Chapter 23
Question Answering and Summarization

'Alright', said Deep Thought. 'The Answer to the Great Question...'
'Yes!'
'Of Life The Universe and Everything...' said Deep Thought.
'Yes!'
'Is...'
'Yes...!!!...?'
'Forty-two', said Deep Thought, with infinite majesty and calm...

Douglas Adams, *The Hitchhiker's Guide to the Galaxy*

I read *War and Peace*... It's about Russia...
Woody Allen, *Without Feathers*

Because so much text information is available generally on the Web, or in specialized collections such as PubMed, or even on the hard drives of our laptops, the single most important use of language processing these days is to help us query and extract meaning from these large repositories. If we have a very structured idea of what we are looking for, we can use the information extraction algorithms of the previous chapter. But many times we have an information need that is best expressed more informally in words or sentences, and we want to find either a specific answer fact, or a specific document, or something in between.

In this chapter we introduce the tasks of **question answering** (**QA**) and **summarization**, tasks which produce specific phrases, sentences, or short passages, often in response to a user's need for information expressed in a natural language query. In studying these topics, we will also cover highlights from the field of **information retrieval** (**IR**), the task of returning documents which are relevant to a particular natural language query. IR is a complete field in its own right, and we will only be giving a brief introduction to it here, but one that is essential for understand QA and summarization.

In this chapter we focus on a central idea behind all of these subfields, the idea of meeting a user's information needs by **extracting** passages directly from documents or from document collections like the Web.

Information retrieval (**IR**) is an extremely broad field, encompassing a wide-range of topics pertaining to the storage, analysis, and retrieval of all manner of media, including text, photographs, audio, and video (Baeza-Yates and Ribeiro-Neto, 1999). Our concern in this chapter is solely with the storage and retrieval of text documents in response to users' word-based queries for information. In section 23.1 we present the **vector space model**, some variant of which is used in most current systems, including most web search engines.

Rather than make the user read through an entire document, we'd often prefer to give a single concise short answer. Researchers have been trying to automate this process of **question answering** since the earliest days of computational linguistics (Simmons, 1965).

The simplest form of question answering is dealing with **factoid questions** . As the name implies, the answers to factoid questions are simple facts that can be found in short text strings. The following are canonical examples of this kind of question.

(23.1) Who founded Virgin Airlines?

(23.2) What is the average age of the onset of autism?

(23.3) Where is Apple Computer based?

Each of these questions can be answered directly with a text string that contain the name of person, a temporal expression, or a location, respectively. Factoid questions, therefore, are questions whose answers can be found in short spans of text and correspond to a specific, easily characterized, category, often a named entity of the kind we discussed in Ch. 22. These answers may be found on the Web, or alternatively within some smaller text collection. For example a system might answer questions about a company's product line by searching for answers in documents on a particular corporate website or internal set of documents. Effective techniques for answering these kinds of questions are described in Sec. 23.2.

Sometimes we are seeking information whose scope is greater than a single factoid, but less than an entire document. In such cases we might need a **summary** of a document or set of documents. The goal of **text summarization** is to produce an abridged version of a text which contains the important or relevant information. For example we might want to generate an **abstract** of a scientific article, a **summary** of email threads, a **headline** for a news article, or generate the short **snippets** that web search engines like Google return to the user to describe each retrieved document. For example, Fig. 23.1 shows some sample snippets from Google summarizing the first four documents returned from the query *German Expressionism Brücke*.

Snippet

To produce these various kinds of summaries, we'll introduce algorithms for summarizing single documents, and those for producing summaries of multiple documents by combining information from different textual sources.

Finally, we turn to a field that tries to go beyond factoid question answering by borrowing techniques from summarization to try to answer more complex questions like the following:

(23.4) Who is Celia Cruz?

(23.5) What is a Hajj?

(23.6) In children with an acute febrile illness, what is the efficacy of single-medication therapy with acetaminophen or ibuprofen in reducing fever?

Answers to questions such as these do not consist of simple named entity strings. Rather they involve potentially lengthy coherent texts that knit together an array of associated facts to produce a biography, a complete definition, a summary of current events, or a comparison of clinic results on particular medical interventions. In addition to the complexity and style differences in these answers, the facts that go into such answers may be context, user, and time dependent.

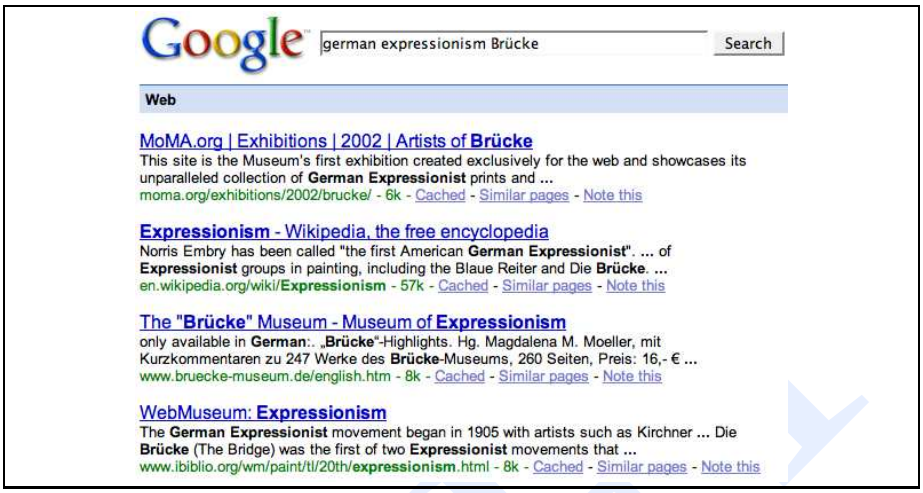

Figure 23.1 The first 4 snippets from Google for *German Expressionism Brücke*.

Complex question

Current methods answer these kinds of **complex questions** by piecing together relevant text segments that come from summarizing longer documents. For example we might construct an answer from text segments extracted from a corporate report, a set of medical research journal articles, or a set of relevant news articles or web pages. This idea of summarizing text in response to a user query is called **query-based summarization** or **focused summarization**, and will be explored in Sec. 23.5.

Query-based summarization

Finally, we reserve for Ch. 24 all discussion of the role that questions play in extended dialogues; this chapter focuses only on responding to a single query.

23.1 Information Retrieval

Information retrieval
IR

Information retrieval (**IR**) is a growing field that encompasses a wide range of topics related to the storage and retrieval of all manner of media. The focus of this section is with the storage of text documents and their subsequent retrieval in response to users' requests for information. In this section our goal is just to give a sufficient overview of information retrieval techniques to lay a foundation for the following sections on question answering and summarization. Readers with more interest specifically in information retrieval should see the references at the end of the chapter.

Most current information retrieval systems are based on a kind of extreme version of compositional semantics in which the meaning of a document resides solely in the set of words it contains. To revisit the Mad Hatter's quote from the beginning of Ch. 19, in these systems *I see what I eat* and *I eat what I see* mean precisely the same thing. The ordering and constituency of the words that make up the sentences that make up documents play no role in determining their meaning. Because they ignore syntactic information, these approaches are often referred to as **bag-of-words** models.

Bag-of-words

Before moving on, we need to introduce some new terminology. In information

Document

retrieval, a **document** refers generically to the unit of text indexed in the system and available for retrieval. Depending on the application, a document can refer to anything from intuitive notions like newspaper articles, or encyclopedia entries, to smaller units such as paragraphs and sentences. In web-based applications, it can refer to a web page, a part of a page, or to an entire website. A **collection** refers to a set of documents being used to satisfy user requests. A **term** refers to a lexical item that occurs in a collection, but it may also include phrases. Finally, a **query** represents a user's information need expressed as a set of terms.

Collection

Term

Query

Ad hoc retrieval

The specific information retrieval task that we will consider in detail is known as **ad hoc retrieval**. In this task, it is assumed that an unaided user poses a query to a retrieval system, which then returns a possibly ordered set of potentially useful documents. The high level architecture is shown in Fig. 23.2.

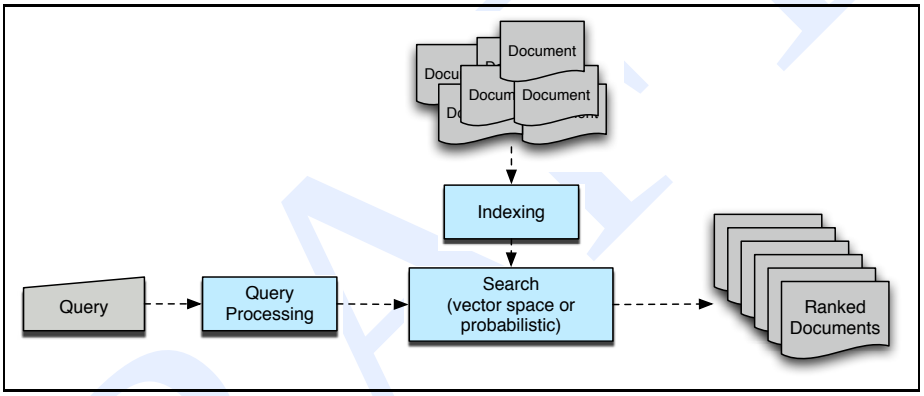

Figure 23.2 The architecture of an ad hoc IR system.

23.1.1 The Vector Space Model

Vector space model

In the **vector space model** of information retrieval, documents and queries are represented as vectors of features representing the terms (words) that occur within the collection (Salton, 1971).

Term weight

The value of each feature is called the **term weight** and is usually a function of the term's frequency in the document, along with other factors.

For example, in a fried chicken recipe we found on the Web the four terms *chicken*, *fried*, *oil*, and *pepper* occur with term frequencies 8, 2, 7, and 4, respectively. So if we just used simple term frequency as our weights, and assuming we pretended only these 4 words occurred in the collection and we put the features are in the above order, the vector for this document (call it j) would be:

$$\vec{d_j} = (8,2,7,4)$$

More generally, we represent a vector for a document d_j as

$$\vec{d_j} = (w_{1,j}, w_{2,j}, w_{3,j}, \cdots, w_{n,j})$$

where $\vec{d}_j$ denotes a particular document, and the vector contains a weight feature for each of the N terms that occur in the collection as a whole; $w_{2,j}$ thus refers to the weight that term 2 has in document j.

We can also represent a query in the same way. For example, a query q for *fried chicken* would have the representation:

$$\vec{q} = (1,1,0,0)$$

More generally,

$$\vec{q} = (w_{1,q}, w_{2,q}, w_{3,q}, \cdots, w_{n,q})$$

Note that N, the number of dimensions in the vector, is the total number of terms in the whole collection. This can be hundreds of thousands of words, even if (as is often done) we don't consider some function words in the set of possible terms. But of course a query or even a long document can't contain very many of these hundreds of thousands of terms. Thus most of the values of the query and document vectors will be zero. Thus in practice we don't actually store all the zeros (we use hashes and other sparse representations).

Now consider a different document, a recipe for poached chicken; here the counts are:

$$\vec{d}_k = (6,0,0,0)$$

Intuitively we'd like the query q *fried chicken* to match document d_j (the fried chicken recipe) rather than document d_k (the poached chicken recipe). A brief glance at the feature suggests that this might be the case; both the query and the fried chicken recipe have the words *fried* and *chicken*, while the poached chicken recipe is missing the word *fried*.

It is useful to view the features used to represent documents and queries in this model as dimensions in a multi-dimensional space, where the feature weights serve to locate documents in that space. When a user's query is translated into a vector it denotes a point in that space. Documents that are located close to the query can then be judged as being more relevant than documents that are farther away.

Fig. 23.3 shows a graphical illustration, plotting the first two dimensions (*chicken* and *fried*) for all three vectors. Note that if we measure the similarity between vectors by the angle between the vectors, that q is more similar to d_j than to d_k, because the angle between q and d_j is smaller.

Cosine

In vector-based information retrieval we standardly use the **cosine** metric that we introduced in Ch. 20 rather than the actual angle. We measure the distance between two documents by the **cosine** of the angle between their vectors. When two documents are identical they will receive a cosine of one; when they are orthogonal (share no common terms) they will receive a cosine of zero. The equation for cosine is:

$$sim(\vec{q},\vec{d}_j) = \frac{\sum_{i=1}^{N} w_{i,q} \times w_{i,j}}{\sqrt{\sum_{i=1}^{N} w_{i,q}^2} \times \sqrt{\sum_{i=1}^{N} w_{i,j}^2}} \tag{23.7}$$

Recall from Ch. 20 that another way to think of the cosine ias as the **normalized dot product**. That is, the cosine is the dot product between the two vectors divided by

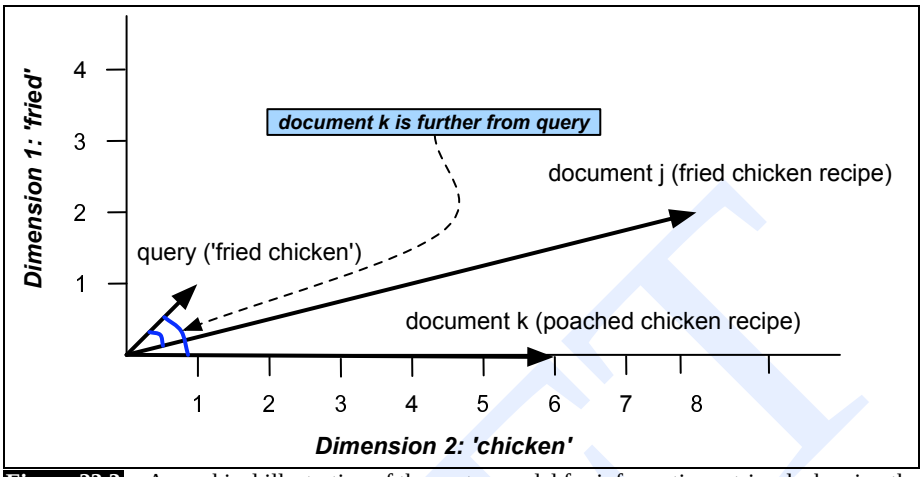

Figure 23.3 A graphical illustration of the vector model for information retrieval, showing the first two dimensions (*fried* and *chicken*) assuming that we use raw frequency in the document as the feature weights.

