Advance Access publication February 16, 2009

Political Analysis (2008) 16:372–403
doi:10.1093/pan/mpn018

Fightin' Words: Lexical Feature Selection and Evaluation for Identifying the Content of Political Conflict

Burt L. Monroe
Department of Political Science, Quantitative Social Science Initiative, The Pennsylvania State University, e-mail: burtmonroe@psu.edu (corresponding author)

Michael P. Colaresi
Department of Political Science, Michigan State University, e-mail: colaresi@msu.edu

Kevin M. Quinn
Department of Government and Institute for Quantitative Social Science, Harvard University, e-mail: kevin_quinn@harvard.edu

Entries in the burgeoning ''text-as-data'' movement are often accompanied by lists or visualizations of how word (or other lexical feature) usage differs across some pair or set of documents. These are intended either to establish some target semantic concept (like the content of partisan frames) to estimate word-specific measures that feed forward into another analysis (like locating parties in ideological space) or both. We discuss a variety of techniques for selecting words that capture partisan, or other, differences in political speech and for evaluating the relative importance of those words. We introduce and emphasize several new approaches based on Bayesian shrinkage and regularization. We illustrate the relative utility of these approaches with analyses of partisan, gender, and distributive speech in the U.S. Senate.

1 Introduction

As new approaches to, and applications of, the ''text-as-data'' movement emerge, we find ourselves presented with many collections of disembodied words. Newspaper articles, blogs, and academic papers burst with lists of words that vary or discriminate across groups of documents (Gentzkow and Shapiro 2006; Quinn et al. 2006; Diermeier et al. 2007; Sacerdote and Zidar 2008; Yu et al. 2008), pictures of words scaled in some political space (Schonhardt-Bailey 2008), or both (Monroe and Maeda 2004; Slapin and Proksch 2008). These word or feature lists and graphics are one of the most intuitive ways to convey the

Author's note: We would like to thank Mike Crespin, Jim Dillard, Jeff Lewis, Will Lowe, Mike MacKuen, Andrew Martin, Prasenjit Mitra, Phil Schrodt, Corwin Smidt, Denise Solomon, Jim Stimson, Anton Westveld, Chris Zorn, and participants in seminars at the University of North Carolina, Washington University, and Pennsylvania State University for helpful comments on earlier and related efforts. Any opinions, findings, and conclusions or recommendations expressed in the paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

key insight from such analyses—the content of a politically interesting difference. Accordingly, we notice when such analyses 1) produce key word lists with odd conceptual matches,[1] 2) remove words from lists before presenting them to the reader (Diermeier et al. 2007) or 3) produce no word lists or pictures at all (Hillard et al. 2007; Hopkins and King 2007).

These word visualizations and lists are common because they serve three important roles. First, they are often intended (implicitly) to offer semantic validity to an automated content analysis—to ensure that substantive meaning is being captured by some text-as-data measure (Krippendorff 2004). That is, the visualizations reflect either a selection of words or a word-specific measure that is intended to characterize some semantic political concept of direct interest, for example, topic (Quinn et al. 2006), ideology (Diermeier et al. 2007), or competitive frame (Schonhardt-Bailey 2008). Second, the visualizations reflect either a selection of words or a word-specific measure that is intended to feed forward to some other analysis. For example, these word estimates might be used to train a classifier for uncoded documents (Yu et al. 2008), to scale future party manifestos against past ones (Laver et al. 2003), to scale individual legislators relative to their parties (Monroe and Maeda 2004), or to evaluate partisan bias in media sources (Gentzkow and Shapiro 2006). Third and more broadly, the political content of the words themselves—words tell us what they mean—allows word lists and visualizations of words to compactly present the very political content and cleavages that justify the enterprise. If we cannot find principled ways to meaningfully sort, organize, and summarize the substantial and at times overwhelming information captured in speech, the promise of using speeches and words as observations to be statistically analyzed is severely compromised.

In this paper, we take a closer look at methods for identifying and weighting words and features that are used distinctly by one or more political groups or actors. We demonstrate some fundamental problems with common methods used for this purpose. Major problems include failure to account for sampling variation and overfitting of idiosyncratic differences. Having diagnosed these problems, we offer two new approaches to the problem of identifying political content. Our proposed solution relies on a model-based approach to avoid inefficiency and shrinkage and regularization to avoid both infinite estimates and overfitting the sample data. We illustrate the usefulness of these methods by examining partisan framing, the content of representation and polarization over time, and the dimensionality of politics in the U.S. Senate.

2 The Objectives: Feature Selection and Evaluation

There are two slightly different goals to be considered here: *feature selection* and *feature evaluation*. With feature selection, the primary goal is a binary decision—in or out—for the inclusion of features in some subsequent analysis. We might, for example, want to know *which words* are reliably used differently by two political parties. This might be for the purpose of using these features in a subsequent model of individual speaker positioning or for a qualitative evaluation of the ideological content of party competition to give two examples. In the former case, fear of overfitting leads us to prefer parsimonious specifications, as with variable selection in regression. In the latter case, computers can estimate but

[1] *Motorway* is a Labour word? (Laver et al. 2003). The single most Republican word is *meth*? (Yu et al. 2008). The word that best defines the ideological left in Germany is *pornographie*? (Slapin and Proksch 2008). Martin Luther King and Ronald Reagan's speeches were distinguished by the use of *him* (MLK) and *she* (Reagan)? (Sacerdote and Zidar 2008).

not interpret our models; data reduction is necessary to reduce the quantity of information that must be processed by the analyst. Further, the scale of speech data is immense. Feature selection is useful because we necessarily need a lower dimensional summary of the sample data. Additionally, we know a priori that different speech patterns across groups can flow from both partisan and nonpartisan sources, including idiosyncrasies of dialect and style.

With feature evaluation, the goal is to quantify our information about different features. We want to know, for example, *the extent to which each word* is used differently by two political parties. This might be used to tell us how to weight features in a subsequent model or allow us, in the qualitative case, to have some impression of the relative importance of each word for defining the content of the partisan conflict. The question is not which of these terms are partisan and which are not, but which are the most partisan, on which side, and by how much. Again, in all these cases, parsimony and clarity are virtues. Overfitting should be guarded against because 1) we are interested, not solely in the sample data, but inferring externally valid regularities from that sample data and 2) a list of thousands of words, all of which are approximately equal in weight, is less useful than a list that is winnowed and summarized by some useful criterion.

3 Methods for Lexical Feature Selection and Evaluation

To fix ideas, we use a running example, identifying and evaluating the linguistic differences between Democrats and Republicans in U.S. Senate speeches on a given topic. The topics we use, like ''Defense'' or ''Judicial Nominations'' or ''Taxes'' or ''Abortion'', are those that emerge from the topic model of Senate speech from 1997 to 2004 discussed in Quinn et al. (2006).[2] In this section, our running example is an analysis of speeches delivered on the topic of ''Abortion'' during the 106th Congress (1999–2000), often the subject of frame analysis (Adams 1997; McCaffrey and Keys 2008; Schonhardt-Bailey 2008).[3] The familiarity of this context makes clear the presence, or absence, of semantic validity under different methods.

The lessons are easily generalized and applied to other contexts (gender, electoral districts, opposition status, multiparty systems, etc.), which we demonstrate in the final section. We take the set of potentially relevant lexical features to be the counts of word stems (or ''lemmas'') produced in aggregate across speeches by those of each party. In the interest of direct communication, we will simply say ''words'' throughout. This generalizes trivially to other feature lists: nonstemmed words, *n*-grams, part-of-speech-tagged words, and so on.[4] So, in short, we wish to select partisan words, evaluate the partisanship of words, or both.

[2]Our data are constructed from the Congressional Speech Corpus under the Dynamics of Rhetoric and Political Representation Project. The raw data are the electronic version of the (public domain) U.S. Congressional Record maintained by the Library of Congress on its THOMAS system. The Congressional corpus includes both the House and the Senate for the period beginning with the 101st Congress (1988–) to the present. For this analysis, we truncate the data at December 31, 2004. We then parse these raw html documents to into tagged XML versions. Each paragraph is tagged as speech or nonspeech, with speeches further tagged by speaker. For the topic coding of the speeches, the unit of analysis is the speech document. That is, all words spoken by the same speaker within a single html document. The speeches are then further parsed to filter out capitalization and punctuation using a set of rules developed by Monroe et al. (2006). Finally, the words are stemmed using the Porter Snowball II stemmer (Porter 2001). This is simply a set of rules that groups words with similar roots (e.g., speed and speeding). These stems are then summed within each speech document and the sums used within the Dynamic Multinomial Topic Coding model (Quinn et al. 2006).

[3]For our particular substantive and theoretical interests, pooling all speech together has several undesirable consequences and topic modeling is a crucial prior step.

[4]The computational demands for preprocessing, storage, and estimation time can vary, as can the statistical fit and semantic interpretability of the results.

In the sections that follow, we consider several sets of approaches. The first are ''classification'' methods from machine learning that are often applied to such problems. The idea would be, in our example, to try to identify the party of a speaker based on the words she used. We provide a brief discussion about why this is inappropriate to the task. Second, we discuss several commonly used nonmodel-based approaches. These use a variety of simple and not-so-simple statistics but share the common feature that there is no stated model of the data-generating process and no implied statements of confidence about the conclusions. This set of approaches includes techniques often used in journalistic and informal applications, as well as techniques associated with more elaborate scoring algorithms. Here, we also discuss a pair of ad hoc techniques that are commonly used to try to correct the problems that appear in such approaches. Third, we discuss some basic model-based approaches that allow for more systematic information accounting. Fourth and finally, we discuss a variety of techniques that use shrinkage, or regularization, to improve results further. Roughly speaking, our judgment of the usefulness of the techniques increases as the discussion progresses, and the two methods in the fourth section are the ones we recommend.

We start with some general definitions of notation that we use throughout. Let $w = 1, \ldots, W$ index words. Let $\mathbf{y}$ denote the W-vector of word frequencies in the corpus, and $\mathbf{y}_k$ the W-vector of word frequencies within any given topic k. We further partition the documents across speakers/authors. Let $i \in I$ index a partition of the documents in the corpus. In some applications, i may correspond to a single document; in other applications, it may correspond to an aggregation, like all speeches by a particular person, by all members of a given party, or by all members of a state delegation, depending on the variable of interest. So, let $\mathbf{y}_k^{(i)}$ denote the W-vector of word frequencies from documents of class i in topic k.

In our running example of this section, we focus on the lexical differences induced by party, so we assume that $I = \{D, R\}$—Democratic and Republican—so that $y_{kw}^{(D)}$ represents the number of times Democratic Senators used word w on topic k, which in this section is exclusively abortion. $y_{kw}^{(R)}$ is analogously defined.

3.1 *Classification*

One approach, standard in the machine learning literature, is to treat this as a classification problem. In our example, we would attempt to find the words (w) that significantly predict partisanship (p). A variety of established machine learning methods could be used: Support Vector Machines (Vapnik 2001), AdaBoost (Freund and Schapire 1997), random forests (Breiman 2001) among many other possibilities.[5] These approaches would attempt to find some classifier function, c, that mapped words to some unknown party label, $c : w \rightarrow p$.

The primary problem of this approach, for our purposes, is that it gets the data generation process backwards. Party is not plausibly a function of word choice. Word choice is (plausibly) a function of party. Any model of some substantive process of political language production—such as the strategic choice of language for heresthetic purposes (Riker 1986)—would need to build the other way.

Nor does this correctly match the inferential problem, implying we (a) observe partisanship and word choice for some subset of our data, (b) observe only word choice for another subset, and (c) wish to develop a model for accurately inferring partisanship for the second subset. Partisanship is perfectly observed. While it is possible to infer future unobserved

[5]A detailed explanation of these approaches is beyond the scope of the paper. See Hastie et al. (2001) for a introduction.

word choice with a model that treated this information as given ($P(p|w)$) by inverting the probability using Bayes' rule, this would entail knowing something about the underlying distribution of words ($P(w)$). More problematically, in many applications we would be using an uncertain probabilistic statement ($P(p|w)$), in place of what we know—the partisan membership of an individual. This effectively discards information unnecessarily.

Hand (2006) has noted several other relevant problems with these, nicely supplemented for political scientists by Hopkins and King (2007). Yu et al. (2008) apply such an approach in the Congressional setting, using language in Congressional speech to classify speakers by party. Their Tables 6–8 list features (words) detected for Democratic-Republican classification by their method, one way of using classification techniques like support vector machines or Naive Bayes for feature selection.

