters to their songs (including "Happy Together"), it owned the band's name as well.

we need to know that "The Turtles" is-a group. The WordNet similarity measures that we use to connect the atype (group in this example) with the answer tokens will not be effective in this case if "The Turtles" is not present in WordNet. If we have a NE tagger that tags Turtles with the "group" tag then it can be easily identified as the answer. With the trainable NE tagger that we have built (refer 3.4), the tag set is not restrictive. If the NE tagger is provided with appropriate training data, it can tag the questions with the required tags.

6.3 Need to model the answer passage

We use a question to query our corpus. In order to retrieve the answer passage, we extract fragments from the question that are useful in getting to the answer passage. The fragments from the question are extracted based on the model that has been proposed for the question. However, after looking at the results obtained from the current QA setup, we find it mandatory to fix a model for the passage too.

Following is the model proposed for the passage that is likely to contain the answer.

- There is atleast one word in the passage that matches a word in the question. Morphing and stemming is allowed for this word(s). These words need to be one of the *hard selectors* identified in the question. These words lie in close proximity to the answer. As per this model, we do not account for the TREC questions where the answer passage provided does not contain even a single word overlap with the question. For example, *What is a nematode?* , the answer passage does not contain the hard selector *nematode*. We do not account for such a question.

- The passage contains atleast one word that has high similarity (is-a similarity measure used) with the atype clue identified in the question. For example, *What state has the most Indians*, for this question, the atype identified *i.e. state* should have high similarity with a passage word. The corpus does not contain any such passage hence we do not account for this question.

6.4 Using the Web as an alternative corpora

A QA system that scales to the Web can have enormous value. This is because the Web is the most popular and ever-expanding corpora of open-domain network of hypertext documents. The issues involved in scaling our system to the web can cause us to deviate from the actual problem that we are addressing. Hence, we use the web as another TREC-like corpus and use the Google search engine [2] for retrieving *passages* from the web corpus. The TREC corpus is limited in information and some questions even from the TREC questions list cannot be answered using the same. To combat this problem, the use of passages obtained from Web documents is effective.

6.4.1 Ranking

The query or the identified selectors are fired to Google and the top 100 passages are retrieved. For most definition questions such as *"What is caldera?"*, the answer is present in top passages. These passages are then cleaned, POS-tagged, NE-tagged and thus prepared for the re-ranking step.

6.4.2 Re-ranking

We use the same setup for re-ranking the passages as the earlier one. The passages retrieved by Google and prepared for re-ranking are re-ranked using the trained logistic regressor. A little different set of features are used. In addition to the earlier set of re-ranking features, the score returned by the focus-word identifier for the question words is also used. This is used with the intuition that the answer tokens will have high similarity with the focus words in the question, *i.e.* the question words for which the regressor for focus word learning returns a high score.

[2] `http://google.com`

Chapter 7

Conclusion

We built a QA system using an approach that breaks the current culture in QA, which relies on human expertise for building the various modules of a QA system. Our current approach helped us to come up with a system that has absolutely no tunable or magical parameters as some of the current QA systems have. The performance of the system is comparable with the high-fliers at competitions such as TREC. Our system is easy to deploy and is available off-the-shelf, as is the case with IR systems.

We wish to make the basics of QA simple enough as are the basics of the vector-space model and TFIDF, used in most of the search engines can be taught in an hour. Thus, in a way, we wish to match the problem of QA to one of the problems that falls under the domain of Information Retrieval (IR). However, our experience with QA, suggests that QA is a far more difficult and profound problem compared to the other problems in Information Retrieval (IR). Most of the problems that fall under the domain of Information Retrieval (IR) involve querying and searching for keywords in a semi-structured corpus of data. However, QA, requires us to use not only those techniques but also ones in Natural Language Processing (NLP), Machine Learning, Artificial Intelligence (AI), Information Extraction (IE) and Mining. This realization cautions us against using techniques predominantly from one of the domains.

It has been possible to get rid of the tunable parameters by making the system learnable with the help of logs of question-answer pairs and to incorporate the IR culture of easy deployment and off-the-shelf availability. However, there is always a possibility of encountering a trade-off between ease of deployment, structure in approach and the system's MRR. We do not want to lose out on the MRR by making the approach simple and structured. Current QA systems blend IR, NLP and mining techniques in varied styles giving emphasis to one domain over the rest. It is difficult to identify the domain to which this problem of Question Answering (QA) exclusively belongs. This difficulty of identifying the domain poses a problem of comparison between the performances of different QA systems. One system may come up with an objective of improving the performance of the system only in terms of MRR while some other systems might put more stress on issues of learnability of the system from past QA pair logs and more structure in the approach. A measure such as the Mean Reciprocal Rank (MRR) to compare such systems with diverse objectives is quite insufficient.

