University of Alberta

Class-Free Answer Typing

by

Christopher James Pinchak

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Fall 2009
Edmonton, Alberta

Examining Committee

Dekang Lin, Computing Science

Davood Rafiei, Computing Science

Grzegorz Kondrak, Computing Science

Mario Nascimento, Computing Science

Ali Shiri, School of Library and Information Studies

Rohini Srihari, Computer Science, State University of New York at Buffalo

Abstract

Answer typing is an important aspect of the question answering process. Most commonly addressed with the use of a fixed set of possible answer classes via question classification, answer typing influences which answers will ultimately be selected as correct. Answer typing introduces the concept of *type-appropriate* responses. Such responses are plausible in the context of question answering when they are believable as answers to a given question. This notion of type-appropriateness is distinct from correctness, as there may exist many type-appropriate responses that are not correct answers. Type-appropriate responses can even exist for other kinds of queries that are not strictly questions.

This work introduces class-free models of answer type for certain kinds of questions as well as models of type-appropriateness useful to the domain of information retrieval. Models built for both open-ended noun phrase questions and how-adjective questions are designed to evaluate the type-appropriateness of a candidate answer directly rather than via the use of an intermediary question class (as is done with question classification). Experiments show a meaningful improvement over alternative typing strategies for these kinds of questions. Ideas from these models are then applied outside of the domain of question answering in an effort to improve traditional information retrieval results. Experiments comparing reranked results with those of the Google search engine show improvements are made in those rare situations for which Google provides less than ideal results.

Acknowledgements

Over the course of a Ph. D., the problems one sets out to solve evolve over time. Without the input and guidance of many people, this thesis would not have come into being. Those most directly involved with the technical aspects of this thesis are my co-supervisors, Drs. Dekang Lin and Davood Rafiei. Dekang spent many hours of his time in helping to develop the ideas herein, and continued to provide advice and supervision even after moving onto other things at Google. I greatly appreciate the fact that Dekang stuck with me until the very end and made this thesis what it is today. Davood also deserves recognition for his willingness to co-supervise me late in my program. He suggested many avenues of investigation that I would not have considered on my own and this has made the thesis that much stronger.

This work was also influenced by the Natural Language Processing group at the University of Alberta. Many ideas were discussed at group meetings that seemed to be only tangentially related to my thesis work, but in fact helped me better understand the place of this thesis within the NLP community as a whole. I would like to especially thank Shane Bergsma, Colin Cherry, and Qin Wang (Dekang's students) for their feedback that strengthened the papers that form the core of this thesis. The papers were always much stronger after your comments had been taken into consideration. I greatly enjoyed the paper that Shane and I co-authored and it is to my everlasting regret that I was not a co-author with Colin or Qin. Perhaps we will have an opportunity to work together once again in the future.

Finally, there are those who contributed to the non-technical aspects of this thesis. Ms. Debra Shiau generously spent many of her hours labeling mind-numbing data sets for appropriate vs. inappropriate candidates and also provided much-needed encouragement to get me to the very end. I would also like to thank my parents, Robert and Sherry Pinchak, for patiently waiting so many years for me to finally finish. The road was long and at times difficult, but I feel that I have ultimately achieved what I set out to do.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In years past, the sum of human knowledge was held in the libraries of the world. With the advent of the computer and the dawning of the information age, this repository of knowledge has gradually migrated from these libraries to the realm of the Internet. Instead of searching libraries far and wide for a piece of desired information, an interested person may simply query this vast repository using an online search engine to at least gain an initial understanding of the topic of their choice. However, the pendulum has begun to swing too far; the wealth of information easily available has begun to outstrip the ability of people to find specific information related to a topic.

The most popular means of searching for information online are the large search engines. These represent large-scale systems based on the principles of information retrieval (IR) in which users are allowed to submit *queries* for which *relevant documents* are returned. Relevance is often based primarily on the overlap between the query contents and the document, which presumes that documents containing query words also contain information the user is looking for. Additional factors can also contribute to relevance, such as PageRank [6] made popular by the Google search engine. Because of this use of overlap, a query for only a few generalized terms results in an overwhelming number of results. This can often prompt a user to refine the query by substituting more specific terms or adding additional terms related to the topic of interest. However, this is not a natural means by which humans meet their information needs.

It is for the combined goals of a natural interface and the reduction of information overload that *question answering* (QA) stands out. Instead of simple keyword-based queries, users may pose syntactically-correct and semantically-rich questions as the basis of their information request. Likewise, the response to these questions will no longer be a simple list of relevant documents (which may or may not include convenience features such as

snippets) but rather specific answers that satisfy the question. In effect, the burden of sifting through documents is shifted from the user (who may be easily overwhelmed) to the QA system.

Beginning with the National Institute of Science and Technology's Eighth Text Retrieval Conference (TREC-8) [67], research into large-scale corpus-based QA has grown dramatically. Organized as a competition, the TREC-8 QA track and subsequent TREC QA tracks spurred rapid development of QA techniques for free-form questions against a large unstructured text corpus. In order for techniques to be successful at TREC, they must address both the issues of handling syntactically well-formed natural language questions and the extraction of responses from unstructured newswire text. The approach advocated by the TREC QA tracks support the overall goal of developing QA techniques that can scale up to large collections such as the Web where a large portion of human knowledge resides.

The use of computational resources to meet or exceed the abilities of humans is a common theme in artificial intelligence research. By playing to the strengths of computational resources over human intelligence, we are able to approximate the means by which users find high-quality information relevant to their information needs. Of course, we must be careful not to overly restrict the amount of information reaching users. IR may be a better solution for exploratory queries (in which a user wishes to gain a better general knowledge of a topic). Additionally, answers alone are seldom acceptable as responses and require the support of one or more documents in the collection.

The goal of *answer typing* is to ensure that a set of potential answers (or *candidate answers*) can be filtered such that only those deemed plausible as responses are retained. Each answer to a question is by definition a plausible response, and responses to query generally fit some notion of appropriateness. Because answer typing forms the basis of this thesis, it is important to note both the distinction between QA and answer typing and the role answer typing plays within the QA task. Whereas QA seeks to find correct answers, answer typing merely attempts to find potentially correct answers according to type appropriateness. This process is an important aspect of the complete QA process, but is also useful in fields for which exact answers are either unexpected or undesirable (e.g., IR). By relaxing the requirement of correct answers to appropriate candidates, we can both focus on a single component of QA and obtain a process from which non-QA applications can benefit.

Answer typing is meant to capture the fact that a user posing a question has some implicit expectation of what encompasses an appropriate response. Simple examples of such questions are those that include a *question focus*, which is defined here as a noun

phrase following the *wh*-word *what* or *which* and coming before the main verb. An example of question focus is *what city hosted the 1988 Winter Olympics?* in which *city* is the focus. Such focus questions clearly state that the desired answer must be a city, and a correct answer must be a city that hosted the 1988 Winter Olympics. Other questions exist in which there is much less explicit type information, but type information is still present. For example, *who* questions, such as *Who was the first Prime Minister of Canada?* do not explicitly state what they are looking for, but a user posing such a question would not accept colours or kinds of automobiles as results. Moreover, the user can confidently exclude such responses as incorrect even in cases in which he or she does not know the correct answer; they are obviously wrong. It is this implicit knowledge of type-appropriateness that we wish to capture and exploit to improve the responses we return.

The previous examples foreshadow the most common means by which answer typing is performed in QA. Most often a list of pre-defined answer type labels is created with both the set of expected questions and some named entity extraction strategy in mind. This pre-defined set depends greatly on the ingenuity of its creator as well as the ability of the named entity recognizer that will attempt to find such entities in text. However, regardless of the set of type labels defined, we still run the risk of encountering an unanticipated question type for which no label is appropriate. In many cases, the set contains a miscellaneous or catch-all type [27, 36] to cover such cases. This behaviour is still a danger even for those typing systems using a large number of types [24]. For these unanticipated questions, answer typing is largely ineffective due to the fact that questions assigned to this class often have very little in common. For unanticipated questions assigned a miscellaneous or catch-all type, we can often make a very meaningful increase in performance over the default strategy of accepting nearly any candidate as appropriate [57].

One of the primary goals of this thesis is to develop an effective answer typing strategy that is both flexible enough to apply to areas outside of QA and does not rely on a set of pre-defined answer type labels. Even though a given set of answer type labels may have high coverage on a specific set of questions, we cannot expect that all future questions will be covered by these labels. Furthermore, as time goes on the questions encountered by a system may begin to drift away from the expected types leading to an increasing number of miscellaneously-classed questions for which labels are ineffective. If we wish to avoid the constant refinement and adjustment of type labels required by pre-defined types, we must evaluate appropriateness in a more direct manner instead of using intermediary types.

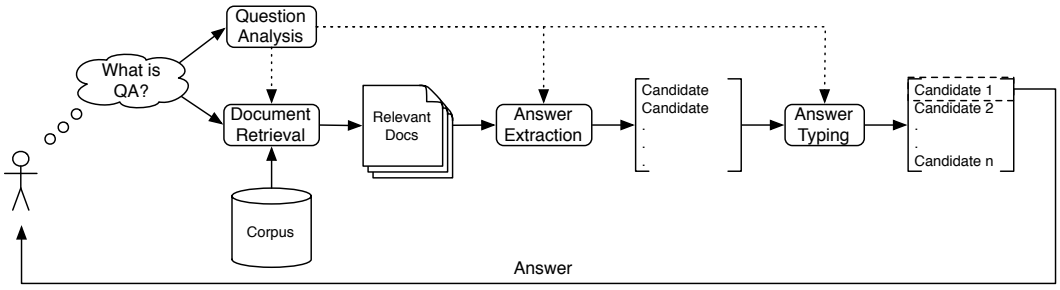


Figure 1.1: Question Answering Framework

1.1 The Question Answering Process

Although only the answer typing aspect of QA is explored in the bulk of this thesis, it is instructive to look at the QA process in its entirety to better understand the limitations and expectations of answer typing. Many of the systems involved in the TREC QA track competitions define a framework similar to the one shown in Figure 1.1 [67]. Although components of Figure 1.1 are shown as a linear pipeline, the pieces can often be rearranged and/or run in parallel to achieve some desired effect. For example, the answer extraction and answer typing modules may be run in parallel and a reranking component may assign weights to each of the two independent processes. Similarly, answer typing may come before answer extraction in the event that the chosen answer extraction module is computationally intensive, allowing the answer typing module to first reduce the number of candidates according to their appropriateness as answers. Regardless of the order in which the modules are applied, this basic framework covers the overall QA process in the majority of cases.

1.1.1 Question Analysis

The question analysis phase of a QA system is meant to extract useful information from the text of a natural language question. The exact information extracted varies by system, but can include information such as a reformulation in keyword format such that a standard IR system can be used, a parse tree or portions thereof [57, 65] for matching against documents, and type information that can be fed into the answer typing module. In some cases, a type category is sent to the document retrieval module for use with an index enhanced with type information [59]. Most often the analysis results are used by the answer extraction and answer typing modules.

Regardless of any module ordering choices, the question analysis module is most likely

to be performed both first and prior to all other modules. This is because the results of question analysis are useful for nearly all subsequent modules and may be employed in a number of ways. Although question analysis may be delayed until its results are required, the results of question analysis can often provide hints on how to handle the given question.

Given the topic of this thesis, we will largely focus on the relationship between question analysis and answer typing. Often the line is blurred due to the fact that much of the information provided by question analysis, such as a desired named entity type, is used exclusively by answer typing. From this perspective, we largely combine the discussion of question analysis (as it relates to answer typing) and answer typing itself (i.e., the identification of type-appropriate responses) into the discussion of answer typing.

1.1.2 Document Retrieval

An information retrieval (often called document retrieval in QA) engine plays an important role in any QA system. Because of the large size of the document collection in which an answer is likely to be found, document retrieval techniques provide an efficient means of filtering out the large number of documents that are unrelated in any way to the question. The bulk of contemporary information retrieval research can be applied to this problem with little or no modification, but a few important differences must be noted.

Information retrieval is concerned with keyword-based queries in which each keyword or phrase is expected to be in some way related to the information sought. A natural language question, however, contains words that would be misleading to an IR system should they appear in a query. For example, in the question *who is James Bond?* the words *who* and *is* are not important to finding documents relevant to *James Bond*. Including them in a keyword query may result in too few or irrelevant document being returned, especially if the IR system implicitly requires that all query terms be present in the document (an AND query). To address this issue, queries must be formulated to include only those words that are expected to appear in documents containing an answer. This may be performed either by the QA system prior to invoking the IR engine, or by a modified IR engine that is capable of accepting questions and filtering out the irrelevant words.

IR systems make use of document scores when producing a sorted list of relevant documents. Ordinarily, a human user of an IR system is unconcerned with the document scores so long as the sort order implied by them places relevant documents near the top of the list. A QA system, on the other hand, may find these scores useful. If only the sort order is used, a QA system can only assume that each document differs from the preceding and follow-

ing documents by some pre-defined score. At the other extreme, a QA system may simply impose a top-N cutoff and consider all top-N documents as having equivalent score. If the score assigned by the IR engine is made available to the QA system, a soft rather than hard cutoff may be used. For example, if the difference in score between the first and fifth documents returned by an IR engine is negligible, all five should be considered roughly equivalent in terms of containing a correct answer. Should the difference in score between the fifth and sixth documents be large, however, the QA system may choose to consider only the top five documents.

Given that information retrieval is a relatively well-studied topic, this thesis will not focus on the document retrieval module of a QA system. Many existing IR systems can be adapted to handle questions without excessive modifications; in QA competitions, sorted lists of retrieved documents are often made available to participants to shift the burden of QA from the document retrieval module to other components of the system. This has the effect of focusing QA research on the QA-specific portions, rather than on the parts shared by both the IR and QA communities.

1.1.3 Answer Extraction

Ultimately, the answer for a given question must come from some document in the collection, and it is the role of the answer extractor to pinpoint a particular answer. The IR engine serves to narrow down the list of possible documents that could provide an answer, but this is still too much information for a user to be presented with. The answer typing module judges the appropriateness of the various words as potential answers, but performs this task in a very broad manner; just because a word is appropriate does not necessarily mean that it is a correct answer. The answer extractor must look within a document to determine support for words as answers to a question.

Answer typing can be viewed as gathering support for candidate answers from a source of general knowledge (in our case, a large document collection), whereas answer extraction is much more focused on gathering support for a candidate within a specific document. Various features of the candidate, document, and question can indicate which candidates have support as an answer to the question. For example, if the position of the *wh*-word in the parse tree of the question is similar to that of a candidate in its parse tree in the document, this candidate may be a good potential answer. Similarly, if there are many question words located in close proximity to a candidate, this candidate may be a more likely answer. The answer extractor generally may use more complex features that can better pinpoint answers

because it only operates within the context of a single document and not a large collection.

The distinction between what constitutes answer typing and answer extraction is often fuzzy. Answer extraction can make use of information derived from a large collection of documents, such as learned patterns that often match the answer to a certain type of question. Similarly, answer typing can place more emphasis on certain documents of a collection than others, and may use features from the document of a candidate answer when determining appropriateness as an answer. Because the line between the two is ill-defined, this thesis will consider answer typing as scoring candidates according to some function that places more weight on corpus statistics versus answer extraction which assigns more weight to a few documents, for example those returned by the IR engine. Although answer typing or answer extraction may include elements from one another, a component can be classified as one or the other according to whether or not a few documents or an entire corpus play a more important role.

1.1.4 Answer Typing

In order for a QA system to accurately identify answers, it is helpful to have some concept of what candidate answers are appropriate. Humans perform this task naturally when both posing and being presented with a question; an expectation of the correct answer type allows for obviously incorrect candidates to be rejected as incorrect. For example, given a question such as *who discovered insulin?* we may accept *Frederick Banting*, *Charles Best*, and *Stephen Harper* as possible answers but reject words such as *treatment*, *1922*, *known*, *involved*, *diabetes*, etc. Based on the format of the question, an idea of the desired answer type can be formed. In the case above the desired answer is a person. Similarly, candidate answers can be identified as type-appropriate or not. By employing such a method, a QA system can invalidate inappropriate responses and increase the likelihood of returning a correct answer to the question. Conversely, answer typing may occur without the explicit use of types, which will be referred to here as *class-free* answer typing. Class-free answer typing directly scores the appropriateness of words to a question without ever having to use types.

It should be noted that answer typing requires only the question to form a notion of what types would be appropriate. If the types are drawn from some finite fixed set, this process can be called *question classification*. Question classification must be combined with *named entity recognition* to provide a complete answer typing system. Named entity recognition is the task of identifying words or phrases of a specified finite fixed set of types within a given

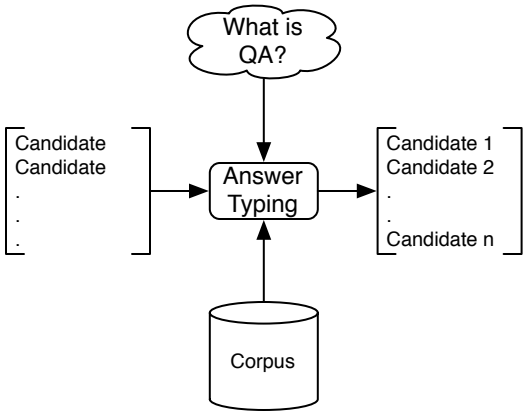


Figure 1.2: Answer Typing Framework

document. For such an answer typing system to be effective, it must be able to accurately classify questions as one or more of the appropriate types and identify words or phrases appropriate to those types.

1.2 Contributions of this Thesis

Given our focus on answer typing alone, we wish to explore effective means of answer typing that are useful for both the task of QA and for any task in which the notion of type-appropriateness is beneficial. For example, an application may wish to store only those items in a database that are type-appropriate. In this context, answer typing can ensure that the database is consistent. We adopt the notion of answer typing depicted in Figure 1.2, representing a portion of Figure 1.1. Some important consequences of this framework are that answer typing is cast as filtering or ranking a pre-defined set of candidate answers and that a text corpus is used. Within this framework, the following sections describe the contributions of this thesis.

1.2.1 Class-Free Answer Typing for Open-Ended Noun Phrase Questions

Given the wide acceptance and use of class-based answer typing approaches, we explore an alternative class-free typing strategy that better accounts for the deficiencies inherent in the use of classes. Although class-based answer typing is appealing because of the separation of question classification and named entity recognition, any set of classes is unable to cover all possible questions. Even if class-based approaches exhibit good performance on questions covered by the classes, we can augment a class-based approach with our class-free typing

strategy in cases where class-based answer typing performs poorly. Therefore, it is useful to explore a class-free strategy that will have a variable (but significant) impact depending on whether or not a class-based approach is used in conjunction.

Class-free answer typing requires that we determine a score for each candidate response directly rather than using an intermediary class. Because a candidate's suitability is computed directly to the question itself, we have a much lower risk of encountering a word which we did not anticipate as a possible answer. In fact, we expect that only rare words and difficult-to-type questions will present problems for our typing strategy, the very same problems that lead to poor performance for class-based strategies. By removing the reliance on intermediary types, we eliminate the possibility of finding no adequate class for a question as well as the errors arising from named entity recognition required of the set of classes.

This thesis introduces an answer typing strategy based on the intrinsic similarity of words and the wealth of information contained within text corpora instead of a predefined set of answer classes. Such a model is shown to be particularly effective for those questions seeking an open-ended noun phrase as a response; these questions typically use the word *what* or *which*. Also, questions seeking more closed forms of answers, such as the how-adjective questions (Chapter 4), benefit from a more fine-grained approach to typing. Conversely, more complex questions often require a more involved response and are not as amenable to typing; reasons or causes are not easy to identify as type-appropriate to a question. The model introduced in detail in Chapter 3 is shown to perform well on open-ended noun phrase questions and the results are encouraging for the application of the model in a full question answering system.

1.2.2 Fine-Grained Typing of How-Adjective Questions

The class of how-adjective questions, questions in which the *wh*-word *how* is followed by an adjective/adverb such as *high*, can be generally considered more restricted than the open-ended noun phrases discussed previously. Most often these questions are relegated to the fixed type of *Number* or *Quantity* because of the fact that they seek an answer expressed in terms of a quantity of units. However, just because these questions are always answered by a numeric value does not mean that all numeric values are appropriate responses. If a question is asking *how high* and we respond with a value expressed in terms of pounds or kilograms, the response is incorrect. Therefore, we wish to exploit the appropriateness of units in response to the adjective in the question.

We introduce a method in Chapter 4 to automatically discover and rank those units that are appropriate for numeric quantities of a given question. By identifying those units that are appropriate to the adjective, we can achieve a finer granularity of typing than that provided by simply looking for numbers. This reduction in appropriate candidates can make the task of answering how-adjective questions easier by allowing only responses expressed in terms of appropriate units to be further considered. Experimental results show better precision over the entire range of recall when compared with both the coarse grain numeric type and alternative finer-grained approaches.

1.2.3 Extending Answer Typing Beyond QA

Answer typing is most easily thought of in the context of QA. However, answer typing does not need to be restricted to QA, and it can play a vital role in any application for which a notion of type-appropriateness is useful. For example, we may wish to extend the database natural join operation that typically joins based on tuples that share an equivalent attribute value. Answer typing could be used in this instance to implement a fuzzy-join in which the attributes are type-compatible rather than equivalent. This fuzzy join would result in many more result tuples than a natural join, but would allow for combinations not previously possible. Such application of answer typing to domains other than QA is a logical next step.

In this thesis, we extend answer typing into the domain of information retrieval, an area closely related to QA but with subtle differences. IR users interact with the system via keyword queries to search for specific answers (as if the query were a modified question) or general interest topics (a "tell me more about *X*" request). Even when searching for specific answers, the queries are often much less structured than questions seeking the same information (for example, a query would be *age Stephen Harper* rather than *how old is Stephen Harper?*). Finally, users expect documents as results rather than short answers, and so we must be careful to order documents rather than terms according to type appropriateness.

To apply answer typing to the domain of IR, we extend our open-ended noun phrase answer typing model with additional features more suited to IR. Because our noun phrase typing model is designed with flexibility in mind, small changes to the notion of type-appropriateness are easily accommodated. Experimental results show that information retrieval results for queries seeking a short answer are significantly improved by a notion of type appropriateness. We compare with the popular Google search engine so that the comparison ordering is as good as is reasonably possible given the wealth of information on the

Web and Google's desire to produce accurate results for queries.

1.3 Outline of the Thesis

To serve as both a summary and preview of the remainder of this thesis, this overview is intended to give a general overview of the upcoming chapters.

Related Work

Chapter 2 introduces, in general terms, the body of work related to the topic of answer typing. Almost all QA systems incorporate answer typing in one form or other, and we identify common themes in the methods commonly employed. By far the most popular is question classification (introduced previously) for which the set of types can vary greatly (from less than 20 to over 140). Methods to assign these types can be based either on hand-defined or learned rules or with the use of a learned classifier based on features of the question. Finally, a few systems depart from the traditional class-based notion of type and attempt to directly evaluate the suitability of a candidate answer. In particular, the system we focus on uses ideas borrowed from theorem proving to connect a candidate answer with concepts in the question.

Typing Open-Ended Noun Phrase Questions

The first set of questions for which we build an answer typing model is the set of open-ended noun phrase questions. A generative model intended to show how pieces of the question contribute typing information followed by a discriminative model that is able to take advantage of more diverse features, are introduced in Chapter 3. Open-ended noun phrase question, specifically those identified by *what* or *which* as the *wh*-word, contain a surprising amount of typing information. The way in which a question is stated reveals clues about what a potential answer must satisfy. For example, a question such as *what city hosted the 1988 Winter Olympics?* must be answered by 1) a city, and 2) something that has hosted an Olympics. If we wish, we can adjust the level of detail by taking more general or specific information from the question. For the above example, we may wish to make the second requirement more specific and require a city that hosted a Winter Olympics. This information is particularly easy to extract from questions because the *wh*-word largely acts as a placeholder for the answer. If we parse the question with Minipar [39], the *wh*-word takes on the syntactic role of the as-yet unknown answer.

Once typing information is extracted from questions, we compare it with similar information extracted from a text corpus as a whole. The best way to find out what words are appropriate for "things" hosting a Winter Olympics is to search text for mentions of things hosting a Winter Olympics. If we have an unlimited amount of text, we can expect to eventually find mentions of all things that have ever hosted a Winter Olympics, the correct answer included. Unfortunately, such an unlimited quantity of text does not exist and so we receive only partial information. To increase coverage, we use automatically generated similar word clusters to essentially fill-out these incomplete lists of mentions. This similar word expansion is not exact, but allows us to form a fuzzy idea of what terms are type appropriate for a particular aspect of a question and the question as a whole. Given a measure of appropriateness, or expected likelihood of seeing a mention, the next step is to combine the values for multiple aspects of a single question into a score for the question as a whole. For example, a candidate such as *Calgary* will have different values for *Calgary is a city* and *Calgary hosted a Winter Olympics*. We introduce two methods of typing in Chapter 3, varying with respect to the weights of question contexts and the inclusion of additional features.