3.2 *Nonmodel-Based Approaches*

Many of the most familiar approaches to the problems of feature selection and evaluation are not based on probabilistic models. These include a variety of simple statistics, as well as two slightly more elaborate algorithmic methods for ''scoring'' words. The latter include the *tf.idf* measure in broad use in computational linguistics and the WordScores method (Laver et al. 2003) prominent in political science. In this section, we demonstrate these methods for our running example, discuss how they are related, and evaluate potential problems in the semantic validity of their results.

3.2.1 Difference of frequencies

We start with a variety of simple statistics. One very naive index is simply the difference in the absolute word-use frequency by parties: $y_{kw}^{(D)} - y_{kw}^{(R)}$. If Republicans say *freedom* more than Democrats, then it is a Republican word. The problem with this, of course, is that it is overwhelmed by whoever speaks more. So, in our running example of speech on abortion, we find that the most common words (*the*, *of*, *and*, *is*, *to*, *a*) are considered Republican. The mistake here is obvious in our data, but perhaps less obvious when one does something like compare the number of database hits for *freedom* in a year of the *New York Times* and the *Washington Post*, without making some effort to ascertain how big each database is. This is common in journalistic accounts[6] and common enough in linguistics that Geoffrey Nunberg devotes a methodological appendix in *Talking Right* to explaining why it is a mistake (Nunberg 2006, 209–10).

3.2.2 Difference of proportions

Taking the obvious step of normalizing the word vectors to reflect word proportions rather than word counts, a better measure is the difference of proportions on each word. Defining the observed proportions by $f_{kw}^{(i)} = y_{kw}^{(i)}/n_k^{(i)}$. Now, the evaluation measure becomes $f_{kw}^{(D)} - f_{kw}^{(R)}$.

Figure 1 shows the results of applying this measure to evaluate partisanship of words on the topic of abortion during the 106th (1999–2000) Senate. The scatter cloud plots these values for each word against the (logged) total frequency of the word in this collection of

[6]For example, a recent widely circulated graphic in *The New York Times* (available at http://www.nytimes.com/interactive/2008/09/04/us/politics/20080905_WORDS_GRAPHIC.html) provided a comparative table of the absolute frequencies of selected words and phrases in the presidential convention speeches of four Republicans and four Democrats (Ericson 2008).

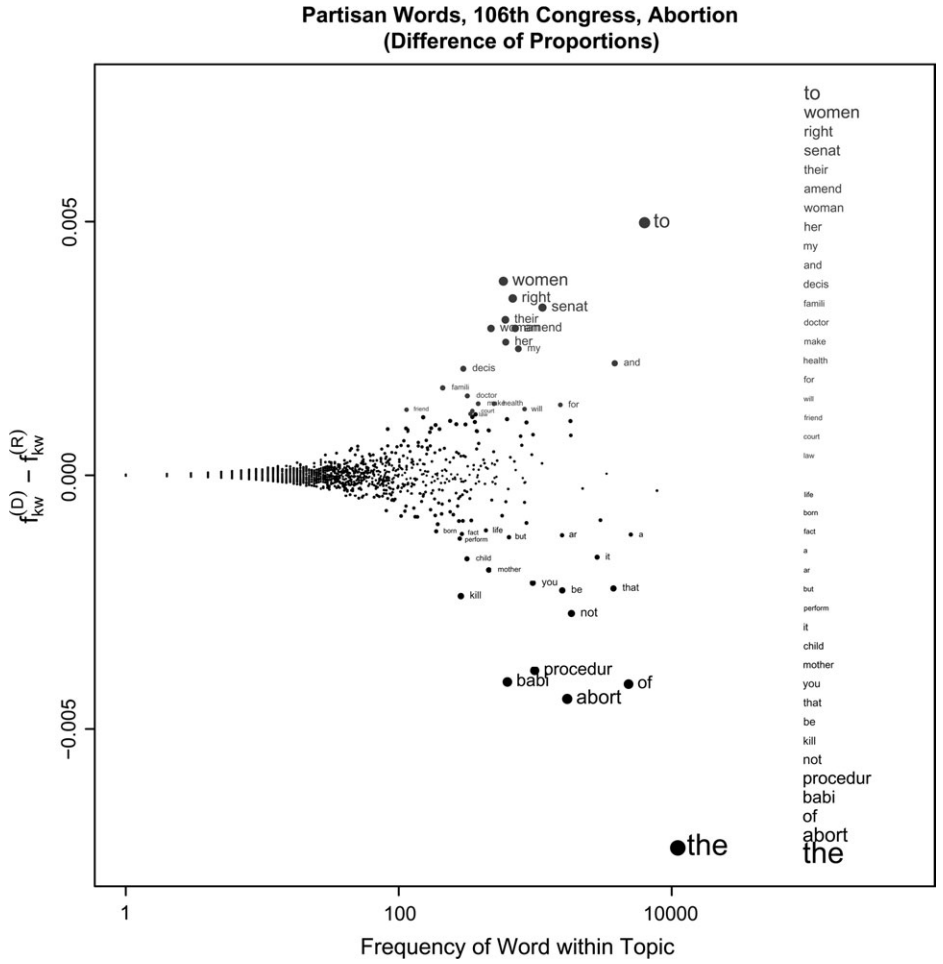


Fig. 1 Feature evaluation and selection using $f_{kw}^{(D)} - f_{kw}^{(R)}$. Plot size is proportional to evaluation weight, $|f_{kw}^{(D)} - f_{kw}^{(R)}|$. The top 20 Democratic and Republican words are labeled and listed in rank order to the right. The results are almost identical for two other measures discussed in the text: unlogged *tf.idf* and frequency-weighted WordScores.

speeches. In this and subsequent figures for the running example, the *y*-axis, size of point, and size of text all reflects the evaluation measure under consideration, in this case $f_{kw}^{(D)} - f_{kw}^{(R)}$. For the top 20 most Democratic and most Republican words, the dots have been labeled with the word, again plotted proportionally to the evaluation measure. These 40 words are repeated, from most Democratic to most Republican, down the right-hand side of the plot.

This is an improvement. There is no generic partisan bias based on volume of speech, and we see several of the key frames that capture known differences in how the parties frame the issue of abortion. For example, Republicans encourage thinking about the issue from the point of view of *babies*, whereas Democrats encourage thinking about the issue from the point of view of *women*. But the lack of overall semantic validity is clear in the overemphasis on high-frequency words. The top Democratic word list is dominated by *to* and includes *my*, *and*, and *for*; the top Republican word list is dominated by *the* and

includes *of*, *not*, *be*, *that*, *you*, *it*, and *a*. As is obvious in Figure 1, the sampling variation in difference of proportions is greatest in high-frequency words. These are not partisan words; they are just common ones.

The difference of proportions statistic is ubiquitous. It appears in journalistic (e.g., the top half of the convention speech graphic discussed in footnote 6, which notes, e.g., that ''Republican speakers have talked about reform and character far more frequently than the Democrats'') and academic accounts.[7] The problem with high-frequency words is often masked by referring to only selected words of interest in isolation.

3.2.3 Correction: removing stop words

A common response to this problem in many natural language processing applications is to eliminate ''function'' or ''stop'' words that are deemed unlikely to contain meaning. This is also true in this particular instance, as many related applications are based on difference of proportions on non–stop words only. This is the algorithm underlying several ''word cloud'' applications increasingly familiar in journalistic and blog settings,[8] as well as more formal academic applications like Sacerdote and Zidar (2008).

We note, however, the practice of stop word elimination has been found generally to create more problems than it solves, across natural language processing applications. Manning et al. (2008) observe: ''The general trend . . . over time has been from standard use of quite large stop lists (200–300 terms) to very small stop lists (7–12 terms) to no stop list whatsoever'' (p. 27). They give particular emphasis to the problems of searching for phrases that might disappear or change meaning without stop words (e.g., ''to be or not to be''). In our example, a stop list would eliminate a word like *her*, which almost definitely has political content in the context of abortion,[9] and a word like *their*, which might (e.g., *women and their doctors*).

More to the point, this ad hoc solution diagnoses the problem incorrectly. Function words are not dominant in the partisan word lists here because they are function words, but because they are frequent. They are more likely to give extreme values in differences of proportions just from sampling variability. Eliminating function words not only eliminates words like *her* inappropriately but it also elevates high-frequency non–stop words inappropriately. The best example here is *Senate*, which is deemed Democratic by difference of proportions, but is, in this context, simply a high-frequency word with high sampling variability.

3.2.4 Odds

Alternatively, we can examine the proportions in ratio form, through odds. The observed ''odds'' (we assert no probability model yet) of word w in group i's usage are defined as $O_{kw}^{(i)} = f_{kw}^{(i)}/(1 - f_{kw}^{(i)})$. The odds ratio between the two parties is $\theta_{kw}^{(D-R)} = O_{kw}^{(D)}/O_{kw}^{(R)}$.[10]

[7] For example, ''Hillary Clinton uses the word security 8.8 times per 10,000 words while Obama . . .uses the word about 6.8 times per 10,000 words'' (Sacerdote and Zidar 2008).

[8] Examples (in which the exact algorithm is proprietary) include Wordle (http://wordle.net) and tag clouds from IBM's Many Eyes visualization site (http://services.alphaworks.ibm.com/manyeyes/page/Tag_Cloud.html).

[9] Try making a statement about the Democratic position on abortion without using the word *her*.

[10] We could also work with risk ratios, $f_{kw}^{(D)}/f_{kw}^{(R)}$, which function more or less identically with low probability events, like the use of any particular word.

This is generally presented for single words in isolation or as a metric for ranking words. Examples can be found across the social sciences, including psychology[11] and sociology.[12] In our data, the odds of a Republican using a variant of *babi* are 5.4 times those of a Democratic when talking about abortion, which seems informative. But, the odds of a Republican using the word *April* are 7.4 times those of a Democrat when talking about abortion, which seems less so.

3.2.5 Log-odds-ratio

Lack of symmetry also makes odds difficult to interpret. Logging the odds ratio provides a measure that is symmetric between the two parties. Working naively, it is unclear what we are to do with words that are spoken by only one party and therefore have infinite odds ratios. If we let the infinite values have infinite weight, the partisan word list consists of only those spoken by a single party. The most Democratic words are then *bankruptci*, *Snow*[*e*], *ratifi*, *confidenti*, and *church*, and the most Republican words are *infant*, *admit*, *Chines*, *industri*, and *40*. If we instead throw out the words with zero counts in one party, the most Democratic words are *treati*, *discrim*, *abroad*, *domest*, and *privacy*, and the most Republican words are *perfect*, *subsid*, *percent*, *overrid*, and *cell*.

One compromise between these two comes from the common advice to ''add a little bit to the zeroes'' (say, 0.5, as in Agresti [2002, 70–1]). If we calculate a smoothed log-odds-ratio from such supplemented frequencies, $\tilde{f}_{kw}^{(i)} = f_{kw}^{(i)} + \varepsilon$, we get the results as shown in Figure 2.

Note that regardless of the zero treatment, the most extreme words are obscure ones. These word lists are strange in a way opposite from that produced by the difference of proportions shown in Figure 1. These do not seem like words that most fundamentally define the partisan division on abortion. Words with plausible partisan content on abortion (*infant*, *church*) are overwhelmed by oddities that require quite a bit more investigation to interpret (*Chines*, *bankruptci*). In short, the semantic validity of this measure is limited.[13]

The problem is again the failure to account for sampling variability. With log-odds-ratios, the sampling variation goes down with increased frequency, as is clear in Figure 2. So, this measure will be inappropriately dominated by obscure words.

3.2.6 Correction: eliminating low-frequency words

Although this is a very basic statistical idea, it is commonly unacknowledged in simple feature selection and related ranking exercises. A common response is to set some frequency ''threshold'' for features to ''qualify'' for consideration. Generally, this simply removes the most problematic features without resolving the issue.

For an example of the latter, consider the list of Levitt and Dubner (2005; 194–8), in their freakonomic discussion of baby names, of ''The Twenty White Girl [Boy] Names That Best Signify High-Education Parents.'' They identify these, from California records,

[11] One study in which preschoolers were observed as they talked found that girls were six times more likely to use the word *love* and twice as likely to use the word *sad* (Senay, Newberger, and Waters 2004, 68).