By way of solving the QA problem, we have come up with several generic techniques that are applicable to other problems in the IR domain. The idea of focus words, WordNet-based similarity measures and application of Machine Learning techniques to avoid intense human efforts that are required for certain tasks, are independent of the QA problem. These ideas can be used for solving problems where shallow NL-techniques will suffice.

Bibliography

[1] S. Abney, M. Collins, and A. Singhal. Answer extraction. In *ANLP*, 2000.

[2] E. Agichtein, S. Lawrence, and L. Gravano. Learning search engine specific query transformations for question answering. In *Proceedings of the 10th World Wide Web Conference (WWW10)*, pages 169–178, 2001.

[3] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases–an introduction. *Journal of Language Engineering*, 1(1):29–81, 1995.

[4] Apache Software Group. Jakarta lucene text search engine. GPL Library, 2002. `http://jakarta.apache.org/lucene/`.

[5] E. Brill, S. Dumais, and M. Banko. An analysis of the AskMSR question answering system. AskMSR, 2002.

[6] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at EDBT'98*, 1998.

[7] M. E. Califf and R. J. Mooney. Bottom-up relational learning of pattern matching rules for information extraction. In *Journal of Machine Learning Research*, pages 177–210, 2003.

[8] E. Charniak. A maximum-entropy-inspired parser. Technical Report CS-99-12, 1999.

[9] C. Clarke, G. Cormack, D. Kisman, and T. Lynam. Question answering by passage selection (MultiText experiments for TREC9). In E. Voorhees and D. Herman, editors, *The Ninth Text REtrieval Conference (TREC 9)*, NIST Special Publication 500-249, pages 673–683. NIST, 2000.

[10] C. L. A. Clarke, G. V. Cormack, and T. R. Lynam. Exploiting redundancy in question answering. In *Research and Development in Information Retrieval*, pages 358–365, 2001.

[11] M. J. Collins and M. P. Marcus. *Head-driven statistical models for natural language parsing.* PhD thesis, 1999.

[12] H. Cunningham, K. Humphreys, R. Gaizauskas, and Y. Wilks. GATE: A TIPSTER-based general architecture for text engineering. In *Proceedings of the TIPSTER Text Program (Phase III) 6 Month Workshop*. Morgan-Kaufmann, may 1997.

[13] M. De Boni and S. Manandhar. Automated discovery of telic relations for wordnet. In *Proceedings of the first International WordNet conference*, India, 2002.

[14] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. Semtag and seeker: bootstrapping the semantic web via automated semantic annotation. In *The Twelfth International World Wide Web Conference*, 2003.

[15] S. Dumais, M. Banko, E. Brill, J. Lin, and A. Ng. Web question answering: Is more always better? In *SIGIR*, pages 291–298, aug 2002.

[16] C. Fellbaum. Wordnet: An electronic lexical database. In *MIT Press*, 1998.

[17] M. Fleischman, E. Hovy, and A. Echihabi. Offline strategies for online question answering: Answering questions before they are asked. In *ACL*, pages 1–8, 2003.

[18] P. B. Ganesh Ramakrishnanan. Text representation with wordnet synsets using soft sense disambiguation. In *NLDB*, pages 214–227, 2003.

[19] S. Harabagiu, D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Girju, V. Rus, and P. Morarescu. FALCON: Boosting knowledge for answer engines. In *Proc. of Ninth Text REtrieval Conference (TREC 9)*, pages 479–488, 2000.

[20] S. Harabagiu, D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Girju, V. Rus, and P. Morarescu. Falcon: Boosting knowledge for answer engines. In *Proc. of Ninth Text REtrieval Conference (TREC 9)*, pages 479–488, 2000.

[21] S. M. Harabagiu, G. A. Miller, and D. I. Moldovan. Wordnet 2 a morphologically and semantically enhanced resource. In *Proceedings of SIGLEX-99*, pages 1–8, Univ. of Maryland, 1999.