Fine-Grained Typing of How-Adjective Questions

As discussed in Section 1.2.2, one of the primary contributions of this thesis is to consider the class of how-adjective questions at a finer level of detail. Although it is true that how-adjective questions are answered by numeric responses, not all numeric responses are satisfactory as answers. A *how tall* question cannot be answered in units of weight or mass, regardless of whether or not the numeric quantity itself is properly identified. Chapter 4 introduces the idea that the units themselves are the basis for type appropriateness. We can automatically discover what units are appropriate for *how tall* questions by looking for words related to the adjective *tall*, such as *height* and *tallness*, and mentions of the phrase *height is measured in X*. Once again, mentions form the basis of what is type appropriate, and similar word expansion can again fill out the list of mentions. These mentions contain units that are type appropriate for the given adjective.

Once a set of potential units has been discovered, we must apply some filtering, as more than just units are discovered, especially when we use unparsed web documents as a source. The result is a list of high-quality units in which answers to the question may be expressed. It should be noted that all questions sharing the same adjective, such as *tall*, will share the same unit list, and there is no need to repeatedly search for units appropriate to every new

question. Given the expected type of a question, determining type-appropriate responses is then as simple as matching the unit of the response with a unit on the list; the list is ordered and matching higher up can be considered a stronger indicator of appropriateness.

Beyond QA: Answer Typing for IR

The logical next step of answer typing is to extend beyond the bounds of QA and into the realm of IR. Chapter 5 extends the flexible model for open-ended noun phrases to queries submitted to an online IR engine. Of course, not all IR queries can benefit from answer typing. Many IR queries are searching for general topics and not specific answers. However, a significant portion of IR queries are looking for short, specific responses. Also introduced is a rudimentary means of identifying such queries along with an extended typing model that can take advantage of additional features specific to the IR task.

In general, IR queries contain much less information than questions, and any information they do contain is typically expressed in a different manner. We focus on *prepositional queries*, such as *cities of Canada*, that are a noun phrase modified by a prepositional phrase. Although this class of queries seems small at first glance, prior work has established a class of noun phrase (not prepositional) queries that can be transformed into this format. Once we have a query in prepositional format, we proceed to analyze the query in much the same way that we analyze questions.

The only further complication introduced by IR is the fact that users expect an ordered list of documents instead of a single specific answer as a response from an IR engine. To preserve this behaviour, we must rerank the results of an IR engine with respect to whether or not results (snippets or documents) contain terms that are type appropriate to the query. The ultimate goal is to produce an ordering that is more favourable to users than the results in their original ordering (for some queries), and experiments show that answer typing improves upon the original Google ordering for those cases in which a query has a short answer.

Conclusions

Chapter 6 concludes with a summary of the primary contributions introduced in this thesis. The connecting theme of all aspects of this thesis is the central notion of the type appropriateness of terms without the need for intermediary types. Although a class-based approach is certainly useful for typing, it cannot cover all cases. For those cases in which a class-based approach is insufficient (such as open-ended noun phrases and fine-grained

how-adjective questions), a class-free approach can truly shine. Answer typing can even apply beyond the realm of QA, a subject that will hopefully be explored further in the future.

Chapter 2

Related Work

Answer typing is an often-neglected aspect of QA. Many times it is mentioned in passing so that more attention can be paid to aspects of a QA system that distinguish the system in question from other similar systems. Often these distinguishing features are part of the answer extraction process, for which the bulk of work is done. This glossing-over of answer typing details is especially prevalent when simple answer typing methods are used. However, more attention is paid whenever answer typing forms a relatively important part of a QA system; heavy reliance on typing and/or sophisticated typing often garners a more detailed description of how the typing is performed. We examine some examples of both simple and complex approaches to answer typing in this chapter.

In most contemporary systems, answer typing plays the role of either *verification* or *guidance* (or both) for the question answering system as a whole. When used for verification, the answer type model provides an answer to the question *Is this answer candidate appropriate?* The decision can be binary (yes or no) or a score that then needs to be combined with the scores obtained in the remainder of the question answering process. Although no known QA system uses what can be considered a pure verification approach, in which the type is only used to verify candidates selected as answers by a separate answer extraction module, some systems, such as that of Burger and Bayer [8], make use of answer type information late in the question answering process. Answer typing features are combined with answer extraction features to create a unified set for which weights must be learned.

When answer typing is used in a guidance role, question analysis first provides a type to go along with the question that may then be used by subsequent stages of question answering. These subsequent stages can take into account the difference between the desired type and the type of a candidate answer when evaluating the appropriateness of that candidate. In some ways, guidance is a more natural way to incorporate the notion of answer

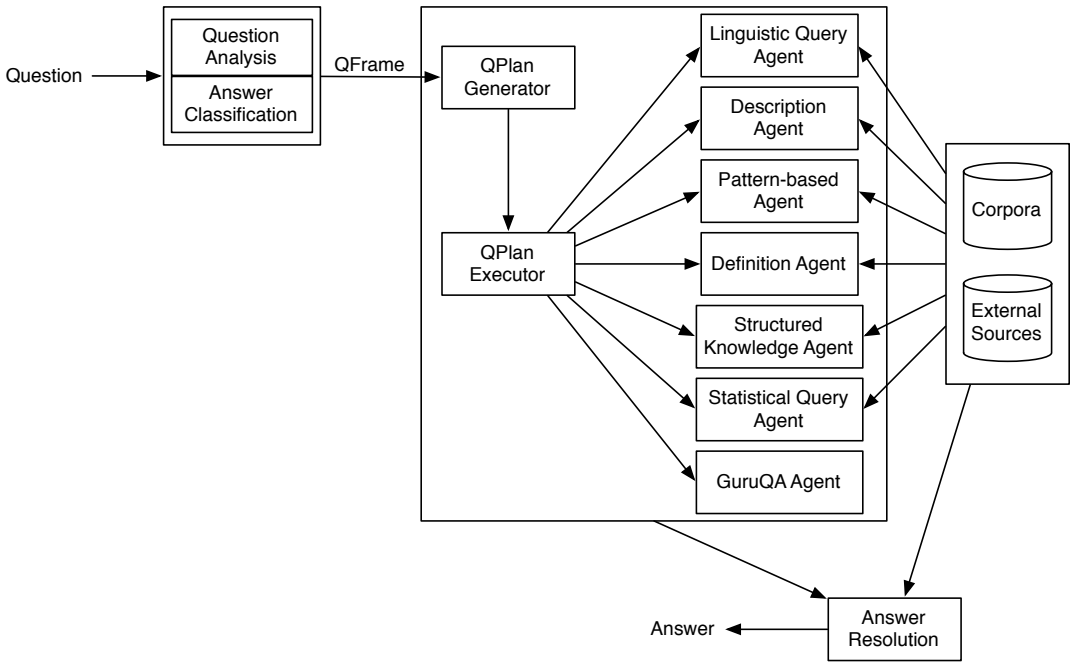


Figure 2.1: Architecture of Prager et al. [60]

type into a question answering system. However, using answer typing for guidance is not conducive to compartmentalizing the question analysis and the answer typing model as a single component of the system; question analysis passes the type to all stages of question answering, resulting in a potentially more difficult development process. Guidance is by far the more popular method of incorporating answer typing into QA. A notion of type can be incorporated during any stage of the question answering process, and is most often added early on. For example, the system of Prager et al. [60] (see Figure 2.1) passes a *QFrame* containing answer type information from the initial question analysis phase to the subsequent answer extraction modules which may include many parallel sub-models. Each of the parallel sub-modules may make use of the desired answer type when searching for answer candidates.

Regardless of their use for verification or guidance, current approaches to answer typing can be considered as either *class-based* or *class-free*. Class-based approaches make use of an explicit set of answer types. These methods assign a type from a given set of types to each question and each answer candidate. In other words, a class-based system must perform two explicit steps: 1) compute likelihood that a type T is assigned to a question Q, and 2) compute the likelihood that a candidate c is labeled as T; the former is referred to as *question classification* whereas the latter is known as *named entity recognition*. Conversely,

a class-free approach refrains from using a set of explicit answer types in favour of computing some notion of appropriateness directly. A class-free system computes the likelihood that c is appropriate for Q without requiring *a priori* knowledge of types $\mathcal{T}$.

2.1 Class-Based Approaches

QA systems that employ a class-based approach to answer typing must both assign types to questions and identify that same type (or types) in the set of answer candidates. Often, the set of types is taken to be of that of an existing named entity recognition system, such as the MUC-7 types [11]. The MUC-7 competition (and previous MUC competitions) embodies a great deal of prior work on named entity recognition. Given that named entity recognition was explored in this prior work and continues to be developed, the focus of this section will be on question classification where questions are assigned a fixed type (or types) and entities of the same types must be identified in text. However, it should be noted that performance of any class-based approach is dependent on both the accuracy of question classification as well as the accuracy of entity recognition for that same set of types.

The majority of class-based approaches can be broken down with respect to how they assign types to a question. The types can be assigned either by some predefined rules (static) or by a learned classification model that associates features of a question with particular types (learned). Static approaches are based on knowledge engineering; someone must sit down and define rules that associate certain aspects of a question with a particular type. For example, one may create the rule (question word $=$ *what* $\wedge$ question contains author $\Rightarrow$ type $=$ *Person*). In contrast, a learned approach employs machine learning techniques, often with supervised training data, to associate features derived from a question with particular types. The learned model may then be applied to future questions to determine types.

2.1.1 Answer Types

The choice of a set of answer types is not a straightforward process. The set may be selected to be small and general so that both the question classification task as well as entity recognition task have relatively high accuracy. Conversely, the set may be chosen to be large with complex organization so that answer types convey a great deal of information. They may also be derived using outside sources of information, such as a predefined ontology. This section explores the sets of answer types used by various question answering systems.

The simplest and most popular sets of answer types are those based on the MUC answer types (Table 2.1). The Message Understanding Conference (MUC) was created to

Type	Definition from MUC-7 [10]
ORGANIZATION	named corporate, governmental, or other organizational entity
PERSON	named person or family
LOCATION	name of politically or geographically defined location (cities, provinces, countries, bodies of water, mountains, etc.)
DATE	complete or partial date expression
TIME	complete or partial expression of time of day
MONEY	monetary expression
PERCENT	percentage

Table 2.1: MUC-7 Types

further research in the field of Information Extraction (IE). One of the tasks of IE is named entity recognition within documents. That is, entities must be identified as one of a fixed set of types. Because of the large amount of research undertaken using the MUC types, it is a logical starting point for class-based answer typing. Existing systems have been designed to identify the MUC types (for example, Maynard et al. [44]) and so the challenge of identifying appropriate types in text has already been addressed for this set of types.

A number of systems make use of what appear to be modified or enhanced sets of the types used as part of MUC. The IBM system [26], based on Maximum Entropy learning [1], originally began using the MUC types augmented with *PHRASE* (a miscellaneous type) and *REASON* (a miscellaneous type applicable to *why* questions). Eventually this system grew to use a total of 31 answer types [27] along with *PHRASE* and *REASON*, shown in Table 2.2. The AnswerFinder system at Macquire University [49] makes use of a set of thirteen possible answer types [48], including MUC types such as *person* and *location* while adding some additional types such as *river*, *state*, and *unknown*. Finally, the Cymphony system [35] uses a set of types with a top-level hierarchy roughly based on the MUC types.

A system developed at the University of Albany SUNY [71] departs from the standard MUC types and instead uses the BBN IdentiFinder [3] types with the addition of three extra types. Also departing from the MUC types, He et al. [22] make use of the set of types as used by the Li and Roth [36] classifier (described later when discussing learned class-based approaches). Both approaches show that the MUC types (along with enhancements) need not be the only basis for a fixed set of answer types.

The ISI Webclopedia system contains over 140 possible entity types, referred to as *QTargets* [24]. QTargets are organized into a multi-level hierarchy and are divided into groups based on the kinds of answer, and often the question, they represent. The entire type list and its hierarchy is specified manually, representing a great deal of effort for type

Category	Types
Name Expressions	Person, Salutation, Organization, Location, Country, Product
Time Expressions	Date, Date-Reference, Time
Number Expressions	Percent, Money, Cardinal, Ordinal, Age, Measure, Duration
Earth Entities	Geological Objects, Areas, Weather, Plant, Animal, Substance, Attraction
Human Entities	Events, Organ, Disease, Occupation, Title-of-Work, Law, People, Company-Roles

Table 2.2: Ittycheriah et al. [27] types

set creation. Despite its wide scope and fine granularity, it is still prone to encountering questions for which no type in the hierarchy is appropriate. In fact, the authors state that the *S-NP* (or general noun phrase) type is the default QTarget and will be used unless something more appropriate matches.

2.1.2 Rule-Based Approaches

By far the most common and simple approaches to answer typing employ a set of predefined rules to map from a given input question to one or more explicit answer types. The rules may be carefully constructed by hand over a development set in accordance with the types of questions developers expect to encounter. The rules may also be learned from a training set of example questions along with the type that is appropriate for each question. Often, the systems employing fixed rules for answer type assignment do not explicitly state the method by which these rules are developed (for example, Sun et al. [65]). For these systems, and similarly for our work, the emphasis is not on the derivation of rules but on the advantages and drawbacks of using a set of rules that have been crystallized prior to running the answer typing system. Presumably, hand-crafted rules exhibit higher precision as opposed to those built automatically, although they may show lower coverage.

The use of a rule-based class-based approach for answer typing does not imply poor overall performance of a system. On the contrary, two of the top-ranked systems at past TREC competitions, described by Harabagiu et al. [21] and Sun et al. [65], make use of rules to assign answer types to questions, although Harabagiu et al. make use of a class-free approach in certain cases (which will be discussed later on). Neither system explicitly states how these rules are derived.

Some systems explicitly state the use of hand-crafted rules. For example, the work of Wu et al. [71] describes rules in which any question beginning with *When*, *Where*, *Who*

are assigned the types *Date*, *Location*, *Person*, respectively. The *How* ⟨*ADJ*⟩ questions are mapped via special rules, and the *What/Which* ⟨*BE*⟩ and *What/Which* ⟨*ENTITY*⟩ are handled using a special mapping from a key noun (either in ⟨*ENTITY*⟩ or following ⟨*BE*⟩) to particular answer types. The use of hand-crafted rules allows for easy and fairly accurate assignment of type(s) to question, but can potentially miss certain cases. For example, the system above would incorrectly assign a type of *Person* to the question *Who manufacturers the world's fastest car?* For this question, the correct answer type would likely be *Organization* rather than *Person*.

The AnswerFinder system [48] makes use of 29 regular expressions to assign answer types to questions. The regular expressions were derived from the TREC 2002 questions, and presumably are created by hand. If a new question matches one of the regular expressions, the type assigned to that question is the one specified by the expression. AnswerFinder uses the GATE system [20] to identify named entities within the text as being of one of the possible answer types. In this particular context, AnswerFinder is an example of a system that takes advantage of prior efforts of MUC for named entity recognition rather than designing their own specific extraction program.

Returning to the ISI Webclopedia system [24], answer types are assigned with the help of a parser and 276 hand-built rules [23]. Given the large number of types and the complexity of the hierarchy, rules are applied in a back-off scheme such that the assignment falls back to increasingly general types. This back-off scheme allows the 140+ type hierarchy to be useful without an incredible number of rules for deciding when either a general or specific type is the best choice for a given question.

2.1.3 Learned Class-Based Approaches

Similar to rule-based approaches, learned approaches make use of a fixed set of answer types that can be assigned to questions. The decisions for type assignments are made based on a learned model of associations between features (derived from the question) and answer types. New questions presented to the model are assigned a type based on how well their features map to types under that model. Often a learned model can discover non-obvious associations that may be missed by manual rules, but may also make errors due to noise in the training data because of limitations introduced by transforming questions into features.

One of the first attempts at learning a model for question typing is presented by Ittycheriah et al. [25] in which a Maximum Entropy approach is used to learn a typing model. The types used are those of MUC, and Maximum Entropy is used for both question classi-

Coarse Class	Fine Sub-classes
ABBREVIATION	abb, exp
ENTITY	animal, body, colour, creative, disease/medicine, event, food, instrument, language, letter, other, plant, product, religion, sport, substance, symbol, technique, term, vehicle, word
DESCRIPTION	definition, description, manner, reason
HUMAN	group, individual, title, description
LOCATION	city, country, mountain, other, state
NUMERIC	code, count, date, distance, money, order, other, period, percent, speed, temp, size, weight

Table 2.3: Li and Roth [36] classes

fication and named entity tagging. Ittycheriah et al. specify an additional miscellaneous or catch-all class (*Phrase*) that can be used when no other class is appropriate for a question. The feature extraction module considers the question as a bag of words, n-grams, part of speech tags, and WordNet-based [45] expansion terms. Error analysis shows that the question classification module and named entity tagger are the two components with the lowest error rate in their entire question answering system.

Li and Roth [36], probably the most well-known of the class-based learned approaches, take learned answer typing a step further by employing a larger set of answer types along with the Sparse Network of Winnow (SNoW) learning architecture (rather than Maximum Entropy). A two-level hierarchical classifier is built on features including the words in the question, the part-of-speech tags for those words, chunked noun phrases, named entities, head chunks for a sentence, and certain key words that often occur with a particular class. Features are also combined to produce additional features, although not all combinations are used. This represents a typical feature space for learned class-based approaches.

Li and Roth's use of a two-level hierarchy is meant to allow for the use of the general class (of which there are six) when deciding on the more specific classes (46 possible). The classes used by Li and Roth are found in Table 2.3. Unfortunately, the performance benefit of first deciding on a coarse class and using it as an additional input when deciding on more refined classes is very small. This does not mean that the classifier in general is poor, however, as Li and Roth report a classification accuracy of 95% on their set of fine classes. Even higher performance is reported on the coarse set of classes, for which the classifier is correct 98.8% of the time. This coarse set of classes better represents the original MUC types, although it likely does not have a sufficient degree of granularity to perform good answer typing.

One missing detail, typical of many class-based approaches, is that of an entity tagger or detector capable of identifying entities of these 46 possible types. Of course, it may be possible for a QA system to modify its behaviour based on the type(s) assigned to the question, but in general a type such as *plant* or *religion* would best be handled by identifying words that are members of that class. Interestingly enough, the classes of Table 2.3 include a partial breakdown for the numeric types (such as *distance* and *speed*). This is similar to our approach, discussed in Chapter 4, of examining the how-adjective questions in greater detail rather than simply resorting to the use of a single numeric label.

Zhang and Lee [72] present a comparison of machine learning techniques for learning a model to assign types to questions. They compare a nearest neighbour approach, a naïve Bayes approach, a decision tree approach, a SNoW-based approach [36], and a novel support vector machine (SVM) approach. The SNoW approach was meant to approximate or recreate the system used by Li and Roth. The use of an SVM to perform essentially the same task as Li and Roth represents a general shift toward SVM learning in contemporary natural language processing. Performance is given at 80% accuracy when classifying TREC-10 questions as one of the types used by Li and Roth (see Figure 2.3), a slight improvement over the 77% for the SNoW-based classifier. By incorporating a kernel based on the parse structure of the question, Zhang and Lee are able to make a slight improvement in performance, but this is only tested on the coarse classes of Figure 2.3.

2.1.4 Narrow-Class Questions

Many questions, including the how-adjective questions, fall into the realm of having either an obvious single or few classes. The *when* questions are always seeking a time or date, and appropriate responses must be temporal references (e.g., [41]). For other questions, such as *who* (looking for a person or organization when not a definition question), it can suffice to identity named-entity types in text based on the MUC classes [11]. Of course, each of these kinds of narrow-class question can be handled in a fine-grained manner, and we focus on the how-adjective questions as an example of such narrow-class questions.

Wu et al. [71] handle how-adjective questions differently than other questions. They use special hand-crafted rules to assign a particular answer target during the answer typing phase (although no examples of rules are ever stated). This is a common trend for the discussion of answer typing in the QA literature; answer typing is seldom considered important enough to be explained in detail, often because the most obvious means are used to perform answer typing (such as the use of hand-crafted rules). Regardless of the rules used,

Wu et al. take advantage of the structure inherent in how-adjective questions rather than just treating them as questions seeking a number. However, manually hand-crafting types is costly, and would have to be repeated if the system was moved to a new language or a new query domain. Our automatic approach does not suffer from this drawback.

2.1.5 Limitations of Class-Based Approaches

Class-based approaches work with a fixed set of answer types that have been defined *a priori*. That is, prior to any *new* question being seen, the set of types has been crystallized and cannot be changed. This results in the problem of what to do in the event that no existing type is appropriate for a question. This can manifest itself as situations in which a question receives no type label (for example, due to no appropriate rule in a static approach) or a significant amount of uncertainty about the type(s) (for example, all types receiving a uniform probability). In such situations, a *miscellaneous*, or *catch-all*, type either implicitly or explicitly exists. This miscellaneous category is reserved for questions seeking an answer that is not one of the predefined types and is effectively a non-type. For example, if our set of types is taken to be {person, place, location} and we are given the question *What animal can run the fastest?* we must assign the miscellaneous type to the question because none of the other types are appropriate. A question receiving the miscellaneous type is looking for something other than one of the given types. However, due to the fact that the majority of words in text are often not one of a predefined set of types (depending on the size of the type set and the accuracy of entity recognition), the majority of words and phrases have no type and are therefore appropriate for a miscellaneous question. In other words, when we have a question seeking a miscellaneous type, we can expect the set of appropriate answer candidates to be large and the relative benefit of class-based approaches to be small.

Light et al. [37] performed a study to measure *answer confusability* in systems that use a fixed set of answer types. Using the set of TREC-9 questions [68], along with a fixed set of 24 answer types, Light et al. manually assign one of the 24 types to each question. Furthermore, they manually identify sentences containing a correct answer, and manually extract all terms of the appropriate type from these sentences. Answer confusability for a question is then measured as the ratio of the number of correct answers to the number of appropriate terms extracted from the sentences. Thus, if the number of correct answers is constant but the number of appropriate candidates increases, the answer confusability score goes down. Averaging answer confusability over a set of questions provides an upper bound on performance because perfect question typing, sentence extraction, and entity

recognition are unrealistic in an automatic system. Overall, Light et al. find that this perfect answer typing would result in only 59% average accuracy. In other words, on average, each question has three correct answers for every five appropriate terms, leaving two terms as appropriate yet incorrect. When considering questions assigned as their *DefaultNP*, which is essentially the miscellaneous class, performance drops to 25%. For the *DefaultNP* class, a system randomly guessing one of the appropriate terms as an answer can be expected to be correct only one out of every four times. Light et al. expose the underlying flaw in using a fixed set of answer types; multiple entities of the same type are indistinguishable by the answer typing system unless some other technique is employed to compensate for this shortcoming.

2.2 Class-Free Approaches

In contrast to class-based approaches, class-free approaches require no explicit set of answer types to perform answer typing. Instead of explicit answer types, other methods are used to associate appropriate answers with the questions. These methods still rely on general information derived from an entire corpus or a set of external sources, and are still considered to be answer typing rather than answer extraction strategies.

Interestingly enough, class-free answer typing is a very uncommon approach to answer typing. Only the system described by Moldovan et al. [47] performs a form of class-free answer typing. Moldovan et al. use a class-based approach for most standard types of questions, but adopt a different approach when faced with a question seeking a reason (such as *how did James Dean die?*). Assigning these sorts of question a fixed type (for example, a *Reason* type) is not particularly helpful because of the difficulty in identifying reasons and because the space of all possible reasons is quite large. Instead, Moldovan et al. derive semantic relations from the question that are used in conjunction with predicates derived from WordNet [45]. An example of a semantic relation derived from the question *how did James Dean die?* is *manner(x, die)* in which *x* would be replaced by a manner in which somebody or something could die. Later on, during answer extraction, this semantic relation would need to be resolved with a combination of the WordNet predicates and predicates from a candidate answer sentence (in which *x* is replaced by some manner of death). Essentially, the whole of WordNet is transformed into a typing strategy for connecting known question concepts (such as *manner(x, die)*) to a potential answer without the need for a fixed type label.

An example of their logic prover, COGEX, in action is provided in [46] and is used here. Much of the terminology used in their description is left to the interpretation of the reader, including the definition of the individual predicates forming the basis of the logic proof. In general, they use x_i variables to indicate concepts and e_j variables to represent relationships between concepts. A question, such as *which company created Mosaic?*, is transformed into a conjunction of these predicates. In this case, the question becomes:

> $\exists e_1, x_2, x_3, x_4, x_5$ (organization_AT(x_2) & company_NN(x_2) & create_VB(e_1, x_2, x_5) & mosaic_NN(x_5))

Roughly translated, this format states that we are looking for something that is both an organization and a company and this entity created Mosaic, the browser that popularized the Web. The candidate answer sentence is also transformed into this logic format, although the logic form is omitted here because of the size and complexity:

> In particular, a program called Mosaic, developed by the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana - Champaign , is gaining popularity as an easy to use point and click interface for searching portions of the Internet.