[12] Americans used the word *frontier* in business names . . . more than 4 times more often than France (Kleinfeld and Kleinfeld 2004).

[13] There *is* abortion partisanship in these words. For example, Democrat Charles Schumer introduced an amendment to a bankruptcy reform bill designed to prevent abortion clinic protesters from avoiding fines via bankruptcy protections. We argue only that these do not have face validity as the *most important* words.

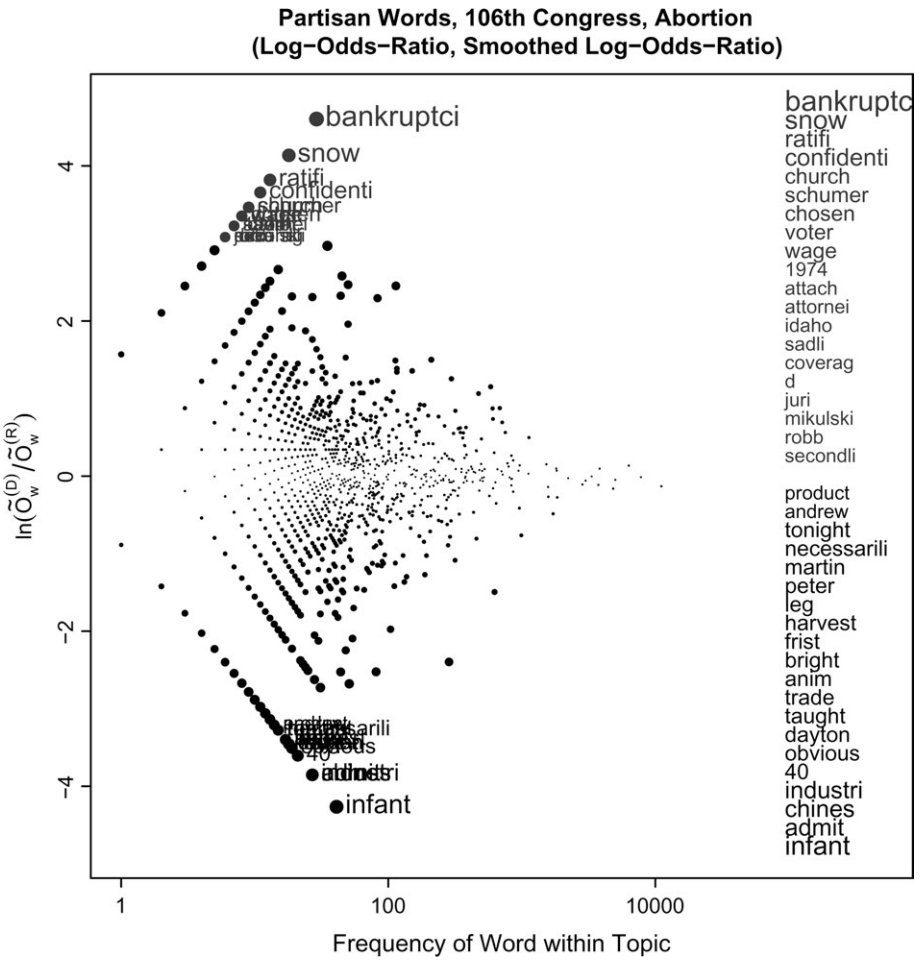


Fig. 2 Feature evaluation and selection using $\hat{\delta}_{kw}^{(D-R)}$. Plot size is proportional to evaluation weight, $\left|\hat{\delta}_{kw}^{(D-R)}\right|$. Top 20 Democratic and Republican words are labeled and listed in rank order. The results are identical to another measure discussed in the text: the log-odds-ratio with uninformative Dirichlet prior.

with a list ranked by average mother's education. The top five girl names are *Lucienne*, *Marie-Claire*, *Glynnis*, *Adair*, and *Meira*; the top five boy names are *Dov*, *Akiva*, *Sander*, *Yannick*, and *Sacha*. A footnote clarifies that a name must appear at least 10 times to make the list (presumably because a list that allowed names used only once or twice might have looked *ridiculous*). It seems likely that each of these words was used exactly, or not many more than, 10 times in the sample. We would get a similar effect by drawing a line at 10 or 100 minimum uses of a word in Figure 2, with the method then selecting the most extreme examples from the least frequent qualifying words.

This mistake is also common in many of the emergent text-as-data discussions in political science. One example is given by Slapin and Proksch (2008), made more striking because they provide a plot that clearly demonstrates the heteroskedasticity. Their Figure 2 looks like Figure 2 turned on its side. The resulting lists of their Table 1 contain many obscurities. This is probably only an issue of interpretation. The item response model they describe, like that of Monroe and Maeda (2004) (see also Lowe [2008]), accounts for

variance when the word parameters are used to estimate speaker/author positions. That is, despite their use of captions like ''word weights'' and ''Top Ten Words placing parties on the left and right,'' these are really the words with the 10 leftmost and rightmost point estimates, not the words that have the most influence in the estimates of actor positions.

3.2.7 tf.idf (Computer Science)

It is common practice in the computational linguistics applications of classification (e.g., Which bin does this document belong in?) and search (e.g., Which document(s) should this set of terms be matched to?) to model documents not by their words but by words that have been weighted by their *tf.idf*, or *term frequency—inverse document frequency*. Term frequency refers to the relative frequency (proportion) with which a word appears in the document; document frequency refers to the relative frequency with which a word appears, at all, in documents across the collection. The logic of *tf.idf* is that the words containing the greatest information about a particular document are the words that appear many times in that document, but in relatively few others. *tf.idf* is recommended in standard textbooks (Jurafsky and Martin (2000, 651–4) (Manning and Schütze 1999, 541–4) and is widely used in document search and information retrieval tasks.[14] To the extent *tf.idf* reliably captures what is distinctive about a particular document, it could be interpreted as a feature evaluation technique.

The most common variant of *tf.idf* logs the *idf* term—this is the ''ntn'' variant (natural *tf* term, logged *df* term, no normalization, see Manning and Schütze [1999, 544]). So, letting df_{kw} denote the fraction of groups that use the word w on topic k at least once, then:

$$tf.idf^{(i)}_{kw}(\text{ntn}) = f^{i}_{kw} \ln\left(1/df_{kw}\right). \tag{1}$$

Qualitatively, the results from this approach are identical to the infinite log-odds-ratio results given earlier. The most partisan words are the words spoken the most by one party, while spoken not once by the other (*bankruptci*, *infant*).[15] Clearly, the logic of logging the document frequency[16] breaks down in a collection of two documents.

Alternatively, we can use an unlogged document frequency term—the ''nnn'' (natural tf term, natural *df* term, no normalization; see Manning and Schütze [1999, 544]) variant of *tf.idf*.

$$tf.idf^{(i)}_{kw}(\text{nnn}) = f^{i}_{kw} / df_{kw}. \tag{2}$$

The results for our running example are nearly identical, qualitatively and quantitatively with those from raw difference of proportions, shown in Figure 1. The weights are correlated (at +0.997 in this case) and differ only in doubling the very low weights of the relatively low-frequency words used by only one party.[17] In any case, for our purposes, neither version of *tf.idf* has clear value. See Hiemstra (2000) and Aizawa (2003) for efforts to put *tf.idf* on a probabilistic or information-theoretic footing.

[14] We note over 15,000 hits for the term in Google Scholar.
[15] The degenerate graphic of this result is omitted for space reasons, but available in the web appendix.
[16] Due to the large number of documents in many collections, this measure is usually squashed with a log function (Jurafsky and Martin 2000, 653).
[17] We omit this mostly redundant graphic here. It is available in the web appendix.

3.2.8 WordScores (Political Science)

Perhaps the most prominent text-as-data approach in political science is the WordScores procedure (Laver et al. 2003), which embeds a feature evaluation technique in its algorithm. The first step of the algorithm establishes scores for words based on their frequencies within ''reference'' texts, which are then used to scale other ''virgin'' texts (for further detail, see Lowe [2008]).

In our running example, we calculate these by setting the Democrats at $+1$ and the Republicans at -1. Then the raw WordScore for each word is:

$$W_{kw}^{(D-R)} = \frac{y_{kw}^{(D)}/n_k^{(D)} - y_{kw}^{(R)}/n_k^{(R)}}{y_{kw}^{(D)}/n_k^{(D)} + y_{kw}^{(R)}/n_k^{(R)}}. \tag{3}$$

Figure 3 shows the results of applying this stage of the WordScores algorithm to our running example. The results bear qualitative resemblance to those with the smoothed log-odds-ratios shown in Figure 2. As shown in Figure 3, the extreme W_{kw} are received by the words spoken by only one party. As with several previous measures, the maximal words are obscure low-frequency words.

The ultimate use for WordScores, however, is for the spatial placement of documents. When the W_{kw} are taken forward to the next step, the impact of any word is proportional to its relative frequency. That is, the implicit evaluation measure, $W_{kw}^{*(D-R)}$, is

$$W_{kw}^{*(D-R)} = \frac{y_{kw}^{(D)}/n_k^{(D)} - y_{kw}^{(R)}/n_k^{(R)}}{y_{kw}^{(D)}/n_k^{(D)} + y_{kw}^{(R)}/n_k^{(R)}} n_{kw}. \tag{4}$$

In the case of two ''documents,'' as is the case here, this is nearly identical to the difference of proportions measure. In this example, they correlate at over $+0.998$. So, WordScores demonstrates the same failure to account for sampling variation and the same overweighting of high-frequency words. Lowe (2008) (in this volume) gives much greater detail on the workings of the WordScores procedure and how it might be given a probabilistic footing.

3.3 *Model-Based Approaches*

In our preferred model-based approaches, we model the choice of word as a function of party, $P(w|p)$. We begin by discussing a model for the full collection of documents and then show how this can be used as a starting point and baseline for subgroup-specific models. In general, our strategy is to first model word usage in the full collection of documents and to then investigate how subgroup-specific word usage diverges from that in the full collection of documents.

3.3.1 The likelihood

Consider the following model. We will start without subscripts and consider the counts in the entire corpus, $\mathbf{y}$:

$$\mathbf{y} \sim \text{Multinomial}(n, \boldsymbol{\pi}), \tag{5}$$

where $n = \sum_{w=1}^{W} y_w$ and $\boldsymbol{\pi}$ is a W-vector of multinomial probabilities. Since $\boldsymbol{\pi}$ is a vector of multinomial probabilities, it is constrained to be in the $(W-1)$–dimensional simplex. In some variants below, we reparameterize and use the (unbounded) log odds transformation

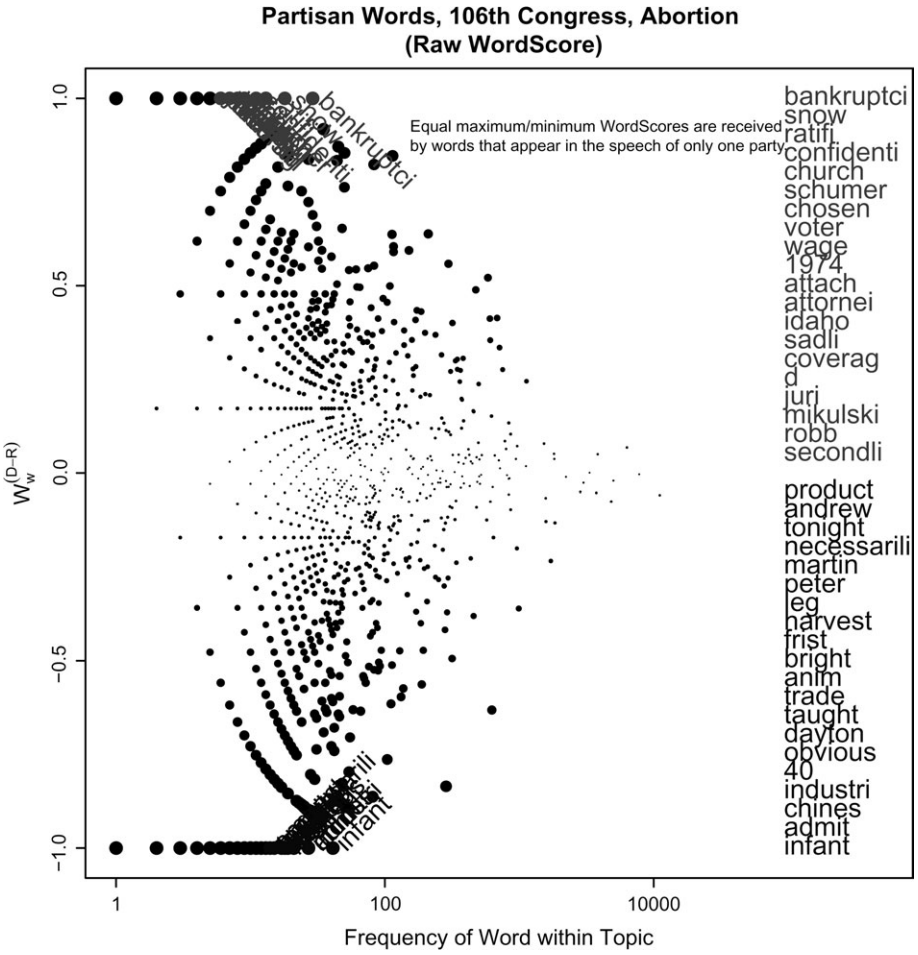


Fig. 3 Feature evaluation using W_{kw}. Plot size is proportional to evaluation weight, $|W_{kw}|$. The top 20 Democratic and Republican words are labeled and listed in rank order to the right.