[22] E. . Hovy, L. Gerber, U. Hermjakob, M. Junk, and C.-Y. Lin. Question answering in webclopedia. In *Proceedings of the TREC-9 Conference. NIST, Gaithersburg, MD*, 2000.

[23] V. Hristidis, Y. Papakonstantinon, and A. Balmin. Keyword proximity search on xml graphs. In *Proceedings of 18th International Conference on Data Engineering, San Jose, USA*, 2002.

[24] A. Jadhav. Question answering for the web. Technical report, Department of Computer Science and Engg, IIT Bombay, mar 2003.

[25] N. Jain. Graph proximity for question answering. Technical report, Department of Computer Science and Engg, IIT Bombay, may 2003.

[26] B. Katz and J. Lin. Selectively using relations to improve precision in question answering. In *Proceedings of the EACL-2003 Workshop on Natural Language Processing for Question Answering, April 2003*, 2003.

[27] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Proceedings of the ACL Conference*, 2003.

[28] D. Klein and C. D. Manning. Fast exact inference with a factored model for natural language parsing. In S. T. S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 3–10. MIT Press, Cambridge, MA, 2003.

[29] C. C. T. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. In *Proceedings of the 10th World Wide Web Conference (WWW'10)*, pages 150–161, 2001.

[30] S. K. Lam, D. M. Pennock, D. Cosley, and S. Lawrence. 1 billion pages = 1 million dollars? mining the web to play "who wants to be a millionaire?". In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence, UAI 2003*, Acapulco, Mexico, August 8–10 2003.

[31] D. Lin. An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304, San Francisco, CA, 1998. Morgan Kaufmann.

[32] J. Lin, D. Quan, V. Sinha, K. Bakshi, D. Huynh, B. Katz, and D. R. Karger. What makes a good answer? the role of context in question answering. In *Proceedings of the Ninth IFIP TC13 International Conference on Human-Computer Interaction (INTERACT 2003)*. Zurich, Switzerland, 2003.

[33] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing.* MIT Press, Cambridge, MA, 1999.

[34] O. Mason. Qtag - a portable pos tagger. Technical report.

[35] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Introduction to WordNet: An online lexical database. *International Journal of Lexicography*, 1993.

[36] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to wordnet: an on-line lexical database. *International Journal of Lexicography 3 (4)*, pages 235 – 244, 1990.

[37] D. I. Moldovan and V. Rus. Explaining answers with extended wordnet. 2003.

[38] C. Monz and M. de Rijke. Tequesta : The university of amsterdam's textual question answering system. Tequesta : The university of Amsterdam's Textual Question Answering System, 2002.

[39] S. R. and W.Li. A question answering system supported by information extraction. In *Proceedings of the 1 st Meeting of the North American Chapter of the Association for Computational Linguistics (ANLP-NAACL-00)*, pages 166–172, 2000.

[40] D. Radev, W. Fan, H. Qi, H. Wu, and A. Grewal. Probabilistic question answering on the web. In *World-wide Web Conference*, pages 408–419, 2002.

[41] G. Ramakrishnan. Semantic disambiguation of text. Technical report, Department of Computer Science and Engg, IIT Bombay, mar 2002.

[42] G. Ramakrishnan and P. Bhattacharyya. A soft sense disambiguation approach. In *8th International Conference on Applications of Natural Language to Information Systems*, Burg, Germany, 2003.

[43] G. Ramakrishnan, S. Chakrabarti, D. Paranjpe, and P. Bhattacharya. Is question answering an acquired skill? In *Proceedings of the 13th conference on World Wide Web*, 2004.

[44] G. Ramakrishnan, A. Jadhav, A. Joshi, S. Chakrabarti, and P. Bhattacharyya. Question answering via bayesian inference on lexical relations. 2003.

[45] Ravichandran and Hovy. Learning surface text patterns for a question answering system. In *Proceedings of ACL*, 2002.

[46] S. E. Robertson, S. Walker, M. Hancock-Beaulieu, A. Gull, and M. Lau. Okapi at TREC. In *Text REtrieval Conference*, pages 21–30, 1992.

[47] R. Shachter. Bayes-ball: The rational pastime for determining irrelevance and requisite information in belief networks and influence diagrams. In *Uncertainty in Artificial Intelligence*, June 1998.

[48] D. D. Sleator and D. Temperley. Parsing English with a link grammar. In *Third International Workshop on Parsing Technologies*, 1993.