Named-entity recognition is used to establish the fact that NCSA is an organization:

> $\forall x_3, x_4, x_5, x_6, x_7, x_8$ (national_NN(x_3) & center_NN(x_4) & for_NN(x_5) & supercomputing_NN(x_6) & application_NN(x_7) & nn_NNC(x_8, x_3, x_4, x_5, x_6, x_7) $\rightarrow$ organization_AT(x_8))

and transformed WordNet glosses establish the relationship between *develop* and *make* and *make* and *create*, thereby encapsulating the general-world knowledge of the equivalence of *develop* and *create* (at least in the realm of software):

> $\exists x_2, x_3, x_4, \forall e_2, x_1, x_7$ (develop_VB(e_2, x_7, x_1) $\leftrightarrow$ make_VB(e_2, x_7, x_1) & something_NN(x_1) & new_JJ(x_1) & such_JJ(x_1) & product_NN(x_2) & or_CC(x_4, x_1, x_3) & mental_JJ(x_3) & artistic_JJ(x_3) & creation_NN(x_3))
>
> $\forall e_1, x_1, x_2$ (make_VB(e_1, x_1, x_2) $\leftrightarrow$ create_VB(e_1, x_1, x_2) & manufacture_VB(e_1, x_1, x_2) & man-made_JJ(x_2) & product_NN(x_2))

Once the rules have been derived from the question and candidate answer sentence (and from WordNet as a whole), all that remains is to derive a specific answer term from the

combination of all the logic forms. For this particular example, a sequence of 374 steps are required. Should a proof be impossible, some conditions that prevent proof completion are relaxed and a penalty is imposed for finding answers with relaxed conditions, thereby reducing the score for answers that are inappropriate (due to WordNet rules being relaxed). Understanding this complex process can be difficult, but the ultimate goal of connecting question concepts with answer concepts via general world knowledge (i.e., WordNet) is clear.

Aside from this method for class-free answer typing used by Moldovan et al., no other known question answering system uses a class-free approach to answer typing. One of the goals of this thesis is to develop a class-free answer typing strategy that is more general than Moldovan et al. so that it can be applied to all questions rather than only ones seeking an abstract *Reason* answer. More specifically, we hope to show that class-free answer typing performs better on those questions for which the answer type is unknown or poorly defined.

2.3 Summary

Although answer typing is an important component of a QA system, the most common approach to typing is often very simple. Class-based approaches using (often hand-crafted) rules to assign types are the most prevalent means of performing typing, even though these classes are known to be deficient in at least some cases where a *Miscellaneous* type must be assigned. Because rules are often difficult to create and can have poor coverage, learned class-based typing approaches can account for a more diverse set of questions being mapped to a given set of types. Class-free approaches avoid the problems inherent with any class-based approach (be it rule-based or learned) by not relying on a set of predefined types that have variable coverage. However, class-free approaches are the most rare and often the most complex, meaning that they require a great deal of effort to create.

In the next chapter, we explore the use of a novel class-free answer typing strategy. Although class-based approaches often exhibit good performance for those questions that can be assigned one of the classes, a class-free model can assist for those questions that are either miscellaneous or have a type assigned with low confidence. Therefore, all typing approaches should be considered to be mutually beneficial rather than mutually exclusive.

Chapter 3

Typing Open-Ended Noun Phrase Questions

The first class of questions considered in this thesis are the open-ended noun phrase questions. In general, these are the questions denoted by the *wh*-word *which*, *what*, and *who* that are not seeking the definition or explanation of some entity or concept. Definition questions often require a long response and are not as suited to typing (such as *who is Stephen Harper?*). Open-ended noun phrase questions are of particular interest for answer typing because the desired answer can be just about anything that can be expressed as a noun phrase; from animals to movies to kinds of automobiles. Conversely, other kinds of questions are significantly restricted in the types they seek. *When* questions in particular are always looking for a temporal reference (i.e., time or date), which is a specific kind of noun phrase. The open-ended noun phrase questions essentially represent those questions that can be answered by something that can be named; if a question is looking for the world's fastest land animal, the answer is the animal named by the word *cheetah*. Nameable answers are crucial to our system, as responses that cannot be named by a term are more complex in nature and are either 1) multi-part, such as *when and where was the Battle of Gettysburg fought?*, or 2) complex, requiring more of an explanation than a short answer (e.g., *how do* questions).

The open-ended noun phrase questions are best able to take advantage of a fixed set of answer types. For example, the *LOCATION* type of Table 2.3 has as subtypes *city, country, mountain* and *state*. Each of these types correspond to something nameable, such as *Edmonton*, *Canada*, *Mt. Robson*, and *California*, respectively. If we encounter questions looking for an answer such as *Edmonton*, the subtype *city* would be useful. Likewise, if Table 2.3 included a *river* type and we encountered a question such as *what is the longest river on the world?*, then the *river* type is an excellent match. However, if no *river* type

exists in our fixed set of types, then we must fall back to using something more general such as *LOCATION* or possibly *miscellaneous*. In this instance, a better solution would be to perform answer typing without an explicit set of answer types.

In this chapter, we study the creation of a class-free answer typing strategy for open-ended noun phrase questions. We begin by presenting a generative model in which the question is decomposed into multiple complementary pieces where each piece contributes to the notion of type-appropriateness for the question as a whole. This generative model explains the way in which the pieces of the decomposed questions interact, but is not able to easily incorporate new and diverse features. Moving to a discriminative model results in a loss of intuitive clarity as to how information from the question contributes to type, but is a great deal more flexible when it comes to adding new and diverse features.

The remainder of this chapter is organized as follows. We begin in Section 3.1 by introducing the resources available for a model of answer type, especially those that represent the general world knowledge necessary for typing. We will then introduce a generative model in Section 3.2 that makes use of these resources to perform class-free typing. Because of some of the shortcomings inherent in the generative model, we move to a discriminative model in Section 3.3 that is both better-performing and more flexible. Both models are experimentally evaluated and their performance is compared to alternative typing strategies in Section 3.4. Conclusions are drawn in Section 3.5 before moving to the next chapter where typing for a new class of questions is studied.

3.1 Resources

Before developing an answer typing model, it is useful to describe those linguistic resources that are available for analyzing a question and determining responses that are most suited to it. The resources listed here are key components of both the initial generative answer typing model of Section 3.2 and of the follow-on discriminative preference ranking model of Section 3.3. Therefore, these resources may be considered as the basis of our open-ended noun phrase typing as a whole.

The model we develop for open-ended noun phrase questions is based exclusively on limited corpus statistics rather than the Web. A number of systems (e.g., [5]) have used the idea of *answer projection* to find answers to a given question on the Web and then *project* the answer upon the corpus (i.e., find documents in the corpus that have the answer found on the Web). This approach has high performance and does not require complex techniques,

but is only applicable to general questions for which the Web is an appropriate corpus. If we instead move to domain-specific QA (e.g., medical), we cannot expect to have Web-scale resources at our disposal and must once again rely on limited corpus statistics. For this reason, we restrict ourselves to limited corpus statistics for general open-ended noun phrase questions even though answer projection from the Web would likely provide better overall performance.

3.1.1 Word Clusters

The data we find in a corpus is extremely sparse. We cannot expect to see every possible occurrence of every word in any corpus, regardless of its size. Instead, we wish to abstract specific words to their general concepts. For example, it may not be as important to keep track of what words follow the verb *eat*, but rather that most tend to fall into the concept of *food*. Word clusters allow us to perform this abstraction from specific words to general concepts. An abstract cluster may not correspond to any particular word, but instead represents a concept that includes words that were not observed occurring in that particular context in the corpus.

Word clusters may be created in a number of different ways. Some are as simple as a thesaurus whereas others use more advanced techniques [7, 16, 53, 31]. The techniques used to obtain word clusters are not the focus of this thesis; what is important is that we have word clusters based on some notion of similarity between words. Therefore, we use the Clustering By Committee (CBC) algorithm [51] developed locally at the University of Alberta [50]. CBC uses the idea of a *committee* that is the core of each cluster; the committee contributes to the cluster centroid rather than all cluster elements. A new element must be close to this reduced centroid to be added to this cluster. CBC has shown good performance and can be built automatically on a corpus of text with no manual labeling. To get a feel for the kinds of clusters CBC creates, the following is an example cluster created for the *negative feeling* concept, although CBC is not currently capable of assigning such a meaningful cluster name:

> tension, anger, anxiety, tensions, frustration, resentment, uncertainty, confusion, conflict, discontent, insecurity, controversy, unease, bitterness, dispute, disagreement, nervousness, sadness, despair, animosity, hostility, outrage, discord, pessimism, anguish, ...

In general, a word belongs to as many clusters as there are senses for that word. However, the idea of a sense under the CBC clustering is not always what a human annotator

Word	Clusters
suite	*Cluster*$_1$: software, network, wireless, ...
	Cluster$_2$: rooms, bathrooms, restrooms, ...
	Cluster$_3$: meeting room, conference room, ...
ghost	*Cluster*$_1$: rabbit, squirrel, duck, elephant, frog, ...
	Cluster$_2$: goblins, ghosts, vampires, ghouls, ...
	Cluster$_3$: punk, reggae, fold, pop, hip-hop, ...
	Cluster$_4$: huge, larger, vast, significant, ...
	Cluster$_5$: coming-of-age, true-life, ...
	Cluster$_6$: clouds, cloud, fog, haze, mist, ...

Table 3.1: Example clusters for words

would select. Rather, the senses are based on how the word is used in text relative to other similar words. Table 3.1 shows two example words and the clusters which loosely correspond to their multiple senses. For our cluster resources, we build CBC clusters on 10 GB of English newswire text sourced from the AQUAINT corpus and TREC disks to obtain a total of 3607 clusters.

3.1.2 Contexts

The contexts in which a word appears, whether it be nearby words or portions of a parse tree, often gives clues as the semantic type of that word. For example, the word *monitor* used in the context *computer monitor* indicates that the word refers to a particular device. If we see the same word in the context *hall monitor* instead, we know that it is referring to a person rather than a device. This basic idea of using context to infer some notion of type has been exploited by many proposals for distributional similarity and clustering [13, 38, 53].

Following the work of Lin and Pantel [40], we define contexts of a word to be the undirected paths in dependency trees involving that word at either the beginning or the end. Minipar [39] is used to produce the dependency tree parses for sentences from which the contexts are extracted. An example dependency tree parse for the sentence *the quick brown fox jumped over the lazy dog* is shown in Figure 3.1.

The links in the tree of Figure 3.1 represent dependency relationships. Dependency relationships are in the form of *X modifies Y* which means that *X* provides additional information to the basic information provided by *Y*. We can also say that *Y* is the *head* of this relationship, whereas *X* is the *modifier*. The direction of the arrow representing a link is from the head to the modifier in the relationship. Labels associated with the links represent the type of relation connecting the head with the modifier. Within a context we may wish to

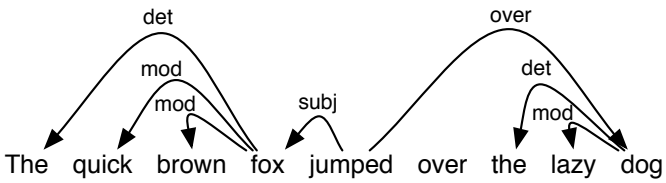


Figure 3.1: Parse tree for *The quick brown fox jumped over the lazy dog*

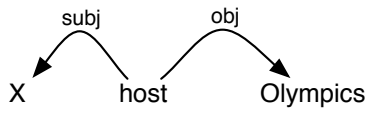
leave one endpoint empty and replace it with a variable *X*. We call any word that replaces *X*, as found in a corpus, to be a *filler* of the context. The concepts of contexts and fillers enables the separation of the two for easier manipulation, especially if we want to look up which clusters (not just words) are most likely to occur in a given context.

The links of Figure 3.1 can be chained together to form *paths* through the dependency tree from one word to another. These paths behave similarly to the individual links (or *relations*), except that longer paths are increasingly more specific. We can use the length of the path to balance between the specificity of the context (i.e., a length-2 path is more specific than a length-1 path) and the sparseness of data. Longer paths contain more information but have fewer fillers. With only a few fillers observed in a corpus, we cannot draw strong conclusions about what is likely to fill a given context. As a result, we restrict ourselves to paths of length at most two (involving at most three words), although we also make use of shorter paths.

To create a database of contexts and fillers, we parsed the entire AQUAINT corpus composed of 3 GB of newswire text with the Minipar dependency parser [39] and collected the frequency counts for words appearing in all contexts. This parsing and collection of contexts need only be done once and can be used for all of our experiments, both in this chapter and in subsequent chapters. An example context and its fillers are shown in Figure 3.2.

Question Contexts

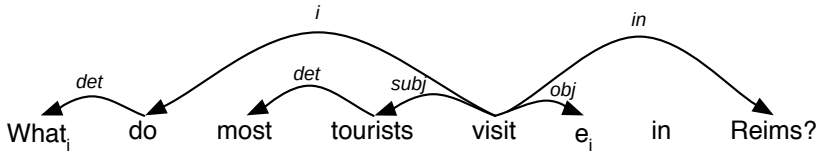
Dependency tree path contexts are useful for both questions and sentences we find in our corpus. For *question contexts*, we parse the question in the same way as a regular sentence. However, the question requires an as-yet unknown answer that is represented by the *wh*-word. This is generally true for the *what* and *which* open-ended noun phrase questions we consider in this chapter. Therefore, question contexts concern the *wh*-word which is a placeholder for the answer. We extract two kinds of question contexts. The first rule applies



Filler	Count	Filler	Count
Athens	16	bidding	4
city	13	money	4
bid	5	Osaka	4
continent	5	Zakopane	4
Moscow	5	Atlanta	3
Beijing	4	China	3

Figure 3.2: An example context and most frequent fillers

if the parse tree contains a trace for the *wh*-word. Minipar generates a trace for a *wh*-word to indicate the role of the answer in the deep structure of a sentence. If a trace is present, it indicates how the answer would appear in an idealized form of the question. For example, a question such as *what do most tourists visit in Reims?* is parsed as:



Here, the symbol e_i is the trace of the *wh*-word *what*$_i$. This parse tree provides us the following question contexts:

Context	Explanation
X ←obj— visit —subj→ tourist	tourist visits X
X ←obj— visit —in→ Reims	visit X in Reims

The second rule deals with situations where the wh-word is a determiner, as in the question *which city hosted the 1988 Winter Olympics?* (the parse tree for which is in Figure 3.3). For these questions, the noun the determiner is attached to is said to provide a *question focus*. In such cases, we extract an additional question context involving the noun that is modified by the determiner. The additional context for the above sentence is:

Context	Explanation
X ←subj— city	X is a city

We create this context because the question explicitly states that the desired answer is a city. For the generative model introduced in Section 3.2, we use this additional context

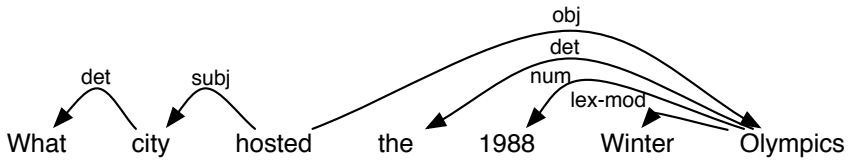


Figure 3.3: Parse tree for *what city hosted the 1988 Winter Olympics?*

alone (i.e., it overrides any other contexts) whenever we can extract it because the question explicitly states the desired answer (in this case, a city). Experiments during development showed that using this context in conjunction with other contexts extracted from the question produces lower performance than using this context alone for the generative model.

In the event that a context extracted from a question is not found in the database, we shorten the context in one of two ways. We start by replacing the word at the end of the path with a wildcard that matches any word. If this fails to yield entries in the context database, we shorten the context to length one and replace the end word with automatically determined similar words instead of a wildcard. This context shortening is of particular benefit to the generative model introduced in this Section 3.2, but is not as useful for the subsequent discriminative model and so is only used in the generative case.

Candidate Contexts

Candidate contexts are very similar in form to question contexts except for one important difference: candidate contexts are extracted from the parse trees containing candidate answers rather than the question. In natural language some words may have more than one meaning. For example, *Washington* may refer to a person, a city, or a state. The occurrence of *Washington* in *Washington's descendants* and *suburban Washington* should not be given the same score when the question is seeking a location. Given that the sense of a word is largely determined by its local context [12], candidate contexts allow the model to take into account the candidate answers' senses implicitly.

Candidate contexts are not always available to our system. For the model of Section 3.2, we will introduce two variants; one with candidate contexts and one without. The use of candidate contexts is dependent on whether or not we have the resources available to parse the candidate answer and its sentence. This requirement is not always reasonable, especially when this means either parsing a very large corpus ahead of time (e.g., the Web) or parsing results on the fly (which will add a great deal of overhead to answer typing). As a result, for those experiments in which we use the AQUAINT corpus as a source of

candidate answers, we have pre-parsed sentences available to provide candidate contexts. For those experiments in which we use another source for candidate answers, we do not use candidate contexts.

3.2 An Initial Generative Model

Building on the resources of Section 3.1, we introduce a probabilistic generative model to determine the appropriateness of candidate answers without the use of fixed types (i.e., a class-free approach). This model was first presented by Pinchak and Lin [57]. The goal of this model is to provide a probability of some given candidate answer t being appropriate as a response to a given question Q. Given the resources available to us above, we say that the problem of finding whether or not t fits the contexts extracted from Q is equivalent to the problem of whether or not t is appropriate to Q. This means that our model will assign high probability to a candidate t *iff* it is likely to appear in the question contexts. We believe this is a reasonable expectation because the question contexts are extracted from the question in such a way that the desired, but unknown, answer is the missing piece to be occupied by fillers.

Once probabilities are assigned, the candidates can be sorted according to how likely each candidate is to be appropriate and we expect appropriate answers to appear near the top. This ranking allows for some leniency in terms of accuracy. Admitting an incorrect answer near the top of the list is not a catastrophic mistake; rather we just allow the possibility of an inappropriate candidate being selected as the correct answer. This model is not designed to be used in isolation; it must be combined with document retrieval and answer extraction modules to form a true model of whether or not an answer is correct.

Given our assumption of the equivalence of appropriateness with likelihood of occurring in question contexts, we now present a model for the likelihood of occurrence in question contexts. We must assume that some set of candidate answers are provided by an external source, be it an answer extraction module or a simple script to collect all nouns from a set of documents. This means that our model does not consider $\Pr(t)$ when computing the likelihood of t occurring in the set of question contexts Γ_Q. In other words, we are computing $\Pr(in(t, \Gamma_Q)|t)$, that is the probability that candidate t fits into the question contexts Γ_Q ($in(t, \Gamma_Q)$) given the fact that we know t has occurred (or has been selected *a priori* as a potential answer).

We can evaluate the value of $\Pr(in(t, \Gamma_Q)|t)$ directly, but there are several complicating

factors. The first complication is the fact that our corpus does not contain enough data to show every possible t in Γ_Q. This data sparseness will be present regardless of the corpus we choose, although larger corpora tend to mitigate this problem somewhat. This means that for some t, we may have never observed it occurring in any or all of the question contexts Γ_Q and so we cannot compute an accurate probability. To address this, we introduce a hidden variable C that represents the clusters to which t belongs. The introduction of C to our model gives:

$$\Pr(in(t, \Gamma_Q)|t) = \sum_C \Pr(in(t, \Gamma_Q), C|t) \tag{3.1}$$

$$= \sum_C \Pr(C|t)\Pr(in(t, \Gamma_Q)|C, t) \tag{3.2}$$

We make a further assumption here that the cluster C supersedes the information contained in the word t. That is, C now is a representative for t, at least in terms of how likely it is to occur in the question contexts Γ_Q. As an example, consider a case in which we have the candidate *Washington* as above. If our current cluster for *Washington* represents major U.S. cities, then we are abstracting the knowledge that we are dealing with *Washington* in particular and are asking how likely is it for a U.S. city to appear in the question contexts Γ_Q instead. This provides a convenient generalization from word (*Washington*) to cluster (U.S. cities) that will help with data sparseness. This substitution is formalized as:

$$\Pr(in(t, \Gamma_Q)|C, t) \approx \Pr(in(C, \Gamma_Q)|C) \tag{3.3}$$

We can now rewrite Equation 3.2 as:

$$\Pr(in(t, \Gamma_Q)|t) \approx \sum_C \Pr(C|t)\Pr(in(C, \Gamma_Q)|C) \tag{3.4}$$

Our model is now conveniently split into two pieces. The first deals with how likely a cluster C is given a candidate t and the second deals with how likely a cluster C is to appear in the question contexts Γ_Q. The former of these pieces is relatively straightforward to consider as we merely have to ask how likely a candidate is to belong to some cluster. The latter piece presents a problem when Γ_Q consists of multiple question contexts derived from the question. In cases where Γ_Q consists of multiple contexts, we make a naïve Bayes assumption that each individual context $\gamma_Q \in \Gamma_Q$ is independent of all other contexts given the cluster C. With this assumption, our model becomes:

$$\Pr(in(t, \Gamma_Q)|t) \approx \sum_C \Pr(C|t) \prod_{\gamma_Q \in \Gamma_Q} \Pr(in(C, \gamma_Q)|C) \tag{3.5}$$

This is our initial model that does not rely on candidate contexts. We have resources available to assist in computing each of the components of Equation 3.5, although the details of exactly how these parameters are estimated are explained in Section 3.2.2. Next we build upon our initial model and add candidate contexts.

3.2.1 Introducing Candidate Contexts

The model of Equation 3.5 assigns the same likelihood to every sense of a candidate answer. This means that the model only considers a candidate such as *Washington* as being equally likely to be a city as a person. However, the way this instance of *Washington* is used in text will have an influence on which sense (and hence cluster) is more likely to represent it. For this case, we are actually computing the value $\Pr(in(t,\Gamma_Q)|t,in(t,\Gamma_t))$. The extra information here is knowledge that t occurs in the candidate contexts Γ_t. If we once again introduce clusters as hidden variables and make the assumption that clusters supersede words, we arrive at the following model:

$$\Pr(in(t,\Gamma_Q)|t,in(t,\Gamma_t)) = \sum_C \Pr(in(t,\Gamma_Q),C|t,in(t,\Gamma_t) \tag{3.6}$$

$$\approx \sum_C \Pr(C|t,in(t,\Gamma_t))\Pr(in(C,\Gamma_Q)|C) \tag{3.7}$$

Once again, the model we arrive at in Equation 3.7 is split into two parts. The latter part, $\Pr(in(C,\Gamma_Q)|C)$ is the same as in Equation 3.4 but the former part now depends on $in(t,\Gamma_t)$ ($\Pr(C|t,in(t,\Gamma_t))$). This means that the cluster C representing candidate t now depends on the contextual information around t (the candidate contexts). Because the candidate contexts, like the question contexts, may include multiple elements, we once again use the naïve Bayes assumption and compute this term as:

$$\Pr(C|t,in(t,\Gamma_t)) = \frac{\Pr(in(t,\Gamma_t)|t,C)\Pr(t,C)}{\Pr(in(t,\Gamma_t|t)\Pr(t)} \tag{3.8}$$

$$\approx \left(\frac{\prod_{\gamma_t\in\Gamma_t}\Pr(in(t,\gamma_t)|t,C)}{\prod_{\gamma_t\in\Gamma_t}\Pr(in(t,\gamma_t)|t)}\right)\Pr(C|t) \tag{3.9}$$

$$= \prod_{\gamma_t\in\Gamma_t}\left(\frac{\Pr(C|t,in(t,\gamma_t))}{\Pr(C|t)}\right)\Pr(C|t) \tag{3.10}$$

It should be noted that for many cases, the set of candidate contexts contains only one context. Therefore, Equation 3.10 is unnecessary and we can simply replace Γ_t with γ_t in Equation 3.7.

Now that we have a generative model with and without the inclusion of candidate contexts (Equations 3.10 and 3.5, respectively), we must describe how the resources of Section

3.1 are used in the estimation of the parameters necessary for their computation. The details of these estimations are the subject of the next section.

3.2.2 Parameter Estimation

The generative probabilistic model relies on a number of parameters that must be estimated from our available resources. Equation 3.5, the probabilistic model without the use of candidate contexts, requires parameters $\Pr(C|t)$ and $\Pr(in(C,\gamma)|C)$. Equation 3.10 requires the same parameters in addition to $\Pr(C|t, in(t,\gamma_t))$. This section describes how these parameters are calculated given the resources of Section 3.1.

The context resources of Section 3.1 provide us with a database of contexts and fillers from which we can collect the joint and marginal frequency counts of contexts and their fillers, i.e., $|in(t,\gamma)|$, $|in(*,\gamma)|$ and $|in(t,*)|$. This further allows us to estimate basic probabilities directly, such as $\Pr(in(t,\gamma))$, $\Pr(in(t,*))$, and $\Pr(in(*,\gamma))$. We can also compute the probability $\Pr(in(t,\gamma)|t)$ directly, which will come in useful for the discriminative preference ranking model. Our word cluster resources provide us with knowledge of which clusters C a candidate t may appear in, and also the similarity between two terms t_1 and t_2 $sim(t_1,t_2)$. Word similarity is a positive real number that corresponds to the strength of similarity between two words and is a by-product of cluster creation. Most often we expect two words in the same cluster to have a higher similarity value than two words that share no cluster.