$$\beta_w = \log(\pi_w) - \log(\pi_1) w = 1, \ldots, W. \tag{6}$$

and work with $\boldsymbol{\beta}$ instead of $\boldsymbol{\pi}$. The inverse transformation

$$\pi_w = \frac{\exp(\beta_w)}{\sum_{j=1}^{W} \exp(\beta_j)}. \tag{7}$$

allows us to transform $\boldsymbol{\beta}$ estimates back to estimates for $\boldsymbol{\pi}$. The likelihood and log-likelihood functions are:

$$L(\boldsymbol{\beta}|\mathbf{y}) = \prod_{w=1}^{W} \left(\frac{\exp(\boldsymbol{\beta}_w)}{\sum_{j=1}^{W} \exp\left(\boldsymbol{\beta}_j\right)} \right)^{y_w}, \tag{8}$$

and

$$\ell(\boldsymbol{\beta}|\mathbf{y}) = \sum_{w=1}^{W} y_w \log\left(\frac{\exp(\boldsymbol{\beta}_w)}{\sum_{j=1}^{W}\exp\left(\boldsymbol{\beta}_j\right)}\right). \tag{9}$$

Within any topic, k, the model to this point goes through with addition of subscripts:

$$\mathbf{y}_k \sim \text{Multinomial}(n_k, \boldsymbol{\pi}_k), \tag{10}$$

with parameters of interest, β_{kw}, and log-likelihood, $\ell(\boldsymbol{\beta}_k|\mathbf{y}_k)$, defined analogously.

Further, within any group-topic partition, indexed by i and k, we superscript for group to model:

$$\mathbf{y}_k^{(i)} \sim \text{Multinomial}\left(n_k^{(i)}, \boldsymbol{\pi}_k^{(i)}\right), \tag{11}$$

with parameters of interest, $\beta_{kw}^{(i)}$, and log-likelihood, $\ell(\boldsymbol{\beta}_k^{(i)}|\mathbf{y}_k^{(i)})$, defined analogously.

If we wish to proceed directly to ML estimation, the lack of covariates results in an immediately available analytical solution for the MLE of $\beta_{kw}^{(i)}$. We calculate

$$\hat{\boldsymbol{\pi}}^{\text{MLE}} = \mathbf{f} = \mathbf{y} \cdot (1/n), \tag{12}$$

and $\hat{\boldsymbol{\beta}}^{\text{MLE}}$ follows after transforming.

3.3.2 Prior

The simplest Bayesian model proceeds by specifying the prior using the conjugate for the multinomial distribution, the Dirichlet:

$$\boldsymbol{\pi} \sim \text{Dirichlet}(\boldsymbol{\alpha}), \tag{13}$$

where $\boldsymbol{\alpha}$ is a W-vector, $\alpha_w > 0$, with a very clean interpretation in terms of "prior sample size." That is, use of any particular Dirichlet prior defined by $\boldsymbol{\alpha}$ affects the posterior exactly as if we had observed in the data an additional $\alpha_w - 1$ instances of word w. This can be arbitrarily uninformative, for example, $\alpha_w = 0.01$ for all w. Again, we can carry this model through to topics and topic-group partitions with appropriate sub- and superscripting.

3.3.3 Estimation

Due to the conjugacy, the full Bayesian estimate using the Dirichlet prior is also analytically available in analogous form:

$$\hat{\boldsymbol{\pi}} = \left(\mathbf{y} + \boldsymbol{\alpha}\right) \cdot 1/\left(n + \alpha_0\right). \tag{14}$$

where $\alpha_0 = \sum_{w=1}^{W} \alpha_w$.

Again, all this goes directly through to partitions with appropriate subscripts if desired.

3.3.4 Feature evaluation

What we have to this point is sufficient to suggest the first approach to feature evaluation. Denote the odds (now with probabilistic meaning) of word w, relative to all others, as

$\Omega_w = \pi_w/(1-\pi_w)$, again with additional sub- and superscripts for specific partitions. Since the Ω_w are functions of the π_w, estimates of these follow directly from the $\hat{\pi}_w$.

Within any one topic, k, we are interested in how the usage of a word by group i differs from usage of the word in the topic by all groups, which we can capture with the log-odds-ratio, which we will now define as $\delta_w^{(i)} = \log(\Omega_w^{(i)}/\Omega_w)$. The point estimate for this is

$$\hat{\delta}_{kw}^{(i)} = \log\left[\frac{\left(y_{kw}^{(i)} + \alpha_{kw}^{(i)}\right)}{\left(n_k^{(i)} + \alpha_{k0}^{(i)} - y_{kw}^{(i)} - \alpha_{kw}^{(i)}\right)}\right] - \log\left[\frac{(y_{kw} + \alpha_{kw})}{(n_k + \alpha_{k0} - y_{kw} - \alpha_{kw})}\right]. \quad (15)$$

In certain cases, we may be more interested in the comparison of two specific groups. This is the case in our running example, where we will have exactly two groups, Democrats and Republicans. The usage difference is then captured by the log-odds-ratio between the two groups, $\delta_w^{(i-j)}$, which is estimated by

$$\hat{\delta}_{kw}^{(i-j)} = \log\left[\frac{\left(y_{kw}^{(i)} + \alpha_{kw}^{(i)}\right)}{\left(n_k^{(i)} + \alpha_{k0}^{(i)} - y_{kw}^{(i)} - \alpha_{kw}^{(i)}\right)}\right] - \log\left[\frac{\left(y_{kw}^{(j)} + \alpha_{kw}^{(j)}\right)}{\left(n_k^{(j)} + \alpha_{k0}^{(j)} - y_{kw}^{(j)} - \alpha_{kw}^{(j)}\right)}\right]. \quad (16)$$

Without the prior, this is of course simply the observed log-odds-ratio. This would emerge from viewing word counts as conventional categorical data in a contingency table or a logit. For each word, imagine a 2×2 contingency table, with the cells including the counts, for each of the two groups, of word w and of all other words. Or, we can specify a logit of the binary choice, word w versus any other word, with our party group indicator the only regressor. With more than two groups, the same information, with slightly more manipulation (via risk ratios), can be recovered from a multinomial logit or an appropriately constrained Poisson regression (Agresti 2002). With the prior, this is a relabeling of the smoothed log-odds-ratio discussed before.

So, if we apply the measure, with equivalent prior, we get results identical to those shown in Figure 2. This has the same problems, with the dominant words still the same list of obscurities. The problem is clearly that the estimates for infrequently spoken words have higher variance than frequently spoken ones.

We can now exploit the first advantage of having specified a model. Under the given model, the variance of these estimates is approximately:

$$\sigma^2\left(\hat{\delta}_{kw}^{(i)}\right) \approx \frac{1}{\left(y_{kw}^{(i)} + \alpha_{kw}^{(i)}\right)} + \frac{1}{\left(n_k^{(i)} + \alpha_{k0}^{(i)} - y_{kw}^{(i)} - \alpha_{kw}^{(i)}\right)} + \frac{1}{(y_{kw} + \alpha_{kw})} + \frac{1}{(n_k + \alpha_{k0} - y_{kw} - \alpha_{kw})}, \quad (17)$$

$$\approx \frac{1}{\left(y_{kw}^{(i)} + \alpha_{kw}^{(i)}\right)} + \frac{1}{(y_{kw} + \alpha_{kw})}, \quad (18)$$

and

$$\sigma^2\left(\hat{\delta}_{kw}^{(i-j)}\right) \approx \frac{1}{\left(y_{kw}^{(i)} + \alpha_{kw}^{(i)}\right)} + \frac{1}{\left(n_k^{(i)} + \alpha_{k0}^{(i)} - y_{kw}^{(i)} - \alpha_{kw}^{(i)}\right)} + \frac{1}{\left(y_{kw}^{(j)} + \alpha_{kw}^{(j)}\right)} + \frac{1}{\left(n_k^{(j)} + \alpha_{k0}^{(j)} - y_{kw}^{(j)} - \alpha_{kw}^{(j)}\right)}, \quad (19)$$

$$\approx \frac{1}{\left(y_{kw}^{(i)} + \alpha_{kw}^{(i)}\right)} + \frac{1}{\left(y_{kw}^{(j)} + \alpha_{kw}^{(j)}\right)}. \tag{20}$$

Where the approximations in Equations 17 and 19 assume $y_{kw}^{(i)} \gg \alpha_{kw}^{(i)}$, $y_{kw} \gg \alpha_{kw}$ and ignore covariance terms that will typically be close to 0 while Equations 18 and 20 additionally assume that $n_k^{(i)} \gg y_{kw}^{(i)}$ and $n_k \gg y_{kw}$. The approximations are unnecessary but reasonable for documents of moderate size (at 1000 words only the fourth decimal place is affected) and help clarify the variance equation. Variance is based on the absolute frequency of a word in all, or both, documents of interest, and its implied absolute frequency in the associated priors.

3.4 *Accounting for Variance*

Now we can evaluate features not just by their point estimates but also by our certainty about those estimates. Specifically, we will use as the evaluation measure the z-scores of the log-odds-ratios, which we denote with ζ:

$$\hat{\zeta}_{kw}^{(i)} = \hat{\delta}_{kw}^{(i)} / \sqrt{\sigma^2\left(\hat{\delta}_{kw}^{(i)}\right)}, \tag{21}$$

and

$$\hat{\zeta}_{kw}^{(i-j)} = \hat{\delta}_{kw}^{(i-j)} / \sqrt{\sigma^2\left(\hat{\delta}_{kw}^{(i-j)}\right)}. \tag{22}$$

Figure 4 shows the feature weightings based on the $\hat{\zeta}_{kw}^{(D-R)}$ for our running example. The prominent features now much more clearly capture the core partisan differences. Republican word choice reflects framing the debate from the point of view of the *baby/child* and emphasizing the details of the *partial birth abortion procedure*. In contrast, Democrats framed their speech from the point of view of the *woman/women* and *her/their right* to *choose*.

There are still problems here that can be observed in Figure 4. Although function words no longer dominate the lists as with several previous techniques, some still appear to be too prominent in their implied partisanship, among them *the*, *of*, *be*, *not*, and *my*. A related problem is that, although there is variation, every word in the vocabulary receives nonzero weight, which could lead to overfitting if these estimates are used in some subsequent model. We could perhaps use some threshold value of $\hat{\zeta}_{kw}^{(D-R)}$ as a selection mechanism (Figure 4 demonstrates with the familiar cutoff at 1.96.)

A nearly identical selection mechanism, also following from binomial sampling foundations, can be developed by subjecting each word to a χ^2 test and selecting those above some threshold. Coverage is nearly identical to the z-test. Or words can be ranked by p-value of χ^2 test, again nearly identically to ranking by p-value of z-tests. This was the approach used by Gentzkow and Shapiro (2006). There is no implied directionality (e.g., Democratic vs. Republican), however, and the χ^2 value itself does not have any obvious interpretation as an evaluation weight.