[49] S. Tellex, B. Katz, J. Lin, A. Fernandes, and G. Marton. Quantitative evaluation of passage retrieval algorithms for question answering. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 41–47. ACM Press, 2003.

[50] M. L. The. Analyses for elucidating current question answering technology.

[51] I. H. Witten and E. Frank. *Data mining: practical machine learning tools and techniques with Java implementations.* Morgan Kaufmann Publishers Inc., 2000.

[52] Z. Zheng. Answerbus question answering system. In *Human Language Technology Conference (HLT)*, 2002.

Appendix A

Primers

A.1 Natural Language Parsing

Natural Language typically contains two kinds of structures, one is the *syntactic* structure and the second is the *semantic* structure. Syntactic structure has standardly been described in terms of categories (phrasal labels and word classes), with little mention of particular words. This is possible, since, with the exception of certain common function words, the acceptable syntactic configurations of a language are largely independent of the particular words that fill out a sentence. Semantic structure is usually described by the various lexical dependencies in the sentences and across sentences.

Syntactic structure can be captured by a context-free-grammar and a PCFG kind of a model suffices in many cases. Unlike syntactic structure where no use of particular words is required, for resolving the important attachment ambiguities of modifiers and arguments, lexical preferences are known to be very effective. Additionally, methods based only on key lexical dependencies have been shown to be very effective in choosing between valid syntactic forms. For identifying the lexical dependencies between sentence tokens, a *dependency syntax* based *dependency grammar* is used.

A.1.2 Lexicalized Probabilistic Parser

Statistical parsers such as [11, 8] standardly use complex joint models of over both category labels and lexical items, where "everything is conditioned on everything" to the extent possible within the limits of data sparseness and finite computer memory. For example, the probability that a verb phrase will take a noun phrase object depends on the head word of the verb phrase. A VP headed by "acquired" will likely take an object, while a VP headed by "agreed" will likely not. There are certainly statistical interactions between syntactic and semantic structure, and, if deeper underlying variables of communication are not modeled, everything tends to be dependent on everything else in language [4]. However, the above considerations suggest that there might be considerable value in a factored model, which provides separate models of syntactic configurations and lexical dependencies, and then combines them to determine optimal parses. For example, under this view, we may know that "acquired" takes right dependents headed by nouns such as company or division, while "agreed" takes no noun-headed right dependents at all. If so, there is no need to explicitly model the phrasal selection on top of the lexical selection.

[28] presents a generative model for natural language tree structures in which semantic (lexical dependency) and syntactic (PCFG) structures are scored with separate models. It is claimed there-in that the factorization provides conceptual simplicity, straightforward opportunities for separately improving the component models, and a level of performance comparable to similar, non-factored models. It admits to be using an extremely effective A* parsing algorithm, which enables efficient, exact inference in the Lexicalized Probabilistic parser that they have come up with.

The lexicalized probabilistic parser implements a factored product-of-experts model, with separate PCFG phrase structure and lexical dependency experts, whose preferences are combined by efficient exact inference, using an A* algorithm.

The parser API returns a parse tree for a piece of text. One can get the various phrases at the different heights in the parse tree. For a sentence such as, *The Ganga is a sacred river.*, the output of the parser is as follows.

```
At depth = 0
The, Ganga, is, a, sacred, river, .

At depth = 1
(DT The), (NNP Ganga), (VBZ is), (DT a), (JJ sacred), (NN river), (. .)

At depth = 2
(NP (DT The) (NNP Ganga)), (NP (DT a) (JJ sacred) (NN river))

At depth = 3
(VP (VBZ is) (NP (DT a) (JJ sacred) (NN river)))
```

```
Corresponding parse tree :

                VP
               /  \
   NP         /    NP
  /  \       /    /  | \
 DT  NNP   VBZ DT   JJ  NN
  |    |     |  |    |    |
 The Ganga is a sacred river
```

Some of the drawbacks of the parser include the need for plenty of memory (a minimum of 120Mb to run as a PCFG parser on sentences up to 40 words in length; typically around 512Mb of memory to be able to parse similarly long typical-of-newswire sentences using the factored model).