Dot product

the lengths of each of the two vectors. This is because the numerator of the cosine is the **dot product**:

$$\text{dot-product}(\vec{x},\vec{y}) = \vec{x}\cdot\vec{y} = \sum_{i=1}^{N} x_i \times u_i \tag{23.8}$$

Vector length

while the denominator of the cosine contains terms for the lengths of the two vectors; recall that **vector length** is defined as:

$$|\vec{x}| = \sqrt{\sum_{i=1}^{N} x_i^2} \tag{23.9}$$

This characterization of documents and queries as vectors provides all the basic parts for an ad hoc retrieval system. A document retrieval system can simply accept a user's query, create a vector representation for it, compare it against the vectors representing all known documents, and sort the results. The result is a list of documents rank ordered by their similarity to the query.

Term-by-document matrix

A further note on representation; the characterization of documents as vectors of term weights allows us to view the document collection as a whole as a (sparse) matrix of weights, where $w_{i,j}$ represents the weight of term i in document j. This weight matrix is typically called a **term-by-document matrix**. Under this view, the columns of the matrix represent the documents in the collection, and the rows represent the terms. The term-by-document matrix for the two recipe documents above (again using only the raw term frequency counts as the term weights) would be:

$$A = \begin{pmatrix} 8 & 6 \\ 2 & 0 \\ 7 & 0 \\ 4 & 0 \end{pmatrix}$$

23.1.2 Term Weighting

In the examples above, we assumed that the term weights were set as the simple frequency counts of the terms in the documents. This is a simplification of what we do in practice. The method used to assign terms weights in the document and query vectors has an enormous impact on the effectiveness of a retrieval system. Two factors have proven to be critical in deriving effective term weights. We have already seen the first, the term frequency, in its simplest form the raw frequency of a term within a document (Luhn, 1957). This reflects the intuition that terms that occur frequently within a document may reflect its meaning more strongly than terms that occur less frequently and should thus have higher weights.

The second factor is used to give a higher weight to words that only occur in a few documents. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection, while terms that occur frequently across the entire collection aren't as helpful. documents. The **inverse document frequency** or **IDF** term weight (Sparck Jones, 1972) is one way of assigning higher weights to these more discriminative words. IDF is defined via the fraction N/n_i, where N is the total number of documents in the collection, and n_i is the number of documents in which term i occurs, The fewer documents a term occurs in, the higher this weight. The lowest weight of 1 is assigned to terms that occur in all the documents. Due to the large number of documents in many collections, this measure is usually squashed with a log function. The resulting definition for inverse document frequency (IDF) is thus:

Inverse document frequency

IDF

$$\text{idf}_i = \log\left(\frac{N}{n_i}\right) \tag{23.10}$$

tf-idf

Combining term frequency with IDF results in a scheme known as **tf-idf** weighting:

$$w_{i,j} = \text{tf}_{i,j} \times \text{idf}_i \tag{23.11}$$

In tf-idf weighting, the weight of term i in the vector for document j is the product of its overall frequency in j with the log of its inverse document frequency in the collection (sometimes the term frequency is logged as well). Tf-idf thus prefers words which are frequent in the current document j but rare overall in the collection. Let's repeat the cosine formula for query-document comparison with tf-idf weights added. We'll modify the formula slightly, since as we noted earlier, most values for any query or document vector will be zero. This means that in practice we don't compute the cosine by iterating over all the (mostly zero) dimensions. Instead we only compute over the words that are present, as suggested by the following equation for the **tf-idf weighted cosine** between a query q and a document d:

$$sim(\vec{q},\vec{d}) = \frac{\sum_{w\in q,d} tf_{w,q}tf_{w,d}(idf_w)^2}{\sqrt{\sum_{q_i\in q}(tf_{q_i,q}idf_{q_i})^2} \times \sqrt{\sum_{d_i\in d}(tf_{d_i,d}idf_{d_i})^2}} \tag{23.12}$$

With some minor variations, this tf-idf weighting scheme is used to assign term weights to documents in nearly all vector space retrieval models. The tf-idf scheme

is also used in many other aspects of language processing; we'll see it again when we introduce **summarization** on page 809.

23.1.3 Term Selection and Creation

Thus far, we have been assuming that it is precisely the words that occur in a collection that are used to index the documents in the collection. Two common variations on this assumption involve the use of **stemming**, and a **stop list**.

Stemming

Stemming, as we discussed in Ch. 3, is the process of collapsing the morphological variants of a word together. For example, without stemming, the terms *process*, *processing* and *processed* will be treated as distinct items with separate term frequencies in a term-by-document matrix; with stemming they will be conflated to the single term *process* with a single summed frequency count. The major advantage to using stemming is that it allows a particular query term to match documents containing any of the morphological variants of the term. The Porter stemmer (Porter, 1980) described in Ch. 3 is frequently used for retrieval from collections of English documents.

A problem with this approach is that it throws away useful distinctions. For example, consider the use of the Porter stemmer on documents and queries containing the words *stocks* and *stockings*. In this case, the Porter stemmer reduces these surface forms to the single term *stock*. Of course, the result of this is that queries concerning *stock prices* will return documents about *stockings*, and queries about *stockings* will find documents about *stocks*. Additionally we probably don't want to stem, e.g., the word *Illustrator* to *illustrate*, since the capitalized form *Illustrator* tends to refer to the software package. Most modern web search engines therefore need to use more sophisticated methods for stemming.

Stop list

A second common technique involves the use of stop lists, which address the issue of what words should be allowed into the index. A **stop list** is simply a list of high frequency words that are eliminated from the representation of both documents and queries. Two motivations are normally given for this strategy: high frequency, closed-class terms are seen as carrying little semantic weight and are thus unlikely to help with retrieval, and eliminating them can save considerable space in the inverted index files used to map from terms to the documents that contain them. The downside of using a stop list is that it makes it difficult to search for phrases that contain words in the stop list. For example, a common stop list presented in Frakes and Baeza-Yates (1992), would reduce the phrase *to be or not to be* to the phrase *not*.

23.1.4 Evaluating Information Retrieval Systems

The basic tools used to measure the performance of ranked retrieval system are the **precision** and **recall** measures we employed in earlier settings. Here we assume that the returned items can be divided into two categories: those relevant to our purposes and those that are not. Therefore, precision is the fraction of the returned documents that are relevant, while recall is the fraction of all possible relevant documents that are contained in the return set. More formally, let's assume that we have been given a total of T ranked documents in response to a given information request, a subset of these documents, R, consists of relevant documents, and a disjoint subset, N, consists of the

remaining irrelevant documents, and finally let's assume that there are U documents in the collection as a whole that are relevant to this particular request. Given all this we can define our precision and recall measures to be:

$$Precision = \frac{|R|}{|T|} \tag{23.13}$$

$$Recall = \frac{|R|}{|U|} \tag{23.14}$$

Unfortunately, these metrics are not quite sufficient to measure the performance of a system that *ranks* the documents it return. That is, if we are comparing the performance of two ranked retrieval systems, we require a metric that will prefer the one that ranks the relevant documents higher. Simple precision and recall as defined above are not dependent on rank in any way; we need to adapt them to capture how well a system does at putting relevant documents higher in the ranking. The two standard methods in information retrieval for accomplishing this are based on plotting precision/recall curves and on averaging precision measures in various ways.

Rank	Judgment	Precision_{Rank}	Recall_{Rank}
1	R	1.0	.11
2	N	.50	.11
3	R	.66	.22
4	N	.50	.22
5	R	.60	.33
6	R	.66	.44
7	N	.57	.44
8	R	.63	.55
9	N	.55	.55
10	N	.50	.55
11	R	.55	.66
12	N	.50	.66
13	N	.46	.66
14	N	.43	.66
15	R	.47	.77
16	N	.44	.77
17	N	.44	.77
18	R	.44	.88
19	N	.42	.88
20	N	.40	.88
21	N	.38	.88
22	N	.36	.88
23	N	.35	.88
24	N	.33	.88
25	R	.36	1.0

Figure 23.4 Rank-specific precision and recall values calculated as we proceed down through a set of ranked documents.

Let's consider each of these methods in turn using the data given in the table in Fig. 23.4. This table provides rank-specific precision and recall values calculated as we proceed down through a set of ranked items. That is, the precision numbers are the fraction of relevant documents seen at a given rank, and recall is the fraction of relevant documents found at the same rank. The recall measures in this example are based on this query having 9 relevant documents in the collection as a whole. Note that recall is non-decreasing as we proceed, when relevant items are encountered recall increases and when non-relevant documents are found it remains unchanged. Precision on the other hand hops up and down, increasing when relevant documents are found and decreasing otherwise.

One common way to get a handle on this kind of data is to plot precision against recall on a single graph using data gathered from across a set of queries. To do this we'll need a way to average the recall and precision values across a set of queries. The standard way to do this is to plot averaged precision values at 11 fixed levels of recall (0 to 100, in steps of 10). Of course, as is illustrated by our earlier table we're not likely to have datapoints at these exact levels for all (or any) of the queries in our evaluation set. We'll therefore use **interpolated precision** values for the 11 recall values from the data points we do have. This is accomplished by choosing the maximum precision value achieved at any level of recall at or above the one we're calculating. In other words,

Interpolated precision

$$IntPrecision(r) = \max_{i>=r} Precision(i) \tag{23.15}$$

Note that this interpolation scheme not only provides us with the means to average performance over a set of queries, but it also provides a sensible way to smooth over the irregular precision values in the original data. This particular smoothing method is designed to give systems the benefit of the doubt by assigning the maximum precision value achieved at higher levels of recall from the one being measured. The interpolated data points for our earlier example are given in the following table and plotted in Fig. 23.5.

Interpolated Precision	Recall
1.0	0.0
1.0	.10
.66	.20
.66	.30
.66	.40
.63	.50
.55	.60
.47	.70
.44	.80
.36	.90
.36	1.0

Figure 23.5 Interpolated data points from Fig. 23.4.

Given curves such as this we can compare two systems or approaches by comparing

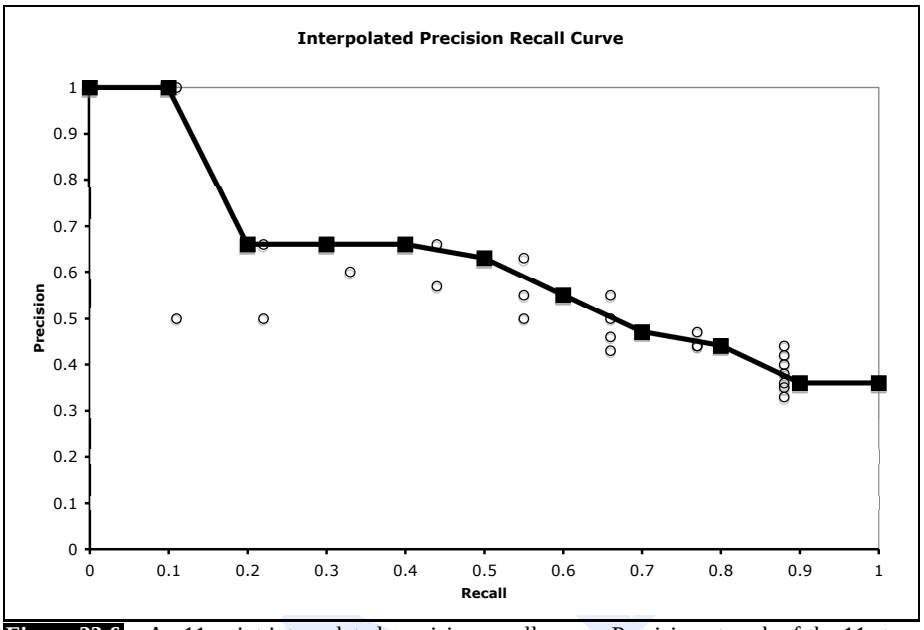

Figure 23.6 An 11 point interpolated precision-recall curve. Precision at each of the 11 standard recall levels is interpolated for each query from the maximum at any higher level of recall. The original measured precision recall points are also shown.

their curves. Clearly curves that are higher in precision across all recall values are preferred. However, these curves can also provide insight into the overall behavior of a system. Systems that are higher in precision towards the left may favor precision over recall, while systems that are more geared towards recall will be higher at higher levels of recall (to the right).

Mean average precision

A second popular way to evaluate ranked retrieval systems is known as **mean average precision**(MAP). In this approach, we again descend through the ranked list of items and note the precision only at those points where a relevant item has been encountered. For a single query, we average these individual precision measurements over the return set up to some fixed cutoff. More formally, if we assume that R_r is the set of relevant documents at or above r, then the average precision for a single query is:

$$\frac{1}{|R_r|} \sum_{d \in R_r} Precision_r(d) \tag{23.16}$$

where $Precision_r(d)$ is the precision measured at the rank where document d was found. For an ensemble of queries, we then average over these averages, giving us our mean average precision measure. Applying this technique to the data in Fig. 23.5 yields a MAP measure of 0.6 for this single retrieval.

MAP has the advantage of providing a single crisp metric that can be used to compare competing systems or approaches. Note, that MAP will tend to favor systems that provide relevant documents at high ranks. Of course, this isn't really a problem since

that is a big part of what we're looking for in a retrieval system. But since the measure essentially ignores recall, it can favor those systems that are tuned to return small sets of documents in which they are highly confident, at the expense of systems that attempt to be more comprehensive by trying to attain higher levels of recall.

The U.S. government-sponsored TREC (Text REtrieval Conference) evaluations, run annually since 1992, provide a rigorous testbed for the evaluation of a variety of information retrieval tasks and techniques. TREC provides large document sets for both training and testing, along with a uniform scoring system. Training materials consist of sets of documents accompanied by sets of queries (called topics in TREC) and relevance judgments. TREC subtasks over the years have included question answering, IR in Chinese and Spanish, interactive IR, retrieval from speech and video, and others. See Voorhees and Harman (2005). Details of all of the meetings can be found at the TREC page on the National Institute of Standards and Technology website.

23.1.5 Homonymy, Polysemy, and Synonymy

Since the vector space model is based solely on the use of simple terms, it is useful to consider the effect that various lexical semantic phenomena may have on the model. Consider a query containing the word *canine*, a word that has senses meaning something like *tooth* and *dog*. A query containing *canine* will be judged similar to documents making use of either of these senses. However, given that users are probably only interested in one of these senses, the documents containing the other sense will be judged non-relevant. Homonymy and polysemy, therefore, can have the effect of *reducing precision* by leading a system to return documents irrelevant to the user's information need.

Now consider a query consisting of the lexeme *dog*. This query will be judged close to documents that make frequent use of the term *dog*, but may fail to match documents that use close synonyms like *canine*, as well as documents that use hyponyms such as *Malamute*. Synonymy and hyponymy, therefore, can have the effect of *reducing recall* by causing the retrieval system to miss relevant documents.