3.5 *Shrinkage and Regularization*

The problems we have seen to this point are typical of many contemporary data-rich problems, emerging not just from computational linguistics but from other similarly scaled or ''ill-posed'' machine and statistical learning problems in areas like image processing and gene expression. In these fields, attempts to avoid overfitting are referred to as *regularization*. A related concept, and more familiar in political science, is Bayesian *shrinkage*. There are several approaches, but the notion in common is to put a strong conservative prior on

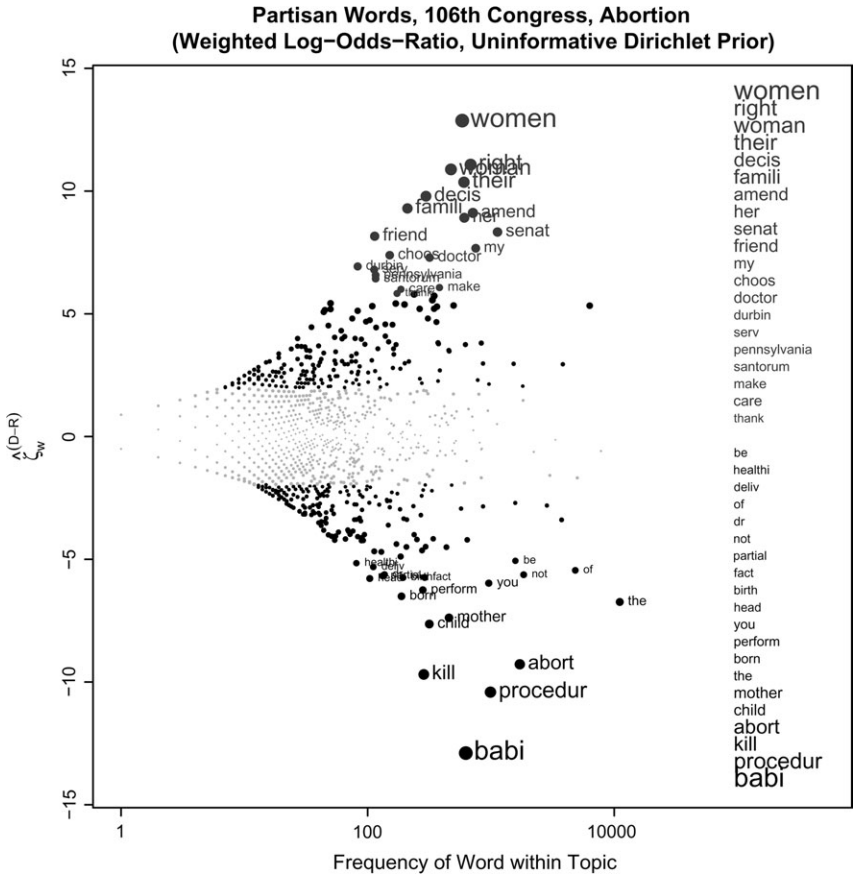


Fig. 4 Feature evaluation and selection using $\hat{\zeta}_{kw}^{(D-R)}$. Plot size is proportional to evaluation weight, $\left|\hat{\zeta}_{kw}^{(D-R)}\right|$; those with $\left|\hat{\zeta}_{kw}^{(D-R)}\right|<1.96$ are gray. The top 20 Democratic and Republican words are labeled and listed in rank order to the right.

the model. We bias the model toward the conclusion of no partisan differences, requiring the data to speak very loudly if such a difference is to be declared.

In this section, we discuss two approaches. In the first, we use the same model as above, but put considerably more information into the Dirichlet prior. In the second, we use a different (Laplace) functional form for the prior distribution.

3.5.1 Informative Dirichlet prior

One approach is to use more of what we know about the expected distribution of words. We can do this by specifying a prior proportional to the expected distribution of features in a random text. That is, we know *the* is used much more often than *nuclear*, and our prior can reflect that information. In our running example, we can use the observed proportion of words in the vocabulary in the context of Senate speech, but across multiple Senate topics.[18] That is,

[18] Although this is technically not a legitimate subjective prior because the data are being used twice, nearly all the prior information is coming from data that are not used in the analysis. Qualitatively similar empirical Bayes results could be obtained by basing the prior on speeches on all topics other than the topic in question or, for that matter, on general word frequency information from other sources altogether.

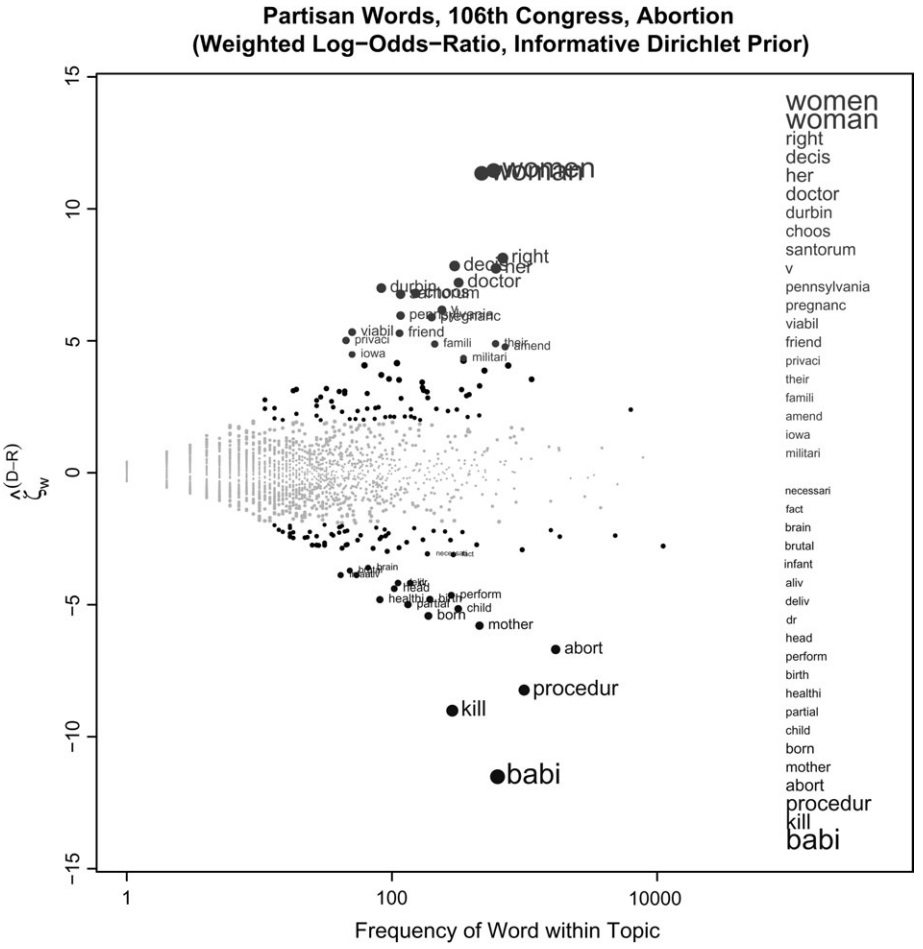


Fig. 5 Feature evaluation and selection based on $\hat{\zeta}_{kw}^{(D-R)}$. Plot size is proportional to evaluation weight, $\hat{\zeta}_{kw}^{(D-R)}$; those with $\left|\hat{\zeta}_{kw}^{(D-R)}\right|<1.96$ are gray. The top 20 Democratic and Republican words are labeled and listed in rank order to the right.

$$\alpha_{kw}^{(i)} = \alpha_{k0}^{(i)} \hat{\boldsymbol{\pi}}^{\text{MLE}} = \mathbf{y} \cdot \frac{\alpha_0}{n} \tag{23}$$

where $\alpha_{k0}^{(i)}$ determines the implied amount of information in the prior. This prior shrinks the $\pi_{kw}^{(i)}$ and $\Omega_{kw}^{(i)}$ to the global values, and shrinks the feature evaluation measures, the $\zeta_{kw}^{(i)}$ and the $\zeta_{kw}^{(i-j)}$, toward zero. The greater $\alpha_{k0}^{(i)}$ is, the more shrinkage we will have.

Figure 5 illustrates results from the running example using this approach. We set α_0 to imply a ''prior sample'' of 500 words per party every day, roughly the average number of words per day used per party on each topic in the data set.

As is shown in Figure 5, this has a desirable effect on the function-word problem noted above. For example, the Republican top 20 list has shuffled, with *the*, *you*, *not*, *of*, and *be* being replaced by the considerably more evocative *aliv*, *infant*, *brutal*, *brain*, and *necessari*.

We also see an apparent improvement in selection, with fewer words now exceeding the same ζ threshold.

The amount of shrinkage is determined by the scale parameter in the Dirichlet prior relative to the absolute frequency of speech in the topic. The prior used in this example has almost no effect on estimates in topics with much more speech and strong partisan differences (like defense, post-Iraq), and overwhelms estimates—as it should—in topics with very little speech or no partisan differences (like symbolic constituent tributes).

Relative to the technique in the next section, this simple approach has several advantages. First, because of the analytical solution, it requires very little computation. Second, this low computational overhead makes it relatively straightforward to develop more flexible variants, such as a dynamic version we discuss below. Third, the fact that these estimates could be recovered from appropriate standard Bayesian generalized linear models means the approach is easily extended beyond the setting of a strict partitioning of the document space into discrete groups. Multiple observable characteristics (party *and* gender *and* geographic district) can be modeled simultaneously, subject to the identification constraints typical of a regression problem.

This approach still suffers as a technique for feature selection: it is unclear where one draws the line between words that qualitatively matter and words that qualitatively do not. So, there is still the problem for the qualitative analyst of interpreting a long (weighted/ordered) list and the problem for the quantitative analyst of possibly overfitting if these are used in subsequent modeling.

3.5.2 Laplace prior

We address this here by using an alternative functional form for the prior, one which shrinks most contrasts between estimates not toward zero, but *to* zero. In this approach, we specify a prior for $\boldsymbol{\beta}_k^{(i)}$. Specifically, we assume

$$\beta_{kw}^{(i)} \overset{ind.}{\sim} \text{Laplace}(\hat{\beta}_{kw}^{\text{MLE}}, \gamma), \quad w = 2, \ldots, W. \tag{24}$$

Or, even more precisely,

$$p(\boldsymbol{\beta}_k^{(i)}|\gamma) = \prod_{w=2}^{W} \frac{\gamma}{2} \exp\Big(-\gamma \Big| \beta_{kw}^{(i)} - \hat{\beta}_{kw}^{\text{MLE}} \Big| \Big). \tag{25}$$

Here β_k^{MLE} is the maximum likelihood estimate from an analysis of the pooled $\mathbf{y}$ which serves as the ''prior'' mode for $\beta_k^{(i)}$ and $\gamma > 0$ serves as an inverse scale parameter.

It is fairly well known that such a prior is equivalent to an L1 regularization penalty (Williams 1995). In practice, such a prior (or regularization penalty) causes many, if not most, of the elements of the posterior mode of $\boldsymbol{\beta}_k^{(i)}$ to be *exactly* equal to the prior mode $\hat{\boldsymbol{\beta}}_k^{\text{MLE}}$.

To finish the prior specification, we consider two types of hyper-priors for γ. The first is that γ follows an exponential distribution.

$$p_E(\gamma) = a \, \exp(-a\gamma). \tag{26}$$

The second is the improper Jeffreys prior:

$$p_J(\gamma) \propto 1/\gamma. \tag{27}$$

It is well known that this corresponds to a uniform prior for $\log(\gamma)$.