A.2 WordNet Primer

WordNet is an online lexical reference system in which English nouns, verbs, adjectives and adverbs are organized into synonym sets or synsets, each representing one underlying lexical concept. Noun synsets are related to each other through hypernymy (generalization), hyponymy (specialization), holonymy (whole of) and meronymy (part of) relations. Of these,(hypernymy, hyponymy) and (meronymy,holonymy) are complementary pairs. The verb and adjective synsets are very sparsely connected with each other. No relation is available between noun and verb synsets. However, 4500 adjective synsets are related to noun synsets with pertainyms (pertaining to) and attra (attributed with) relations.

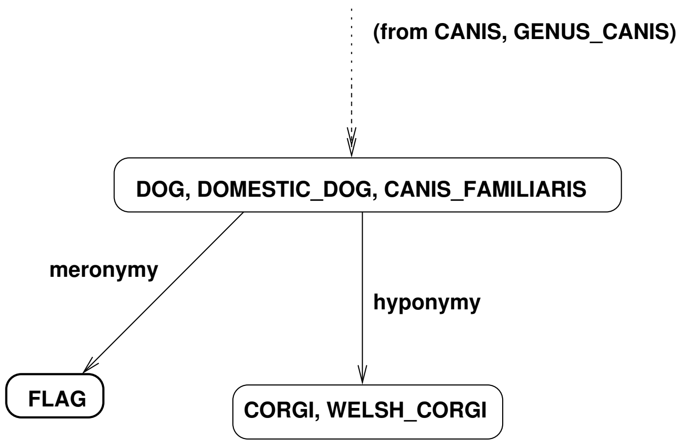

Figure A.1: WordNet relations

Figure A.1 shows that the synset {dog, domestic dog, canis familiaris} has a hyponymy link to { corgi, welshcorgi} and meronymy link to flag (*a conspicuously marked or shaped tail*). While the hyponymy link helps us answer the question *A corgi is a kind of what?*, the meronymy connection here is perhaps more confusing than useful: this sense of flag is rare.

For our purposes, WordNet is a graph where nodes are concepts and edges are relations between concepts. A concept is called a "synset" because is described by a set of synonyms, also called "lemmas". A synset may be described by many lemmas. Conversely, a lemma (like "match") can describe many synsets, in which case it is highly *polysemous*. We only consider *hypernym* and *hyponym* edges which represent IS-A relations. E.g., in this chain of generalization:

> horse, Equus caballus $\rightarrow$ equine, equid $\rightarrow \cdots \rightarrow$ ungulate, hoofed mammal $\rightarrow$ placental, placental mammal, eutherian, eutherian mammal $\rightarrow$ mammal $\rightarrow \cdots \rightarrow$ animal, animate being, beast, brute, creature, fauna $\rightarrow \cdots \rightarrow$ entity

beast and *brute* are synonyms, *equid* is a **hyponym** of *ungulate*, *horse* is a **hyponym descendant** of *mammal*, *equid* is a **hypernym** of *horse*, and *entity* is a **hypernym ancestor** of *horse*.

A.3 Lucene Primer

Lucene is a free text-indexing and text-searching API written in Java[1]. There are three main parts to know about Lucene and they are the indexing, searching and ranking/scoring mechanisms.

A.3.1 Indexing

The Lucene index format is a directory structure of several files. An index contains a sequence of documents.

- **Documents:** Documents are the primary retrievable units from a Lucene query. Documents consist of a sequence of fields. Fields have a name ("contents" and "filename" in our example). Field values are a sequence of terms.

- **Terms:** A term is the smallest piece of a particular field. Fields have three attributes of interest,

  1. Stored – Original text is available in the documents returned from a search.
  2. Indexed – Makes this field searchable.
  3. Tokenized – The text added is run through an analyzer and broken into relevant pieces (only makes sense for indexed fields).

  Stored fields are handy for immediately having the original text available from a search, such as a database primary key or filename. Stored fields can dramatically increase the index size, so use them wisely. Indexed field information is stored extremely efficiently, such that the same term in the same field name across multiple documents is only stored once, with pointers to the documents that contain it.

- **Analysis:** Does case matter for searching? What are word boundaries? Are acronyms, email addresses, URLs, and other such textual constructs kept intact and made searchable? If a singular word is indexed, should searching on the plural form return the document? These are all very interesting and complex questions to ask when deciding on which analyzer to use, or whether to create your own. Lucene has its built-in StandardAnalyzer and there are other built-in analyzers as well as some optional ones (found currently in the Lucene "sandbox" CVS repository) that can be used.