Note that it is inaccurate to state flatly that polysemy reduces precision, and synonymy reduces recall since, as we discuss on page 789, both measures are relative to a fixed cutoff. As a result, every non-relevant document that rises above the cutoff due to polysemy takes up a slot in the fixed size return set, and may thus push a relevant document below threshold, thus reducing recall. Similarly, when a document is missed due to synonymy, a slot is opened in the return set for a non-relevant document, potentially reducing precision as well.

These issues lead naturally to the question of whether or not word sense disambiguation can help in information retrieval. The current evidence on this point is mixed, with some experiments reporting a gain using disambiguation-like techniques (Schütze and Pedersen, 1995), and others reporting either no gain, or a degradation in performance (Krovetz and Croft, 1992; Sanderson, 1994; Voorhees, 1998).

23.1.6 Improving User Queries

One of the most effective ways to improve retrieval performance is to find a way to improve user queries. The techniques presented in this section have been shown to varying degrees to be effective at this task.

Relevance feedback

The single most effective way to improve retrieval performance in the vector space model is the use of **relevance feedback** (Rocchio, 1971). In this method, a user presents a query to the system and is presented with a small set of retrieved documents. The user is then asked to specify which of these documents appears relevant to their need. The user's original query is then reformulated based on the distribution of terms in the relevant and non-relevant documents that the user examined. This reformulated query is then passed to the system as a *new* query with the new results being shown to the user. Typically an enormous improvement is seen after a single iteration of this technique.

The formal basis for the implementation of this technique falls out directly from some of the basic geometric intuitions of the vector model. In particular, we would like to *push* the vector representing the user's original query toward the documents that have been found to be relevant, and away from the documents judged not relevant. This can be accomplished by adding an averaged vector representing the relevant documents to the original query, and subtracting an averaged vector representing the non-relevant documents.

More formally, let's assume that $\vec{q}_i$ represents the user's original query, R is the number of relevant documents returned from the original query, S is the number of non-relevant documents, and documents in the relevant and non-relevant sets are denoted as $\vec{r}$ and $\vec{s}$, respectively. In addition, assume that β and γ range from 0 to 1 and that $\beta + \gamma = 1$. Given these assumptions, the following represents a standard relevance feedback update formula:

$$\vec{q}_{i+1} = \vec{q}_i + \frac{\beta}{R}\sum_{j=1}^{R}\vec{r}_j - \frac{\gamma}{S}\sum_{k=1}^{S}\vec{s}_k$$

The factors β and γ in this formula represent parameters that can be adjusted experimentally. Intuitively, β represents how far the new vector should be pushed towards the relevant documents, and γ represents how far it should be pushed away from the non-relevant ones. Salton and Buckley (1990) report good results with $\beta = .75$ and $\gamma = .25$.

We should note that evaluating systems that use relevance feedback is rather tricky. In particular, an enormous improvement is often seen in the documents retrieved by the first reformulated query. This should not be too surprising since it includes the documents that the user told the system were relevant on the first round. The preferred way to avoid this inflation is to only compute recall and precision measures for what is called the **residual collection**, the original collection without any of the documents shown to the user on any previous round. This usually has the effect of driving the system's raw performance below that achieved with the first query, since the most highly relevant documents have now been eliminated. Nevertheless, this is an effective technique to use when comparing distinct relevance feedback mechanisms.

Residual collection

Query expansion

An alternative approach to query improvement focuses on terms that comprise the query vector. In **query expansion**, the user's original query is expanded by adding terms that are synonymous with or related to the original terms. Query expansion is thus a technique for improving recall, perhaps at the expense of precision. For example the query *Steve Jobs* could be expanded by adding terms like *Apple*, *Macintosh*, and *personal computer*.

Thesaurus

The terms to be added to the query are taken from a **thesaurus**. It is possible to use a hand-built resource like WordNet or UMLS as the thesaurus for query expansion, when the domain is appropriate. But often these thesauruses are not suitable for the collection, and instead, we do **thesaurus generation**, generating a thesaurus automatically from documents in the collection. We can do this by clustering the words in the collection, a method known as **term clustering**. Recall from our characterization of the term-by-document matrix that the columns in the matrix represent the documents and the rows represent the terms. Thus, in thesaurus generation, the rows can be clustered to form sets of synonyms, which can then be added to the user's original query to improve its recall. The distance metric for clustering can be simple cosine, or any of the other distributional methods for word relatedness discussed in Ch. 20.

Thesaurus generation

Term clustering

The thesaurus can be generated once from the document collection as a whole (Crouch and Yang, 1992), or sets of synonym-like terms can be generated dynamically from the returned set for the original query (Attar and Fraenkel, 1977). Note that this second approach entails far more effort, since in effect a small thesaurus is generated for the documents returned for every query, rather than once for the entire collection.

23.2 Factoid Question Answering

There are many situations where the user wants a particular piece of information rather than an entire document or document set. We use the term **question answering** for the task of returning a particular piece of information to the user in response to a question. We call the task **factoid question answering** if the information is a simple fact, and particularly if this fact has to do with a **named entity** like a person, organization, or location.

The task of a factoid question answering system is thus to answer questions by finding, either from the Web or some other collection of documents, short text segments that are likely to contain answers to questions, reformatting them, and presenting them to the user. Fig. 23.7 shows some sample factoid questions together with their answers.

Since factoid question answering is based on information retrieval techniques to find these segments, it is subject to the same difficulties as information retrieval. That is, the fundamental problem in factoid question answering is the gap between the way that questions are posed and the way that answers are expressed in a text. Consider the following question/answer pair from the TREC question answering task:

User Question: What company sells the most greeting cards?

Potential Document Answer: Hallmark remains the largest maker of greeting cards.

Here the user uses the verbal phrase *sells the most* while the document segment

Question	Answer
Where is the Louvre Museum located?	in Paris, France
What's the abbreviation for limited partnership?	L.P.
What are the names of Odin's ravens?	Huginn and Muninn
What currency is used in China?	the yuan
What kind of nuts are used in marzipan?	almonds
What instrument does Max Roach play?	drums
What's the official language of Algeria?	Arabic
What is the telephone number for the University of Colorado, Boulder?	(303)492-1411
How many pounds are there in a stone?	14

Figure 23.7 Some sample factoid questions and their answers.

uses a nominal *the largest maker*. The solution to the possible mismatches between question and answer form lies in the ability to robustly process *both* questions and candidate answer texts in such a way that a measure of similarity between the question and putative answers can be performed. As we'll see, this process involves many of the techniques that we have introduced in earlier chapters including limited forms of morphological analysis, part-of-speech tagging, syntactic parsing, semantic role labelling, named-entity recognition, and information retrieval.

Because it is impractical to employ these relatively expensive NLP techniques like parsing or role labeling on vast amounts of textual data, question answering systems generally use information retrieval methods to first retrieve a smallish number of potential documents. The most expensive techniques then used in a second pass on these smaller numbers of candidate relevant texts.

Fig. 23.8 shows the three phases of a modern factoid question answering system: question processing, passage retrieval and ranking, and answer processing.

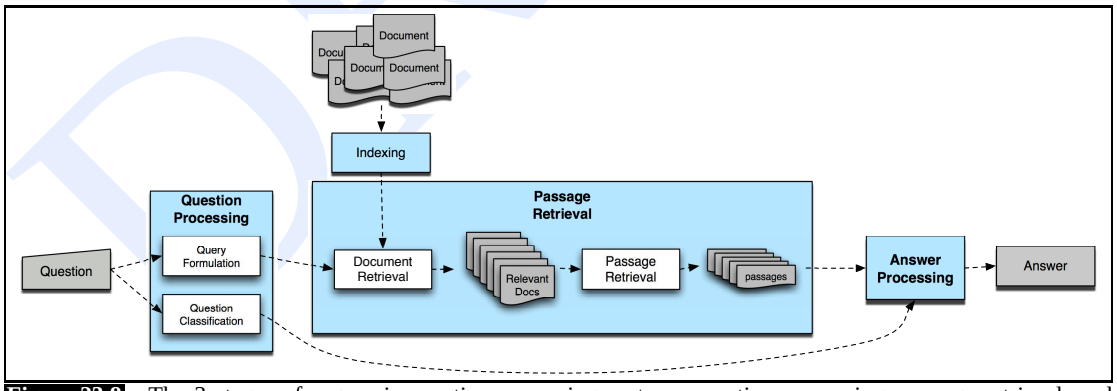

Figure 23.8 The 3 stages of a generic question answering system: question processing, passage retrieval, and answer processing..

23.2.1 Question Processing

The goal of the question processing phase is to extract two things from the question: a keyword **query** suitable as input to an IR system and an **answer type**, a specification of the kind of entity that would constitute a reasonable answer to the question.

Query Formulation

The process of **query formulation** is very similar to the processing done on other IR queries. Our goal is to create from the question a list of keywords that forms an IR query.

Exactly what query to form depends on the question answering application. If question answering is applied to the Web, we might simply create a keyword from every word in the question, letting the web search engine automatically remove any stopwords. Often we leave out the question word (*where*, *when*, etc). Alternatively, keywords can be formed from only the terms found in the noun phrases in the question, applying stopword lists to ignore function words and high-frequency, low-content verbs.

When question answering is applied to smaller sets of documents, for example to answer questions about corporate information pages, we still use an IR engine to search our documents for us. But for this smaller set of documents we generally need to apply query expansion. On the Web the answer to a question might appear in many different forms, and so if we search with words from the question we'll probably find an answer written in the same form. In smaller sets of corporate pages, by contrast, an answer might appear only once, and the exact wording might look nothing like the question. Thus query expansion methods can add query terms hoping to match the particular form of the answer as it appears.

Thus we might add to the query all morphological variants of the content words in the question, as well as applying the thesaurus-based or other query expansion algorithms discussed in the previous section to get a larger set of keywords for the query. Many systems use WordNet as a thesaurus, while others rely on special-purpose thesauruses that are specifically hand-built for question-answering.

Query reformulation

Another query formulation approach that is sometimes used when questioning the Web is to apply a set of **query reformulation** rules to the query. The rules rephrase the question to make it look like a substring of possible declarative answers. For example the question *"when was the laser invented?"* would be reformulated as *the laser was invented*; the question *"where is the Valley of the Kings?"* might be reformulated as *"the Valley of the Kings is located in"*. We can apply multiple such rules to the query, and pass all the resulting reformulated queries to the web search engine. Here are some sample hand-written reformulation rules from Lin (2007):

(23.17) *wh-word* did A *verb* B → ... A *verb*+ed B

(23.18) Where is A → A is located in

Question Classification

Answer type

The second task in question processing is to classify the question by its expected **answer type**. For example a question like *"Who founded Virgin Airlines"* expects an

Question classification
Answer type recognition

answer of type PERSON. A question like *"What Canadian city has the largest population?"* expects an answer of type CITY. This task is called **question classification** or **answer type recognition**. If we know the answer type for a question, we can avoid looking at every sentence or noun phrase in the entire suite of documents for the answer, instead focusing on, e.g., just people or cities. Knowing an answer type is also important for presenting the answer. A DEFINITION question like *"What is a prism"* might use a simple answer template like *"A prism is..."* while an answer to a BIOGRAPHY question like *"Who is Zhou Enlai?"* might use a biography-specific template, perhaps beginning with the persons nationality and proceeding to their dates of birth and other biographical information.

As some of the above examples suggest, we might draw the set of possible answer types for a question classifier from a set of named entities like the PERSON, LOCATION, and ORGANIZATION described in Ch. 22. Usually, however, a somewhat richer set of answer types is used. These richer tagsets are often hierarchical, and so we usually call them an **answer type taxonomy** or a **question ontology**. Such taxonomies can be built semi-automatically and dynamically, for example from WordNet (Harabagiu et al., 2000; Pasca, 2003), or they can be designed by hand.

Answer type taxonomy
Question ontology

Fig. 23.2.1 shows one such hand-built ontology, the hierarchical Li and Roth (2005) tagset. In this tagset, each question can be labeled with a coarse-grained tag like HUMAN, or a fine-grained tag like HUMAN:DESCRIPTION, HUMAN:GROUP, HUMAN:IND, and so on. Similar tags are used in other systems; the type HUMAN:DESCRIPTION is often called a BIOGRAPHY question, because the answer requires giving a brief biography of the person, rather than just a name.

Question classifiers can be built by hand-writing rules, via supervised machine learning, or via some combination. The Webclopedia QA Typology, for example, contains 276 hand-written rules associated with the approximately 180 answer types in the typology (Hovy et al., 2002). A regular expression rule for detecting an answer type like BIOGRAPHY (which assumes the question has been named-entity tagged) might be:

(23.19) who {is | was | are | were} PERSON

Most modern question classifiers, however, are based on supervised machine learning techniques. These classifiers are trained on databases of questions that have been hand-labeled with an answer type such as the corpus of Li and Roth (2002). Typical features used for classification include the words in the questions, the part-of-speech of each word, and named entities in the questions.

Often a single word in the question gives extra information about the answer type, and its identity is used as a feature. This word is sometimes called the question **headword** or the **answer type word**, and may be defined as the headword of the first NP after the question's *wh-word*; headwords are indicated in boldface in the following examples:

(23.20) Which **city** in China has the largest number of foreign financial companies?

(23.21) What is the state **flower** of California?

Finally, it often helps to use semantic information about the words in the questions. The WordNet synset id of the word can be used as a feature, as can the ids of the hypernym and hyponyms of each word in the question.