Under the exponential hyper-prior for γ, the full posterior becomes:

$$p(\boldsymbol{\beta}_k^{(i)}, \gamma | y_k^{(i)}) \propto p(\mathbf{y}_k^{(i)} | \boldsymbol{\beta}_k^{(i)}) p(\boldsymbol{\beta}_k^{(i)} | \gamma) p_E(\gamma)$$

$$\propto \left\{ \prod_{w=1}^{W} \left(\frac{\exp(\beta_{kw}^{(i)})}{\sum_{i=1}^{W} \exp(\beta_{ki}^{(i)})} \right)^{y_{kw}} \right\} \left\{ \prod_{w=2}^{W} \frac{\gamma}{2} \exp\left(-\gamma \left| \beta_{kw}^{(i)} - \hat{\beta}_{kw}^{\text{MLE}} \right| \right) \right\} a \exp(-a\gamma) \qquad (28)$$

and under the Jeffreys hyper-prior for γ, the full posterior becomes:

$$p(\boldsymbol{\beta}_k^{(i)}, \gamma | y_k^{(i)}) \propto p(\mathbf{y}_k^{(i)} | \boldsymbol{\beta}_k^{(i)}) p(\boldsymbol{\beta}_k^{(i)} | \gamma) p_J(\gamma)$$

$$\propto \left\{ \prod_{w=1}^{W} \left(\frac{\exp(\beta_{kw}^{(i)})}{\sum_{i=1}^{W} \exp\left(\beta_{ki}^{(i)} \right)} \right)^{y_{kw}} \right\} \left\{ \prod_{i=2}^{W} \frac{\gamma}{2} \exp\left(-\gamma \left| \beta_{kw}^{(i)} - \hat{\beta}_{kw}^{\text{MLE}} \right| \right) \right\} 1/\gamma. \qquad (29)$$

The sharp mode of the Laplace prior means that words whose partisanship is not clear will receive partisan contrasts that are exactly zero. This is useful for feature selection. The nondifferentiability at the mode does considerably increase the difficulty and computational overhead associated with estimation. This procedure is discussed in the Appendix.

With the Laplace model, we do not downweight (further) by variance, but look directly at the $\hat{\delta}_{kw}^{(D-R)}$. Figure 6 shows the results for our running example. Note first that Figure 6 is largely blank, with the vast majority of words having estimates shrunk to zero. The estimates are dominated by a few words: *kill*, *babi*, *procedur*, *child*, *mother*, and *abort* for Republicans; *women*, *woman*, *decis*, *famili*, *friend*, and *right* for Democrats. A higher smoothing parameter would result in even fewer nonzero features; a smaller one would result in more.[19] As with the above, the effect of any particular smoothing parameter will be affected by the relative volume of speech and the polarization of word use. This is ideal for an application where we value parsimony in the output.

This has a few drawbacks. Most notably, the nondifferentiability of the Laplace prior dramatically increases computational complexity even for point estimates. Extending the model—dynamically, for example—is also difficult. These properties make this approach less than ideal for feature evaluation and applications where the estimates feed forward to another analysis. With a sufficiently high smoothing parameter, however, this is an excellent choice for feature selection.

4 Applications of Feature Selection and Evaluation

In this section we explore the broader usefulness of our proposed models for analyzing substantive questions relating to issue framing, representation, polarization, and dimensionality. Although the merging of the text-as-data approach with shrinkage methods could fill several articles for each of these topics, our goal is to illustrate how future research might utilize these methods to answer substantive questions in political science. In turn, we show how our

[19] For these calculations, we define the prior with $a = 100$.

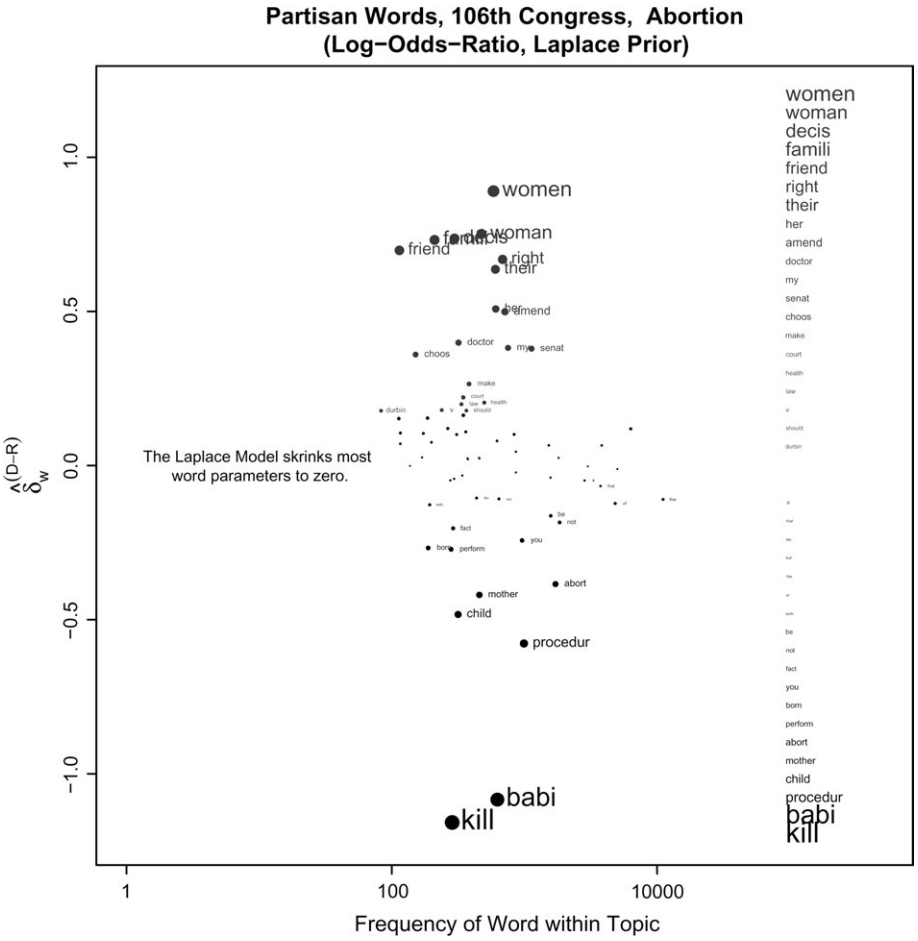


Fig. 6 Feature evaluation and selection using $\hat{\delta}_{kw}^{(D-R)}$. Plot size is proportional to evaluation weight, $\hat{\delta}_{kw}^{(D-R)}$. The top 20 Democratic and Republican words are labeled and listed in rank order to the right.

methods can be easily expanded to analyze different issues, extra-partisan grouping, change over time, and political divisions across more than two categories.

We begin by expanding the issues within which we search for partisan frames. Specifically, we are interested in whether our methods find frames where they are likely to be (as in the abortion topic) and do not find partisan frames where there are likely to few. Next, we show that distinctions in word use can be identified not only across party but across gender (within party) also, as has been argued in the representation literature (Williams 1998; Diaz 2005). This is followed by examples that illustrate the changes in partisan word usage over time. Finally, we explore how feature selection and weighting across geographic units can be used to understand the underlying dimensionality of politics.

4.1 *Partisan framing*

Chong and Druckman (2007, 106) note that the analysis of ''frames in communication'' is an endeavor that ''has become a virtual cottage industry.'' In such analyses, ''an initial set of frames for an issue is identified inductively to create a coding scheme'' (Chong and

Druckman 2007, 107). The result of this initial stage is something like the 65-argument coding scheme developed by Baumgartner et al. (2008) to exhaustively capture the pro- and antiframes used in debate over the death penality (Baumgartner et al. 2008, 103–12, 243–51). These then serve as the basis for a content analysis of relevant texts, like newspaper articles, to evaluate variations in frame across source and across time. The feature selection techniques we describe here can be useful for such a purpose.

Consider the topic of taxes. Figure 7 shows the results of applying the Laplace model, again in the 106th Congress. Note the more compressed *y*-axis, relative to the abortion analysis of Figure 6, indicating that during this period taxation was a considerably more partisan topic. The prominent features identify the most prominent partisan frames. Republicans were pushing for elimination of the *death tax*, lowering of *capital gains taxes*, elimination of the *marriage penalty*, and *flatter brackets*; Democrats advocated for a tax credit for *prescription drug coverage*, for *college*, and against lowering the *estate tax* or giving *breaks* to the *wealthiest*. Indeed, *death tax* and *marriage penalty* are two of the more famous and successful examples of party-coordinated language choice.

Such features could then be used to evaluate something like media use of taxation frames. A full analysis is well beyond our scope here, but a brief example is suggestive of the potential here. We conducted Lexis-Nexis searches of two newspapers, *The New York Times* and *The Washington Times*, and two broadcast news sources, *National Public Radio* and *Fox News*, widely thought to be, relatively speaking, at different partisan extremes. We counted articles or transcripts that used variants of the word *tax* along with variants of Democratic features (*estate*, *prescription*, *wage*, *wealthiest*) and four prominent Republican frames (*death*, *bracket*, *flat*, *penalty*), for 1999–2000. Sticking to the familiar metric of odds ratios, the odds of *The New York Times* using these Democratic over Republican frames are roughly 20% higher than those of the *Washington Times*. Similarly, the odds of NPR using Democratic over Republican frames are roughly 63% higher than Fox News.[20] This is a toy example, of course, but suggests possibilities for a variety of more interesting applications across source, time, or political topic.

More central to our motivations for the Laplace model are the advantages of regularization. In this context, one notable advantage is that the model will not suggest differences where there are none. For example, our Senate analysis finds a large number of ''sports'' speeches congratulating a home state team. We would not expect these to have partisan structure. If we apply exactly the same Laplace model as above to these speeches, the model finds only a tiny handful of words with very small partisan content.[21] We would correctly conclude that there is virtually no partisan content to such speech.

4.2 *Women's Representation*

Not only is it useful to estimate our model on different topics but we can also analyze extra-partisan distinctions. For example, a long-standing question in the representation literature concerns the relative roles of descriptive representation, symbolic representation, and substantive representation instantiated by diversity in the demographics of

[20]The counts of *estate*, *prescription*, *wage*, *wealthiest*, *death*, *bracket*, *flat*, *penalty*, respectively, are (2078, 620, 1159, 208, 1280, 164, 431, 743) for *The New York Times*, (1032, 376, 692, 84, 731, 189, 399, 720) for *The Washington Times*, (172, 214, 224, 68, 209, 47, 68, 174) for National Public Radio, and (167, 362, 238, 54, 450, 55, 197, 281) for Fox News.

[21]Specifically, *football* is very slightly Republican, whereas *great* and *women* are slightly Democratic. The graphic for this is omitted because it is largely blank, as expected, but is available in the web appendix.

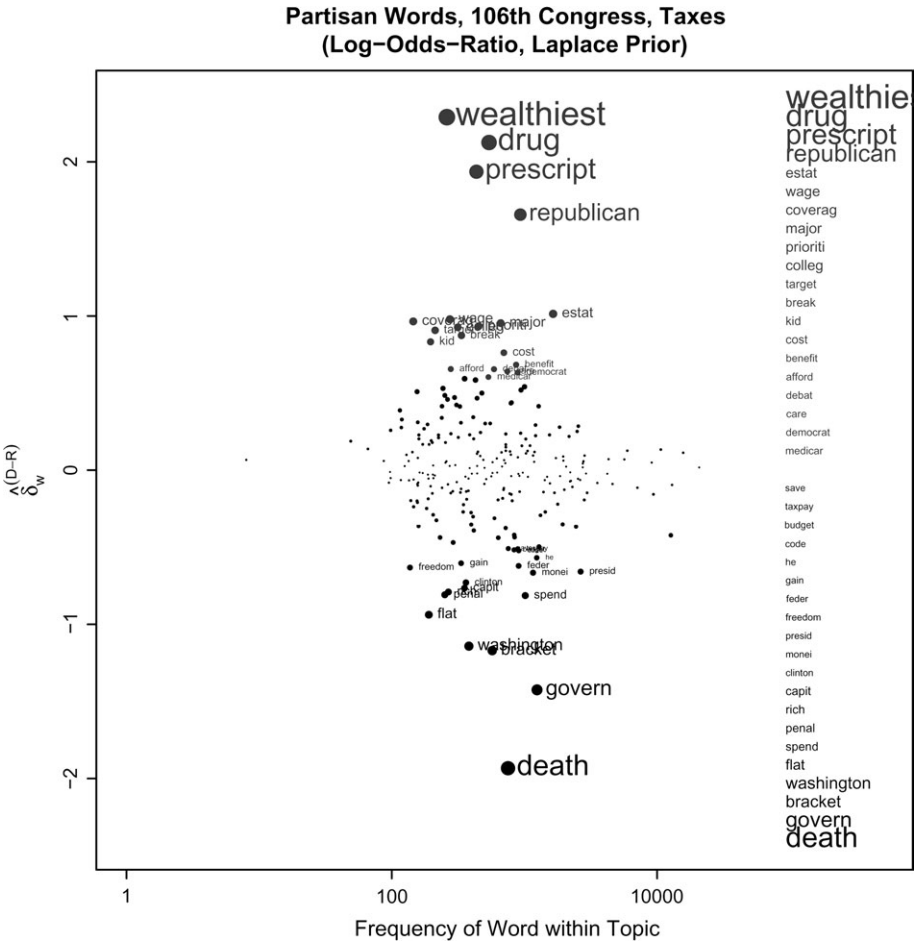


Fig. 7 Feature evaluation and selection using $\hat{\delta}_{kw}^{(D-R)}$. Plot size is proportional to evaluation weight, $\hat{\delta}_{kw}^{(D-R)}$. The top 20 Democratic and Republican words are labeled and listed in rank order to the right.

a legislature. One of the more prominent is the concern over whether women legislators behave differently than male legislators and whether they offer a fundamentally different form of representation by speaking ''in a different voice'' (Gilligan 1993; Williams 1998; Diaz 2005).