Because the calculation of $\Pr(in(C,\gamma)|C)$ requires the use of $\Pr(C|t)$, the means by which $\Pr(C|t)$ is computed will be described first. For a given candidate t, we cannot sample how often t is used in the sense represented by C because each occurrence of t in text is not labeled with the sense that is used. To combat this, we turn to the average weighted guesses (or votes) made by those neighbours of t, in which a neighbour of t is a term with high similarity to t. The idea here is that if t and its neighbour t' are similar words ($t' \in S(t)$), then $\Pr(C|t)$ and $\Pr(C|t')$ will have similar values. Averaging across multiple neighbours gives a better picture of how likely t is to be used in a certain sense:

$$\Pr(C|t) = \frac{\sum_{t' \in S(t)} sim(t,t') \times \Pr_u(C|t')}{\sum_{\{C'|t \in C'\}} \sum_{t' \in S(t)} sim(t,t') \times \Pr_u(C'|t')} \tag{3.11}$$

Of course, the calculation of $\Pr(C|t')$ in Equation 3.11 suffers from the same problem. A solution, which is adopted here, is to replace $\Pr(C|t)$ with a uniform probability estimate

$\Pr_u(C|t')$:

$$\Pr_u(C|t) = \begin{cases} \frac{1}{|\{C'|t \in C'\}|} & \text{if } t \in C, \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

This uniform estimate is very crude, but the weighted average of such crude estimates can quite often be correct. Essentially we are saying that t' occurs with uniform probability in all of its clusters and votes with weight $1/|C'|$ for its neighbour t to also be a member of some particular cluster C. Although crude estimates can be inaccurate, the weighted average of neighbours means $\Pr(C|t)$ is likely to be closer to the true value than to a uniform distribution.

Estimation of $\Pr(C|t, in(t, \gamma_t))$ departs from the simple case of $\Pr(C|t)$ by requiring that neighbours of word t must also occur in the candidate context γ_t:

$$\Pr(C|t, in(t, \gamma_t)) = \frac{\sum_{t' \in S(t) \wedge in(t', \gamma_t)} sim(t, t') \times \Pr_u(C|t')}{\sum_{\{C'|t \in C'\}} \sum_{(t' \in S(t) \wedge in(t', \gamma_t))} sim(t, t') \times \Pr_u(C'|t')} \tag{3.13}$$

This means that a neighbour t' must both share a cluster with t *and* share the context γ_t with t. This will admit fewer neighbours from which the weighted average of crude estimates (Equation 3.12) is created. The only drawback to this approach is that we may not have enough neighbours t' that actually appear in a given candidate context γ_t. For this reason we use short candidate contexts, in particular contexts of length one. Also, if γ_t does not have sufficient fillers in our database, we fall back to using $\Pr(C|t)$. Therefore, we make use of the candidate context when possible and fall back to a simpler calculation when either the candidate context is too rare or we do not have any candidate contexts at all.

The final parameter necessary for our model is $\Pr(in(C, \gamma)|C)$, and is calculated according to:

$$\Pr(in(C, \gamma)|C) = \frac{\sum_{t' \in C} \Pr(C|t') \times |in(t', \gamma)| + \Pr(in(*, \gamma))}{\sum_{t' \in C} \Pr(C|t') \times |in(t', *)| + 1} \tag{3.14}$$

For this component, we start by assuming that each instance of t in text (specifically in context γ) actually represents fractional counts for all the clusters C of t. Therefore, if we would like to know how often the concept for C occurs in text, we look for how often the words of C occur and sum up their fractional counts. For example, if we want to know how often the cluster *major Canadian cities* occurs in text, we find the occurrences for words

such as *Toronto, Montreal, Vancouver, Calgary, Edmonton, ...* and add their fractional counts to the total. These words do not contribute their full counts because they may occur in other clusters corresponding to other senses (e.g., *Vancouver* is a city as well as an island). To smooth this distribution, we use add-one smoothing [9].

3.2.3 Summary

The generative model introduced here relies mainly on the ideas of question contexts derived from the question itself and fillers for those contexts derived from a parsed corpus. Both of these resources are created automatically from existing unlabeled data. Adding candidate contexts increases the complexity of the model, especially in terms of how we decide which clusters best represent a given candidate. However, when contextual information is available for a candidate, making use of it can help improve the model as shown in the empirical results of Section 3.4.

3.3 Discriminative Preference Ranking

The generative model of Section 3.2 combines various corpus statistics to give a probabilistic model of appropriateness, but is unfortunately hampered by the fact that including additional and diverse information is very difficult. For example, if some of these probability distributions are derived from dramatically different sources, we can expect a significant degree of error in the resultant probabilities. Even worse is the inability to include new features that were not considered when developing the generative model. For example, we may wish to include some bias based on the frequency of occurrence of a candidate in the candidate list. This feature would be very difficult to incorporate into the models of Section 3.2 because this number is not based on any property related to the corpus.

Discriminative models specialize in their ability to combine a diverse set of features when making decisions on output values. Feature values can be real-valued or binary and can be drawn from any number of different distributions representing any desired concept. We have already mentioned one such discriminative learning technique, Maximum Entropy [1], in Chapter 2, when discussing class-based learned techniques of Ittycheriah et al. [26, 27]. More recently, language research has moved from maximum entropy to maximum margin approaches, most often represented as Support Vector Machines (SVMs). Because software packages are freely available for SVM-based binary classification [28], multi-class classification [15], regression or function fitting [62], and preference ranking [30], we adopt the use of SVM-based discriminative learning techniques to improve upon our answer type

model. In particular, we use the SVM^{light} package[1], a freely available implementation of SVM learning, including the quadratic program solver at the heart of the SVM learning task.

A SVM gives our model a degree of flexibility that is not possible with hand-built generative models. This increased flexibility allows us to add as-yet unanticipated features, migrate the model to a new domain, and increase performance by addressing some of the specific drawbacks of the particular generative model of Section 3.2. It is shown in Pinchak et al. [58] that application of these discriminative techniques improves performance. Furthermore, in Chapter 5, we show how this flexibility allows for the model to be used in domains for which the model was not originally designed.

3.3.1 On the Use of Preference Ranking

The ultimate goal of answer typing, like that of IR, is to produce some ranked list of responses according to how appropriate they are to the query. Although QA in general must find a single correct answer, answer typing must find multiple type-appropriate responses from a set of candidate answers. Initially, this seems like a straightforward binary classification problem; a candidate is either appropriate or not. However, a set of candidates are all often related in some way to the question and so we wish to determine which candidates are *more* appropriate than others. In effect, we want to produce a ranked list according to type-appropriateness.

Preference ranking naturally lends itself to any problem in which the relative ordering among examples is more important than the specific labels or values assigned to those examples. The classic example application of preference ranking is that of information retrieval results ranking [30]. Generally, information retrieval results are presented in some ordering such that those higher on the list are either more relevant to the query or would be of greater interest to the user. Information retrieval results can be influenced by a number of signals including the document contents, highlighted or prominent words in the document (such as the title or headline), and the general popularity of the source of the document. Discriminative preference ranking allows the inclusion of these various features when determining how highly in a ranking a result should appear.

In a general preference ranking task we have a set of candidates $c_1, c_2, ..., c_n$, and a ranking r such that the relation $c_i <_r c_j$ holds *iff* candidate c_i should be ranked higher than c_j, for $1 \leq i, j \leq n$ and $i \neq j$. The ranking r can form a total ordering, as for

[1] `http://svmlight.joachims.org/`

information retrieval results, or a partial ordering in which we have both $c_i \not<_r c_j$ and $c_j \not<_r c_i$. Partial orderings are of particular interest for answer typing because they can be used to specify candidates that are of an equivalent rank. That is, we may specify two answer candidates that are either both appropriate or both inappropriate. Answer typing is only meant to discover appropriate candidates and we cannot assess the relative magnitude of two appropriate (or inappropriate) candidates. Therefore, a partial ordering can be used to specify that appropriate candidates be ranked higher than inappropriate candidates, but say nothing about how appropriate candidates are ranked in relation to one another.

Given some $c_i <_r c_j$, preference ranking only considers the difference between the feature representations of c_i and c_j (respectively denoted by $\Phi(c_i)$ and $\Phi(c_j)$) as evidence. The particular features used with this model are introduced in Section 3.3.3, but any set of features is applicable in the general case. We want to learn some weight vector $\vec{w}$ such that $\vec{w} \cdot \Phi(c_i) > \vec{w} \cdot \Phi(c_j)$ holds for all pairs c_i and c_j that have the relation $c_i <_r c_j$. In other words, we want $\vec{w} \cdot (\Phi(c_i) - \Phi(c_j)) > 0$. Merely requiring that $\vec{w} \cdot \Phi(c_i) > \vec{w} \cdot \Phi(c_j)$ does not generalize well to future examples, and so we introduce a *margin* to help correctly evaluate unseen candidates. The use of a margin follows the techniques of Support Vector Machine learning (using max-margin) [29] and has shown to have good performance. In the context of Support Vector Machines [30], we are trying to find the parameters that minimize:

$$\underset{\vec{w},\xi}{\operatorname{argmin}} \left(\frac{1}{2} \vec{w} \cdot \vec{w} + C \sum \xi_{i,j} \right) \tag{3.15}$$

subject to the constraints:

$$\forall (c_i <_r c_j) : \vec{w} \cdot (\Phi(c_i) - \Phi(c_j)) \geq 1 - \xi_{i,j} \tag{3.16}$$

$$\forall i, j : \xi_{i,j} \geq 0 \tag{3.17}$$

The margin incorporates slack variables $\xi_{i,j}$ for problems that are not linearly separable. These slack variables essentially allow for violations of the constraints by removing them from consideration and imposing a penalty in their place. For example, if $\Phi(c_j) < \Phi(c_i)$ for some candidates c_i and c_j, we can either modify the weight vector $\vec{w}$ and hence the separating hyperplane in terms of a SVM or we can leave $\vec{w}$ unchanged and instead apply the penalty $\xi_{i,j}$. In general, slack variables allow for learning a classifier for a problem that is linearly non-separable, hence a SVM without slack variables would not be able to find a solution. This ranking task is analogous to the SVM classification task on the pairwise difference vectors $(\Phi(c_i) - \Phi(c_j))$, known as *rank constraints*. Unlike classification, no

explicit negative evidence is required as $\vec{w} \cdot (\Phi(c_i) - \Phi(c_j)) = (-1)\vec{w} \cdot (\Phi(c_j) - \Phi(c_i))$. It is also important to note that no rank constraints are generated for candidates for which no order relation exists under the ranking r.

3.3.2 Preference Ranking in the Context of Answer Typing

Answer typing is not a task that can be easily defined as classification or function approximation. This is in large part because typing must often deal with candidates that are all related in some way to the question, whether it be by appropriateness or not. The feature representation of questions cannot always capture the difference between appropriate and inappropriate and a simple binary classification of *is-appropriate* or *is-not-appropriate* is insufficient. Similarly, we cannot assign a real value representing how appropriate a given candidate is because such numbers are extremely difficult for even annotators to assess. For these reasons, we approach answer typing from the perspective of preference ranking.

Preference ranking is not the only possible method by which we can build a model of answer type. Although classification is not conceptually as fitting as preference ranking, it still allows for the creation of a typing model should the problem of unbalanced training data be overcome. Answer typing provides a highly-unbalanced set of labeled examples; there are far more negative examples (inappropriate candidates) than positive examples (appropriate candidates). In general, classifiers have difficulty with unbalanced examples because there is an incentive to simply label all examples as negative and accept the small number of errors on the positive side. For example, if we have 100 training examples and 90 are negative examples, we can achieve a 10% error rate by assigning all 100 training examples as negative (10 errors for the positive examples). In answer typing we often have an imbalance close to the above example and so we run the risk of assigning all future examples a negative label. Attempts to balance the data by both increasing the weight of positive examples or selecting a balanced set of random negative examples both provide worse performance than preference ranking [58].

To apply preference ranking to answer typing, we learn a model over a set of questions $q_1, ..., q_n$. For each question q_i we have as training data a list of appropriate candidate answers $a_{(i,1)}, ..., a_{(i,u)}$ and a list of inappropriate candidate answers $b_{(i,1)}, ..., b_{(i,v)}$. The partial ordering for our answer type model, r, is the set:

$$\forall i, j, k : \{a_{(i,j)} <_r b_{(i,k)}\} \tag{3.18}$$

This means that rank constraints are only generated for candidate answers $a_{(i,j)}$ and $b_{(i,k)}$

Pattern	Description
$E(t, \gamma)$	Estimated count of candidate t in context γ
$N(t, \gamma)$	Observed count of candidate t in context γ
$\sum_{t'} N(t', \gamma)$	Count of all fillers of context γ
$\sum_{\gamma'} N(t, \gamma')$	Count of candidate t regardless of context
$F(t)$	Count of the times candidate t occurs in the candidate list

Table 3.2: Discriminative feature templates

for question q_i and not between candidates $a_{(i,j)}$ and $b_{(l,k)}$ where $i \neq l$. For example, the candidate answers for the question *what city hosted the 1988 Winter Olympics?* are not compared with those for *what colour is the sky?* because our partial ordering r does not attempt to rank candidates for one question in relation to candidates for another. Moreover, no rank constraints are generated between $a_{(i,j)}$ and $a_{(i,k)}$ nor $b_{(i,j)}$ and $b_{(i,k)}$ because the partial ordering does not include orderings between two candidates of the same class. Given the question *what city hosted the 1988 Winter Olympics?* and two appropriate candidates *New York* and *Calgary*, rank constraints will not be created for the pair (*New York*, *Calgary*).

3.3.3 Model Details

The basis of our discriminative preference ranking model is built upon the resources of Section 3.1. Using these resources, in combination with the question and the list of candidate answers, we arrive at the feature templates of Table 3.2. These features are considerably easier to obtain than the parameters of our generative model and for this reason answer typing based on discriminative preference ranking is an attractive alternative to the generative model.

Because our discriminative preference ranking model is learned based on features obtained from both the question and a candidate answer, we must be careful not to provide a space of too many unique features. For example, if a feature such as $N(t, \gamma)$ (the observed count of candidate t in question context γ) contains a γ unique to this question, any values in the weight vector for the element corresponding to $N(t, \gamma)$ will have no effect on future questions; no future question ever uses this question context γ. Although in the limit we can expect there to be eventual repetition of question contexts (especially when they are short), we often do not see this for limited training and testing data.

To address the potential uniqueness of question contexts, we remove lexical elements from question contexts. This step is performed after feature values are obtained for the fully lexicalized path; the removal of lexical elements simply allows many similar contexts

to share a single learned weight. For example, the term *Calgary* in context *X* $\leftarrow$ *subj* $\leftarrow$ host $\rightarrow$ *obj* $\rightarrow$ Olympics, *X* hosted Olympics, is used to obtain a feature value of 3 that is assigned to a feature such as $N(\textit{Calgary}, X \leftarrow subject \leftarrow * \rightarrow object \rightarrow *) = 3$. Removal of lexical elements greatly reduces the number of contexts and hence the number of features produced by the templates of Table 3.2.

The expected number of candidate answer t in context γ, $E(t, \gamma)$, is based on a portion of the generative model and is calculated as:

$$E(t, \gamma) = \sum_{C} \Pr(C|t) N(C, \gamma) \tag{3.19}$$

Essentially, this equation computes an expected count for candidate t in question context γ by observing how likely t is to be part of a cluster C ($\Pr(C|t)$) and then observing how often terms of cluster C occur in context γ ($N(C, \gamma)$). This is a simplified version of what is used in the discriminative model of Equation 3.5 in which the products of probabilities $\Pr(in(C, \gamma')|C)$ over all γ' is computed instead of $N(C, \gamma)$ for a single γ. The generative model is unable to assign individual weights to different question contexts, even though not all question contexts are equally important. For example, the generative model is forced to consider a question focus context (such as "*X* is a city") to be of equal importance to non-focus contexts (such as "*X* host Olympics"). However, we have observed that it is more important that candidate *X* is a city than it hosted an Olympics in this instance. In fact, for the results reported in Pinchak and Lin [57] it was shown that higher performance is obtained by ignoring all contexts other than a focus context when a focus context is present in a question. We address this problem with the use of discriminative methods.

The observed count features of candidate t in context γ, $N(t, \gamma)$, are included to allow for combination with the estimated values described previously. Because the estimated counts make use of cluster-based smoothing, errors may occur. By including the observed counts of candidate t in context γ, we hope to allow for the use of more accurate statistics whenever they are available, and for the smoothed counts in cases for which they are not.

Finally, we include the frequency of a candidate t in the list of candidates, $F(t)$. The idea here is that the correct and/or appropriate answers are likely to be repeated many times in a list of candidate answers. Terms that are strongly associated with the question and appear often in results are likely to be what the question is looking for.

The two features $N(t, \gamma)$ and $F(t)$ are significantly different than the values used in the generative model and can be incorporated into the generative model with varying degrees of difficulty. The value of $F(t)$ in particular is highly dependent on the means used to

obtain the candidate list, and the distribution of words over the candidate list is often very different from the distribution of words in the corpus. For this very reason, the generative model computes the conditional probability $\Pr(in(t, \Gamma_Q)|t)$ instead of the joint probability $\Pr(in(t, \Gamma_Q), t)$. However, we wish to include some knowledge of how frequent a word is in the candidate list into the model even though factors that influence its inclusion in the candidate list may not be based in any way on the corpus. Because this feature value comes from a different source than our other features, it would be difficult to use this feature in a non-discriminative model.

We introduce four variants of the discriminative preference ranking answer typing model, varying on how the candidates are labeled as appropriate:

Correctness Model Although appropriateness and correctness are not synonymous, this model deals with distinguishing correct from incorrect candidates in the hopes that the resulting model will be able to perform well on finding both correct and appropriate answers. For learning, correct answers are placed at a rank above that of incorrect candidates, regardless of whether or not those candidates are appropriate. This represents the strictest definition of appropriateness and requires no annotator labels beyond those for correct answers. Such labels are freely available from TREC data sets but must be provided by human annotators in other cases.

Appropriateness Model The correctness model assumes only correct answers are appropriate. In reality, this is seldom the case. For example, documents or snippets returned for the question *what country did Catherine the Great rule?* will contain not only *Russia* (the correct answer), but also *Germany* (the nationality of her parents) and *Poland* (her modern-day birthplace). To better address this overly strict definition of appropriateness, we rank all candidates labeled as appropriate above those labeled as inappropriate, without regards to correctness. Because we want to learn a model for appropriateness, training on appropriateness rather than correctness information may produce a model closer to what we desire.

Combined Model Discriminative preference ranking is not limited to only two ranks. We combine the ideas of correctness and appropriateness together to form a three-rank combined model. This model places correct answers above appropriate-but-incorrect candidates, which are in turn placed above inappropriate-and-incorrect candidates.

Reduced Model Both the appropriateness model and the combined model incorporate a

large number of rank constraints. We can reduce the number of rank constraints generated by simply removing all appropriate, but incorrect, candidates from consideration and otherwise following the correctness model. The main difference is that many appropriate (incorrect) candidates are no longer assigned a low rank. By removing appropriate, but incorrect, candidates from the generation of rank constraints, we no longer rank correct answers above appropriate candidates.

3.3.4 Summary

The four aforementioned models of answer type based on discriminative preference ranking are meant to allow for a greater diversity of features and a greater degree of flexibility in feature weighting than the generative model permits. The only significant drawback of these models is the requirement of having supervised training data in the form of annotator-labeled appropriate candidates, although the correctness model variant needs only correct answers. The discriminative preference ranking method of answer typing can be viewed as an extension of the generative model in which the feature values are easier to obtain and more diverse in their composition. As we will see in the following section, this increase in flexibility provides a considerable performance gain over the generative model.

3.4 Experiments

The ultimate goal of our answer typing models for open-ended noun phrase questions is to identify a set of high-quality type-appropriate potential answers from a set of answer candidates. In the context of Figure 1.2, we want to reorder the input such that those closest to the top are those most appropriate as responses. In what follows, we show how our generative class-free answer typing model improves performance over a strategy based on the use of a fixed set of types (a class-based approach) and then how discriminative preference ranking is able to improve upon the performance of the generative model.

It is worth noting here that the proposed answer typing models used in these experiments are often limited in scope due to the relatively small corpus from which the resources described in Section 3.1 (the AQUAINT corpus) are derived. This means that for some portion of questions, our models can provide none of the parameters required by the model (no parameter estimates in the generative model or no feature values in the discriminative models). For example, a question such as *what organelle is responsible for photosynthesis?* would likely have no question context information in a newswire corpus due to the fact that it uses biology terms. Because of the high degree of performance offered by compo-

sitional QA approaches (such as that of Prager et al. [60]), we suggest the application of these models in combination with baseline models and/or other typing strategies. Therefore, in our experiments we focus on those questions for which we can actually apply our model and omit those for which our model has insufficient data. Although this set may seem small, increasing the corpus size for our resources (for example, Web-scale corpora) will also increase the number of questions to which our models can be applied.

3.4.1 Benefits of the Generative Model

The primary goal of our generative model is to show the opportunity for improvement over a class-based approach. To this end, we compare our generative model with two alternative systems, both of which are idealized to some degree. The first of these systems, the *Oracle* system, is given perfect knowledge of the question class and a set of only those candidates that are appropriate for that type (manually labeled in both cases). The *ANNIE*-based approach is also given perfect knowledge of the question class but must make use of the ANNIE named-entity recognition system [44] to decide which candidates are type-appropriate. The input candidate list is derived by applying the ANNIE named-entity tagger to the top 10 matching documents of each question, as ranked by NIST's PRISE[2] search engine. For the class-based comparison systems, we use a set of MUC-based named entity types augmented with *Thing-name* representing proper names of inanimate objects and *Miscellaneous* to catch unknown questions and entities. Some examples of *Thing-name* are *Guinness Book of World Records*, *Thriller*, *Mars Pathfinder*, and *Grey Cup*. Examples of *Miscellaneous* candidates are *copper*, *oil*, *red*, and *iris*.

We evaluate each answer typing system by measuring its performance in filtering type-appropriate candidates. Each answer candidate is scored by the answer typing system and the list is sorted in descending order of score. We then observe the proportion of candidates that must be accepted by the filter so that at least one correct answer is accepted. A model that allows a low percentage of candidates to pass while still allowing at least one correct answer through is favorable to a model in which a high number of candidates must pass. The evaluation data consist of 154 questions from the TREC-2003 QA Track [70] satisfying the following criteria:

- The *wh*-word of the question is *what*, *which*, or *who*. This restricted the evaluation to questions that often seek open-ended noun phrases as answers. It should be noted

[2] `http://www.itl.nist.gov/iad/894.02/works/papers/zp2/zp2.html`

that this set of questions from this particular TREC conference are primarily factoid, and include very few definition questions.

- There exists entry for the question in the answer patterns[3] so that the question is known to be answerable.

- At least one of the top-10 documents from which candidates are extracted contains a correct answer.

The generative model of Section 3.2 assigns a probability value for each candidate, and these probabilities can be used for the purpose of ranking. For the comparison systems, the score for a candidate is either 1 if it is tagged as the same type as the question or 0 otherwise. With only two possible scores for each candidate, there are potentially many ties which results in an arbitrary ranking in which a candidate is ranked low despite having the same score as a highly-ranked candidate. To address this problem, we compute the probability of the first correct answer appearing at rank $R = k$ as follows:

$$\Pr(R = k) = \left(\prod_{i=0}^{k-2} \frac{t - c - i}{t - i} \right) \frac{c}{t - k + 1} \tag{3.20}$$

where t is the number of unique candidate answers that are of the appropriate type and c is the number of unique candidate answers that are correct. The first portion of Equation 3.20 is the probability that the first $k - 1$ positions are occupied by incorrect answers with a correct type and the second part is the probability that a correct answer is in the k^{th} position. Using the probabilities computed by Equation 3.20, we compute the expected rank, $E(R)$, of the first correct answer of a given question in the system as:

$$E(R) = \sum_{k=1}^{t-c+1} k \Pr(R = k) \tag{3.21}$$

The median number or percent of candidates that are accepted by a filter over the questions of our evaluation data is one measure of overall performance of the model. Another measure is to observe the number of questions with at least one correct answer in the top $N\%$ for various values of N. By examining the number of correct answers found in the top $N\%$ we can better understand what an effective filter cutoff would be.

The overall results of our comparison can be found in Table 3.3. For a comparison with a simple, yet effective, strategy, we have added the results of a system that scores candidates based on their frequency within the top 10 documents. The second column is the median

[3] `http://trec.nist.gov/data/qa/2003_qadata/03QA.tasks/t12.pats.txt`

System	Median %	Top 1%	Top 5%	Top 10%	Top 50%
Oracle	0.7%	89 (57%)	123 (79%)	131 (85%)	154 (100%)
Frequency	7.7%	31 (20%)	67 (44%)	86 (56%)	112 (73%)
Our model	**1.2%**	**71 (46%)**	**106 (69%)**	**119 (77%)**	**146 (95%)**
Γ_Q **only**	**2.2%**	**58 (38%)**	**102 (66%)**	**113 (73%)**	**145 (94%)**
ANNIE	4.0%	54 (35%)	79 (51%)	93 (60%)	123 (80%)

Table 3.3: Generative model performance summary

percentage of where the highest scored correct answer appears in the sorted candidate list. Low percentage values mean the answer is usually found high in the sorted list. The remaining columns list the number of questions that have a correct answer somewhere in the top $N\%$ of their sorted lists. This is meant to show the effects of imposing a strict cutoff prior to running the answer type model.