Tag	Example
ABBREVIATION	
abb	What's the abbreviation for limited partnership?
exp	What does the "c" stand for in the equation E=mc2?
DESCRIPTION	
definition	What are tannins ?
description	What are the words to the Canadian National anthem?
manner	How can you get rust stains out of clothing?
reason	What caused the Titanic to sink ?
ENTITY	
animal	What are the names of Odin's ravens?
body	What part of your body contains the corpus callosum ?
color	What colors make up a rainbow ?
creative	In what book can I find the story of Aladdin?
currency	What currency is used in China?
disease/medicine	What does Salk vaccine prevent ?
event	What war involved the battle of Chapultepec?
food	What kind of nuts are used in marzipan?
instrument	What instrument does Max Roach play?
lang	What's the official language of Algeria?
letter	What letter appears on the cold-water tap in Spain?
other	What is the name of King Arthur's sword?
plant	What are some fragrant white climbing roses?
product	What is the fastest computer ?
religion	What religion has the most members ?
sport	What was the name of the ball game played by the Mayans?
substance	What fuel do airplanes use?
symbol	What is the chemical symbol for nitrogen ?
technique	What is the best way to remove wallpaper?
term	How do you say " Grandma " in Irish ?
vehicle	What was the name of Captain Bligh's ship ?
word	What's the singular of dice?
HUMAN	
description	Who was Confucius?
group	What are the major companies that are part of Dow Jones ?
ind	Who was the first Russian astronaut to do a spacewalk?
title	What was Queen Victoria's title regarding India?
LOCATION	
city	What's the oldest capital city in the Americas ?
country	What country borders the most others?
mountain	What is the highest peak in Africa?
other	What river runs through Liverpool?
state	What states do not have state income tax?
NUMERIC	
code	What is the telephone number for the University of Colorado?
count	About how many soldiers died in World War II?
date	What is the date of Boxing Day?
distance	How long was Mao's 1930s Long March?
money	How much did a McDonald's hamburger cost in 1963?
order	Where does Shanghai rank among world cities in population?
other	What is the population of Mexico?
period	What was the average life expectancy during the Stone Age?
percent	
speed	What is the speed of the Mississippi River?
temp	How fast must a spacecraft travel to escape Earth's gravity?
size	What is the size of Argentina?
weight	How many pounds are there in a stone?

Figure 23.9 Question typology from Li and Roth (2002, 2005). Example sentences are from their corpus of 5500 labeled questions. A question can be labeled either with a coarse-grained tag like HUMAN or NUMERIC, or a fine-grained tag like HUMAN:DESCRIPTION, HUMAN:GROUP, HUMAN:IND, and so on.

In general question classification accuracies are relatively high on easy question types like PERSON, LOCATION, and TIME questions; detecting REASON and DESCRIPTION questions can be much harder.

23.2.2 Passage Retrieval

The query that was created in the question processing phase is next used to query an information retrieval system, either a general IR engine over a proprietary set of indexed documents or a web search engine. The result of this document retrieval stage is a set of documents.

Although the set of documents is generally ranked by relevance, the top-ranked document is probably not the answer to the question. This is because documents are not an appropriate unit to rank with respect to the goals of a question answering system. A highly relevant and large document that does not prominently answer a question is not an ideal candidate for further processing.

Therefore, the next stage is to extract a set of potential answer passages from the retrieved set of documents. The definition of a passage is necessarily system dependent, but the typical units include sections, paragraphs and sentences. For example, we might run a paragraph segmentation algorithm of the type discussed in Ch. 21 on all the returned documents and treat each paragraph as a segment.

Passage retrieval

We next perform **passage retrieval**. In this stage we first filter out passages in the returned documents that don't contain potential answers, and then rank the rest according to how likely they are to contain an answer to the question. The first step in this process is to run a named entity or answer-type classification on the retrieved passages. The answer type that we determined from the question tells us the possible answer types (extended named entities) we expect to see in the answer. We can therefore filter out documents that don't contain any entities of the right type.

The remaining passages are then ranked; either via hand-crafted rules or supervised training with machine learning techniques. In either case, the ranking is based on a relatively small set of features that can be easily and efficiently extracted from a potentially large number of answer passages. Among the more common features are:

- The number of **named entities** of the right type in the passage
- The number of **question keywords** in the passage
- The longest exact sequence of question keywords that occurs in the passage
- The rank of the document from which the passage was extracted
- The **proximity** of the keywords from the original query to each other:
  For each passage identify the shortest span that covers the keywords contained in that passage. Prefer smaller spans that include more keywords (Pasca, 2003; Monz, 2004).
- The N**-gram overlap** between the passage and the question:
  Count the N-grams in the question and the N-grams in the answer passages. Prefer the passages with higher N-gram overlap with the question (Brill et al., 2002).

For question answering from the Web, instead of extracting passages from all the returned documents, we can rely on the web search to do passage extraction for us. We

do this by using **snippets** produced by the web search engine as the returned passages. For example, Fig. 23.10 shows some snippets for the first 5 document returned from the Google search engine for the query *When was movable type metal printing invented in Korea?*

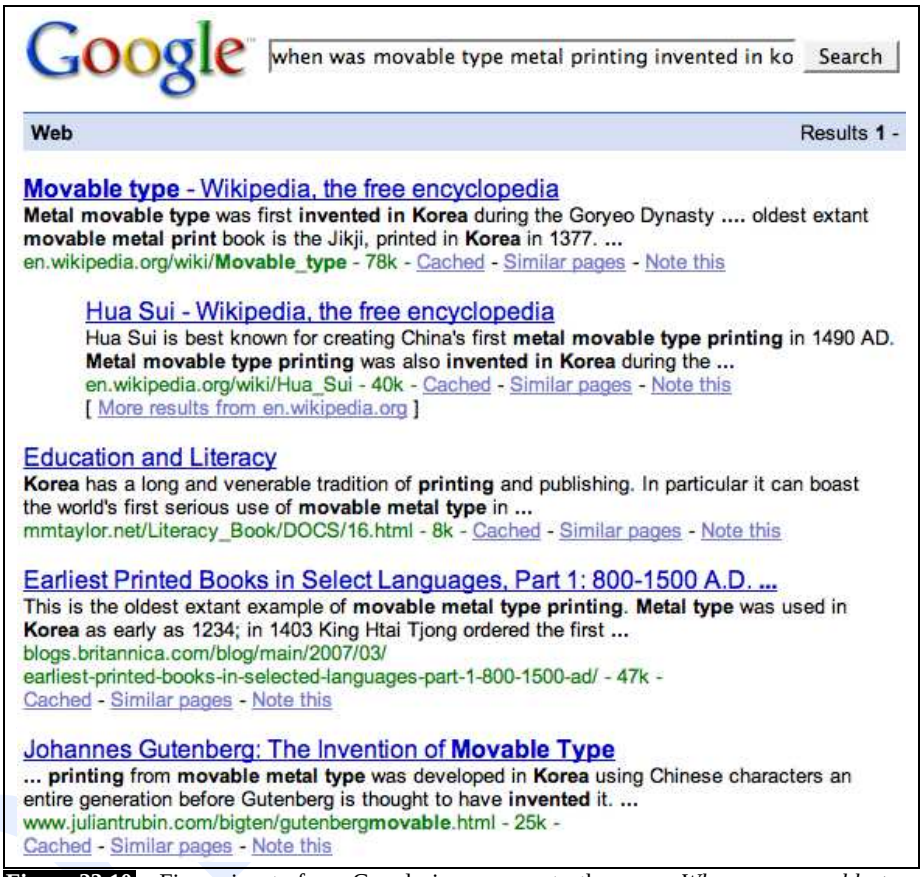

Figure 23.10 Five snippets from Google in response to the query *When was movable type metal printing invented in Korea?*

23.2.3 Answer Processing

The final stage of question answering is to extract a specific answer from the passage, so as to be able to present the user with an answer like *300 million* to the question *"What is the current population of the United States"*.

Two classes of algorithms have been applied to the answer extraction task, one based on **answer-type pattern extraction** and one based on *N*-**gram tiling**.

In the **pattern extraction** methods for answer processing, we use information about the expected answer type together with regular expression patterns. For example, for questions with a HUMAN answer type we run the answer type or named entity tagger on

the candidate passage or sentence, and return whatever entity is labeled with type HUMAN. Thus in the following examples, the underlined named entities are extracted from the candidate answer passages as the answer to the HUMAN and DISTANCE-QUANTITY questions:

"Who is the prime minister of India"
Manmohan Singh, Prime Minister of India, had told left leaders that the deal would not be renegotiated.

"How tall is Mt. Everest?
The official height of Mount Everest is 29035 feet

Unfortunately, the answers to some questions, such as DEFINITION questions, don't tend to be of a particular named entity type. For some questions, then, instead of using answer types, we use handwritten regular expression patterns to help extract the answer. These patterns are also useful in cases where a passage contains multiple examples of the same named entity type. Fig. 23.2.3 shows some patterns from Pasca (2003) for the question phrase (QP) and answer phrase (AP) of definition questions.

Pattern	Question	Answer
<AP> such as <QP>	What is autism?	", developmental disorders such as autism"
<QP> (an <AP>)	What is a caldera?	"the Long Valley caldera, a volcanic crater 19 miles long"

Figure 23.11 Some answer extraction patterns for definition questions (Pasca, 2003).

The patterns are specific to each question type, and can either be written by hand or learned automatically.

The automatic pattern learning method of Ravichandran and Hovy (2002), Echihabi et al. (2005), for example, makes use of the pattern-based methods for relation extraction we introduced in Ch. 20 and Ch. 22 (Brin, 1998; Agichtein and Gravano, 2000). The goal of the pattern learning method is to learn a relation between a particular answer type such as YEAR-OF-BIRTH, and a particular aspect of the question, in this case the name of the person whose birth year we want. We are thus trying to learn patterns which are good cues for a relation between two phrases (PERSON-NAME/YEAR-OF-BIRTH, or TERM-TO-BE-DEFINED/DEFINITION, etc). This task is thus very similar to the task of learning hyponym/hyponym relations between WordNet synsets introduced in Ch. 20 , or learning ACE relations between words from Ch. 22. Here is a sketch of the algorithm as applied to question-answer relation extraction:

1. For a given relation between two terms (i.e. person-name→year-of-birth), we start with a hand-built list of correct pairs (e.g., "gandhi:1869", "mozart:1756", etc).
2. Now query the Web with instances of these pairs (e.g., "gandhi" and "1869", etc) and examine the top X returned documents.
3. Break each document into sentences, and keep only sentences containing both terms (e.g., PERSON-NAME and BIRTH-YEAR).
4. Extract a regular expression pattern representing the words and punctuation that occur between and around the two terms.

5. Keep all patterns that are sufficiently high-precision.

In Ch. 20 and Ch. 22 we discussed various ways to measure accuracy of the patterns. A method used in question-answer pattern matching is to keep patterns which are **high-precision**. Precision is measured by performing a query with only the question terms, but not the answer terms (i.e. query with just "gandhi" or "mozart"). We then run the resulting patterns on the sentences from the document, and extract a birth-date. Since we know the correct birth-date, we can compute the percentage of times this pattern produced a correct birthdate. This percentage is the precision of the pattern.

For the YEAR-OF-BIRTH answer type, this method learns patterns like the following:

```
<NAME> (<BD>-<DD>)
<NAME> (<BD>-<DD>),
<NAME> was born on <BD>
```

These two methods, named entity detection and question-answer pattern extraction, are still not sufficient for answer extraction. Not every relation is signaled by unambiguous surrounding words or punctuation, and often multiple instances of the same named-entity type occur in the answer passages. The most successful answer-extraction method is thus to combine all these methods, using them together with other information as features in a classifier that ranks candidate answers. We extract potential answers using named entities or patterns or even just looking at every sentence returned from passage retrieval, and rank them using a classifier with features like the following:

Answer type match: True if the candidate answer contains a phrase with the correct answer type.

Pattern match: The identity of a pattern that matches the candidate answer.

Number of matched question keywords: How many question keywords are contained in the candidate answer.

Keyword distance: The distance between the candidate answer and query keywords (measured in average number of words, or as the number of keywords that occur in the same syntactic phrase as the candidate answer.

Novelty factor: True if at least one word in the candidate answer is novel, i.e. not in the query.

Apposition features: True if the candidate answer is an appositive to a phrase containing many question terms. Can be approximated by the number of question terms separated from the candidate answer through at most three words and one comma Pasca (2003).

Punctuation location: True if the candidate answer is immediately followed by a comma, period, quotation marks, semicolon, or exclamation mark.

Sequences of question terms: The length of the longest sequence of question terms that occurs in the candidate answer.

N-gram tiling

An alternative approach to answer extraction, used solely in web search, is based on N**-gram tiling**, sometimes called the **redundancy-based approach** (Brill et al., 2002; Lin, 2007). This simplified method begins with the snippets returned from the web

N-gram mining

search engine, produced by a reformulated query. In the first step of the method, ***N*-gram mining**, every unigram, bigram, and trigram occurring in the snippet is extracted and weighted. The weight is a function of the number of snippets the *N*-gram occurred in, and the weight of the query reformulation pattern that returned it. In the ***N*-gram filtering** step, *N*-grams are scored by how well they match the predicted answer type. These scores are computed by hand-written filters built for each answer type. Finally, an ***N*-gram tiling** algorithm concatenates overlapping *N*-gram fragments into longer answers. A standard greedy method is to start with the highest-scoring candidate and try to tile each other candidate with this candidate. The best scoring concatenation is added to the set of candidates, the lower scoring candidate is removed, and the process continues until a single answer is built.

N-gram filtering

For any of these answer extraction methods, the exact answer phrase can just be presented to the user by itself. In practice, however, users are rarely satisfied with an unadorned number or noun as an answer; they prefer to see the answer accompanied by enough passage information to substantiate the answer. Thus we often give the user an entire passage with the exact answer inside it highlighted or boldfaced.

23.2.4 Evaluation of Factoid Answers

A wide variety of techniques have been employed to evaluate question answering systems. By far the most influential evaluation framework has been provided by the TREC Q/A track first introduced in 1999.

Mean reciprocal rank
MRR

The primary measure used in TREC is an **intrinsic** or **in vitro** evaluation metric known as **mean reciprocal rank**, or **MRR**. As with the ad hoc information retrieval task described in Sec. 23.1, MRR assumes a test set of questions that have been human-labeled with correct answers. MRR also assumes that systems are returning a short **ranked** list of answers, or passages containing answers. Each question is then scored based on the reciprocal of the **rank** of the first correct answer. For example if the system returned 5 answers but the first 3 are wrong and hence the highest-ranked correct answer is ranked 4, the reciprocal rank score for that question would be $\frac{1}{4}$. Questions with return sets that do not contain any correct answers are assigned a zero. The score of a system is then the average of the score for each question in the set. More formally, for an evaluation of a system returning M ranked answers for test set consisting of N questions, the MRR is defined as:

$$\text{MRR} = \frac{\sum_{i=1}^{N} \frac{1}{rank_i}}{N} \tag{23.22}$$

23.3 Summarization

The algorithms we have described so far in this chapter present the user an entire document (information retrieval), or a short factoid answer phrase (factoid question answering). But sometimes the user wants something that lies in between these extremes: something like a **summary** of a document or set of documents.