These models are natural for such questions. Not only can we look at the relative differences in language use between men and women, as we did with party, but we can also do so while controlling for party. This is important because of a general gender gap across the two parties both in core values (Rokeach 1973) and in framing (Lakoff 2004). Democrats are both more likely than Republicans to address ''women's issues,'' and also have more women members serving in the Senate than do the Republicans.

Figure 8 shows a simple example of gender differences, within Democrats, in language use for our running example of abortion in the 106th Senate, using the informative Dirichlet model. The most prominent and obvious difference between female and male Senators is the distinction between discussing the adult involved in an abortion as a *woman*, or as a *mother*. We keep it simple here, but more elaborate types of cross-nesting of categories,

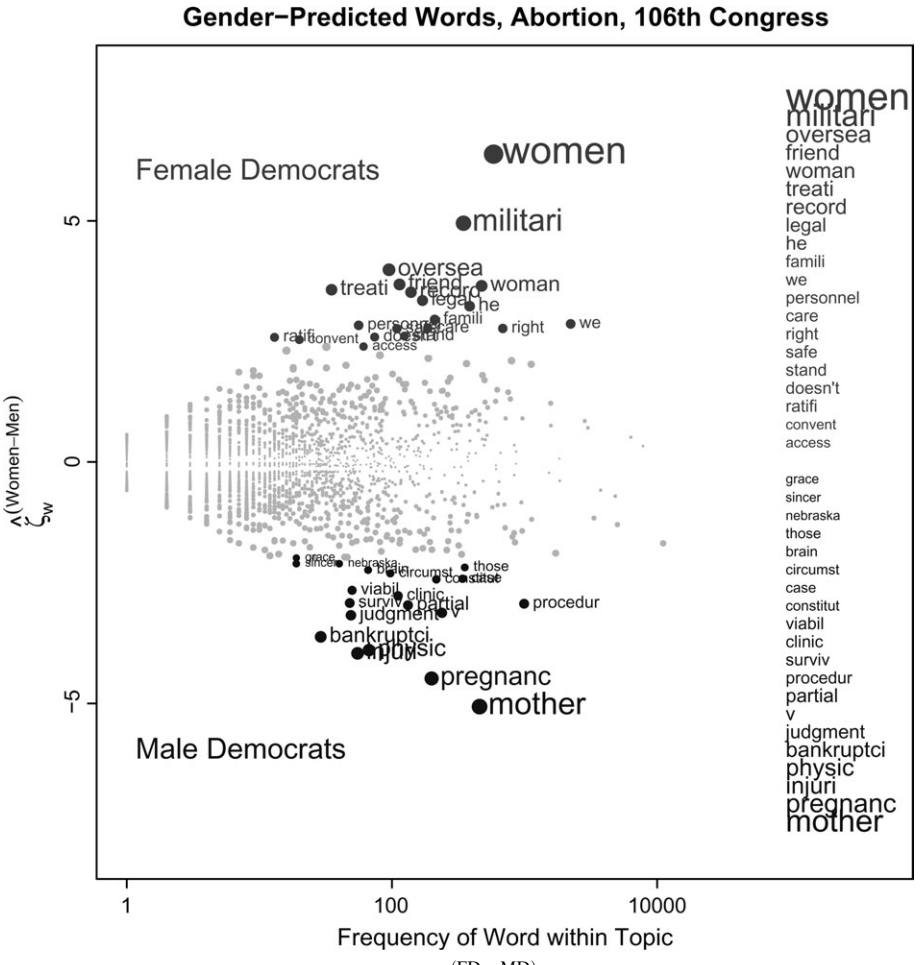


Fig. 8 Feature evaluation and selection using $\hat{\zeta}_{kw}^{(\mathrm{FD}-\mathrm{MD})}$. Plot size is proportional to evaluation weight, $\hat{\zeta}_{kw}^{(\mathrm{FD}-\mathrm{MD})}$. Top 20 Female (Democrat) and Male (Democrat) words are labeled and listed in rank order.

as well as continuous covariates (like urbanity of state), are simple extensions of the Dirichlet model.

4.3 *Dynamics: Issue Evolutions and Semantic Realignment*

We can also extend our analysis across time to analyze dynamics across partisan (or extra-partisan) word use. One of the fundamental questions in American politics has concerned the existence, content, timing, and mechanisms of parties shifting against the ideological background (Carmines and Stimson 1990; Poole and Rosenthal 1997). A large-scale example clearly matching the criteria for a ''realignment'' or an ''issue evolution'' is the reversal of the parties on race and civil rights issues over the course of the 20th century; a small-scale example, which may or may not meet the criteria, is the polarization over time of the parties on the issue of abortion.

Empirical investigation of such historical phenomena is difficult. One approach, exemplified by Mucciaroni and Quirk's *Deliberative Choices* (2006) is to examine in great detail

the traceable behaviors—in this case, Congressional speeches, testimony, etc.—for a particular policy debate held over a relatively short time scale. A second approach, exemplified by Carmine and Stimson's *Issue Evolution* (1990) and similar work that followed its lead (Adams 1997), looks for changes across more abstract traceable behaviors—for example, rollcall votes in Congress, survey responses—in a single issue area over a long time scale: race for Carmines and Stimson, abortion and others for later work. A third approach, exemplified by Poole and Rosenthal's *Congress* (1997), looks at a single behavior—Congressional rollcall votes—across all issues and long time scales, looking for abstract changes in the partisan relationship, such as the dimensionality of a model required to explain the behaviors.

Speech offers interesting potential leverage on the problem. We have seen that static snapshots can illuminate the content of a particular partisan conflict. If these techniques can be extended dynamically, we can look directly at how the content of partisan conflict changes.

The analytic solution of the Dirichlet model makes dynamic extensions straightforward. First, the y are extremely noisy on a day-to-day basisso we apply a smoother to the data.[22] Then we calculate *zeta* over a moving time window of the data.

A genuine partisan realignment or issue evolution would be evidenced by massive shifts in the partisanship of language. That is, a necessary condition for an issue evolution would be for some words to switch sides, to flip sign in our measure. This generally occurred (or been agreed to have occurred) only over fairly long time frames. So, over the relatively short time frame of these data, 8 years, we would not expect to find much that would qualify as such. However, we do find many microscale movements that suggest such analyses can prove fruitful.

For example, we find several instances where Republicans ''own'' an issue or frame while in opposition to Clinton are successful in obtaining a policy change during the Bush honeymoon period of early 2001 or in the rally period after 9/11, only to find Democrats ''owning'' the issue when the policy becomes unpopular. Dramatic examples include *debt* when discussing the budget, *oil* when discussing energy policy, and *Iraq* when discussing defense policy.

Figure 9 shows an estimate of the dynamic partisanship[23] of *iraq* on the topic of defense, an example which provides a unique window into the dynamics of partisan conflict. In order to observe shifts in the tectonic plates of partisan conflict, we need both salience (the parties and presumably the public must care) and some exogenous shock or change in the status quo (to motivate change). Iraq supplies both of these, and in turn, we see relative nonpartisanship pre-9/11 shift to a Republican talking point post-9/11 only to see events on the ground and the cost of the war increase Democratic leverage on the issue. This technique may be of particular importance in the future for any analyst interested in tracing whether the Iraq issue settles into a nonpartisan or renewed partisan equilibrium.

Similarly, dynamics can reveal how institutions affect the ''ideological'' justifications utilized by parties. Figure 10 illustrates one example, the dynamic partisanship of the word

[22]Specifically, we smooth the data by first calculating a b-day moving average, m, of word use, and apply an exponential smoother, with smoothing factor A: $m^{(i)}_{kwt} = \sum_{\tau=t-b}^{t} y^{(i)}_{kw\tau}$; $s^{(i)}_{kw(b+1)} = m^{(i)}_{kw(b+1)}$; $s^{(i)}_{kwt} = Am^{(i)}_{kwt} + \left(1-A\right)s^{(i)}_{kw(t-1)}$. From this we calculate the ζ_{kwt} for each day, using the $s^{(i)}_{kwt}$ where we would have used the $y^{(i)}_{kwt}$ previously.

[23]These particular graphs reflect daily estimates based on a $b = 180$ day (6 months) moving average component and an $A = .01$ smoothing factor.

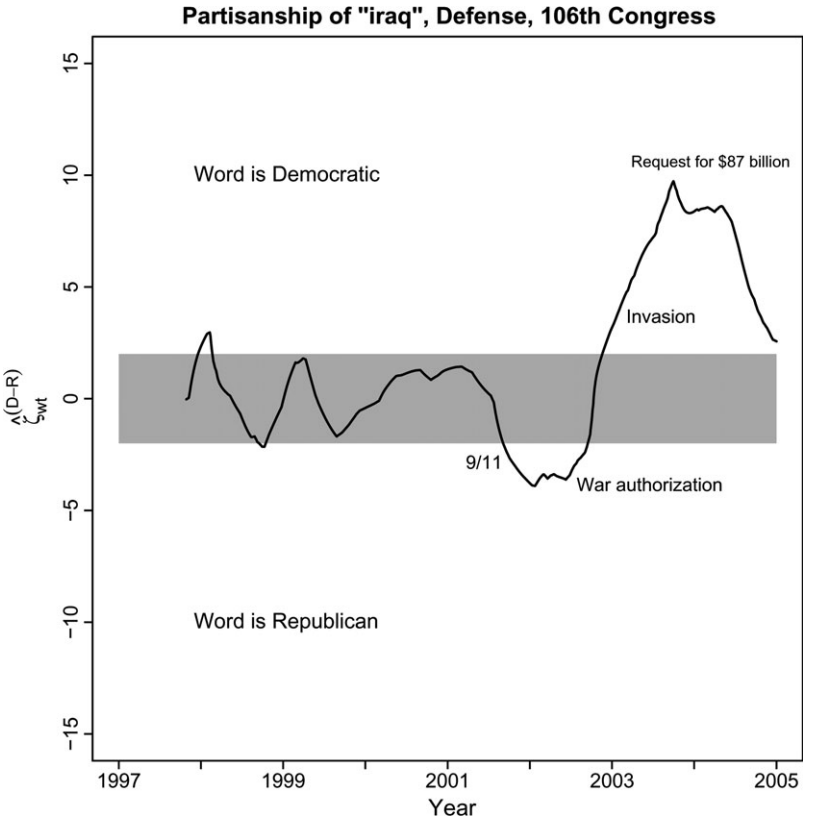


Fig. 9 Dynamic partisanship of ''iraq'' in the context of defense.

vote on the issue of judicial nominations. During the Clinton era, Republicans consistently blocked his judicial nominees. Democrats demanded that votes be held, with increasing urgency as the Clinton presidency neared its end. During the Bush era, Democrats consistently filibustered his judicial nominees, and Republicans discovered a deeply held belief in presidential deference and up-or-down votes. On the theoretical issue of majority rule, where you stand depends on where you sit.[24]

4.4 *Polarization*

Partisan polarization has been a central focus for much recent scholarship American politics.(McCarty et al. 2006; Sinclair 2006; Theriault 2008) The primary measure of polarization at the elite level has come from rollcall analysis, following Poole and Rosenthal (1997). Variation in language use offers an alternative measure, one that can be differentiated across political topics and at a finer dynamic grain.

[24]We can also look at the entire vocabulary over time in an animation. An example for the topic of defense is given here: http://qssi.psu.edu/PartisanDefenseWords.html.