Oracle performs best, as it benefits from both ideal (manual) assignment of type to question and ideal named entity recognition. If entity recognition is performed by an automatic system (as it is for *ANNIE*), the performance drops noticeably. Our generative model performs better than *ANNIE* and achieves approximately 2/3 of the performance of *Oracle*. Table 3.3 also shows that the use of candidate contexts as discussed in Section 3.2.1 increases the performance of our answer type model. Evidence for this claim can be seen in the fact that the Γ_Q only model, our model without candidate contexts, performs slightly worse than the model including candidate contexts.

To understand where gains are being made by the generative model, and to see if the generative model offers any performance benefits over the *Oracle* system, we show a breakdown of performance for questions based on the manually-assigned answer class in Table 3.4. When compared with *Oracle*, our model performs worse overall for questions of all types except for those seeking *miscellaneous* answers. For *miscellaneous* questions, *Oracle* identifies all tokens that do not belong to one of the other known categories as possible answers. This means that the expected rank of the first correct answer is computed from nearly the entire list of candidates instead of a small subset that are of the appropriate type. For all questions of non-*miscellaneous* type, only a small subset of the candidates are marked appropriate. In particular, our model performs worse than *Oracle* for questions seeking persons and thing-names. *Person* questions often seek rare person names, which occur in few contexts and are difficult to reliably cluster. *Thing-name* questions are easy for a human to identify but difficult for automatic system to identify. *Thing-names* are a diverse category and are not strongly associated with any identifying contexts.

System	Measure	Question Type (# of questions) All (154)	*Loc* (57)	*Person* (17)	*Org* (19)	*Thing-name* (17)	*Misc* (37)	Other (7)
Ours	Median	1.2%	0.8%	2.0%	1.3%	3.7%	3.5%	12.2%
	Top 1%	71	34	6	9	7	13	2
	Top 5%	106	53	11	11	10	19	2
	Top 10%	119	55	12	17	10	22	3
	Top 50%	146	56	16	18	17	34	5
Oracle	Median	0.7%	0.4%	1.0%	0.3%	0.4%	16.0%	0.3%
	Top 1%	89	44	8	16	14	1	6
	Top 5%	123	57	17	19	17	6	7
	Top 10%	131	57	17	19	17	14	7
	Top 50%	154	57	17	19	17	37	7
ANNIE	Median	4.0%	0.6%	1.4%	6.1%	100%	16.7%	50.0%
	Top 1%	54	39	5	7	0	0	3
	Top 5%	79	53	12	9	0	2	3
	Top 10%	93	54	13	11	0	12	3
	Top 50%	123	56	16	15	5	28	3

Table 3.4: Detailed breakdown of generative model performance

Our model outperforms the *ANNIE* system in general, and for questions seeking *organizations*, *thing-names*, and *miscellaneous* targets in particular. *ANNIE* may have low coverage on organization names, resulting in reduced performance. Like *Oracle*, *ANNIE* treats all candidates not belonging to one of the other categories as appropriate for *miscellaneous* questions. Because *ANNIE* cannot identify *thing-names*, they are treated as *miscellaneous*. *ANNIE* shows very low performance on *thing-names* because words incorrectly tagged with types are sorted to the bottom of the list for *miscellaneous* and *thing-name* questions. If the correct answer is incorrectly tagged with a type, then it will be sorted near the bottom, resulting in a poor score.

The results of Tables 3.3 and 3.4 show the advantages of our class-free generative answer typing model over idealized class-based models. Of particular interest is the *miscellaneous* class. These 37 questions are ones that cannot be assigned a more appropriate class in our simple type set. As was mentioned earlier, such questions will always exist no matter how carefully the set of fixed types has been specified. Although the relative size of this class may vary depending on the distribution of new questions and the exact composition of the question set, we can never hope to truly eliminate it with a class-based approach. The generative model is shown here to be an especially good choice for use on these questions and produces reasonable performance for the other types as well.

3.4.2 Further Improvements with Discriminative Preference Ranking

The generative model improves upon class-based approaches, and we wish to further extend this model by introducing a greater degree of flexibility as to how various parameters are used via the discriminative preference ranking model of Section 3.3. The goal of these experiments is to show that the discriminative preference ranking model improves upon the performance of the generative model and is even capable of reaching a level of performance comparable to that of full QA systems on a limited subset of questions (discussed below). The results presented here (and originally reported in [58]) show that the additional flexibility afforded by discriminative preference ranking provides a better model of answer type than does the generative model.

Because our discriminative preference ranking models are built using supervised learning, we require a set of known correct answers and a list of annotated appropriate candidates for each training question. Correct answers to our set of questions are obtained from the TREC 2002-2006 results [69]. For appropriateness labels we turn to human annotators. Two annotators were instructed to label a candidate as appropriate if that candidate was believable as an answer, even if that candidate was not correct. For a question such as *what city hosted the 1988 Winter Olympics?*, all cities should be labeled as appropriate even though only *Calgary* is correct. This task comes with a moderate degree of difficulty, especially when dealing with questions for which appropriate answers are less obvious (such as *what kind of a community is a Kibbutz?*). We observed an inter-annotator (κ) agreement of 0.73, which indicates substantial but not perfect agreement. This value of κ conveys the difficulty that even human annotators have when trying to decide which candidates are appropriate for a given question. Because of this value of κ, we adopt strict gold standard appropriateness labels that are the intersection of the two annotators' labels in which a candidate is only appropriate if both annotators label it as such, otherwise it is inappropriate.

To compare with the generative model, we use a set of *what* and *which* questions with a question focus; questions with a noun phrase following the *wh*-word. These are a subset of the more general *what*, *which*, and *who* questions handled by the generative model. Although the discriminative preference ranking models can accommodate a wide range of *what*, *which*, and *who* questions, the focused *what* and *which* questions are an easily identifiable subclass that are rarely definitional or otherwise complex in terms of the desired answer. Additionally, such questions are often composed of more than one question context, which presents an opportunity to weight multiple contexts differently. We took the

set of focused *what* and *which* questions from TREC 2002-2006 [69] comprising a total of 385 questions and performed 9-fold cross-validation, with one dedicated development partition (the tenth partition). The development partition was used to tune the regularization parameter of the final SVM used at testing time.

Candidates are obtained by submitting the question as-is to the Google search engine[4] and chunking the top 20 snippets returned, resulting in an average of 140 candidates per question. Google snippets create a better confusion set than random words for appropriate and inappropriate candidates; many of the terms found in Google snippets are related in some way to the question. To ensure a correct answer is present (where possible), we append the list of correct answers to the list of candidates.

As a measure of performance, we adopt Mean Reciprocal Rank (MRR) for both correct and appropriate answers, as well as precision vs. recall for appropriate answers. MRR is computed as the average of the reciprocal rank (RR) over the set of test questions. Reciprocal rank is:

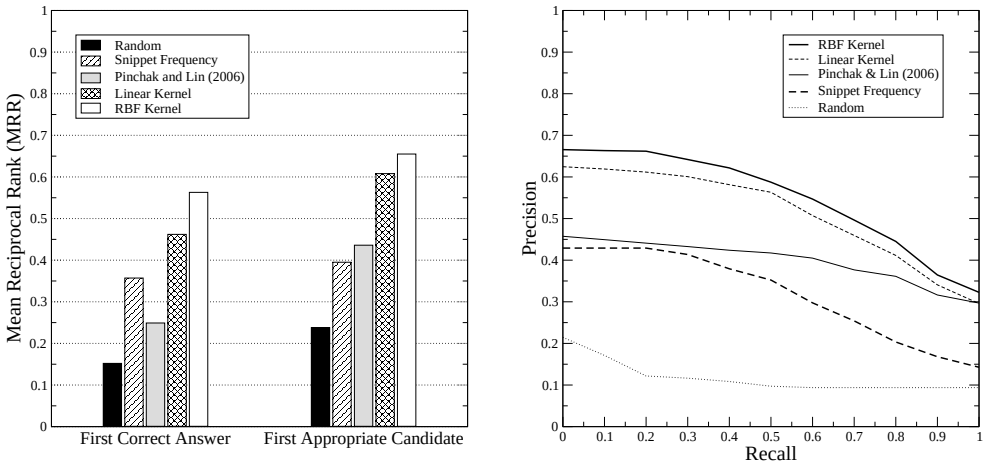
$$RR(Q) = \frac{1}{\text{rank of first correct answer for } Q} \tag{3.22}$$

MRR is often used as a measure of overall QA system performance [69], but is based only on the top correct or appropriate answer encountered in a ranked list. For this reason, we also show the precision vs. recall curve to better understand how our models perform.

We compare our models with three alternative approaches, the simplest of which is *Random*. For *Random*, the candidate answers are randomly shuffled and performance is averaged over 100 runs. The performance of *Random* is influenced by how often a candidate appears in Google snippets and is therefore dependent upon Google's ability to find both correct and appropriate answers. The *snippet frequency* approach orders candidates based on their frequency of occurrence in the Google snippets, and is just the $F(t)$ feature of Table 3.2 in isolation. Given that question words tend to be very frequent in snippets, we remove terms comprised solely of question words from all approaches to prevent question words from being selected as answers. The last of our alternative systems is the generative model of Section 3.2 with candidates ranked according to their probabilities as assigned by the model.

Figure 3.4 shows the MRR results and precision/recall curve of our correctness model against the alternative approaches. In comparison to these alternative systems, we show two versions of our correctness model (Section 3.3.3). The first uses a linear kernel and is able to outperform the alternative approaches. The second uses a radial basis function

[4] `http://www.google.com`



(a) MRR

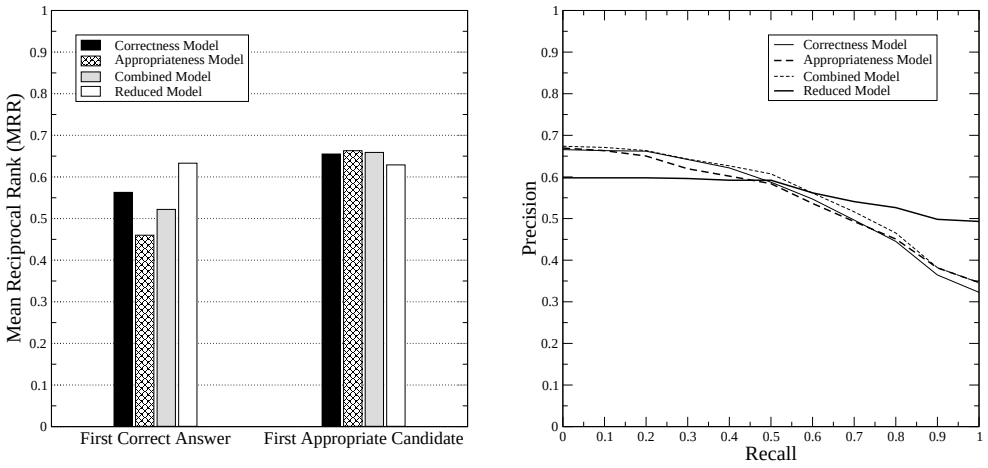
(b) Precision/Recall

Figure 3.4: Results for the correctness model

(RBF) kernel and exhibits performance superior to that of the linear kernel. This suggests a degree of non-linearity present in the data that cannot be captured by the linear kernel alone. We do not tune the RBF kernel parameters because the purpose of using the RBF kernel is to show that higher performance can be achieved by using a non-linear kernel. Custom kernel functions will likely be able to outperform the RBF kernel, although such extended kernels are not explored here. Both the training and running times of the RBF kernel are considerably larger than that of the linear kernel. The accuracy gain of the RBF kernel must therefore be weighed against the increased time required to use the model.

Figure 3.5 gives the MRR results and precision/recall curves for our additional models in comparison to that of the correctness model. Although losses in MRR and precision are observed for both the appropriate and combined model using the RBF kernel, the linear kernel versions of these models show slight performance gains.

The results of our correctness model, found in Figure 3.4, show considerable gains over our alternative systems, including our generative model. These results show that our discriminative preference ranking approach creates a better model of both correctness and appropriateness via weighting of contexts, preference rank learning, and with the incorporation of additional related features as listed in Table 3.2. The snippet frequency feature $F(t)$ is not particularly strong on its own, but can be easily incorporated into our discriminative model. The ability to add a wide variety of potentially helpful features is one of the strengths of discriminative techniques in general.



(a) MRR

(b) Precision/Recall

Figure 3.5: Results for all models (RBF kernel)

By moving away from correct answers in the correctness model and incorporating labeled appropriate examples in various ways, we are able to further improve upon the performance of our approach (Figure 3.5). Training on appropriateness labels instead of correct answers results in a loss in MRR for the first correct answer, but a gain in MRR for the first appropriate candidate. Unfortunately, this does not carry over to the entire range of precision over recall. However, when using a linear kernel instead of a RBF kernel, our three additional models (appropriateness, combined, and reduced) show consistent improvements over the correctness model. In this particular instance only the reduced model provides a meaningful change in performance for the better-performing RBF kernel.

The precision-recall curves of Figures 3.4(b) and 3.5(b) show remarkable consistency across the full range of recall, despite the fact that candidates exist for which feature values cannot easily be obtained. Due to errors in obtaining candidates from Google results, malformed candidates may exist that are judged appropriate by the annotators. For example, *explorer Hernando Soto* is a candidate marked appropriate by both annotators to the question *what Spanish explorer discovered the Mississippi River?* However, our corpus and hence resources do not include the phrase *explorer Hernando Soto* meaning that many of the important features will have their values set to zero. The net effect is that these malformed appropriate candidates are likely to receive a lower rank despite the fact that they are actually appropriate. Despite these occasional problems, our models are able to rank most correct and appropriate candidates high in a ranked list.

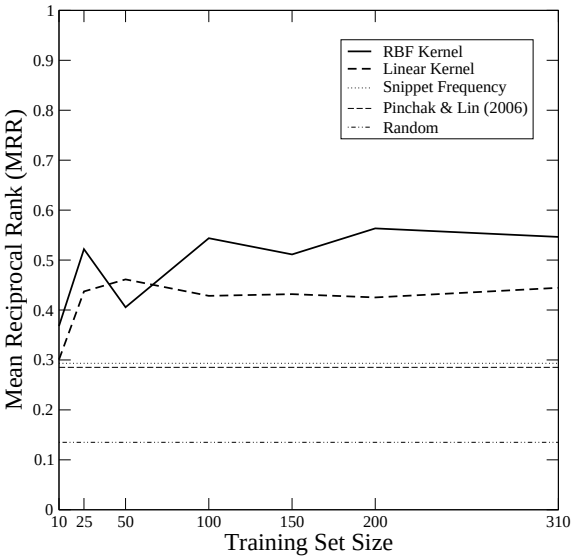


Figure 3.6: Learning curve (MRR, correct answer, correctness model)

Finally, we examine the effects of training set size on MRR. The learning curve for a single partitioning under the correctness model is presented in Figure 3.6. Although the model trained with the RBF kernel exhibits some degree of instability below 100 training questions, both the linear and RBF models quickly achieve high performance. This learning curve shows that both the linear and RBF kernels outperform the alternative systems with as few as ten training questions. Requiring only a relatively small number of training examples means that an effective model can be learned with relatively little input in the form of question-answer pairs or annotated candidate lists.

3.5 Conclusions

Open-ended noun phrase questions are some of the most difficult questions on which to apply answer typing. Given that they can be answered by nearly any noun phrase answer, or typically anything that can be named, we cannot always predict in advance the range of possible answers. However, because there is often a great deal of repetition in what people ask questions about, this class of questions is often addressed with the use of a class-based answer typing strategy.

This chapter has introduced two alternative class-free answer typing strategies meant to augment and/or replace class-based strategies typically used for these questions. The first of our models, the generative model of Section 3.2, shows considerable gains for some answer

classes over idealized systems based on a fixed set of answer types. This generative model is carefully built using resources extracted from a small corpus and requires no supervised training data, making it ideal for domains in which limited data is available. Unfortunately, due to its generative nature, this model is difficult to update with new statistics that may reveal new information about answer type. For this, we require more powerful discriminative techniques.

Moving from a generative model to a discriminative model provides benefits in terms of both increased flexibility by adding new features and increased performance because of this increased flexibility. Whereas the generative model is limited to parameters that can be accounted for in the probability model, a discriminative preference ranking model (Section 3.3) can make use of nearly any real-valued feature. Aside from incorporating new features that are difficult to add to the generative model, the discriminative preference ranking model is able to weight individual question contexts. This was lacking in the generative model because of the assumptions made here. Preference ranking also allows us to better model the task at hand by reranking candidate answers according to appropriateness. Experimental results show an increase in performance and it is possible that new and as-yet unknown features could increase this performance even further. We will revisit this model when we introduce the application of answer typing to information retrieval in Chapter 5.

Chapter 4

Fine-Grained Typing of How-Adjective Questions

Some questions, such as the how-adjective questions examined in this chapter, initially seem trivial to type. How-adjective questions, i.e., questions in which the *wh*-word *how* is followed by an adjective or adverb, are always answered with a numerical value. Because numerical values are very easy to identify, locating type-appropriate answers in text is very simple. However, a closer examination of these types, from a more fine-grained perspective, reveals opportunities to improve performance via answer typing.

Although it is true that all how-adjective questions are answered by numerical values, not all numerical values are equally appropriate. For example, a question such as *how tall is the CN tower?* is answerable only in numerical quantities of units that can measure height. A response of *33 years* is inappropriate and unacceptable as an answer even though it is obviously a numerical value. Making the distinction between a correct answer of *1815 feet* and an inappropriate response of *33 years* requires consideration of the question and candidates at a level of detail much finer than is typically used for these questions. We specifically exclude the *how many* and *how much* questions from consideration because *how many* questions almost always specify the appropriate units following the word *many*, as in *how many people live in Canada? How much* questions, on the other hand, are very often answered in terms of monetary values.

The key to fine-grained typing of how-adjective questions lies in the adjective specified in the question. The adjective (or adverb) is conceptually related to the units that are type-appropriate for a response. For example, the unit *feet* is related to the adjective *tall* via the concept of *height* or *tallness*. We must automatically establish a connection between *feet*, *height*, and *tall* so that we can build lists of appropriate units for adjectives that may be used in a how-adjective question. This connection must be made between all adjectives

used in how-adjective questions and their corresponding candidate units so that we may decide which potential answers are appropriate and which are not.

Given the limited number of popular adjectives, employing hand-crafted rules to decide which units are appropriate is an enticing typing strategy. Wu et al. [71] do exactly this by assigning a specific target based on rules applied to the how-adjective questions. Unfortunately, these rules are insufficient in two dimensions: coverage of possible adjectives and coverage of appropriate units. Rules will often cover the common adjectives such as *high*, *tall*, *heavy*, and *far* (among others), but may miss obscure adjectives such as *viscous*. Furthermore, even common adjectives such as *cool* many have obscure units such as *MegaFonzies*, a fictional unit of *coolness* introduced in the television show *Futurama*. It is highly unlikely that any rule-based system would ever assign *MegaFonzies* as a unit for *cool*, even though it is appropriate in limited cases.

In this chapter, we introduce two variants of an answer typing model specifically for how-adjective questions, originally presented in [56]. Our models automatically discover appropriate units for a given how-adjective question by discovering and leveraging the link from adjective to unit via the use of a structured ontology. The unit discovery can be performed before any new questions are seen by using adjectives found in a query log so as to have unit lists available for a large proportion of how-adjective questions. Experimental results show that fine-grained answer typing of how-adjective questions is superior to that of coarse-grained typing using only a single numerical class. Our results also show that our methods of fine-grained answer typing have advantages over baseline systems, some of which make use of information not available to a real system . Once again, discriminative preference ranking is employed to further boost performance.

This chapter is organized as follows. A description of the resources available to our fine-grained answer typing system is given in Section 4.1. Once the available resources have been established, we describe two variants of our typing model in Section 4.2; one model is probabilistic and the other is a more complex discriminative model employing a greater number of features. Experiments detailing the advantages of our models over both baseline systems and the use of a single numeric type are presented in Section 4.3. Conclusions are drawn in Section 4.4 before we move on to the final aspect of answer typing: the application of typing to the domain of information retrieval.

4.1 Resources

As discussed previously, we wish to create a model of answer type by linking the adjective of a how-adjective question with appropriate units. Because units are not provided as part of the question, we must find a means of automatically discovering a set of potential units. For our purpose, a structured ontology will assist in connecting the adjective with words more useful for finding units, and online search engines (or some other interface to a very large corpus) will provide a pool of potential units that may be appropriate to a given adjective. We discuss the resources available for both of these tasks in the remainder of this section.

4.1.1 WordNet

WordNet [19] is a well-known hierarchically-structured ontology based primarily on the concepts of *synsets* and the *hypernym/hyponym* relationship. Synsets exist for individual concepts and are comprised of synonymous words that represent that concept. Words with multiple senses will be found in multiple synsets, one for each sense. Synsets are organized hierarchically such that they are linked via the hypernym or hyponym relationship (depending on the direction of the link). A synset *X* is a *hyponym* of synset *Y* if the concept represented by *X* entails the concept represented by *Y*. Inversely, *Y* is a *hypernym* of *X*. For example, the word *cat* is a hyponym of *mammal*, which in turn is a hyponym of *animal*. Transitivity applies to these relations meaning that *cat* is indirectly a hyponym of *animal*. These notions of synonymity (via synsets) and specialization (via hyponyms/hypernyms) are the most popular aspects of WordNet.

In addition to these features of WordNet, additional useful information is also recorded in the hierarchy. Gloss information exists that allows WordNet to function similar to a dictionary. In particular, the system of Moldovan et al. [46] transforms this gloss information into predicates that may be used with their logic prover. Of more interest here are the links representing derivationally related forms that link morphological variants of a word. For example, an adjective such as *tall* is derivationally related to the concept of *height* which also includes the synonym *tallness*. The WordNet *attribute* relation serves a similar purpose and so words linked via the attribute relation (when available) are considered in combination with derivationally related forms. These WordNet relations provide a means by which our answer typing model can connect an adjective such as *tall* to a related noun such as *height*. Figure 4.1 displays this information visually. This alternative noun form will allow us to automatically discover units that cannot be discovered with the adjective alone due to

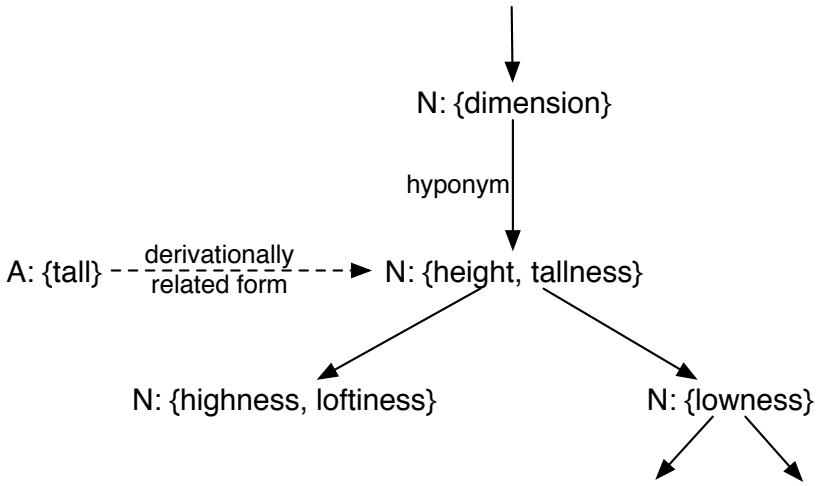


Figure 4.1: A portion of the WordNet hierarchy

grammatical restrictions of the English language.

4.1.2 Google Search Results

The World Wide Web represents the largest existing corpus in the world, but accessing this large quantity of data is not a trivial matter. Because the Web is comprised of many petabytes of data, we cannot iterate over it without a large amount of computing resources. The Google search engine addresses this problem by maintaining an index of a significant portion of the Web. Google accepts user queries in the form of both keywords and phrases and leverages their index of the Web to find relevant pages. Therefore, we can access relevant portions of the Web indirectly via queries to the Google search engine.

One of the goals of fine-grained answer typing for how-adjective questions is to automatically discover units that are appropriate for a given adjective. In order for our model to make a judgement according to type-appropriateness, we must provide it with some candidates that are reasonably likely to be units; evaluating the entire vocabulary of English words is unreasonable in this case. Therefore, we use Web search (via Google) as a means of discovering a small subset of high-quality candidates taken from all terms on the Web. To this end, we search for a keyword or phrase and collect nearby terms (or possibly those terms that co-occur in the snippet). In general, terms co-occurring within a short context window of query keywords or phrases are related in some way to the query terms.