A.3.2 Searching

Lucene provides a searching mechanism for search for a the *query* on its *index* that is built on the required text. Lucene provides a number of query formats for constructing the Query. Using different kinds of queries it supports wildcard searches, boolean searches, range searches, proximity searches and fuzzy searches.

[1] `http://jakarta.apache.org/lucene/`

A.3.3 Scoring

After searching, a Hits collection is returned. The hits returned are ordered by Lucene's determination of score. Lucene uses the well-known Okapi *bm25* weighing scheme. [46] contains the details of the scoring function.

Appendix B

Evaluation Tests

B.1 Mean reciprocal rank (MRR)

Suppose $n_q \geq 1$ is the earliest rank of the passage at which the answer to question $q \in Q$ is found. Then MRR is defined as $(1/|Q|)\sum_q(1/n_q)$. MRR is between 0 and 1, and a higher MRR is better. Note that the penalty for dropping an answer from rank 1 to rank 2 is the same as the penalty for dropping the answer from rank 2 to rank ∞ (i.e., not report it at all). Some people disagree with this policy.

In *lenient* scoring, the answer to a question is "found" in a passage if the passage satisfies a regular expression published by TREC. In *strict* scoring, the passage must also come from one of a list of documents approved for the question. Accordingly, the MRR is called strict MRR or lenient MRR. We have used lenient MRR for our experiments.

For the last few years, NIST (organizers of TREC) have been reporting *accuracy@1*: the fraction of passages at rank 1 that contain answers. This can also be strict or lenient. As Radev *et al.* [40] point out, MRR makes finer distinctions between ranking algorithms than accuracy@1 and better guides algorithm design.

There is also a distinction made between the MRRs that are computed using the ranks only upto 5 and making scores 0 for all the results beyond 5, and the MRRs calculated using all the 100 or so results. We call the first as MRR@5 and the latter MMR@in indicating the MRR being computed till infinity.

B.2 Significance Tests

In order to find out the synsets whose frequency of occurrences stand above the background distribution of frequencies, we use some significance tests. The following steps are used to apply the significance tests.

Significance tests:

1. Collect a background distribution of synset frequencies from the corpus

2. For each node n in the prefix tree and for each synset s at n, test the hypothesis that s appears more often at n than in the background distribution

3. Retain only those synsets that survive the test

	C2	¬C2
C1	c_{11}	c_{12}
¬C1	c_{21}	c_{22}

Figure B.1: Contingency Table

We experimented with three kinds of significance tests namely, the χ^2 test, the *likelihood ratio* and the *t-test*. A short description of the χ^2 test is provided below.

- χ^2 ***test*:** The χ^2 significance test compares the difference between the actual frequencies (the observed frequencies in the data) with those which one would expect if no factor other than chance had been operating (the expected frequencies). The closer these two results are to each other, the greater the probablity that the observed frequencies are influenced by chance alone. The way we use the χ^2 significance test is as follows. We construct a contingency table as shown B.1. C1 and C2 are two events. For example, in the case of this experiment of identifying the mapping between the *wh-words* and the WordNet synsets, the event of a *wh-word* occuring is C1 while the event of a particular synset occuring is C2. The significance level is calculated as follows,

$$S = \frac{N * ((c_{11} * c_{22}) - (c_{12} * c_{21})) * ((c_{11} * c_{22}) - (c_{12} * c_{21}))}{(c_{11} + c_{12}) * (c_{11} + c_{21}) * (c_{12} + c_{22}) * (c_{21} + c_{22})}$$

  where
  S : Significance level
  N : Total number of synsets

Appendix C

TREC 2000 and TREC 2002 Diagnosis

C.1 TREC-2000

C.1.1 Abbreviation Questions

1. What does NAFTA stand for?
2. What does EKG stand for?
3. When was the NFL established?
4. What does NASA stand for?
5. What is a nickname for Mississippi?
6. CNN is an acronym for what?
7. What is the nickname of Pennsylvania?
8. What's the abbreviation for limited partnership?

C.1.2 Definition Questions

1. What is a stratocaster?
2. What is tyvek?
3. What is a nanometer?

C.1.3 Description Questions

1. Why can't ostriches fly?

C.1.4 Questions with generic and inconclusive Atypes

1. What is the wingspan of a condor?
2. What is the English translation for the word "caliente"?
3. What was the distinguishing mark on the "Little Rascals" dog?
4. What does Nicholas Cage do for a living?
5. What is the busiest air travel season?
6. What is Chiricahua the name of?
7. What is the Jewish alphabet called?
8. What English word has the most letters?
9. What is the mascot for Notre Dame University?
10. What is the exchange rate between England and the U.S.?
11. Where is your corpus callosum?
12. The numbering system we use today was introduced to the western world by what culture?