Text summarization

Text summarization is *the process of distilling the most important information from a text to produce an abridged version for a particular task and user* (definition adapted from Mani and Maybury (1999)). Important kinds of summaries that are the focus of current research include:

- **outlines** of any document
- **abstracts** of a scientific article
- **headlines** of a news article
- **snippets** summarizing a web page on a search engine results page
- **action items or other summaries** of a (spoken) business meeting
- **summaries** of email threads
- **compressed sentences** for producing simplified or compressed text
- **answers** to complex questions, constructed by summarizing multiple documents

These kinds of summarization goals are often characterized by their position on two dimensions:

- **single document** versus **multiple document** summarization
- **generic** summarization versus **query-focused** summarization

Single document summarization

In **single document summarization** we are given a single document and produce a summary. Single document summarization is thus used in situations like producing a headline or an outline, where the final goal is to characterize the content of a single document.

Multiple document summarization

In **multiple document summarization**, the input is a group of documents, and our goal is to produce a condensation of the content of the entire group. We might use multiple document summarization when we are summarizing a series of news stories on the same event, or whenever we have web content on the same topic that we'd like to synthesize and condense.

Generic summary

A **generic summary** is one in which we don't consider a particular user or a particular information need; the summary simply gives the important information in the document(s). By contrast, in **query-focused summarization**, also called **focused summarization**, **topic-based summarization** and **user-focused summarization**, the summary is produced in response to a user query. We can think of query-focused summarization as a kind of longer, non-factoid answer to a user question.

Query-focused summarization

In the remainder of this section we give a brief overview of the architecture of automatic text summarization systems; the following sections then give details.

One crucial architectural dimension for text summarizers is whether they are producing an **abstract** or an **extract**. The simplest kind of summary, an **extract**, is formed by selecting (**extracting**) phrases or sentences from the document to be summarized and pasting them together. By contrast, an **abstract** uses different words to describe the contents of the document. We'll illustrate the difference between an extract and an abstract using the well-known Gettysburg address, a famous speech by Abraham Lincoln, shown in Fig. 23.12.[1] Fig. 23.13 shows an extractive summary from the speech followed by an abstract of the speech.

Extract

Abstract

[1] In general one probably wouldn't need a summary of such a short speech, but a short text makes it easier to see how the extract maps to the original for pedagogical purposes. For an amusing alternative application of modern technology to the Gettysburg Address, see Norvig (2005).

Fourscore and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battle- field of that war. We have come to dedicate a portion of that field as a final resting-place for those who here gave their lives that this nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we cannot dedicate...we cannot consecrate...we cannot hallow... this ground. The brave men, living and dead, who struggled here, have consecrated it far above our poor power to add or detract. The world will little note nor long remember what we say here, but it can never forget what they did here. It is for us, the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us...that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion; that we here highly resolve that these dead shall not have died in vain; that this nation, under God, shall have a new birth of freedom; and that government of the people, by the people, for the people, shall not perish from the earth.

Figure 23.12 The Gettysburg Address. Abraham Lincoln, 1863.

Extract from the Gettysburg Address:

Four score and seven years ago our fathers brought forth upon this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation can long endure. We are met on a great battlefield of that war. We have come to dedicate a portion of that field. But the brave men, living and dead, who struggled here, have consecrated it far above our poor power to add or detract. From these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion — that government of the people, by the people for the people shall not perish from the earth.

Abstract of the Gettysburg Address:

This speech by Abraham Lincoln commemorates soldiers who laid down their lives in the Battle of Gettysburg. It reminds the troops that it is the future of freedom in America that they are fighting for.

Figure 23.13 An extract versus an abstract from the Gettysburg Address (abstract from Mani (2001)).

Most current text summarizers are extractive, since extraction is much easier than abstracting; the transition to more sophisticated abstractive summarization is a key goal of recent research.

Text summarization systems and, as it turns out, **natural language generation** systems as well, are generally described by their solutions to the following three problems:

1. **Content Selection:** What information to select from the document(s) we are summarizing. We usually make the simplifying assumption that the granularity of extraction is the sentence or clause. Content selection thus mainly consists of

choosing which sentences or clauses to extract into the summary.

2. **Information Ordering:** How to order and structure the extracted units.
3. **Sentence Realization:** What kind of clean up to perform on the extracted units so they are fluent in their new context.

In the next sections we'll show these components in three summarization tasks: **single document** summarization, **multiple document** summarization, and **query-focused summarization**.

23.3.1 Summarizing Single Documents

Let's first consider the task of building an extractive summary for a single document. Assuming that the units being extracted are at the level of the sentence, the three summarization stages for this task are:

1. **Content Selection:** Choose sentences to extract from the document
2. **Information Ordering:** Choose an order to place these sentences in the summary
3. **Sentence Realization:** Clean up the sentences, for example by removing non-essential phrases from each sentence, or fusing multiple sentences into a single sentence, or by fixing problems in coherence.

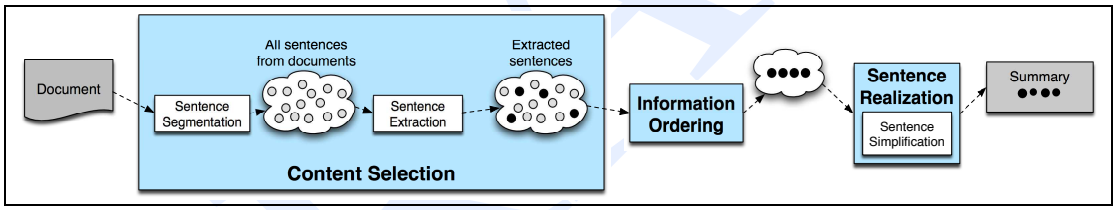

Figure 23.14 The basic architecture of a generic single document summarizer.

We'll first describe basic summarization techniques with only one of these components: *content selection*. Indeed, many single document summarizers have no information ordering component, simply ordering the extracted sentences in the order they appeared in the original document. In addition, we'll assume for now that sentences are not combined or cleaned up after they are extracted, although we'll briefly mention later how this is done.

Unsupervised Content Selection

Content selection

The **content selection** task of extracting sentences is often treated as a classification task. The goal of the classifier is to label each sentence in a document with a binary label: *important* versus *unimportant* (or *extract-worthy* versus *not extractworthy*). We begin with some unsupervised algorithms for sentence classification and then turn to supervised algorithms in the next section.

The simplest unsupervised algorithm, based on an intuition that dates back to the early summarizer of (Luhn, 1958), is to select sentences that have more **salient** or **informative** words. Sentences that contain more informative words tend to be more

Topic signature
Signature terms

extract-worthy. Saliency is usually defined by computing the **topic signature**, a set of **salient** or **signature terms**, each of whose saliency scores is greater than some threshold θ.

Saliency could be measured in terms of simple word frequency, but frequency has the problem that a word might have a high probability in English in general but not be particularly topical to a particular document. Therefore weighting schemes like **tf-idf** or **log-likelihood ratio** are more often used.

Recall from page 786 that the tf-idf scheme gives a high weight to words that appear frequently in the current document, but rarely in the overall document collection, suggesting that the word is particularly relevant to this document. For each term i that occurs in the sentence to be evaluated, we compute its count in the current document j $\text{tf}_{i,j}$, and multiply by the inverse document frequency over the whole collection idf_i:

$$weight(w_i) = \text{tf}_{i,j} \times \text{idf}_i \tag{23.23}$$

Log likelihood ratio

A better performing method for finding informative words is **log likelihood ratio** (LLR). The log likelihood ratio for a word, generally called $\lambda(w)$, is the ratio between the probability of observing w both in the input and in the background corpus assuming equal probabilities in both corpora, and the probability of observing w in both assuming different probabilities for w in the input and the background corpus. See Dunning (1993), Moore (2004)and Manning and Schütze (1999) for details on log likelihood and how it is calculated.

It turns out for log likelihood ratio that the quantity $-2\log(\lambda)$ is asymptotically well approximated by the χ^2 distribution, which means that a word appears in the input significantly more often than in the background corpus (at $\alpha = 0.001$) if $-2log(\lambda) > 10.8$. Lin and Hovy (2000) first suggested that this made log likelihood ratio particularly appropriate for selecting a topic signature for summarization. Thus the word weight with log likelihood ratio is generally defined as follows:

$$weight(w_i) = \begin{cases} 1 & \text{if } -2\log(\lambda(w_i)) > 10 \\ 0 & \text{otherwise.} \end{cases} \tag{23.24}$$

Eq. 23.24 is used to set a weight of 1 or 0 for each word in the sentence. The score for a sentence s_i is then the average weight of its non-stop words:

$$weight(s_i) = \sum_{w \in s_i} \frac{weight(w)}{|\{w | w \in s_i\}|} \tag{23.25}$$

The summarization algorithms computes this weight for every sentence, and then ranks all sentences by their score. The extracted summary consists of the top ranked sentences.

The family of algorithms that this thresholded LLR algorithm belongs to is called **centroid-based summarization** because we can view the set of signature terms as a pseudo-sentence which is the 'centroid' of all the sentences in the document and we are looking for sentences which are as close as possible to this centroid sentence.

Centrality

A common alternative to the log likelihood ratio/centroid method is to use a different model of sentence **centrality**. These other centrality based methods resemble the centroid method described above, in that their goal is to rank the input sentences

in terms of how central they are in representing the information present in the document. But rather than just ranking sentences by whether they contain salient words, centrality based methods compute distances between each candidate sentence and each other sentence and choose sentences that are on average closer to other sentences. To compute centrality, we can represent each sentence as a bag-of-words vector of length N as described in Ch. 20. For each pair of sentences x and y, we compute the tf-idf weighted cosine as described in Eq. 23.12 above.

Each of the k sentences in the input is then assigned a centrality score which is its average cosine with all other sentences:

$$centrality(x) = \frac{1}{K}\sum_{y} \textit{tf-idf-cosine}(x,y) \tag{23.26}$$

Sentences are ranked by this centrality score, and the sentence which has the highest average cosine across all pairs, i.e. is most like other sentences, is chosen as the most 'representative' or 'topical' of all the sentences in the input.

It is also possible to extend this centrality score to use more complex graph-based measures of centrality like PageRank (Erkan and Radev, 2004).

Unsupervised Summarization based on Rhetorical Parsing

The sentence extraction algorithm we introduced above for content extraction relied solely on a single shallow feature, word saliency, ignoring possible higher-level cues such as discourse information. In this section we briefly summarize a way to get more sophisticated discourse knowledge into the summarization task.

The summarization algorithm we'll describe makes use of **coherence relations** such as the RST (rhetorical structure theory) relations described in Ch. 21. Recall that RST relations are often expressed in terms of a **satellite** and a **nucleus**; nucleus sentence are more likely to be appropriate for a summary. For example, consider the following two paragraphs taken from the Scientific American magazine text that we introduced in Sec. 21.2.1:

> With its distant orbit – 50 percent farther from the sun than Earth – and slim atmospheric blanket, Mars experiences frigid weather conditions. Surface temperatures typically average about -70 degrees Fahrenheit at the equator, and can dip to -123 degrees C near the poles.
>
> Only the midday sun at tropical latitudes is warm enough to thaw ice on occasion, but any liquid water formed in this way would evaporate almost instantly because of the low atmospheric pressure. Although the atmosphere holds a small amount of water, and water-ice clouds sometimes develop, most Martian weather involves blowing dust or carbon dioxide.

The first two discourse units in this passage are related by the RST JUSTIFICATION relation, with the first discourse unit justifying the second unit, as shown in Fig. 23.15. The second unit (*"Mars experiences frigid weather conditions"*) is thus the nucleus, and captures better what this part of the document is about.

We can use this intuition for summarization by first applying a discourse parser of the type discussed in Ch. 21 to compute the coherence relations between each discourse

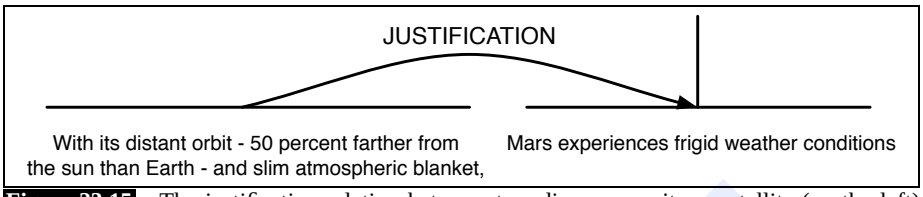

Figure 23.15 The justification relation between two discourse units, a satellite (on the left) and a nucleus (on the right).

unit. Once a sentence has been parsed into a coherence relation graph or parse tree, we can use the intuition that the nuclear units are important for summarization by recursively extracting the salient units of a text.

Consider the coherence parse tree in Fig. 23.16. The salience of each node in the tree can be defined recursively as follows:

- Base case: The salient unit of a leaf node is the leaf node itself
- Recursive case: The salient units of an intermediate node are the union of the salient units of its immediate *nuclear* children

By this definition, discourse unit (2) is the most salient unit of the entire text (since the root node spanning units 1-8 has the node spanning units 1-6 as its nucleus, and unit 2 is the nucleus of the node spanning units 1-6.)

If we rank each discourse unit by the height of the nodes that it is the nucleus of, we can assign a partial ordering of salience to units; the algorithm of Marcu (1995) assigns the following partial ordering to this discourse:

(23.27) $$2 > 8 > 3 > 1,4,5,7 > 6$$

See Marcu (1995, 2000b) for the details of exactly how this partial order is computed, and Teufel and Moens (2002) for another method for using rhetorical structure in summarization.

Supervised Content Selection

While the use of topic signatures for unsupervised content selection is an extremely effective method, topic signatures is only a single cue for finding extractworthy sentences. Many other cues exist, including the alternative saliency methods discussed above like centrality and PageRank mathods, as well as other cues like the position of the sentence in the document (sentences at the very beginning or end of the document tend to be more important), the length of each sentence, and so on. We'd like a method that can weigh and combine all of these cues.