Google also provides estimates of the frequency of a given query in documents on the Web, a value known as *hits*. Even though these estimates are well-known to be inaccurate

and unstable [32], their relative magnitudes are representative of their true frequencies. For example, estimates for a word such as *cat* may be around 900 million one day and 700 million another. The fact that they are closer to 1 billion than 100 million does not change; the order of magnitude has not changed much even though the counts differ by 200 million. Often, the best way to deal with this estimation error is to use log values, as the order of magnitude is often what we wish to measure when thinking in terms of Web counts. Both the frequency estimates and the ability to search a large corpus make Google an ideal choice as an interface to the Web as a source of candidate units.

4.2 Models

Like for the open-ended noun phrase questions of Chapter 3, a model of answer type for how-adjective questions must assign scores to candidates to produce a sorted list. Unlike open-ended noun phrase questions, we are dealing with candidate *units* rather than complete candidate answers. Nearly any noun representing a concrete entity can be used as a unit. This means that the space of candidates is conceptually the same as for open-ended noun phrase questions, but in practice the nouns encountered by the how-adjective model are quite different from the open-ended noun phrase model due to the differing sources of candidates.

We introduce in this section two models for answer typing of how-adjective questions mirroring those introduced for open-ended noun phrase questions. The first model is based on probabilities derived from simple statistics of the resources of Section 4.1. This probabilistic model uses WordNet to connect the adjective in the question with nouns that are appropriate for use with Google search. These nouns, placed into a specific phrasal pattern, are then used to discover a set of potential units that must be evaluated according to a model based on frequency values for the terms.

Our second model builds upon the probabilistic model by introducing discriminative preference ranking (as was done in Section 3.3). Discriminative preference ranking allows for the introduction of additional features that contribute to improving performance of unit ranking. This concept of discriminative preference ranking applies to more than just open-ended noun phrase questions; any typing that relies on building a sorted list of relevant responses is well-suited to discriminative preference ranking.

4.2.1 Probabilistic Model

Our probabilistic answer typing model for how-adjective questions seeks to create a mapping $T : A \rightarrow U_A$ in which A is a set of adjectives that may be used in a how-adjective question and U_A is a set of lists of units appropriate to each adjective. We build this mapping *a priori* from a set of already encountered or anticipated adjectives so that a list of units is ready ahead of time and so that we may use statistics based on all lists (which would change if we add new adjectives).

As described in the resources of Section 4.1, WordNet provides a means to link the adjective found in the question with related nouns via the derivationally related and attribute relations. Once we obtain these related nouns, we construct a pattern for each consisting of the phrase "*<adj> is measured in*." This phrase is then used as a phrasal query for Google search for which documents containing the phrase are returned as results. A relevant document is guaranteed by the semantics of Google search to contain at least one occurrence of this phrase, and so we may simply iterate through as many results as we desire to collect occurrences.

Once occurrences have been located in documents, we use the Minipar dependency parser [39] to parse the sentence containing the query phrase. From the parse tree, we can determine which term is prepositionally attached to *measured* via *in*. Because the most straightforward way to state that a particular concept (such as *height*) can be measured with a particular unit (such as *feet*) is via the statement *height is measured in feet*, we search for the first portion of this phrase with the expectation that it will lead to a candidate set rich with appropriate units. In practice this is generally true, although we cannot discover a large quantity of appropriate units without first analyzing a large number of documents (an issue we will further address below).

During the analysis of occurrences of the phrasal pattern in documents, we record the number of times we encounter each candidate unit. If r is a noun related to adjective a via WordNet, then each single occurrence of the unit u in the pattern containing r contributes a single count to the statistic $N(r, u)$. That is, $N(r, u)$ is the sum of all occurrences of the pattern containing r with u as the discovered unit (r *is measured in* u). Because there may be more than one noun r related to adjective a via WordNet (the set R_a), the count for $N(a, u)$ is:

$$N(a, u) = \sum_{r \in R_a} N(r, u) \tag{4.1}$$

These values are used when filtering the list of candidate units to remove spurious non-

applicable or inappropriate nouns that are present due to unanticipated pattern matches and parser errors.

Before moving on to discuss filtering of the unit lists, it must be noted that the lists created from pattern matches alone are relatively sparse. We cannot request a large number of documents from Google due to restrictions on Google resources along with the time required to fetch and process documents, especially from slow to respond servers. Because of this, we expand the list of candidate units with a list of similar words derived from clusters used for open-ended noun phrase questions in Section 3.1. Of course, given that these units are not observed to occur, we must estimate the value for $N(a, v)$ for some expanded candidate unit v. If V_u are the expanded candidates containing the neighbours of candidate unit u and U_a is the set of observed units, we calculate $N(a, v)$ for some unseen v as:

$$N(a, v) = \sum_{u \in U_a} sim(u, v) N(a, u) \tag{4.2}$$

Essentially, this equation creates counts for unseen units v by observing neighbours of v that actually occurred and taking weighted fractional counts of them. The counts are weighted by the similarity between two words, a number between 0 and 1 derived from our cluster resources. At the end of this process, we have a new set of units W_a:

$$W_a = U_a \cup \bigcup_{u \in U_a} V_u \tag{4.3}$$

For a given adjective a and a particular unit u with instance counts $N(a, u)$, we define two important statistics:

$$P(u|a) = \frac{N(a, u)}{\sum_{u' \in U} N(a, u')} \tag{4.4}$$

$$P(a|u) = \frac{N(a, u)}{\sum_{a' \in A} N(a', u)} \tag{4.5}$$

in which U is the set of all candidate units for *all* adjectives (i.e., the union over all W_a) and A is the set of all adjectives for which we build unit lists. Equation 4.4 measures the likelihood of a unit u being an appropriate unit for a how-adjective question with the given adjective a. Equation 4.5 measures, for some unit u, how likely a how-adjective question with adjective a is answered in terms of u. The second measure is particularly useful in cases where a unit u co-occurs with a number of different adjectives. These units are inherently less useful for identifying correct responses. For example, if the word *terms* occurs on the unit list of adjectives such as *high*, *long*, and *heavy*, it may indicate that *terms*

is not an appropriate unit for any of these concepts, but rather a word likely to co-occur with adjectives in general.

The values for Pr$(u|a)$ and Pr$(u|a)$Pr$(a|u)$ are useful as ranking functions for unit appropriateness. Pr$(a|u)$ alone showed poor performance on a development set [56] and so is not further considered as a means of ranking and filtering. Pr$(u|a)$Pr$(a|u)$ roughly approximates the standard *tf.idf* measure [61] in which Pr$(u|a)$ is the term frequency *tf* in the unit list and $P(a|u)$ is the inverse document frequency *idf* of the unit over all unit lists. Using these measures, we can create a final unit list for an adjective a as

$$W_a = \{u : Score(u, a) \geq \mathcal{T}\} \tag{4.6}$$

in which $Score(u, a)$ is one of Pr$(u|a)$ or Pr$(u|a)$Pr$(a|u)$ and $\mathcal{T}$ is some threshold imposed to deal with the amount of noise present in the unit lists. This threshold allows us to vary the strength of the association between the unit and the adjective required to consider the unit appropriate. In the experiments of Section 4.3 we vary this threshold to obtain a tradeoff between precision and recall of the resulting unit list. The value $Score(u, a)$ can also be passed to downstream modules of the question answering process (such as the answer extractor) which may then exploit the association value directly, although this possibility is not explored in this thesis.

This simple probabilistic model, based on values easily collected from Google search results, is a powerful fine-grained model of answer type for how-adjective questions. In combination with this model, we may initially assume that all appropriate answers to how-adjective questions are numeric responses. This means that our fine-grained model of answer type need only consider numeric entities that are expressed in terms of appropriate units, a much more restrictive requirement than numeric entities alone. The next section expands upon this simple probabilistic model with discriminative methods to produce a more accurate and flexible model of type for how-adjective questions.

4.2.2 Discriminative Model

Discriminative preference ranking is applicable to more than the domain of open-ended noun phrase questions. We apply the same techniques introduced in Section 3.3 to the how-adjective questions. Discriminative preference ranking has been established as a means of increasing both flexibility (in terms of features) and performance, and so we expect a model based on discriminative preference ranking to further improve upon the performance of our probabilistic model of Section 4.2.1. Little additional description is necessary regarding

Pattern	Description
$N(a, u)$	Actual count of candidate unit u in pattern for a
$\sum_{u'} N(a, u')$	Count of all units for a
$\sum_{a'} N(a', u)$	Total for u over all adjectives
$len(u)$	String length
$hits(a, u)$	Hits where u and a co-occur
$phra(a, u)$	Hits for "a is measured in u"
$quant(u, q)$	Hits where u follows some quantifier q
$punct(u)$	u contains punctuation

Table 4.1: How-adjective question feature templates

the application of discriminative preference ranking to typing how-adjective questions. In general, discriminative preference ranking can be broadly applied to answer typing by simply identifying a set of features that may indicate appropriateness of a candidate answer (or unit) in conjunction with portions of the question, and ranking these candidates according to a function over these features. Therefore, the remainder of this section describes the changes necessary for how-adjective questions.

Because discriminative preference ranking can take advantage of a diverse set of features, we include additional indicators of appropriateness for units that are different from the probabilities used by the probabilistic model. The feature templates we use for the discriminative model are summarized in Table 4.1. The features include $N(a, u)$, $\sum_{u'} N(a, u')$, and $\sum_{a'} N(a', u)$ that are the same features used to obtain the parameters of the probabilistic model of Section 4.2.1. To these basic counts we add values for the string length of a candidate unit u with the expectation that long lengths are indicative of poor candidate units. Considering the number of times u and adjective a co-occur on the same page, $hits(a, u)$, is meant to somewhat describe whether or not u is a unit related in some way to a; if u and a never occur on the same page, it is unlikely they are related to one another. The exact phrase count for "a is measured in u," $phra(a, u)$ is not intended to produce a large number of hits but may have a non-zero value for certain adjectives (not related nouns). We expect that if u is truly a unit of measure then it can follow a quantifier $q \in \{$0, 1, 2, 000, 10, 100, 1000, one, two, three, ten, thousand, million$\}$ and so we count the number of times the pattern "q u" occurs on the web. Finally, if u contains punctuation then it may be more likely as an abbreviated units such as *k.p.h.* for *kilometres per hour*.

Unit discovery for the discriminative preference ranking model is identical to that for the probabilistic model of Section 4.2.1. In particular, the list of units are expanded with similar terms to provide a more complete set of candidate features after the templates have been

applied. Due to the increased number of features, the discriminative preference ranking model has a better chance of correctly identifying which of these expanded candidates are truly appropriate units and which are noise introduced by this expansion. In the experiments of the next section (Section 4.3), we will see how expanded units can reduce performance if they are not carefully weighted by the discriminative model.

4.3 Experiments

Measuring the performance of answer typing for how-adjective questions is as difficult as for that of open-ended noun phrase questions. The answer typing model is designed to identify appropriate candidates whereas most QA data available provides only correct answers. Although correct answers are by definition appropriate, there are a large number of appropriate candidates that are not correct. For the experiments presented here, we use a list of known correct units to evaluate the models of Section 4.2.1 and a manually labeled set of appropriate units to evaluate the models of Section 4.2.2.

Once again, we use our models to produce ordered lists of candidate units such that highly-ranked units are more likely to be appropriate to a given question. To produce a list of candidate adjectives for which units must be discovered, we examine how-adjective questions from TREC 2002-2005 [69] resulting in a total of 15 unique adjectives (*accurate, big, close, cold, deep, far, fast, heavy, high, hot, large, late, long, old, tall*). Although this seems like a small set of adjectives to which this approach applies, we can find 152 unique adjectives following the phrase *how <adjective> is/are/was/were* in the set of AOL queries [52]. We restrict ourselves to the 15 adjectives found in TREC 2002-2005 because they have a list of known correct answers (which contain units). A total of 94 TREC questions share these 15 adjectives, 86 of which are used for our experiments and 8 of which are used as a development set.

For a given how-adjective question and a document of interest selected from the set of AQUAINT documents known to contain a correct answer, we use a two-stage process to identify the entities in the document that are suitable answers for the question. First, the named entity recognition of Minipar is used to identify all numerical entities in text. Minipar labels times, dates, monetary amounts, and street addresses with special types and so we exclude these from consideration. We then inspect the context of all numeric entities to determine if a unit exists on the pre-computed unit list for the given adjective. Textual entities that pass both stages of our identification process are deemed appropriate by our

model.

Given that unit discovery is identical for both the probabilistic and discriminative models, we use the same set of discovered candidate units when evaluating both kinds of models. Furthermore, the option to expand the candidate units with similar words results in two parallel sets of candidates. As we will see in the subsequent sections, the benefits of expansion come at a price and we cannot assume that expanding the list of discovered candidate units will always improve performance. We now measure how performance can be improved by using a probabilistic model of answer type and then the further improvements possible with the subsequent discriminative model.

4.3.1 High Precision with the Probabilistic Model

The goal of experiments with the probabilistic models is to show that a probabilistic model of answer type is able to identify units of correct answers more reliably than comparison systems relying on other indicators of type-appropriateness. Although answer typing seeks to find the most appropriate candidate for a given adjective, the performance of a QA system will be evaluated on correct answers and therefore units associated with correct answers. By being either too general or ineffective at identifying correct units, the system will increase the number of possible answers that are incorrect. This is the exact opposite of the desired effect of fine-grained answer typing. By varying the threshold $\mathcal{T}$ in Equation 4.6, we can adjust the number of units deemed appropriate and hence measure the effects on identifying correct answers (and only correct answers).

We evaluate our models with an approach originally termed Answer-Identification Precision/Recall (AIPR) [56]. By adjusting the scoring threshold $\mathcal{T}$ we obtain a unit list for each adjective that can be used to identify potential answers in documents. Answer-identification precision measures the number of correct answers among the candidates identified as appropriate by our unit list. In contrast, answer-identification recall measures the number of correct answers extracted among the total number of correct answers in the answer documents (from the list of correct answers provided by TREC).

A plot of AIPR allows for the identification of the best precision/recall tradeoff possible for the desired use of answer typing in QA. If other stages of QA require a large number of candidates for which some noise is acceptable, a high recall value may be desired so no potential answers are overlooked. On the other hand, if answer typing is used as a means of boosting already-likely answers, high precision may instead be favoured at the cost of missing some good units. As is often the case with precision and recall, accepting a decrease

in one of these measures allows for an increase in the other. Because there is no incentive to use a threshold $\mathcal{T}$ that decreases both precision *and* recall, we report interpolated precision rather than raw precision [42].

Our probabilistic model is not the only means by which we can filter potential units for correct answers. By using variable amounts of perfect knowledge, we can observe how closely our probabilistic models perform to idealized situations that are, in general, unrealistic. In a similar vein, we compare our models with simplistic strategies that do not require the development of a complex model of answer type. These simple strategies rely on properties of the question or the source of candidate answers rather than the resources of Section 4.1. Specifically, our comparison systems are:

Query-Specific Oracle: This ideal system creates a unit list for each how-adjective question separately, choosing only those units known to be correct from the TREC judgements for a specific question.

Adjective-Specific Oracle: This system is designed, like our models, to provide a unit list for each adjective rather than for a specific question. The unit list for a particular adjective contains all the units from all the answers of questions containing that adjective. It is optimal in the sense it will identify every correct answer for each adjective and only use units necessary for this identification.

Fixed-Category: This system gives the performance of a general-purpose, fixed-category answer typing approach applied to how-adjective questions. In a fixed-category strategy, all how-adjective questions are seeking numerical answers, and thus all numerical entities are identified as possible answers.

IR-Document Inferred: Here we infer question units from documents believed to be relevant to the question. The PRISE IR system is given a how-adjective question and returns a set of documents. Every numerical entity in the documents can be considered a possible answer to the question, and the units associated with those values can be collected as the unit list, ranked (and thresholded) by frequency. We remove units that occur in a list of 527 stopwords, and filter numerical modifiers like *hundred*, *thousand*, *million*, etc.

Answer-Document Inferred: This approach is identical to the IR-Document Inferred approach, except only using those documents judged by TREC to contain the answer.

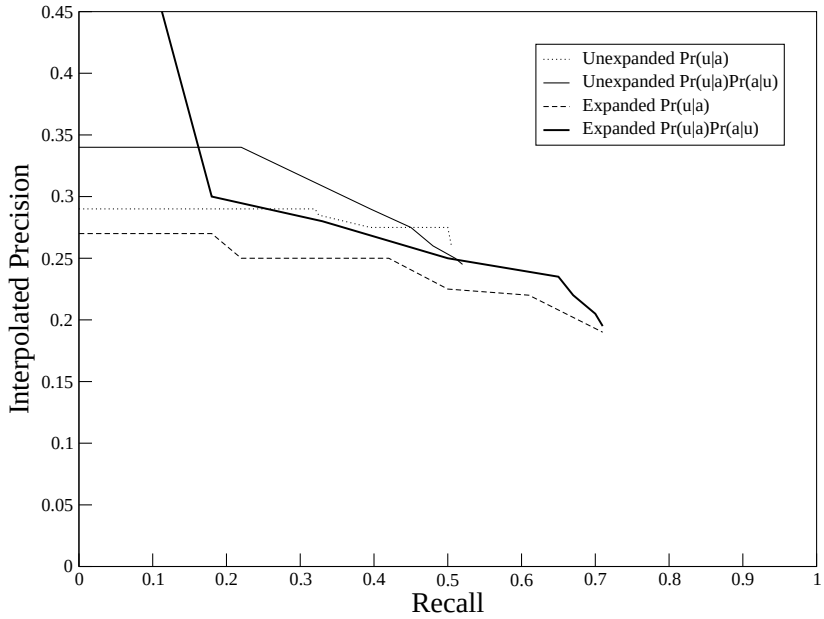


Figure 4.2: Answer-Identification Precision/Recall

In this way the Answer-Document Inferred approach provides an upper bound on inferring units by assuming perfect document retrieval.

Inferring answer units from a set of relevant documents is similar in spirit to the work of Daume and Marcu [18]. In one of their experiments in query-focused summarization, they show competitive summarization performance *without even providing the query*, as the query model is inferred solely from the commonality in relevant documents. In our case, high performance will be possible if the actual answers have the highest frequency among the numerical values in the relevant documents. This expectation is reasonable given the fact that the correct answer is often highly related to the question terms, and this is reinforced when the answer is found repeatedly in many different sources.

Answer-Identification Precision/Recall for various scoring functions over all question-answer pairs is shown in Figure 4.2. The first item of interest in Figure 4.2 is the benefit of ranking with Pr$(u|a)$Pr$(a|u)$ as opposed to only Pr$(u|a)$. Over most of the range of recall, using the combined value Pr$(u|a)$Pr$(a|u)$ results in better precision. The second item of note is that using expanded units can achieve almost 20% higher maximum recall than the corresponding unexpanded units, even at the cost of precision at lower recall values. This provides strong justification for the small overhead of looking up similar words for items on our unit list when a greater degree of recall is required.

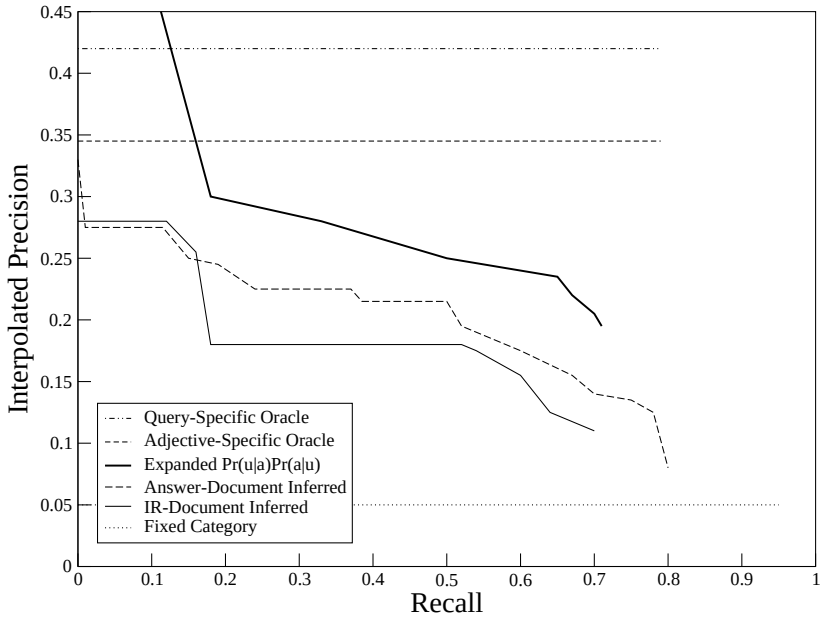


Figure 4.3: AIPR of our approach versus comparison systems

The AIPR performance of our comparison models versus our best-performing probabilistic model is shown in Figure 4.3. The query-specific oracle is able to achieve the highest performance because of perfect knowledge of the units appropriate to a given question. Still, its precision is only 42.2%. Its accuracy is limited because the correct answer shares its units with other numerical entities in the answer documents. Slightly worse is the adjective-specific oracle at 34.2%. Unlike the query-specific oracle, if the question is *how long did WWII last?*, entities with the irrelevant units *metres* and *kilometres* must also be proposed as candidate answers because they occur in answers to other *how long* questions. This adjective-specific oracle thus provides a more realistic upper bound on our model as our model also builds unit lists for individual adjectives rather than specific questions.

In terms of recall, both oracles can find units for 78% of questions (with our expanded systems reaching close to this at about 72%). However, this number is somewhat misleading in that the oracles are expected to find answers for 100% of questions. On inspecting the actual misses, we find that 10 of the 16 missed answers correspond to *how old* questions that are often answered without units (e.g. *at age 52*). Because of this, higher recall would be possible if the system defaults to extracting all numerical entities for *how old* questions. This issue contributes to error in all comparison systems (except for the fixed category baseline) and so is a constant source of error in all results.

Also of note in the results of Figure 4.3 is the clear disadvantage of using a fixed category approach (i.e., numerical entities) for how-question answer typing. The precision for fixed category typing is just under 5%, roughly one half of the lowest precision of any of our approaches at any recall value. However, fixed-category typing does achieve high recall, roughly 96%, missing only numerical entities unrecognized by Minipar. This high recall is possible because fixed-category typing does not miss answers for the *how old* questions.

Both answer-document and IR-document inferred models exhibit lower precision than our model over the entire range of recall. Thus inferring units from relevant documents does not seem promising, as even the unrealistic approach of inferring only from known answer documents cannot achieve precision comparable to that of our model. This is similar for the more realistic IR-document inferred model. Higher recall is possible with the answer-document inferred model due to the unrealistic use of documents known to contain the answer rather than documents returned by the PRISE search engine. Despite their lower performance, note that these inferred approaches are completely orthogonal to our probabilistic models, so it may be possible to combine variants of these models to achieve further increases in performance.

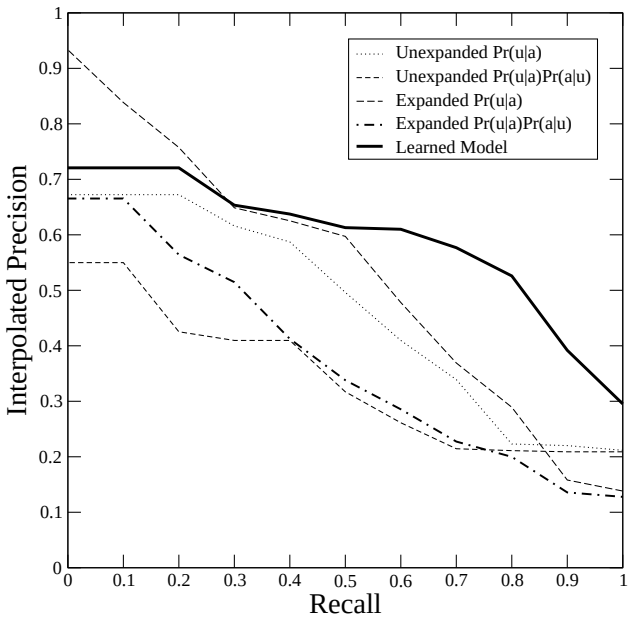
4.3.2 Further Improving Performance with Discriminative Techniques

Once again, the goal of building a discriminative preference ranking model of answer type is to increase the flexibility and performance of typing for how-adjective questions. The evaluation of our model, however, is changed to better reflect this true goal. Instead of simply determining which units appearing on a list are used with a correct answer, we wish to know which units could plausibly be used with some correct answer. For this evaluation, we label a list of (expanded and unexpanded) candidates for each of the adjectives used in TREC 2002-2005. Each candidate unit is assigned one of two possible labels:

Appropriate: The unit is judged to be appropriate. This includes both those units commonly associated with a particular class of question (such as *feet* for *how high* questions) as well as those units that are acceptable but uncommon (such as *microns* for *how tall* questions).

Inappropriate: The unit is inappropriate for this class of how-adjective questions or the unit is not a unit at all. These are the majority of discovered units due to the noisy results from the Google pattern match.