C.1.5 Unknown AType

1. What is the zip code for Parsippany, NJ?
2. What is the chemical formula/name for napalm?

C.1.6 Question with too many common words

1. How many highway miles to the gallon can you get with the Ford Fiesta?
2. What is the hair style called that new military recruits receive?
3. In what country is a stuck-out tongue a friendly greeting?
4. What is the name of the company that manufactures the "American Girl" doll collection?
5. Where does chocolate come from?
6. What time of year is air travel the heaviest?
7. What hair color can I use to just cover a little gray?
8. How much does one ton of cement cost?

C.1.7 Corpus does not conform to the passage model

1. How tall is kilamanjaro?
2. When was Dick Clark born?
3. When is Dick Clark's birthday?
4. What is Dick Clark's date of birth?
5. When was "the Great Depression"?
6. Who wrote the song, "Boys of Summer"?

C.1.8 Insufficient NE tagging

These and the ones given for the reason of inconclusive atypes can overlap.

1. What is the most heavily caffeinated soft drink?
2. What is worth seeing in Reims?

C.1.9 Others

The questions which could not be categorized into the above failure categories are given below. The reason for failure of each of them is given at the side of each question.

1. What is the name of the inventor of silly putty? $->$ passage does not conform to the model
2. How many people watch network television? $->$ need atype alongwith variations to be used in the query
3. What's the population of Biloxi, Mississippi? $->$ answer not present in the corpus
4. What is Giorgio Vasari famous for? $->$ difficult atype
5. When did Muhammad live? $->$ ambiguous question

C.2 TREC-2002

C.2.1 Abbreviation Questions

1. What does CPR stand for?
2. What does DEA stand for?
3. What does the abbreviation WASP mean?
4. What does DNA stand for?
5. What does NASDAQ stand for?
6. What does R&B stand for?

C.2.2 Description Questions

1. If something has petrified, what has it turned into?
2. How did Molly Shannon's mother die?
3. The sun is mostly made up of what two gasses?

C.2.3 Questions with generic and inconclusive Atypes

1. What lays blue eggs?
2. How do you say "house" in Spanish?
3. What is slang for a "five dollar bill"?
4. How much of U.S. power is from nuclear energy?
5. What is the national anthem in England?
6. What country's leader was awarded the 2000 Nobel Peace Prize?
7. What is the North Korean national anthem?
8. What is the scientific name for tobacco?

C.2.4 Question with too many common words

1. What is the name of the professional baseball team in Myrtle Beach, South Carolina?
2. How many states were still united after the southern states seceded?
3. What was the first name of the Gehrig who played for the New York Yankees?

C.2.5 Others

The plausible reason for failure of the following questions is given at the side of each of question.

1. How high is Mount Kinabalu? $->$ passage does not conform to the passage model
2. What is the capital of Syria? $->$ simple but the corpus does not contain the required passage
3. What is the fear of lightning called? $->$ compound word detection required
4. What college did Magic Johnson attend? $->$ for getting connected to the answer, we need to know that in the answer passage, the name is that of a college
5. What is the name given to a collection of poetry? $->$ compound word detection required
6. Who succeeded Ferdinand Marcos? $->$ succeeded is ambiguous
7. How much are tickets to Disney World? $->$ how much should map to currency ($) and given in the query

8. Who said "Music hath charm to soothe the savage beast"? $->$ not present in the corpus
9. What is the state bird of Alaska? $->$ compound word detection required
10. What is the capital of Kentucky? $->$ passage does not conform to the passage model
11. What are the headpieces called that the Saudi Arabians wear? $->$ headpieces are referred to as "headdress"
12. What is the city of brotherly love? $->$ compound word detection required
13. When did the 6-day war begin? $->$ First need to know which is the 6-day war and then when it began. Lucene fails to get the passage containing 6-day war and the year when it began.
14. What president is on a quarter? $->$ Sense of quarter not known
15. How tall was Judy Garland? $->$ the query should contain keyword that extracts the height
16. What year was the PC invented? $->$ need to expand PC and use synonyms of "invented"