The best principled method for weighing and combining evidence is supervised machine learning. For supervised machine learning, we'll need a training set of documents paired with human-created summary extracts, such as the Ziff-Davis corpus (Marcu, 1999). Since these are *extracts*, each sentence in the summary is, by definition, taken from the document. That means we can assign a label to every sentence in the document; **1** if it appears in the extract, **0** if it doesn't. To build our classifier, then, we just need to choose features to extract which are predictive of being a good sentence

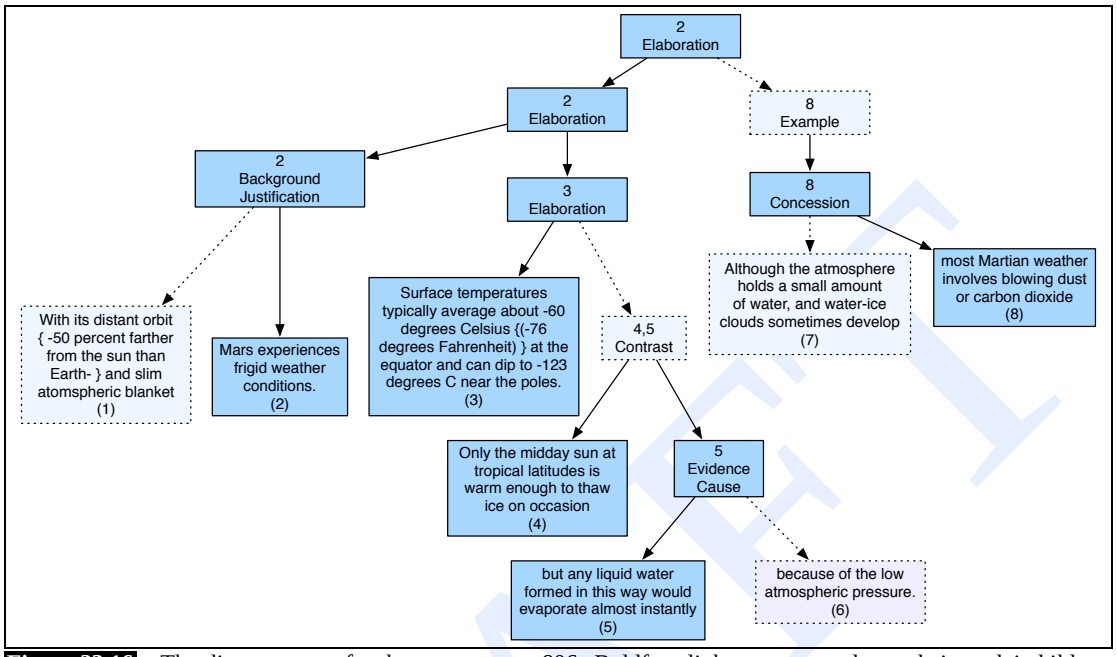

Figure 23.16 The discourse tree for the text on page 806. Boldface links connect nodes to their nuclei children; dotted lines to the satellite children. After Marcu (2000a).

to appear in a summary. Some of the features commonly used in sentence classification are shown in Fig. 23.17.

Each sentence in our training document thus has a label (0 if the sentence is not in the training summary for that document, 1 if it is) and set of extracted feature values like those in Fig. 23.17. We can then train our classifier to estimate these labels for unseen data; for example a probabilistic classifier like naive Bayes or MaxEnt would be computing the probability that a particular sentence s is extractworthy given a set of features $f_1 \ldots f_n$; then we can just extract any sentences for which this probability is greater than 0.5:

$$P(\text{extractworthy}(s) | f_1, f_2, f_3, ..., f_n) \tag{23.28}$$

There is one problem with the algorithm as we've described it: it requires that we have a training summary for each document which consists solely of extracted sentences. If we could weaken this restriction, we could apply the algorithm to a much wider variety of summary-document pairs, such as conference papers or journal articles and their abstracts. Luckily it turns out that when humans write summaries, even with the goal of writing abstractive summaries, they very often use phrases and sentences from the document to compose the summary. But they don't use *only* extracted sentences; they often combine two sentences into one, or change some of the words in the sentences, or write completely new abstractive sentences. Here is an example of an extracted sentence from a human summary that, although modified in the final human summary, was clearly a document sentence that should be labeled as extractworthy:

position	The position of the sentence in the document. For example Hovy and Lin (1999) found that the single most extract-worthy sentence in most newspaper articles is the title sentence. In the Ziff-Davis corpus they examined, the next most informative was the first sentence of paragraph 2 (P1S1), followed by the first sentence of paragraph 3 (P3S1); thus the list of ordinal sentence positions starting from the most informative was: T1, P2S1, P3S1, P4S1, P1S1, P2S2,... Position, like almost all summarization features, is heavily genre-dependent. In Wall Street Journal articles, they found the most important information appeared in the following sentences: T1, P1S1, P1S2,...
cue phrases	Sentences containing phrases like *in summary*, *in conclusion*, or *this paper* are more likely to be extract-worthy. These cue phrases are very dependent on the genre. For example in British House of Lords legal summaries, the phrase *it seems to me that* is a useful cue phrase. (Hachey and Grover, 2005).
word informativeness	Sentences that contain more terms from the **topic signature**, as described in the previous section, are more extractworthy.
sentence length	Very short sentences are rarely appropriate for extracting. We usually capture this fact by using a binary feature based on a cutoff (true if the sentence has more than, say, 5 words).
cohesion	Recall from Ch. 21 that a **lexical chain** is a series of related words that occurs throughout a discourse. Sentences which contain more terms from a lexical chain are often extractworthy because they are indicative of a continuing topic. (Barzilay and Elhadad, 1997). This kind of cohesion can also be computed by graph-based methods (Mani and Bloedorn, 1999). The PageRank graph-based measures of sentence centrality discussed above can also be viewed as a coherence metric (Erkan and Radev, 2004).

Figure 23.17 Some features commonly used in supervised classifiers for determining whether a document sentence should be extracted into a summary;

(23.29) **Human summary**: This paper identifies the desirable features of an ideal multisensor gas monitor and lists the different models currently available.

(23.30) **Original document sentence**: The present part lists the desirable features and the different models of portable, multisensor gas monitors currently available.

Alignment

Thus an important preliminary stage is to *align* each training document with its summary, with the goal of finding which sentences in the document were (completely or mostly) included in the summary. A simple algorithm for **alignment** is to find the source document and abstract sentences with the longest common subsequences of non-stopwords; alternatively minimum edit distance can be computed, or more sophisticated knowledge sources can be used, such as WordNet. Recent work has focused on more complex alignment algorithms such as the use of HMMs (Jing, 2002; Daumé III and Marcu, 2005, inter alia).

Given such alignment algorithms, supervised methods for content selection can

make use of parallel corpora of documents and human abstractive summaries, such as academic papers with their abstracts (Teufel and Moens, 2002).

Sentence Simplification

Sentence compression
Sentence simplification

Once a set of sentences has been extracted and ordered, the final step in single-document summarization is **sentence realization**. One component of sentence realization is **sentence compression** or **sentence simplification**. The following examples, taken by Jing (2000) from a human summary, show that the human summarizer chose to eliminate some of the adjective modifiers and subordinate clauses when expressing the extracted sentence in the summary:

(23.31) **Original sentence:** ~~When it arrives sometime new year in new TV sets,~~ the V-chip will give parents a ~~new and potentially revolutionary~~ device to block out programs they don't want their children to see.

(23.32) **Simplified sentence by humans:** The V-chip will give parents a device to block out programs they don't want their children to see.

The simplest algorithms for sentence simplification use rules to select parts of the sentence to prune or keep, often by running a parser or partial parser over the sentences. Some representative rules from Zajic et al. (2007), Conroy et al. (2006), and Vanderwende et al. (2007a) remove the following:

appositives	Rajam, ~~28, an artist who was living at the time in Philadelphia,~~ found the inspiration in the back of city magazines.
attribution clauses	Rebels agreed to talks with government officials, ~~international observers said Tuesday.~~
PPs without named entities	The commercial fishing restrictions in Washington will not be lifted [SBAR unless the salmon population 329 increases [PP ~~to a sustainable number~~]
initial adverbials	"For example", "On the other hand", "As a matter of fact", "At this point"

More sophisticated models of sentence compression are based on supervised machine learning, in which a parallel corpus of documents together with their human summaries is used to compute the probability that particular words or parse nodes will be pruned. See the end of the chapter for pointers to this extensive recent literature.

23.4 Multi-Document Summarization

Multi-document summarization

When we apply summarization techniques to groups of documents rather than a single document we call the goal **multi-document summarization**. Multi-document summarization is particularly appropriate for web-based applications, for example for building summaries of a particular event in the news by combining information from different news stories, or finding answers to complex questions by including components from extracted from multiple documents.

While multi-document summarization is far from a solved problem, even the current technology can be useful for information-finding tasks. McKeown et al. (2005),

for example, gave human experimental participants documents together with a human summary, an automatically generated summary, or no summary, and had the participants perform time-restricted fact-gathering tasks. The participants had to answer three related questions about an event in the news; subjects who read the automatic summaries gave higher-quality answers to the questions.

Multi-document summarization algorithms are based on the same three steps we've seen before. In many cases we assume that we start with a cluster of documents that we'd like to summarize, and we must then perform **content selection**, **information ordering**, and **sentence realization**, as described in the next three sections and sketched in Fig. 23.18

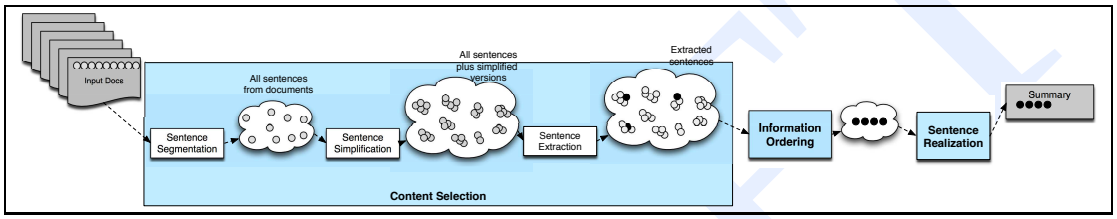

Figure 23.18 The basic architecture of a multi-document summarizer.

23.4.1 Content Selection in Multi-Document Summarization

In single document summarization we used both supervised and unsupervised methods for content selection. For multiple document summarization supervised training sets are less available, and we focus more on unsupervised methods.

The major difference between the tasks of single document and multiple document summarization is the greater amount of **redundancy** when we start with multiple documents. A group of documents can have significant overlap in words, phrases, and concepts, in addition to information that might be unique to each article. While we want each sentence in the summary to be about the topic, we don't want the summary to consist of a set of identical sentences.

For this reason, algorithms for multi-document summarization focus on ways to avoid redundancy when selected sentences for the summary. When adding a new sentence to a list of extracted sentences we need some way to make sure the sentence doesn't overlap too much with the already-extracted sentences.

A simple method of avoiding redundancy is to explicitly include a redundancy factor in the scoring for choosing a sentence to extract. The redundancy factor is based on the similarity between a candidate sentence and the sentences that have already been extracted into the summary; a sentence is penalized if it is too similar to the summary. For example the **MMR** or **Maximal Marginal Relevance** scoring system Carbonell and Goldstein (1998), Goldstein et al. (2000) includes the following penalization term for representing the similarity between a sentence s and the set of sentences already extracted for the summary *Summary*, where λ is a weight that can be tuned and Sim is some similarity function:

MMR
Maximal Marginal Relevance

$$\text{MMR penalization factor}(s) = \lambda max_{s_i \in Summary} \text{Sim}(s, s_i) \tag{23.33}$$

An alternative to MMR-based method is to instead apply a clustering algorithm to all the sentences in the documents to be summarized to produce a number of clusters of related sentences and then to select a single (centroid) sentence from each cluster into the summary.

By adding MMR or clustering methods for avoiding redundancy, we can also do sentence simplification or compression at the content selection stage rather than at the sentence realization stage. A common way to fit simplification into the architecture is to run various sentence simplification rules (Sec. 23.3.1) on each sentence in the input corpus. The result will be multiple versions of the input sentence, each version with different amounts of simplification. For example, the following sentence:

> Former Democratic National Committee finance director Richard Sullivan faced more pointed questioning from Republicans during his second day on the witness stand in the Senate's fund-raising investigation.

might produce different shortened versions:

- Richard Sullivan faced pointed questioning.
- Richard Sullivan faced pointed questioning from Republicans
- Richard Sullivan faced pointed questioning from Republicans during day on stand in Senate fundraising investigation
- Richard Sullivan faced pointed questioning from Republicans in Senate fundraising investigation

This expanded corpus is now used as the input to content extraction. Redundancy methods such as clustering or MMR will choose only the (optimally long) single version of each original sentence.

23.4.2 Information Ordering in Multi-Document Summarization

The second stage of an extractive summarizer is the ordering or structuring of information, where we must decide how to concatenate the extracted sentences into a coherent order. Recall that in single document summarization, we can just use the original article ordering for these sentences. This isn't appropriate for most multiple document applications, although we can certainly apply it if many or all of the extracted sentences happen to come from a single article.

Chronological ordering

For sentences extracted from news stories, one technique is to use the dates associated with the story, a strategy known as **chronological ordering**. It turns out that pure chronological ordering can produce summaries which lack cohesion; this problem can be addressed by ordering slightly larger chunks of sentences rather than single sentences; see Barzilay et al. (2002).

Perhaps the most important factor for information ordering, however, is **coherence**. Recall from Ch. 21 the various devices that contribute to the coherence of a discourse. One is having sensible coherence relations between the sentences; thus we could prefer orderings in summaries that resulting in sensible coherence relations between the sentences. Another aspect of coherence has to do with cohesion and lexical chains; we could for example prefer orderings which have more local cohesion. A final aspect of coherence is coreference; a coherence discourse is one in which entities are mentioned in coherent patterns. We could prefer orderings with coherent entity mention patterns.

All of these kinds of coherence have been used for information ordering. For example we can use *lexical cohesion* as an ordering heuristic by ordering each sentence next to sentences containing similar words. This can done by defining the standard tf-idf cosine distance between each pair of sentences and choosing the overall ordering that minimizes the average distance between neighboring sentences Conroy et al. (2006), or by building models of predictable word sequences across sentences (Soricut and Marcu, 2006).

Coreference-based coherence algorithms have also made use of the intuitions of **Centering**. Recall that the Centering algorithm was based on the idea that each discourse segment has a salient entity, the *focus*. Centering theory proposed that certain syntactic realizations of the focus (i.e. as subject or object) and certain transitions between these realizations (e.g., if the same entity is the subject of adjacent sentences) created a more coherent discourse. Thus we can prefer orderings in which the transition between entity mentions is a preferred one.

For example in the entity-based information approach of Barzilay and Lapata (2005, 2007), a training set of summaries is parsed and labeled for coreference. The resulting sequence of entity realizations can be automatically extracted and represented into an *Entity grid* **entity grid**. Fig. 23.19 shows a simplified version of a parsed summary and the extracted grid. A probabilistic model of particular entity transitions (i.e. $\{S,O,X,-\}$ can then be trained from the entity grid. For example the transitions $\{X,O,S,S\}$ for the head word *Microsoft* exemplify the fact that new entities in a discourse are often introduced first in oblique or object position and then only later appear in subject position. See Barzilay and Lapata (2007) for details.