Figure 4.4: Precision/recall for preference ranking how-adjective questions



Two annotators were used for annotation resulting in a kappa (κ) agreement [14] of 0.71. This value is due mainly to disagreement on the rare units, especially for questions such as *how large* and *how big* that have a potentially large number of plausible units depending on the annotator's interpretation of appropriateness. However, a value of $\kappa = 0.71$ still indicates a substantial amount of agreement between the two annotators.

To build unit lists with discriminative preference ranking we train 15 individual models, each predicting a single class of how-adjective question using 13 of the other classes as training data. For example, the model for *how tall* questions is trained using the data for 13 other question classes and *how tall* information is omitted. For development, we omit a further question class so that a particular set of classes is used to train a model for one class at development time but another at test time. The SVM regularization parameter was set using this held-out 14th question for each model. Prediction accuracy for each of the 15 models is averaged (macro-averaging) to obtain a precision and recall score over the entire set.

Figure 4.4 shows the precision/recall curve for our set of 15 how-adjective question types obtained by varying the number of units taken from the top of the ranked list. Included are the new results for the probabilistic models of Section 4.2.1, showing their performance on a closed set of labeled candidates. As Figure 4.4 shows, the learned model performs

better than all of the probabilistic models, except for the unexpanded $\Pr(u|a)\Pr(a|u)$ model at low recall (< 0.3). In particular, discriminative preference ranking shows the most benefit at higher levels of recall, an area where precision formerly dropped off dramatically. This means that appropriate units are often ranked near the top of the list and can be obtained without permitting too many false positives (i.e., inappropriate "units").

The results of Figure 4.4 show more consistent precision over recall than those exhibited by the probabilistic models. This "flatter" precision/recall curve means that some low-recall precision is sacrificed to obtain greater precision with a larger number of appropriate units. Often the units sought by a particular class of how-adjective question (such as *how tall* questions) can be very diverse, especially when considering those units not usually associated with a such a class of questions. As an example, consider a question such as *how tall is the Empire State Building?* In this case, we would accept an answer in terms of the expected units *feet* or *meters*, but an answer of *the Empire State Building is 102 storeys tall* is also acceptable. Our learned model places the unit *storey* at position 19, but the best position for the unlearned models is no higher than position 23. Repeating this situation many times for the various classes of how-adjective questions results in a loss of overall precision at the low end of recall (at least compared with unexpanded $\Pr(u|a)\Pr(a|u)$) but a gain in precision over the remainder of the range.

From these experiments, we can once again conclude that discriminative preference ranking shows benefits for answer typing performance. Although we seek to find and rank *units* rather than candidate answers, the how-adjective and open-ended noun phrase questions are similar enough to benefit from similar strategies. Building models unique to each adjective allows for the rapid application of the model to incoming questions as well as compartmentalized improvements in the future; handling a new adjective requires the construction of a new individual model rather than the rebuilding of a shared model. This makes the application of discriminative preference ranking to how-adjective questions somewhat different from open-ended noun phrase questions, although in a way that further enhances flexibility for how-adjective questions.

4.4 Conclusions

Applying fine-grained answer to how adjective questions has shown definite benefits in performance. This is of special interest because how-adjective questions are often thought of as easily typed. However, the typical strategy of considering all numerical entities as

potential answers for how-adjective questions results in a large number of incorrect answers identified as appropriate, leading to poor performance. Although some class-based answer typing approaches do include specific classes for the type of quantity measured [71], we cannot easily filter out inappropriate responses. The models introduced in this chapter are shown to capitalize upon this opportunity for improvement.

Typing how-adjective questions relies on the concept of a *unit* that must be discovered and ranked according to how appropriate it is in response to the question. The models presented here automatically create lists of units for particular adjectives such that these lists can be applied immediately to new questions. Depending on the amount of time devoted to answer typing, this could be extended to typing units based on other elements of the question. For example, the units appropriate for a specific *how tall* question vary depending on what entity is included. The CN tower cannot be measured on the same scale as a microchip. Although this issue is not explored here, even more fine-grained typing is possible for these questions.

Experimental results show a very meaningful improvement in performance when using both the probabilistic and preference ranking model. Most comparison systems are idealized in one way or another and would therefore be difficult or impossible to implement in a production system. Depending on the application, we may desire a higher or lower degree of recall from the answer typing model. Should higher recall be required, the discriminative preference ranking model shows obvious advantages by permitting only an equal number of incorrect units (precision = 0.5) at a recall level of 80%. If high precision for only one or a few units is more important, the simple probabilistic models provide excellent performance for very little overhead in terms of required features.

Chapter 5

Beyond QA: Answer Typing for IR

The task of information retrieval (IR) is commonly thought of as identifying documents *relevant* to a keyword-based query. The notion of relevance has long been a core concept of IR [43] with the ultimate goal of increasing the quality of documents returned to the user. However, relevance can be a fluid concept; not all documents that contain information related to a query are equally desirable to an end user. As a result, popular Web search engines have shifted from common notions of relevance in IR such as query overlap to increasing the number of returned pages that a user is likely to click.

Typically, an IR engine creates an inverted index over a document collection that allows documents to be found with keywords. If position information is also stored in the index, phrasal queries can be handled and keyword distance can be factored into a measure of relevance. Phrasal or conjunctive queries are ideal for use with an inverted index representation, and documents containing all possibly adjacent terms are a requirement of these queries. Although an inverted index allows for the selection of documents based on which contain query terms, the index itself does not decide the order of presentation of those documents to the user. Additional processing is required to decide upon such an ordering.

Relevance ranking is an important part of all IR systems. The popularity of the Google search engine depends largely on the superior ranking it returns, especially given the fact that many online search engines index roughly the same number of pages. Features such as term proximity and *tf.idf* scores [61] generally indicate pages with more relevant content, whereas features such as *PageRank* [6] indicate the popularity of a page or the likelihood that a page is to be visited regardless of the query. Relevance ranking can be performed by incorporating any number of features, some of which come from the natural language analysis of both the query and the documents.

NLP techniques have been applied in the past to increase IR performance [64], and we

focus specifically on the application of answer type. There have also been prior efforts to transfer QA techniques into the task of document retrieval [4, 66]. This is the first work to incorporate a notion of automatically-identified answer type into IR, although work on templated queries [33] allows a user to explicitly specify the type of the desired response.

In this chapter, we apply a modified version of the answer typing model introduced in Section 3.3 to the problem of IR. IR users interact with the system by specifying keyword or phrasal queries for which a list of documents, along with snippets, are returned. Instead of a unique specific answer, we must be sure to order the list of results based on a notion of relevance to the query. It is left to the user to sort through the results to find the true information required, as is the case for any modern IR system. However, we wish to ensure that high-ranking documents contain some entity that is type-appropriate to the query whenever the notion of type-appropriateness applies. Therefore, our goal is to provide a meaningful improvement via answer typing for a subset of queries submitted to an existing IR system.

The remainder of the chapter is organized as follows. Section 5.1 introduces the resources available to our model. Because the model is so heavily based on the open-ended noun phrase typing model of Section 3.3, we focus on those resources that are in addition to those of Section 3.1. Once the resources have been introduced, Section 5.2 examines a means of identifying queries in a query stream for which a notion of type-appropriateness applies. Many queries in a search engine log are navigational in nature (i.e., seeking some specific page) and so cannot benefit from a notion of answer type. For those queries deemed typeable, we apply a model of answer type described in Section 5.3 to improve their results. Experiments of Section 5.4 show the benefits of applying answer typing to IR queries; conclusions are drawn in Section 5.5.

5.1 Resources

The answer typing model for IR relies heavily upon the resources of Section 3.1. We make use of the same word clusters and context database as discussed in Sections 3.1.1 and 3.1.2. The model of Section 3.3 can be applied directly to words found in search results, leaving only the means by which result documents are scored to be decided. The Google search engine provides *snippets*, short pieces of text derived from the document often containing the query terms, for each result. We rely heavily upon these snippets as discussed in the following section.

5.1.1 Google Snippets

Along with providing links to result documents for a given input query, the Google search engine provides short text snippets for each document. Snippets are meant to capture the essence of a document as it relates to a query by providing the query terms within the context of the document. A snippet can be viewed as a simple query-directed summary of a document that ideally captures other nearby important terms. Ideally, a snippet contains all of the information relevant to a query, although this is often not possible for queries that are searching for certain pages (navigational) or general concepts (definitional). Because these snippets are summaries of documents and are the first form of feedback users see, we base our answer typing model on snippets rather than entire documents.

The quality of snippets is very important to the perceived quality of search results. As a result, Google has spent a great deal of effort to ensure that the generated snippets are optimized for both length and content. Although snippets are not guaranteed to contain type-appropriate responses to certain queries, we assume that Google snippets are of such high quality that any simple attempt to recreate snippets including type-appropriate responses will fail to provide superior results. Therefore, we leave Google snippets unmodified and instead rerank results based on the existing contents of Google snippets. The only drawback of this decision is that lower-ranked results may be promoted based on snippet contents, leading to a situation in which lower-quality results are deemed better than those previously selected by Google search.

5.2 Finding Typeable Queries

Applying answer typing techniques to IR queries depends greatly on the ability to identify queries that can benefit from such typing. The queries of interest are often those searching for some sort of short open-ended noun phrase answer rather than a long description (more definitional), a complex answer (long-response informational), or a particular page (navigational). Definitional queries are looking for explanations of some particular concept and often cannot be adequately answered with a short answer. Long-response informational queries are typeable in some sense, but they require too long of an explanation to be easily handled by open-ended noun phrase typing (e.g., *causes of World War II*). Navigational queries are searching for some particular page(s) and only that particular page(s) is acceptable as a response. We focus instead on the short-answer informational queries, as they are likely to have type-appropriate open-ended noun phrases as either sufficient or appropriate

Pattern	Description
LHS_w	root noun to the left of the preposition
LHS_c	cluster of the root noun
RHS_w	preposition and the root of the prepositional attachment
RHS_c	preposition and cluster of the root of the prep. attachment
has_picture	query contains the word "picture" or "photo"
has_map	query contains the word "map"
initial_the	query starts with the word "the"
rearranged	find a non-prepositional wording of this query in the AOL data set

Table 5.1: Typeability classifier feature templates

responses.

Compound noun queries are prevalent in Web search logs, although their exact number is difficult to measure. Most queries are short (less than three terms) and are likely to be compound noun queries, as many verbs require both subject and object arguments. The search for short-answer informational queries begins with the observation of Lapata and Keller [34] that compound noun phrases can often be interpreted in more than one way. Lapata and Keller examine ways in which certain noun phrases can be paraphrased as equivalent preposition-containing phrases. For example, the compound noun phrase *war stories* can be paraphrased as the equivalent *stories about war*. This transformation is of importance here because preposition-containing queries are easier to identify as being short-answer typeable than compound noun queries. However, due to the prevalence of compound noun queries, we wish to develop a strategy that covers this large class of queries. Therefore, transforming compound noun queries into preposition-containing queries is a form of *query normalization*. With this in mind, we now introduce a method for identifying typeable preposition-containing queries.

Because a typical query stream such as the AOL query log [52] used here contain a wide variety of queries, we must filter out those queries unlikely to benefit from typing. For this task, we make use of large-margin discriminative learning as implemented in a SVM classifier [29]. Our SVM classifier for identifying typeable queries makes use of the features summarized in Table 5.1. Because we only consider prepositional queries, we can break each query into a left-hand side (LHS) and right-hand side (RHS). The LHS contains only the head noun of the query. The RHS includes both the preposition and the head of the prepositional attachment. For example, the query *cities in Canada* has a LHS of *city* and a RHS of *in Canada*. Cluster information (Section 3.1.1) is used to increase the amount of feature overlap between queries that are not identical. For example, *cities in Canada*

and *towns in Mexico* are both typeable and similar. To these features we add flags for certain words that often indicate untypeable queries, such as queries looking for images, photographs, or maps. The presence of the word *the* at the beginning of a query often indicates the query is a single cohesive phrase and likely to be navigational or definitional. Finally, we search for rearrangements, or paraphrases, of the query in the query log with the hope that paraphrasability is an indicator of typeability.

The set of features in Table 5.1 provide information about whether or not a query is typeable. The features themselves are all simple binary flags and must be weighted such that the combination of weights and features yields a decision on whether or not a query is typeable. Learning these weights can be achieved by the use of a SVM classifier [29] in which we find the weight vector $\vec{w}$ and some scalar b such that:

$$\min \frac{1}{2} ||\vec{w}||^2 + C \left(\sum_i \xi_i \right) \tag{5.1}$$

subject to the constraints:

$$\vec{w} \cdot \Phi(x_i) + b \geq 1 - \xi_i \text{ when } y_i = 1 \tag{5.2}$$

$$\vec{w} \cdot \Phi(x_i) + b \leq -1 + \xi_i \text{ when } y_i = -1 \tag{5.3}$$

$$\forall \xi_i \geq 0 \tag{5.4}$$

Here ξ_i are slack variables for the case when the problem is non-separable, $\Phi(x_i)$ are the feature vectors for the queries comprised of the features described above, and $y_i \in \{+1, -1\}$ is the label of whether or not the query is typeable. Once the weight vector $\vec{w}$ has been obtained, we can label unseen queries x_j by finding $y_j = \vec{w} \cdot \Phi(x_j) + b$. If $y_j > 0$, we say that query x_j is typeable, otherwise it is not.

The purpose of this classifier is to identify queries likely to benefit from our notion of typing. For training data, a single annotator labeled a set of 2000 preposition-containing queries according to this single criterion. If the query seems likely to benefit from typing and can be answered by a short noun phrase then the query was labeled as a positive example. Roughly one-third of these queries were labeled as positive examples. A single annotator was used because of the high degree of subjectivity of the task; it is not clear what exactly defines a typeable query and some familiarity with the answer typing method is required to identify cases in which typing is likely to help.

Experiments on a small test set of 200 queries show an accuracy of 87.5% for the query identification model, with errors being divided according to the proportion of positive and negative examples (one-third and two-thirds, respectively). Although this number is

fairly high, precision is only 77%, meaning a number of untypeable queries are incorrectly identified as typeable. Even though this precision is fairly low, we focus on finding answers for certain queries and false positives will not have any applicable short answer.

5.3 Model Specifics

The goal of answer typing for IR queries is to find snippets containing appropriate responses and boosting their position to improve results. We take queries identified as typeable by the model of Section 5.2 and apply the model of Section 3.3 to words found in the snippets as if the query were a question and the snippet contents were potential answers. To preserve the interface of online search engines, we must accept keyword or phrasal queries instead of fully-formed questions. For responses, we must be sure to preserve snippet contents as-is and rerank based on the contents of these snippets. These two requirements, along with the shift in domain from news corpus-based QA to Web-based IR, necessitate a few changes to the model of Section 3.3.

Because we are dealing with queries rather than fully-formed questions, we extract contexts from queries in a slightly different manner. IR queries in general do not have a *wh*-word that signifies the desired answer. Instead, each prepositional query has a focus (the LHS) along with additional information (RHS). We create a single context from the LHS and possibly multiple contexts of maximum length two from the combination of LHS with RHS. For example, the query *cities in Canada* generates the context *X is a city* from the LHS alone and *X is a city in Canada* from the combination of LHS with RHS. The combination of LHS with RHS is used because queries are often short and provide only a few contexts. This is in contrast to questions in which the conceptual RHS can be used alone because they are often more context-rich.

Given that these queries were originally intended for information retrieval, we submit them as-is to the Google search engine and obtain a ranked list of results along with their snippets. The top 100 snippets, an arbitrary number used to balance quantity of results against the use of Google resources, are tagged [55] and chunked [54] to extract a set of noun phrase candidates that are then ranked by the model. These ranked candidates form an ordered *candidate list* on which snippet scores and hence results are based.

As was observed in Section 3.3, a discriminative preference ranking model grants the flexibility of adding additional features on top of basic probabilities derived from the context database. A summary of the feature templates we use for this model can be found in

Pattern	Description
$E(t, \gamma)$	Expected count of candidate t in context γ
$N(t, \gamma)$	Observed count of candidate t in context γ
$\sum_{t'} N(t', \gamma)$	Count of context γ in the corpus
$\sum_{c'} N(t, \gamma')$	Count of candidate t in the corpus
$F(t)$	Count of the times t occurs in the candidate list
$W(t)$	Estimated depth of t in the WordNet hierarchy
$LHS(t, q)$	Flag for when the LHS of query q is a substring of t
$U(t)$	Flag for when t contains capitalized letters
$T(t)$	Number of terms comprising candidate t

Table 5.2: IR typing model feature templates

Table 5.2. The first five of these features are identical to those found in Table 3.2 for the discriminative model for open-ended noun phrase questions (Section 3.3). To these features we add four additional kinds of features that do not rely on contexts. The first, $W(t)$, is the estimated depth of the candidate t in the WordNet hierarchy [19]. Should the candidate not appear in WordNet, we estimate the depth of the candidate by averaging the depth of terms with high similarity to the candidate, according to our cluster resources (Section 3.1.1). Words with named-entity types assigned by Minipar that are not found in WordNet due to poor coverage of WordNet over proper nouns are assigned an arbitrary fixed depth of 12. This particular fixed depth was close to the average depth of named entities that can be found in WordNet. $LHS(t, q)$ is a flag that fires whenever the LHS of a query q (such as *city* in *cities in Canada*) appears as part of the candidate t. We also include a flag, denoted by $U(t)$, that fires when a candidate contains one or more capitalized letters. The features also include the integer number of space-delineated words in the candidate as denoted by $T(t)$. These features allow our model to prefer capitalized terms and terms of a certain length.

For training data, two annotators labeled appropriate candidates in the top 20 snippets (down from 100 to reduce annotator workload) from a total of 200 queries randomly selected from the positive training examples for query identification (Section 5.2). We observed an inter-annotator agreement of $\kappa = 0.68$, which is relatively low. This indicates the difficulty both in interpreting the meaning of queries and in finding appropriate candidates. Because of this relatively low level of agreement, we tested models built on each annotator's labels, along with the intersection of the labels, on a held-out development set. Given that the intersection model showed the best performance, we chose to train on the intersection of the labels.

Once candidates for a question have been ordered by the model, we select the top 20 candidates for scoring snippets. The decision to use the top 20 candidates was based on experiments with a development set in combination with the fact that most queries have only a limited number of appropriate responses in the top 100 snippets. The score of a snippet s is calculated as the average number of candidates per *fragment*:

$$score(s) = \frac{\sum_{t \in top20} in(t, s)}{frag(s)} \tag{5.5}$$

where $in(t, s)$ is 1 if snippet s contains candidate t and 0 otherwise, and $frag(s)$ is the number of fragments separated by "..." in the snippet. Fewer fragments indicate a more cohesive snippet; highly-fragmented snippets require more appropriate candidates to receive a high score. This simple scoring function is designed to avoid relying solely on the output values of the preference ranker and instead uses membership in the top 20 candidates as evidence of a high-quality snippet.

Many factors contribute to Google relevance rankings that are not exposed externally. Therefore, we view the original ranking as a proxy for the combination of such factors. Given that Google often performs well at returning relevant results, we do not wish to deviate too greatly from the Google ordering without strong evidence. To this end, we use an interpolated model most often employed for smoothing [42]. Our final score for a snippet is therefore:

$$interpolated(s) = \alpha \times MRR_{score(s)} + (1 - \alpha) \times MRR_{original} \tag{5.6}$$

We use the Mean Reciprocal Rank (MRR) [67] of our model combined with that of the original Google ordering because MRR provides a natural decrease in score that roughly corresponds to a user's interest in each subsequent result. Ties under our model are assigned an equivalent rank and thus have an equivalent $MRR_{score(s)}$ value. The parameter α controls the degree to which we are willing to deviate from the original Google ordering.

5.4 Experiments

When an IR query is seeking a short answer, even if it is only one of many possibilities, answer typing has a chance to improve results by boosting snippets containing appropriate terms. Capturing such improvements in an IR experiment is a difficult task. Common measures for IR performance are precision and recall along with the associated F-measure. In contrast, QA often uses the MRR measure. Given that we will be dealing primarily with queries for which an answer is thought to exist (Section 5.2), we adopt the notion

of *first correct answer* as is used with MRR. With this in mind, we propose two kinds of experiments to measure the effect of answer typing applied to IR.

The first kind of experiment is a *side-by-side (SxS)* experiment in which results from two alternatives are compared next to one another. Annotators are then asked to choose which side provides a better result. Consistent selection of one side over another reveals which alternative better responds to the query. In general, a SxS experiment can only reveal when one alternative performs better than the other and not when one or both are performing well overall. Therefore comparing two alternatives that are both exceptionally good or bad will result in an insignificant difference. Clearly we want to produce a system that performs well overall and this cannot be measured accurately with a SxS experiment alone.

The second kind of experiment involves only the MRR of a single system. For this kind of experiment, we look for snippets containing partial or complete answers to a query. This only applies to queries that actually have short answers and cannot measure anything for those that do not. Although it may be the case that search results for queries that have no short answer may be degraded in quality, they do not have short answers for which our system was designed to find. This kind of experiment also has the added benefit of measuring the performance of a system directly rather than in comparison to another. Therefore, the overall performance of a system will be revealed in the MRR score itself.

The two kinds of experiments are applied in the following sections to both the candidates which we use to rank snippets in a SxS experiment and the reranked snippets themselves (MRR experiment). To balance the aggressiveness of our typing model against the high quality of Google results, we set $\alpha = 0.4$ in Equation 5.6. This value of α was determined using a held-out development set. The hope is that our system finds better candidates for a query than a simple frequency-based method would find. To measure the quality of candidates, we use a SxS experiment on the top 20 terms sorted according to model score (Section 5.4.1) and frequency. An MRR experiment on the top 5 snippets provided by our system versus the original Google ordering (Section 5.4.2) further shows that our system can make improvements when the query is looking for a short answer.

5.4.1 Candidate Ranking

The candidate ranking SxS experiment is meant to examine the quality of terms selected to rerank snippets. For this task, we display side-by-side the top 20 candidates according to the sort order of frequency and our model score. Displaying a list of appropriate terms does not conform to the task of IR, but allows us to determine whether or not our model can

Total queries	996
Our terms preferred	365 (37%)
Most frequent terms preferred	194 (19%)
Both good	83 (8%)
Both bad	108 (11%)
Indeterminate	246 (25%)

Table 5.3: Top 20 Terms SxS

find appropriate terms for a query. We hope that in some cases the list contains one or more terms that fulfill the information needs of the user. For example, a list containing the word *Edmonton* would be useful for the query *cities in Canada*.

Annotations for this experiment are collected from the Amazon Mechanical Turk (AMT) system.[1] AMT allows a *Requester* to upload a set of *Human Intelligence Tasks* (*HITs*), each of which is a single example to be annotated. Each HIT is annotated by one or more annotators (known as *Turkers*) who are paid small sums of money for their efforts. AMT results, when averaged, have been shown to have high agreement with expert annotators [63] even though Turkers are not experts. We take advantage of this by requiring a minimum of five judgements on any one HIT. One side is preferred over the other if and only if we observe a majority of votes (i.e., ≥ 3) for that side. This leads to a considerable number of *indeterminate* results for which no choice has a majority and for which we cannot confidently assign a single label. These indeterminate results are therefore largely excluded from further consideration.

In comparison with our typing model, we select the top 20 most frequent terms in the snippets excluding any query terms. The idea here is that appropriate terms may be repeated in many snippets. Snippet frequency has previously been shown to be a reasonable measure for identifying appropriate answers to questions in the model for open-ended noun phrase questions (Section 3.3). Table 5.3 shows the results of comparing our top 20 terms with the 20 most frequent terms for a set of 996 queries judged as typeable by the model of Section 5.2. This set of queries includes queries erroneously labeled as typeable because useful terms can exist for queries that are not answerable by short answers alone.

Turkers were asked to judge which of the two lists of 20 snippets were better as a response to a given query. Both lists were sorted according to score to allow for the implicit expectation that candidates appearing higher in a list are more relevant. The results in Table 5.3 show the clear advantage of our model over the most frequent terms indicating

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

Ranking method	MRR
Original Google order	0.514
Reranked by our model	0.567

Table 5.4: Top-5 MRR

that our model is able to identify terms appropriate to this particular subset of queries. We would expect that if the terms were largely irrelevant or if the model performed poorly, the preference would be for the most frequent terms. These results provide motivation for applying the model to rerank snippets according to Equation 5.6, presented in the following section.

5.4.2 Snippet Reranking

The encouraging results of the previous section lead to a MRR experiment for snippets reranked according to our model in comparison with that of the original Google ordering. The Google search engine is currently the most well-known Web search engine and Google has spent a great deal of effort to provide both high-quality relevance ranking and high-quality snippets for each result. Because of this, we expect that Google search results alone will produce a high MRR score when looking for short answers that are a response to a query.

This experiment deals only with those queries for which there is some short answer. Of the 996 queries used in the prior SxS experiment, 331 are strictly determined to be short-answerable, indicating fairly low precision of our classifier (Section 5.2). However, it is for these questions that our model of answer type is designed. Snippets provide additional information and context for answers and are the expected response to IR queries. As a result, snippets that include a short answer to the query allow a user to find desired information along with some context without having to visit additional external Web pages.