A general way to view all of these methods is as assigning a coherence score to a sequence of sentences via a local coherence score between pairs or sequences of sentences; a single general transition score between sentences could then combine lexical coherence and entity-based coherence. Once we have such a scoring function, choosing an ordering which optimizes all these local pairwise distances is known to be quite difficult. The task of finding the optimal ordering of a set of sentences given a set of pairwise distances between the sentences is equivalent to very hard problems like Cyclic Ordering and the Traveling Salesman Problem.[2] Sentence ordering is thus equivalent to the difficult class of problems known as **NP-complete**. While difficult to solve exactly, there are a number of good approximation methods for solving NP-complete problems that have been applied to the information ordering task. See Althaus et al. (2004), Knight (1999a), Cohen et al. (1999), Brew (1992) for the relevant proofs and approximation techniques.

In the models described above, the information ordering task is completely separate from content extraction. An alternative approach is to learn the two tasks jointly, resulting in a model that both selects sentences and orders them. For example in the HMM model of Barzilay and Lee (2004), the hidden states correspond to document content topics and the observations to sentences. For example for newspaper articles on earthquakes, the hidden states (topics) might be *strength of earthquake*, *location*, *rescue efforts*, and *casualties*. They apply clustering and HMM induction to induce

[2] The Traveling Salesman Problem: given a set of cities and the pairwise distances between them, find the shortest path that visits each city exactly once.

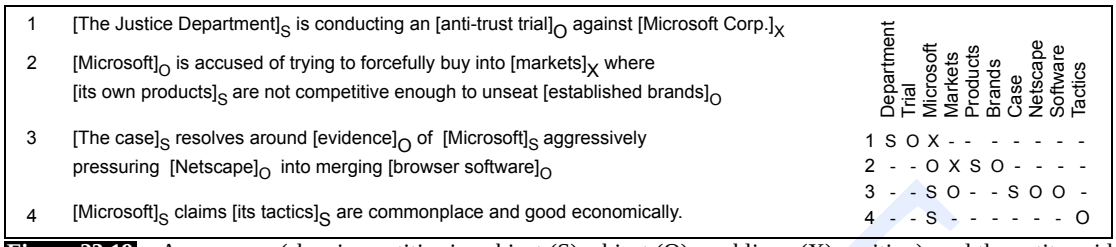

Figure 23.19 A summary (showing entities in subject (S), object (O) or oblique (X) position), and the entity grid that is extracted from it. Adapted from Barzilay and Lapata (2005).

these hidden states and the transitions between them. For example, here are three sentences from the *location* cluster they induce:

(23.34) The Athens seismological institute said the temblor's epicenter was located 380 kilometers (238 miles) south of the capital.

(23.35) Seismologists in Pakistan's Northwest Frontier Province said the temblor's epicenter was about 250 kilometers (155 miles) north of the provincial capital Peshawar.

(23.36) The temblor was centered 60 kilometers (35 miles) northwest of the provincial capital of Kunming, about 2,200 kilometers (1,300 miles) southwest of Beijing, a bureau seismologist said.

The learned structure of the HMM then implicitly represent information ordering facts like *mention 'casualties' prior to 'rescue efforts'* via the HMM transition probabilities.

In summary, we've seen information ordering based on **chronological order**, based on **coherence**, and an ordering that is learned automatically from the data. In the next section on query-focused summarization we'll introduce a final method in which information ordering can be specified according to an ordering template which is predefined advance for different query types.

Sentence Realization

While discourse coherence can be factored in during sentence ordering, the resulting sentences may still have coherence problems. For example, as we saw in Ch. 21, when a referent appears multiple times in a coreference chain in a discourse, the longer or more descriptive noun phrases occur before shorter, reduced, or pronominal forms. But the ordering we choose for the extracted sentences may not respect this coherence preference.

For example the boldfaced names in the original summary in Fig. 23.20 appear in an incoherent order; the full name **U.S. President George W. Bush** occurs only after the shortened form **Bush** has been introduced.

One possible way to address this problem in the sentence realization stage is to apply a coreference resolution algorithm to the output, extracting names and applying some simple cleanup rewrite rules like the following:

(23.37) Use the **full name** at the first mention, and just the **last name** at subsequent mentions.

(23.38) Use a **modified** form for the first mention, but remove appositives or premodifiers from any subsequent mentions.

The rewritten summary in Fig. 23.20 shows how such rules would apply; in general such methods would depend on high-accuracy coreference resolution.

Original summary:
Presidential advisers do not blame **O'Neill**, but they've long recognized that a shakeup of the economic team would help indicate **Bush** was doing everything he could to improve matters. **U.S. President George W. Bush** pushed out **Treasury Secretary Paul O'Neill** and top economic adviser Lawrence Lindsey on Friday, launching the first shake - up of his administration to tackle the ailing economy before the 2004 election campaign.
Rewritten summary:
Presidential advisers do not blame **Treasury Secretary Paul O'Neill**, but they've long recognized that a shakeup of the economic team would help indicate **U.S. President George W. Bush** was doing everything he could to improve matters. **Bush** pushed out **O'Neill** and White House economic adviser Lawrence Lindsey on Friday, launching the first shake-up of his administration to tackle the ailing economy before the 2004 election campaign.

Figure 23.20 Rewriting references, from Nenkova and McKeown (2003)

Sentence fusion

Recent research has also focused on a finer granularity for realization than the extracted sentence, by using **sentence fusion** algorithms to combine phrases or clauses from different sentences into one new sentence. The sentence fusion algorithm of Barzilay and McKeown (2005) parses each sentence, uses multiple-sequence alignment of the parses to find areas of common information, builds a fusion lattice with overlapping information, and creates a fused sentence by linearizing a string of words from the lattice.

23.5 Question Answering Meets Summarization: Focused Summarization

As noted in at the beginning of this chapter, most interesting questions are not factoid questions. User needs require longer, more informative answers than a single phrase can provide. For example, while a DEFINITION question might be answered by a short phrase like "*Autism is* **a developmental disorder**" or "*A caldera is* **a volcanic crater**", a user might want more information, as in the following definition of *water spinach*:

> *Water spinach* (ipomoea aquatica) is a semi-aquatic leafy green plant characterized by long hollow stems and spear-shaped or heart-shaped leaves which is widely grown throughout Asia as a leaf vegetable. The leaves and stems are often eaten stir-fried as greens with salt or salty sauces, or in soups. Other common names include *morning glory vegetable*, *kangkong* (Malay), *rau muong* (Vietnamese), *ong choi* (Cantonese), and *kong xin cai*

(Mandarin). It is not related to spinach, but is closely related to sweet potato and convolvulus.

Complex questions can also be asked in domains like medicine, such as this question about a particular drug intervention:

(23.39) In children with an acute febrile illness, what is the efficacy of single-medication therapy with acetaminophen or ibuprofen in reducing fever?

For this medical question, we'd like to be able to extract an answer of the following type, perhaps giving the document id(s) that the extract came from, and some estimate of our confidence in the result:

> Ibuprofen provided greater temperature decrement and longer duration of antipyresis than acetaminophen when the two drugs were administered in approximately equal doses. (PubMedID: 1621668, Evidence Strength: A)

Questions can be even more complex, such as this one from the Document Understanding Conference annual summarization competition:

(23.40) Where have poachers endangered wildlife, what wildlife has been endangered and what steps have been taken to prevent poaching?

Where a factoid answer might be found in a single phrase in a single document or web page, these kinds of complex questions are likely to require much longer answers which are synthesized from many documents or pages.

For this reason, summarization techniques are often used to build answers to these kinds of complex questions. But unlike the summarization algorithms introduced above, the summaries produced for complex question answering must be relevant to some user question. When a document is summarized for the purpose of answering some user query or information need, we call the goal **query-focused summarization** or sometimes just **focused summarization**. (The terms **topic-based summarization** and **user-focused summarization** are also used.) A query-focused summary is thus really a kind of longer, non-factoid answer to a user question or information need.

Query-focused summarization

Snippet

One kind of query-focused summary is a **snippet**, the kind that web search engines like Google return to the user to describe each retrieved document. Snippets are query-focused summaries of a single document. But since for complex queries we will want to aggregate information from multiple documents, we'll need to summarize multiple documents.

Indeed, the simplest way to do query-focused summarization is to slightly modify the algorithms for multiple document summarization that we introduced in the previous section to make use of the query. For example, when ranking sentences from all the returned documents in the content selection phase, we can require that any extracted sentence must contain at least one word overlapping with the query. Or we can just add the cosine distance from the query as one of the relevance features in sentence extraction. We can characterize such a method of query-focused summarization as a bottom-up, domain-independent method.

An alternative way to do query-focused summarization is to make additional use of top-down or information-extraction techniques, building specific content selection algorithms for different types of complex questions. Thus we could specifically build a

query-focused summarizer for the kinds of advanced questions introduced above, like definition questions, biography questions, certain medical questions. In each case, we use our top-down expectations for what makes a good definition, biography, or medical answer to guide what kinds of sentences we extract.

Genus
Species

For example, a **definition** of a term often includes information about the term's **genus** and **species**. The genus is the hypernym or superordinate of the word; thus a sentence like *The Hajj is a type of ritual* is a genus sentence. The species gives important additional properties of the term that differentiate the term from other hyponyms of the genus; an example is *"The annual hajj begins in the twelfth month of the Islamic year"*. Other kinds of information that can occur in a definition include **synonyms**, **etymology**, **subtypes**, and so on.

In order to build extractive answers for definition questions, we'll need to make sure we extract sentences with the genus information, the species information, and other generally informative sentences. Similarly, a good **biography** of a person contains information such as the person's **birth/death**, **fame factor**, **education**, **nationality** and so on; we'll need to extract sentences with each of these kinds of information. A medical answer that summarizes the results of a study on applying a drug to a medical problem would need to contain information like the **problem** (the medical condition), the **intervention** (the drug or procedure), and the **outcome** (the result of the study).

Fig. 23.21 shows some example predicates for definition, biography, and medical intervention questions.

Definition	
genus	`The Hajj is a type of ritual`
species	`the annual hajj begins in the twelfth month of the Islamic year`
synonym	`The Hajj, or Pilgrimage to Mecca, is the central duty of Islam`
subtype	`Qiran, Tamattu', and Ifrad are three different types of Hajj`
Biography	
dates	`was assassinated on April 4, 1968`
nationality	`was born in Atlanta, Georgia`
education	`entered Boston University as a doctoral student`
Drug efficacy	
population	`37 otherwise healthy children aged 2 to 12 years`
problem	`acute, intercurrent, febrile illness`
intervention	`acetaminophen (10 mg/kg)`
outcome	`ibuprofen provided greater temperature decrement and longer duration of antipyresis than acetaminophen when the two drugs were administered in approximately equal doses`

Figure 23.21 Examples of some different types of information that must be extracted in order to produce answer to certain kinds of complex questions.

In each case we use the **information extraction** methods of Ch. 22 to find specific sentences for genus and species (for definitions), or dates, nationality, and education (for biographies), or problems, interventions and outcomes (for medical questions). We can then use standard domain-independent content selection algorithms to find other good sentences to add on to these.

A typical architecture consists of the four steps shown in Fig. 23.22 from the definition extraction system of Blair-Goldensohn et al. (2004). The input is a definition question T, the number N of documents to retrieve, and the length L of the answer (in sentences).

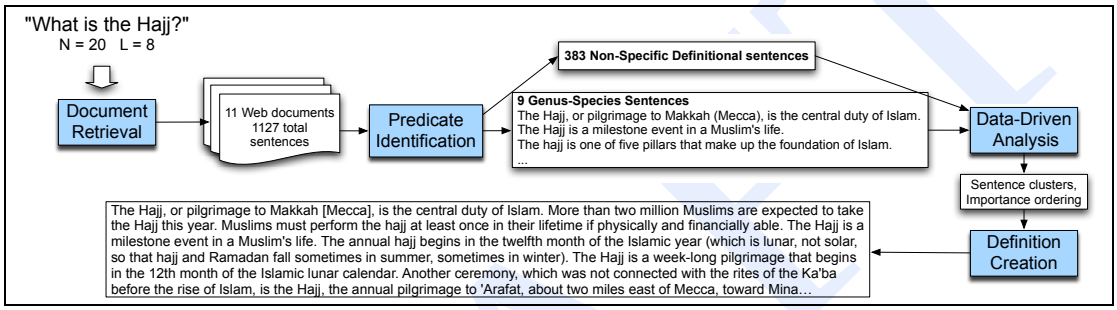

Figure 23.22 Architecture of a query-focused summarizer for definition questions (Blair-Goldensohn et al., 2004).

The first step in any IE-based complex question answering system is information retrieval. In this case a handwritten set of patterns is used to extract the term to be defined from the query T (*Hajj*) and generate a series of queries that are sent to an IR engine. Similarly, in a biography system it would be the name that would be extracted and passed to the IR engine. The returned documents are broken up into sentences.

In the second stage, we apply classifiers to label each sentence with an appropriate set of classes for the domain. For definition questions, Blair-Goldensohn et al. (2004) used of four classes: **genus**, **species**, **other definitional**, or **other**. The third class, **other definitional**, is used to select other sentences that might be added into the summary. These classifiers can be based on any of the information extraction techniques introduced in Ch. 22, including hand-written rules, or supervised machine learning.

In the third stage, we can use the methods described in the section on generic (non-query-focused) multiple domain summarization content selection to add additional sentences to our answer that might not fall into a specific information extraction type. For example for definition questions, all the sentences that are classified as *other definitional* are examined, and a set of relevant sentences is selected from them. This selection can be done by the centroid method, in which we form a TF-IDF vector for each sentence, find the centroid of all the vectors, and then choose the K sentences closest to the centroid. Alternatively we can use a method for avoiding redundancy, like clustering the vectors and choosing the best sentence from each cluster.

Because query-focused summarizers of this type or domain-specific, we can use domain-specific methods for information ordering as well, such as using a fixed hand-built template. For biography questions we might use a template like the following:

(23.41) <NAME> is <WHY FAMOUS>. She was born on <BIRTHDATE> in <BIRTHLOCATION>. She <EDUCATION>. <DESCRIPTIVE SENTENCE>. <DESCRIPTIVE SENTENCE>.

The various sentences or phrases selected in the content selection phase can then be fit into this template. These templates can also be somewhat more abstract. For example, for definitions, we could place a genus-species sentence first, followed by remaining sentences ordered by their saliency scores.