For this experiment, two annotators were asked to identify the position of the first snippet containing a short answer to the query. Two annotators were used instead of Amazon Mechanical Turk due to the attention to detail required to identify short answers in snippet text. The answer was allowed to be partial to cover cases in which a list of answers is sought or for which more than one answer is correct. Only the top 5 snippets of our two systems were presented to the annotators to produce a measure of top-5 reciprocal rank in which annotators can use the label *none* to indicate that an answer is not present in any snippet and for which the subsequent reciprocal rank is 0. Top-5 MRR is the average of these top-5

reciprocal rank values over the set of 331 queries. The results for both systems are presented in Table 5.4. The two annotators agree on over 80% of the queries, indicating high confidence in these values. Agreement exists for queries when the same snippet was chosen by both annotators as having a correct answer. We expect some amount of disagreement for queries in which the true intention of the user is difficult to discern. This expectation is in line with inter-annotator agreement numbers observed in Pinchak et al. [58].

The results of Table 5.4 show a slight improvement over the high performance of Google-ranked snippets. Although the improvement in MRR is slight at about 0.05, the results are significant ($p < 0.01$). It should be noted that a system that places the correct answer at position 2 for every query will achieve a MRR of 0.5 and so any improvement upon this is conceptually an increase in the number of answers that appear near the top of the list. Given that Google results are already above this MRR value of 0.5, we conclude that Google offers relatively few opportunities for which we can improve results. The fact that we show a statistically significant improvement for this set of queries means that we are able to capitalize on those rare situations in which Google provides a generic response rather than snippets containing concrete answers that our model prefers.

5.5 Conclusions

The purpose of information retrieval is to find documents that satisfy a user's request for information. Unlike question answering, IR is permitted to return inexact responses with the expectation that the user will sift through the results for the desired information. This chapter has applied answer typing to the task of IR in an effort to improve the ability of users to find specific answers to certain kinds of queries. Although not all IR queries are appropriate for answer typing, at least some are phrased in such a way as to indicate that they are looking for short, concise answers. We believe that responding to such queries with results (or snippets) known to contain appropriate answers improves the IR experience.

The model we employ for this task is a slightly modified version of that described in Section 3.3 for open-ended noun phrase questions. Typeable queries, partially identified by the presence of a preposition and further filtered by the classifier of Section 5.2, are often very similar in form to open-ended noun phrase questions except that they contain less contextual information that gives clues for type-appropriate responses. Thus the observed range of possible contexts is reduced due to the more regular structure of the queries. Still, enough contextual information can be extracted to produce an accurate model of type as

shown by experiments.

Experiments on applying answer typing to IR queries take the form of comparison with a very high performing baseline: the Google search engine. Google performs well at finding relevant results and providing high-quality snippets for most queries. Our goal is to improve results in those situations for which Google does not include type-appropriate responses in high-ranking results. To this end, we apply our answer typing model to queries deemed answerable with short noun-phrase answers and observe a slight, but significant, improvement in performance. When considered against the fact that Google alone performs very well in many cases, this is an important result that shows the applicability of answer typing to a domain outside of QA, in this case the domain of information retrieval.

Chapter 6

Conclusions

The primary goal of this thesis is the investigation of answer typing as an important and useful component of question answering. All too often answer typing is viewed as a task of convenience; if one can easily determine that a question is looking for some named entity, then simply look for those named entities in text. This thesis has shown that answer typing need not be so simplistic nor opportunistic. There are many opportunities for answer typing to make a difference, especially in places in which existing answer typing strategies (overwhelmingly class-based) either do not apply or are insufficient to cover all cases.

Class-based answer typing approaches are applicable for many questions. However, Light et al. [37] have shown that classes are not a perfect solution. A tradeoff exists between generality, for ease in identifying members of a class, and specificity, for accurately focusing the list of appropriate responses, when deciding how to apply a class-based answer typing approach. At one end of the spectrum lies the MUC types for identifying only the most basic of named entities and at the other the Webclopedia types (over 140 separate types) [24] used to identify very specific entities. The models we have introduced here for open-ended noun phrase questions and how-adjective questions do not rely on classes. Instead, we directly decide whether or not a candidate answer is appropriate. This gives a very high level of specificity (at the term level) while still maintaining high accuracy.

The techniques applied in this thesis are based on core resources that are, in many respects, limited. Work on *answer projection* for QA [5] has shown that the application of Web-scale resources to the task of QA results in large performance gains. Therefore, we can view the resources we use here (especially those of Section 3.1), as overly-restricted for the task of QA in general. Should these ideas be applied to a large- or Web-scale QA system, the resources would be more comprehensive and contain more accurate information. Therefore, the performance results presented for each application of answer typing should be viewed

as a lower bound on potential performance in a production setting.

Chapter 5 introduced the application of answer typing techniques to the related, but distinct, field of information retrieval. The goal was to show that answer typing is applicable to domains outside QA. Any system looking for what could conceptually be considered a short answer may benefit from a notion of answer type. Linking type-appropriate terms with information contained in queries or requirements is a potentially useful concept. Although we have used the notion of *question* (or query) and *answer*, we have developed a means by which any linking, such as between query and snippet, can be exploited. IR is only a single example of such a linking but many others are possible in other diverse domains for which type-appropriateness is a meaningful concept.

We now return to the contributions of Section 1.2 and summarize how each has been achieved by this thesis. This is followed by some of the remaining open issues and potential directions of future work.

6.1 Review of Contributions

The three primary contributions of this thesis were set out in Section 1.2. We now return to how each of these contributions are achieved.

6.1.1 Class-Free Answer Typing for Open-Ended Noun Phrase Questions

Chapter 3 introduced two models for class-free typing of open-ended noun phrase questions; one generative and the other based on discriminative preference ranking. The choice to develop a novel class-free approach rather than refine an existing class-based approach was one of the primary goals of this thesis. A class-free approach may not offer great benefits for more well-defined classes of questions, but we expect it to offer great benefits for *miscellaneous*-classed questions. Light et al. [37] provide strong evidence that *miscellaneous* questions do not exhibit high performance under a class-based approach. Ideally, our class-free model performs as well as or better than class-based methods for all questions thereby superseding a class-based method in all cases.

The initial generative model of Section 3.2 shows performance improvements over baselines with varying degrees of perfect information. The *Oracle* system of Section 3.4.1 is difficult to surpass, but exhibits poor performance on the *miscellaneous* questions, those same questions identified by Light et al. [37] as having particularly low performance. The semi-idealized *ANNIE* system (in which type assignment is perfect but entity recognition is

not) is often surpassed by our non-idealized generative model of answer type. These results alone provide strong evidence for the use of our class-free approach to answer typing.

When moving from a generative probabilistic model to the discriminative preference ranking model of Section 3.3 we observe further increases in performance, strengthening the evidence in favour of a class-free model of answer type. Figure 3.4 of Section 3.4.2 shows a dramatic improvement for the discriminative preference ranking model over the generative model. Both models outperform the snippet frequency model that is a simple indicator of answer type. With MRR values as high as those shown in 3.4(a), this discriminative preference ranking model achieves nearly the performance of a complete QA system for focused open-ended noun phrase questions. Because of this, we can conclude that class-free answer typing is a viable alternative to typical class-based approaches often employed as the sole means of answer typing in a QA system.

6.1.2 Fine-Grained Typing of How-Adjective Questions

Open-ended noun phrase questions are a class of questions representing almost innumerable possibilities; any noun can potentially be the answer to such a question. In contrast, the how-adjective questions, i.e., questions for which the *wh*-word *how* is followed by an adjective or adverb, appear very narrow in that they are always looking for some numerical entity. These questions are commonly assigned a type of *number* by class-based answer typing, although this is often refined into a few finer classes (such as *date*, *percent*, and *money*). Even though this class of questions seems restricted at first glance, we need not take such a general view in our approach to typing them.

Chapter 4 introduces a model for how-adjective questions based on the knowledge that the answers are almost always expressed in terms of some unit. Moreover, not all units are appropriate to all how-adjective questions. *Kilograms* is no more appropriate to a *how tall* question than *metres* is to a *how heavy* question. Recognizing and exploiting this fact is key to adequately typing these kinds of questions. Whereas a class-based answer typing method may assign some generic type, our model is able to provide a list of specific units appropriate as answers.

To maintain accuracy, we do not type at the level of an individual question (e.g., *how tall is the CN tower?*) but rather at the level of individual adjectives (e.g., *how tall*). This has the effect of producing one model per adjective rather than one model per question, and is more fine-grained than class-based answer typing. In addition, we introduce methods for the discovery of appropriate units from online resources. The resulting model does not

require fine-grained named-entity recognition to identify appropriate units. Experiments show that such models of answer type are far better than extracting numeric entities alone, even though numeric entities are often easy to identify in text. Furthermore, the model of Section 4.2.2 reinforces the applicability of discriminative preference ranking to the task of answer typing by producing a superior model for how-adjective questions.

6.1.3 Extending Answer Typing Beyond QA

One often thinks of answer typing only as a component of QA. However, answer typing techniques apply to nearly any task for which a notion of type-appropriateness is reasonable. Although the field of information retrieval is closely related to that of QA, the interface and goals are somewhat different. IR accepts keyword-based or phrasal queries for which a set of ranked documents are returned. It is left to the user to sift through the resulting documents for any pieces of information they desire. Not all IR queries are this generic in nature. Some are, in fact, looking for specific short answers.

Chapter 5 introduces a means of identifying such queries (Section 5.2) and a model for boosting results that include type-appropriate responses (Section 5.3) such that a user can find these short, specific answers directly rather than having to sift through results. We use the snippet returned by Google as the unit of response and attempt to find type-appropriate noun phrases within the snippets themselves. A variant of the model used for open-ended noun phrase questions (Section 3.3) is applied because of the similarity of short-answer queries to questions.

Applying the model of answer type from Section 3.3 to IR queries demonstrates the flexibility of class-free answer typing models introduced in this thesis. Experimental results of Section 5.4 show that even when compared to the high quality of Google search results, our model is able to provide a slight, but significant, improvement in performance. IR queries known to be looking for a short noun phrase answer can indeed benefit from the application of an answer typing model even when Google produces highly-relevant ranked search results. With success in the domain of IR, we believe that answer typing can be applied to other areas outside of QA for which type-appropriateness applies.

6.2 Open Issues and Future Directions

Question answering is an ever-evolving field of research. Consequently, answer typing must evolve along with it. Recent progress at NIST-sponsored conferences have taken the focus away from factoid-based question answering (in which each question is more or less

independent of other questions) and began to progress toward *interactive QA* in which a dialogue is maintained between user and QA system to explore a subject in depth [17]. The work described in this thesis is not as well-placed for this shift in direction due to the fact that we view each question as an independent event, much like queries to an IR engine. Although such events may be related in some way, this assumption is not always safe. As a result, this thesis has focused on the more general problem of single questions rather than question dialogues.

A focus on standalone questions is not necessarily a limitation of the applicability of the ideas contained herein. In fact, this work is much more applicable to the field of *domain-specific QA* in which questions and answers are drawn from some specific domain rather than general text. For example, one could create a specific biomedical QA system utilizing medical articles as a corpus (such as those indexed by PubMed[1]) and allowing for specific biomedical questions to be answered. This would provide obvious benefits to biomedical researchers over existing simple query interfaces. The models described in this thesis are built using general resources and often do not rely heavily on Web-scale data sets. As a result, the model of Section 3.2 could be adapted using a biomedical-aware dependency parser and biomedical word clusters. Class-based answer typing would require entirely new classes for answers (such as *protein*, *disease*, *enzyme*, etc.) along with the ability to identify such entities in biomedical text. In contrast, our model could be applied without manual intervention. Exploring domain-specific QA would be worthwhile, especially given the current level of interest in domains such as medicine and the life sciences.

Past work on *answer projection* in QA has shown obvious benefits of using more data than is available in a small reference corpus. Unfortunately, using Web-scale data in the framework introduced in this thesis (the resources of Section 3.1) would require parsing Web-scale data. Parsing such large quantities of text requires a prohibitive amount of computational resources. However, the context database we extract from the corpus need not rely solely on parsed corpora. Recent work by Bergsma et al. [2] has explored the estimation of likelihood of a given term appearing in limited contexts from a n-gram corpus of the World Wide Web. An occurrence need not be observed in this n-gram corpus but can be estimated in a way similar to the combination of clusters with a context database (Section 3.2). Our models require some measure of how likely a word is to appear in a context, which is similar to the information provided by Bergsma et al. Because Bergsma et al. make use of Web-scale resources without the need to perform expensive operations such as

[1] `http://www.pubmedcentral.nih.gov/`

parsing, we expect further improvements in the performance of answer typing models built upon these much larger resources. This may be of particular use to domain-specific QA for domains (or languages) in which parsing resources are limited but data is plentiful. Making use of Web-scale resources in an intelligent way, rather than simple Web search patterns, is the logical next step of this work, allowing it to continue to improve in usefulness to QA and extend beyond into other areas in which type appropriateness is a useful concept.

Bibliography

[1] A.L. Berger, V.J. Della Pietra, and S.A. Della Pietra. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1):39–72, March 1996.

[2] S. Bergsma, D. Lin, and R. Goebel. Discriminative Learning of Selectional Preference from Unlabeled Text. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 59–68, Waikiki, Hawaii, October 2008.

[3] D. Bikel, R. Schwartz, and R. Weischedel. An Algorithm that Learns What's in a Name. *Machine Learning*, 34(1–3):211–231, February 1999.

[4] M. Bilotti, P. Ogilvie, J. Callan, and E. Nyberg. Structured Retrieval for Question Answering. In *Proceedings of SIGIR 2007*, 2007.

[5] E. Brill, J. Lin, M. Banko, S. Dumais, and A. Ng. Data-Intensive Question Answering. In *Proceedings of the Tenth Text Retrieval Conference (TREC-2001)*, Gaithersburg, Maryland, 2001.

[6] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*, 30(1-7):107–117, 1998.

[7] P.F. Brown, V.J. Della Pietra, P.V. deSouza, J.C. Lai, and R.L. Mercer. Class-based n-gram Models of Natural Language. *Computational Linguistics*, 16(2):79–85, 1990.

[8] J. Burger and S. Bayer. MITRE's Qanda at TREC-14. In *Proceedings of the Fourteenth Text REtrieval Conference (TREC- 2005)*, Gaithersburg, Maryland, 2005.

[9] S.F. Chen and J. Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. Technical Report TR-10-98, Computer Science Group, Harvard University, Cambridge, Massachusetts, August 1998.

[10] N. Chinchor. MUC-7 Named Entity Task Definition. In *Message Understanding Conference Proceedings*, 1998.

[11] N. Chinchor. Overview of MUC-7. In *Message Understanding Conference Proceedings*, 1998.

[12] Y. Choueka and S. Lusignan. Disambiguation by Short Contexts. *Computer and the Humanities*, 19:147–157, 1985.

[13] K. Church and P. Hanks. Word Association Norms, Mutual Information, and Lexicography. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 76–83, Vancouver, British Columbia, Canada, 1989.

[14] J. Cohen. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[15] K. Crammer and Y. Singer. On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *Journal of Machine Learning Research*, 2, 2002.

[16] D.R. Cutting, D. Karger, J. Pedersen, and J.W. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. In *Proceedings of SIGIR-92*, pages 318–329, Copenhagen, Denmark, 1992.

[17] H.T. Dang, D. Kelly, and J. Lin. Overview of the TREC 2007 Question Answering Track. In *Proceedings of the Sixteenth Text Retrieval Conference (TREC 2007)*, Gaithersburg, Maryland, 2007.

[18] H. Daumé III and D. Marcu. Bayesian Query-Focused Summarization. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 305–312, 2006.

[19] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[20] R. Gaizauskas, H. Cunningham, Y. Wilks, P. Rodgers, and K. Humphreys. GATE: An Environment to Support Research and Development in Natural Language Engineering. In *Proceeedings of the Eighth IEEE International Conference on Tools with Artificial Intelligence*, Toulouse, France, 1996.

[21] S. Harabagiu, D. Moldovan, C. Clark, M. Bowden, A. Hickl, and P. Wang. Employing Two Question Answering Systems in TREC-2005. In *Proceedings of the Fourteenth Text REtrieval Conference (TREC-2005)*, Gaithersburg, Maryland, 2005.

[22] J. He, C. Chen, C. Yao, P. Yin, and Y. Bao. Peking University at the TREC-2005 Question Answering Track. In *Proceedings of the Fourteenth Text REtrieval Conference (TREC-2005)*, Gaithersburg, Maryland, 2005.

[23] E. Hovy, U. Hermjakob, C-Y. Lin, and D. Ravichandran. Using Knowledge to Facilitate Factoid Answer Pinpointing. In *Proceedings of the 19th International Conference on Computational Linguistics*, 2002.

[24] E.H. Hovy, U. Hermjakob, and D. Ravichandran. A Question/Answer Typology with Surface Text Patterns. In *Proceedings of the DARPA Human Language Technology Conference (HLT)*, San Diego, CA, 2002.

[25] A. Ittycheriah, M. Franz, W-J. Zhu, and A. Ratnaparkhi. Question Answering Using Maximum-Entropy Components. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL 2001)*, Pittsburgh, PA, 2001.

[26] A. Ittycheriah, M. Franz, W-J. Zhu, A. Ratnaparkhi, and R. Mammone. IBM's Statistical Question Answering System. In *Proceedings of the 9th Text REtrieval Conference (TREC-9)*, Gaithersburg, Maryland, 2000.

[27] A. Ittycheriah and S. Roukos. IBM's Statistical Question Answering System – TREC-11. In *Proceedings of the Eleventh Text REtrieval Conference (TREC-2002)*, Gaithersburg, Maryland, 2002.

[28] T. Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proceedings of the European Conference on Machine Learning*, 1998.

[29] T. Joachims. Making Large-Scale SVM Learning Practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*. MIT-Press, 1999.

[30] T. Joachims. Optimizing Search Engines Using Clickthrough Data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2002.

[31] G. Karypis, E.-H. Han, and V. Kumar. Chameleon: A Hierarchical Clustering Algorithm using Dynamic Modeling. *IEEE Computer: Special Issue on Data Analysis and Mining*, 32(8):68–75, 1999.

[32] A. Kilgarriff. Googleology is Bad Science. *Computational Linguistics*, 33(1):147–151, March 2007.

[33] G. Kumaran and J. Allan. Information Retrieval Techniques for Templated Queries. In *Proceedings of RIAO 2007*, 2007.

[34] M. Lapata and F. Keller. Web-based Models for Natural Language Processing. *ACM Transactions on Speech and Language Processing*, 2(1):1–30, February 2005.

[35] W. Li, R. K. Srihari, X. Li, M. Srikanth, X. Zhang, and Cheng Niu. Extracting Exact Answers to Questions Based on Structural Links. In *Proceedings of Multilingual Summarization and Question Answering at the 19th International Conference on Computational Linguistics (COLING 2002)*, Taipei, Taiwan, 2002.

[36] X. Li and D. Roth. Learning Question Classifiers. In *Proceedings of the International Conference on Computational Linguistics (COLING 2002)*, pages 556–562, 2002.

[37] M. Light, G. Mann, E. Riloff, and E. Breck. Analyses for Elucidating Current Question Answering Technology. *Natural Language Engineering*, 7(4):325–342, 2001.

[38] D. Lin. Automatic Retrieval and Clustering of Similar Words. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL 98)*, Montreal, Québec, August 1998.

[39] D. Lin. Language and Text Analysis Tools. In *Proceedings of the First International Conference on Human Language Technology Research*, pages 222–227, San Diego, California, 2001.

[40] D. Lin and P. Pantel. Discovery of Inference Rules for Question Answering. *Natural Language Engineering*, 7(4):343–360, 2001.

[41] I. Mani and G. Wilson. Robust Temporal Processing of News. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 69–76, 2000.

[42] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[43] M.E. Maron and J.L. Kuhns. On Relevance, Probabilistic Indexing and Information Retrieval. *Journal of the ACM*, 7(3):216–244, July 1960.

[44] D. Maynard, V. Tablan, H. Cunningham, C. Ursu, H. Saggion, K. Bontcheva, and Y. Wilks. Architectural Elements of Language Engineering Robustness. *Natural Language Engineering*, 8(2/3):257–274, 2002.

[45] G. Miller. WordNet: An On-line Lexical Database. *International Journal of Lexicography*, 3(4), 1990.

[46] D. Moldovan, C. Clark, S. Harabagiu, and S. Maiorano. COGEX: A Logic Prover for Question Answering. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, 2003.

[47] D. Moldovan, S. Harabagiu, C. Clark, M. Bowden, J. Lehmann, and J. Williams. Experiments and Analysis of LCC's two QA Systems over TREC 2004. In *Notebook Proceedings of the Thirteenth Text REtrieval Conference (TREC-2004)*, pages 21–33, Gaithersburg, Maryland, 2004.

[48] D. Mollá and M. Gardiner. AnswerFinder - Question Answering by Combining Lexical, Syntactic and Semantic Information. In *Proceedings of the Australian Language Technology Workshop (ALTW 2004*, pages 9–16, Sydney, December 2004.

[49] D. Mollá and M. van Zaanen. AnswerFinder at TREC 2005. In *Proceedings of the Fourteenth Text REtrieval Conference (TREC-2005)*, Gaithersburg, Maryland, 2005.

[50] P. Pantel. *Clustering by Committee*. PhD thesis, University of Alberta, 2003.

[51] P. Pantel and D. Lin. Document Clustering with Committees. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 199–206, Tampere, Finland, 2002.

[52] G. Pass, A. Chowdhury, and C. Torgeson. A Picture of Search. In *The First International Conference on Scalable Information Systems*, 2006.

[53] F. Pereira, N. Tishby, and L. Lee. Distributional Clustering of English Words. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 183–190, 1993.

[54] X. Phan. CRFChunker: CRF English Phrase Chunker. http://crfchunker.sourceforge.net, 2006.

[55] X. Phan. CRFTagger: CRF English POS Tagger. http://crftagger.sourceforge.net, 2006.

[56] C. Pinchak and S. Bergsma. Automatic Answer Typing for How-Questions. In *Proceedings of Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*, pages 516–523, Rochester, New York, April 2007.

[57] C. Pinchak and D. Lin. A Probabilistic Answer Type Model. In *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006)*, Trento, Italy, April 2006.

[58] C. Pinchak, D. Lin, and D. Rafiei. Flexible Answer Typing with Discriminative Preference Ranking. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2009)*, pages 666 – 674, Athens, Greece, 2009.

[59] J. Prager, E. Brown, A. Coden, and D. Radev. Question-Answering by Predictive Annotation. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 184–191, 2000.

[60] J. Prager, J. Chu-Carroll, K. Czuba, C. Welty, A. Ittycheriah, and R. Mahindru. IBM's PIQUANT in TREC2003. In *Proceedings of the Twelfth Text REtrieval Conference (TREC-2003)*, Gaithersburg, Maryland, 2003.

[61] G. Salton and C. Buckley. Term Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[62] A.J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[63] R. Snow, B. O'Connor, D. Jurafsky, and A.Y. Ng. Cheap and Fast – But is it Good? Evalating Non-Expert Annotations for Natural Language Tasks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 254–263, Waikiki, Hawaii, October 2008.

[64] T. Strzalkowski. Natural Language Information Retrieval. *Information Processing & Management*, 31(4):397–417, May-June 1995.

[65] R. Sun, J. Jiang, Y.F. Tan, H. Cui, T-S. Chua, and M-Y. Kan. Using Syntactic and Semantic Relation Analysis in Question Answering. In *Proceedings of the Fourteenth Text REtrieval Conference (TREC-2005)*, Gaithersburg, Maryland, 2005.

[66] J. Tiedemann. Improving Passage Retrieval in Question Answering using NLP. In *Proceedings of the 12th Portuguese Conference on Artificial Intelligence (EPIA)*, 2005.

[67] E.M. Voorhees. The TREC-8 Question Answering Track Report. In *Proceedings of the Eighth Text Retrieval Conference (TREC 8)*, pages 77–82, Gaithersburg, Maryland, 1999.

[68] E.M. Voorhees. Overview of the TREC-9 Question Answering Track. In *Proceedings of the Ninth Text Retrieval Conference (TREC-9)*, Gaithersburg, Maryland, 2000.

[69] E.M. Voorhees. Overview of the TRE 2002 Question Answering Track. In *Proceedings of the Eleventh Text Retrieval Conference (TREC 2002)*, Gaithersburg, Maryland, 2002.

[70] E.M. Voorhees. Overview of the TREC 2003 Question Answering Track. In *Proceedings of the Twelfth Text Retrieval Conference (TREC 2003)*, pages 54–68, Gaithersburg, Maryland, 2003.

[71] M. Wu, M. Duan, S. Shaikh, S. Small, and T. Strzalkowski. ILQUA – An IE-Driven Question Answering System. In *Proceedings of the Fourteenth Text REtrieval Conference (TREC-2005)*, Gaithersburg, Maryland, 2005.

[72] D. Zhang and W.S. Lee. Question Classification using Support Vector Machines. In *Proceedings of the 26th ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2003)*, Toronto, ON, 2003.