23.6 Summarization Evaluation

As is true for other speech and language processing areas like machine translation, there are a wide variety of evaluation metrics for summarization, metrics requiring human annotation, as well as completely automatic metrics.[3]

As we have seen for other tasks, we can evaluate a system via **extrinsic** (task-based) or **intrinsic** (task-independent) methods. We described a kind of extrinsic evaluation of multi-document summarization in Sec. 23.4, in which subjects were asked to perform time-restricted fact-gathering tasks, and were given full documents together with either no summaries, human summaries, or automatically generated summaries to read. The subjects had to answer three related questions about an event in the news. For query-focused single-document summarization (like the task of generating web **snippets**), we can measure how different summarization algorithms affect human performance at the task of deciding if a document is relevant/not-relevant to a query by looking solely at the summary.

ROUGE

The most common intrinsic summarization evaluation metric is an automatic method called **ROUGE**, **Recall-Oriented Understudy for Gisting Evaluation** (Lin and Hovy, 2003; Lin, 2004). ROUGE is inspired by the BLEU metric used for evaluating machine translation output, and like BLEU, automatically scores a machine-generated candidate summary by measuring the amount of N-gram overlap between the candidate and human-generated summaries (the references).

ROUGE-1
ROUGE-2

Recall that BLEU is computed by averaging the number of overlapping N-grams of different length between the hypothesis and reference translations. In ROUGE, by contrast, the length of the N-gram is fixed; **ROUGE-1** uses unigram overlap, while **ROUGE-2** uses bigram overlap. We'll choose to define ROUGE-2; the definitions of all the other ROUGE-N metrics follows. ROUGE-2 is a measure of the bigram recall between the candidate summary and the set of human reference summaries:

$$ROUGE2 = \frac{\sum_{S \in \{ReferenceSummaries\}} \sum_{bigram \in S} \text{Count}_{\text{match}}(bigram)}{\sum_{S \in \{ReferenceSummaries\}} \sum_{bigram \in S} \text{Count}(bigram)} \tag{23.42}$$

[3] We focus here on evaluation of entire summarization algorithms and ignore evaluation of subcomponents such as information ordering, although see for example (Lapata, 2006) on the use of Kendall's τ, a metric of rank correlation, for information ordering.

The function $\text{Count}_{\text{match}}(bigram)$ returns the maximum number of bigrams that co-occur in the candidate summary and the set of reference summaries. ROUGE-1 is the same but counting unigrams instead of bigrams.

Note that ROUGE is a recall-oriented measure, where BLEU is a precision-oriented measure. This is because the denominator of (23.42) is the total sum of the number of bigrams in the reference summaries. By contrast, in BLEU the denominator is the total sum of the number of N-grams in the candidates. Thus ROUGE is measuring something like how many of the human reference summary bigrams are covered by the candidate summary, where BLEU is measuring something like how many of the candidate translation bigrams occurred in the human reference translations.

ROUGE-L
ROUGE-S
ROUGE-SU
Skip bigram

Variants of ROUGE include **ROUGE-L**, which measure the **longest common subsequence** between the reference and candidate summaries, and **ROUGE-S** and **ROUGE-SU** which measure the number of **skip bigrams** between the reference and candidate summaries. A skip bigram is a pair of words in their sentence order, but allowing for any number of other words to appear between the pair.

While ROUGE is the most commonly applied automatic baseline, it is not as applicable to summarization as similar metrics like BLEU are to machine translation. This is because human summarizers seem to disagree strongly about which sentences to include in a summary, making even the overlap of humans with each other very low.

Pyramid Method

This difference in which sentences humans choose to extract has motivated human evaluation methods which attempt to focus more on meaning. One metric, the **Pyramid Method**, is a way of measuring how many units of meaning are shared between the candidate and reference summaries, and also weights the units of meaning by importance; units of meaning which occur in more of the human summaries are weighted more highly. The units of meaning are called **Summary Content Units** (SCU), which are sub-sentential semantic units which roughly correspond to propositions or coherent pieces of propositions.

Summary Content Units

In the Pyramid Method, humans label the Summary Content Units in each reference and candidate summary, and then an overlap measure is computed.

Let's see an example from Nenkova et al. (2007) of how two SCUs are labeled in sentences from six human abstracts. We'll first show sentences from the human summaries indexed by a letter (corresponding to one of the 6 human summaries) and a number (the position of the sentence in the human summary):

A1. The industrial espionage case involving GM and VW began with the hiring of Jose Ignacio Lopez, an employee of GM subsidiary Adam Opel, by VW as a production director.

B3. However, he left GM for VW under circumstances, which along with ensuing events, were described by a German judge as "potentially the biggest-ever case of industrial espionage".

C6. He left GM for VW *in March 1993*.

D6. The issue stems from the alleged recruitment of GM's eccentric and visionary Basque-born procurement chief Jose Ignacio Lopez de Arriortura and seven of Lopez's business colleagues.

E1. *On March 16, 1993*, with Japanese car import quotas to Europe expiring in two years, renowned cost-cutter, Agnacio Lopez De Arriortura, left his job as head

of purchasing at General Motor's Opel, Germany, to become Volkswagen's Purchasing and Production director.

F3. *In March 1993*, Lopez and seven other GM executives moved to VW overnight.

The annotators first identify similar sentences, like those above, and then label SCUs. The underlined and italicized spans of words in the above sentences result in the following two SCUs, each one with a weight corresponding to the number of summaries it appears in (6 for the first SCU, and 3 for the second):

SCU1 (w=6): *Lopez left GM for VW*
A1. the hiring of Jose Ignacio Lopez, an employee of GM . . . by VW
B3. he left GM for VW
C6. He left GM for VW
D6. recruitment of GMs . . . Jose Ignacio Lopez
E1. Agnacio Lopez De Arriortura, left his job . . . at General Motors Opel . . . to become Volkswagens . . . director
F3. Lopez . . . GM . . . moved to VW

SCU2 (w=3) *Lopez changes employers in March 1993*
C6. in March, 1993
E1. On March 16, 1993
F3. In March 1993

Once the annotation is done, the informativeness of a given summary can be measured as the ratio of the sum of the weights of its SCUs to the weight of an optimal summary with the same number of SCUs. See the end of the chapter for more details and pointers to the literature.

Random sentence baseline
Leading sentence baseline

The standard baselines for evaluating summaries are the **random sentences** baseline and the **leading sentences** baseline. Assuming we are evaluating summaries of length N sentences, the random baseline just chooses N random sentences, while the leading baseline chooses the first N sentences. The leading sentences method is quite a strong baseline and many proposed summarization algorithms fail to beat it.

23.7 Summary

- The dominant models of information retrieval represent the meanings of documents and queries as bags of words.
- The **vector space model** views documents and queries as vectors in a large multidimensional space. In this model, the similarity between documents and queries, or other documents, can be measured by the cosine of the angle between the vectors.
- The main components of a factoid question answering system are the **question classification** module to determine the named-entity type of the answer, a **passage retrieval** module to identify relevant passages, and an answer processing module to extract and format the final answer.

- Factoid question answers can be evaluated via **mean reciprocal rank** (**MRR**).
- Summarization can be **abstractive** or **extractive**; most current algorithms are extractive.
- Three components of **summarization algorithms** include **content selection**, **information ordering**, and **sentence realization**.
- Current single document summarization algorithms focus mainly on **sentence extraction**, relying on features like **position** in the discourse, **word informativeness**, **cue phrases**, and **sentence length**.
- Multiple document summarization algorithms often perform **sentence simplification** on document sentences.
- **Redundancy avoidance** is important in multiple document summarization; it is often implemented by adding a redundancy penalization term like **MMR** into sentence extraction.
- **Information ordering** algorithms in multi-document summarization are often based on maintaining **coherence**.
- **Query-focused summarization** can be done using slight modifications to **generic summarization** algorithms, or by using information-extraction methods.

Bibliographical and Historical Notes

Luhn (1957) is generally credited with first advancing the notion of fully automatic indexing of documents based on their contents. Over the years Salton's SMART project (Salton, 1971) at Cornell developed or evaluated many of the most important notions in information retrieval including the vector model, term weighting schemes, relevance feedback, and the use of cosine as a similarity metric. The notion of using inverse document frequency in term weighting is due to Sparck Jones (1972). The original notion of relevance feedback is due to Rocchio (1971).

Probabilistic IR

An alternative to the vector model that we have not covered is the **probabilistic model** originally shown effective by Robinson and Sparck Jones (1976). See Crestani et al. (1998) and Chapter 11 of Manning et al. (2008) on probabilistic models in information retrieval.

Manning et al. (2008) is a comprehensive modern text on information retrieval. Good but slightly older texts include Baeza-Yates and Ribeiro-Neto (1999) and Frakes and Baeza-Yates (1992); older classic texts include Salton and McGill (1983) and van Rijsbergen (1975). Many of the classic papers in the field can be found in Sparck Jones and Willett (1997). Current work is published in the annual proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR). The US National Institute of Standards and Technology (NIST) has run an annual evaluation project for text information retrieval and extraction called the Text REtrieval Conference (TREC) since the early 1990s; the conference proceedings from TREC contain results from these standardized evaluations. The primary journals in the field are the *Journal of the American*

Society of Information Sciences, *ACM Transactions on Information Systems*, *Information Processing and Management*, and *Information Retrieval*.

Question answering was one of the earliest tasks for NLP systems in the 1960's and 1970's (Green et al., 1961; Simmons, 1965; Woods et al., 1972b; Lehnert, 1977), but the field lay dormant for a few decades until the need for querying the Web brought the task back into focus. The U.S. government-sponsored TREC (Text REtrieval Conference) QA track began in 1999 and a wide variety of factoid and non-factoid systems have been competing in annual evaluations since then. See the references in the chapter and Strzalkowski and Harabagiu (2006) for a collection of recent research papers.

Research on text summarization began with the work of Luhn (1958) on extractive methods for the automatic generation of abstracts, focusing on surface features like term frequency, and the later work of Edmunson (1969) incorporating positional features as well. Term-based features were also used in the early application of automatic summarization at Chemical Abstracts Service (Pollock and Zamora, 1975). The 1970s and 1980s saw a number of approaches grounded in AI methodology such as scripts DeJong (1982b), semantic networks Reimer and Hahn (1988), or combinations of AI and statistical methods Rau et al. (1989).

The work of Kupiec et al. (1995) on training a sentence classifier with supervised machine learning led to many statistical methods for sentence extraction. Around the turn of the century, the growth of the Web led naturally to interest in multi-document summarization and query-focused summarization.

SumBasic
Centroid

There have naturally been a wide variety of algorithms for the main components of summarizers. The simple unsupervised log-linear content selection algorithm we describe is simplified from the **SumBasic** algorithm of Nenkova and Vanderwende (2005), Vanderwende et al. (2007b) and the **centroid** algorithm of Radev et al. (2000) and Radev et al. (2001). A number of algorithms for information ordering have used entity coherence, including Kibble and Power (2000), Lapata (2003), Karamanis and Manurung (2002), Karamanis (2003), Barzilay and Lapata (2005, 2007). Algorithms for combining multiple cues for coherence and searching for the optimal ordering include Althaus et al. (2004), based on linear programming, the genetic algorithms of Mellish et al. (1998) and Karamanis and Manurung (2002), and the Soricut and Marcu (2006) algorithm, which uses A* search based on IDL-expressions. Karamanis (2007) showed that adding coherence based on rhetorical relations to entity coherence didn't improve sentence ordering. See Lapata (2006, 2003), Karamanis et al. (2004), Karamanis (2006) on methods for evaluating information ordering.

Sentence compression is a very popular area of research. Early algorithms focused on the use of syntactic knowledge for eliminating less important words or phrases Grefenstette (1998), Mani et al. (1999), Jing (2000). Recent research has focused on using supervised machine learning, in which a parallel corpus of documents together with their human summaries is used to compute the probability that particular words or parse nodes will be pruned. Methods include the use of maximum entropy Riezler et al. (2003), the noisy channel model and synchronous context-free grammars (Galley and McKeown, 2007; Knight and Marcu, 2000; Turner and Charniak, 2005; Daumé III and Marcu, 2002), Integer Linear Programming Clarke and Lapata (2007), and large-margin learning McDonald (2006). These methods rely on various features, especially including syntactic or parse knowledge Jing (2000), Dorr et al. (2003), Sid-

dharthan et al. (2004), Galley and McKeown (2007), Zajic et al. (2007), Conroy et al. (2006), Vanderwende et al. (2007a), but also including coherence information Clarke and Lapata (2007). Alternative recent methods are able to function without these kinds of parallel document/summary corpora (Hori and Furui, 2004; Turner and Charniak, 2005; Clarke and Lapata, 2006).

See Daumé III and Marcu (2006) for a recent Bayesian model of query-focused summarization.

For more information on summarization evaluation, see Nenkova et al. (2007), Passonneau et al. (2005), and Passonneau (2006) for details on the Pyramid method, van Halteren and Teufel (2003) and Teufel and van Halteren (2004) on related semantic-coverage evaluation methods, and Lin and Demner-Fushman (2005) on the link between evaluations for summarization and question answering. A NIST program starting in 2001, the Document Understanding Conference (DUC), has sponsored an annual evaluation of summarization algorithms. These have included single document, multiple document, and query-focused summarization; proceedings from the annual workshop are available online.

Mani and Maybury (1999) is the definitive collection of classic papers on summarization. Sparck Jones (2007) is a good recent survey, and Mani (2001) is the standard textbook.

Paraphrase detection

The task of **paraphrase detection** is an important task related to improving recall in question answering and avoiding redundancy in summarization, and also very relevant for tasks like textual entailment. See Lin and Pantel (2001), Barzilay and Lee (2003), Pang et al. (2003), Dolan et al. (2004), Quirk et al. (2004) for representative papers on techniques for detecting paraphrases.

Text categorization

Another task related to information retrieval and summarization is the **text categorization** task, which is to assign a new document to one of a pre-existing set of document classes. The standard approach is to use supervised machine learning to train classifiers on a set of documents that have been labeled with the correct class. A very important application of text categorization is for **spam detection**.

Spam detection

Exercises

23.1 Do some error analysis on web-based question answering. Choose 10 questions and type them all into two different search engines. Analyze the errors (e.g., what kinds of questions could neither system answer; which kinds of questions did one work better on; was there a type of question that could be answered just from the snippets, etc).

23.2 Read Brill et al. (2002) and reimplement a simple version of the AskMSR system.