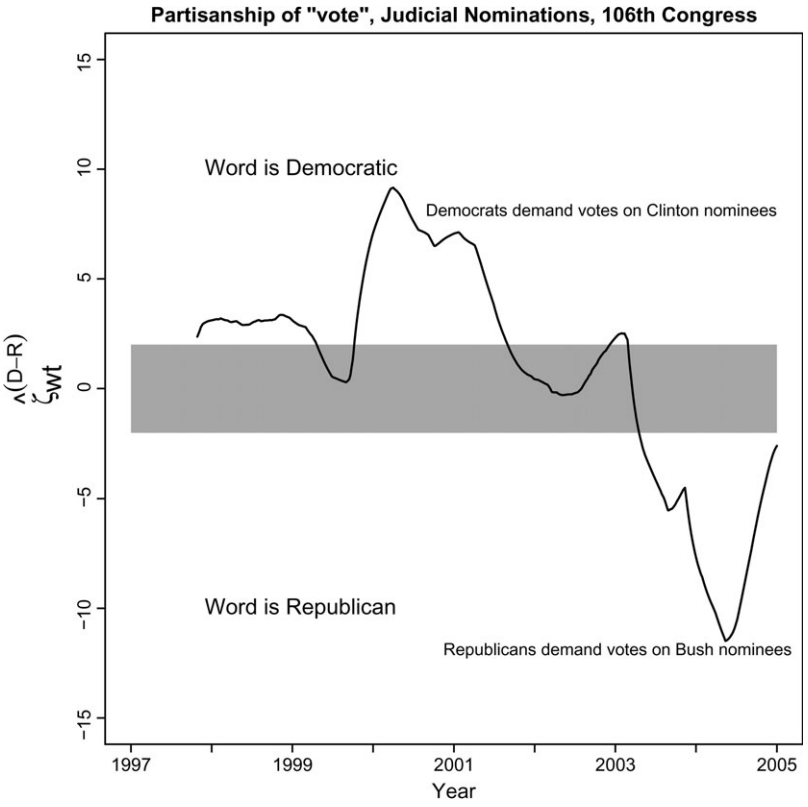


Fig. 10 Dynamic partisanship of ''vote'' in the context of judicial nominations.

A simple measure of polarization is the variance of $\zeta_{kwt}^{(D-R)}$ across all words at any given time, *t*. Figure 11 illustrates this for four topics over the period: budget, defense, education, and sports. Defense is particularly interesting, reaching near zero polarization in the aftermath of 9/11 and dramatically increasing through and after the invasion of Iraq. Note also the buildup and then collapse of polarization around education peaking after the passage of No Child Left Behind, the apparent annual (out of phase) cycles in polarization around the salient topics, and the consistently low partisan polarization of sports.

4.5 *Dimensionality*

Another central concern of the Congressional literature is the dimensionality of the political space in which parties and politicians position themselves. Spatial theorists have argued that underlying preferences must be distributive and multidimensional (Aronson and Ordeshook 1981) and that parties and institutions are reshaped to manage the inherent instability (Black 1958; Riker 1986). Rollcall-based work suggests that a single ideological dimension accounts for virtually all Congressional divisions (Poole 2005). But if bills are structured by a partisan agenda-setter, in the manner of Cox and McCubbins (2005) for example, there will never be enough cross-cutting rollcall votes to detect underlying

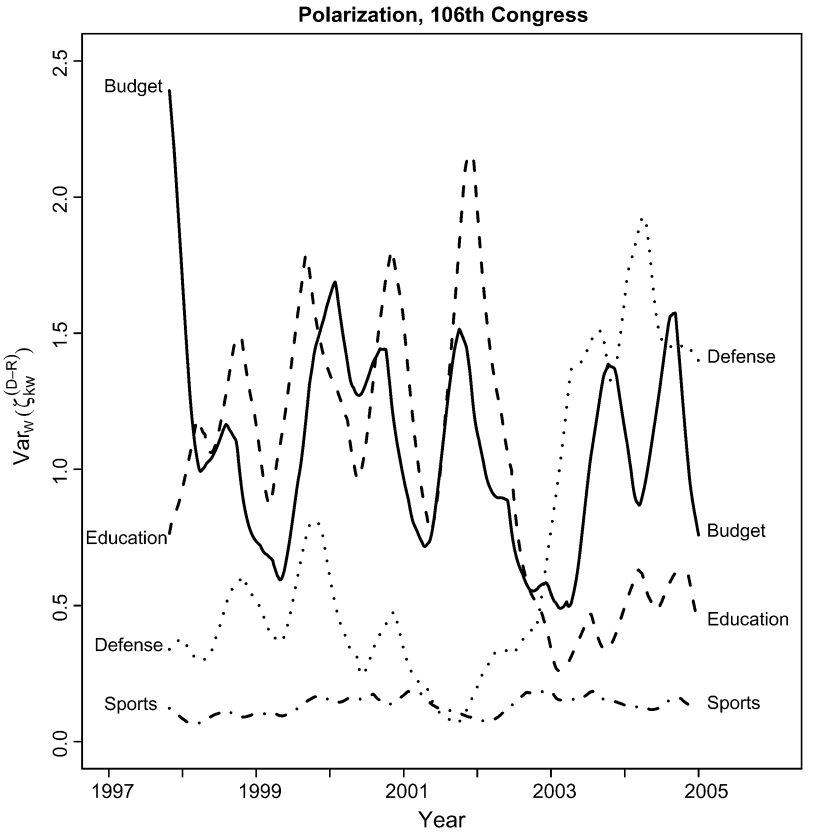


Fig. 11 The dynamics of language polarization.

49-dimensional state versus state preferences. Speech, on the other hand, can reveal such subtleties.

To give one example, Figure 12 shows the state-predicted words on the issue of public lands and water, for four states with a range of likely interests in the subject. The delegations from each state each have some distinctive speech on the issue, with prominent words including the state names, major features (Great Lakes, Finger Lakes), and primary concerns (e.g., invasive species in Michigan, emissions in New York, or hunting in Alabama). There is also clear variation in the intensity of preferences, with great intensity on multiple issues from the Alaska delegation, but limited intensity from the Alabama delegation. This is suggestive that there are measurable distributive (and therefore multidimensional) preference differences across Senators (which are unlikely to be present in rollcall votes once partisan coalitions have been built around particular bills).

5 Discussion and Conclusion

Contemporary representative democratic systems are shaped, perhaps defined, by the relationships among citizens, elected representatives, political parties, and the substance of politics. Creating tools to analyze these relationships, and how they interconnect dynamically over the life of a democracy, is a fundamental challenge to political methodology. One bottleneck has been our measurement and modeling of the relationship between

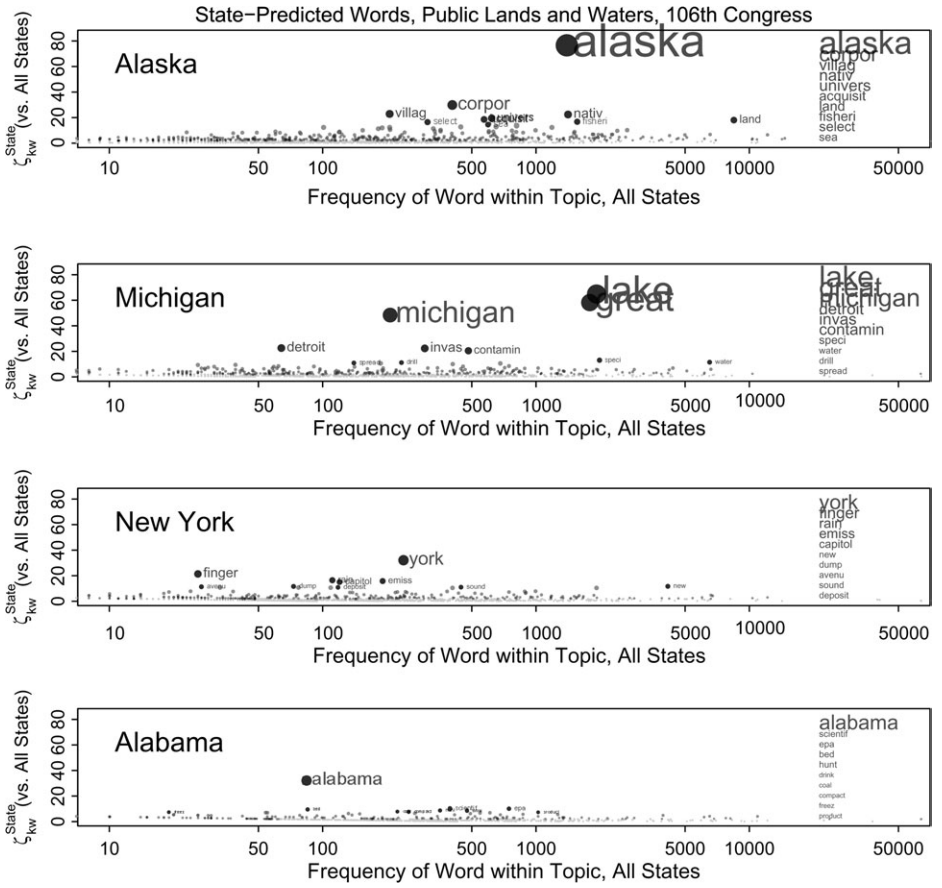


Fig. 12 Feature evaluation and selection using $\hat{\zeta}_{kw}^{(\text{State})}$. Plot size is proportional to evaluation weight, $\hat{\zeta}_{kw}^{(\text{State})}$. Top 10 state-predicted words for each state are labeled and listed in rank order.

political substance and partisan dynamics. Efforts to systematically analyze political conflict, especially in the American context, have previously focused on behaviors that are easily quantified but obscure substance (e.g., votes). More recent attempts to investigating political dynamics that summarize and organize text have fallen into other traps, including inefficiency (e.g., classification techniques), and overfitting idiosyncratic group differences in low- or high-frequency words (e.g., *tf.idf* or difference of proportions). These problems lead researchers to select and weight feature/word lists that have low semantic validity or vary wildly depending on arbitrary decisions.

Here we have highlighted the problems with these existing techniques for feature selection and evaluation as well as presented two potential solutions. By using a model-based approach with Bayesian shrinkage and regularization, we argue, analysts can at least partially side-step previous pitfalls. Most importantly, the word lists and weights produced by our preferred techniques reduce the emphasis placed on words used very frequently across all groups as well as those words that are very infrequently used across both groups. Our methods also do not necessitate that the researcher create an ad hoc filter that erases words from the political vocabulary. Instead, Bayesian shrinkage supplies rules that are consistently applied across the whole of the vocabulary, allowing consistent distinctions across groups to become apparent.

We are, of course, ultimately concerned with whether these tools are useful for political analysis. Our examples have shown that questions relating to issue framing, representation, polarization, and dimensionality can be explicitly explored with text data through our techniques. We expect that others working within these substantive fields will be able to carry out more detailed and extensive applications with these tools.

The next step is to further refine and craft methods that will be useful for unlocking the exciting potential of political texts. Our model-based regularization approach is one such attempt. Although off-the-shelf algorithms and techniques have proven valuable, we must look under the hood to know if we are validly measuring the concepts we care about. There is much that is theoretically and conceptually unique about the production of language in politics, and there is a need for new methods to be developed and applied accordingly.

Funding

National Science Foundation (grant BCS 05-27513 and BCS 07-14688).

Appendix—Estimation of the Laplace Model

Maximization of the posterior densities in equations 28 and 29 is complicated by the discontinuities in the partial derivatives that are introduced by the Laplace prior. We make use of a version of the algorithm of Shevade and Keerthi (2003) that has been modified to work with a multinomial likelihood. Calculation of the relevant derivatives is tedious but not difficult. In addition to moving from a Bernoulli to a multinomial likelihood, we also extend the work of Shevade and Keerthi (2003) and Cawley and Talbot (2006) to allow an exponential prior for γ.

Marginalizing over γ with a Jeffreys Prior

Using results from Cawley and Talbot (2006), it is easy to show how the value of $\boldsymbol{\beta}_k^{(i)}$ that maximizes

$$p(\boldsymbol{\beta}_k^{(i)}|y_k^{(i)}) \propto \int p(\mathbf{y}_k^{(i)}|\boldsymbol{\beta}_k^{(i)}) p(\boldsymbol{\beta}_k^{(i)}|\gamma) p_J(\gamma) \mathrm{d}\gamma \tag{A1}$$

can be calculated via a modified version of the iterative approach of Shevade and Keerthi (2003) in which, at each iteration, a working version, $\tilde{\gamma}$, of γ is defined to be:

$$\tilde{\gamma} \equiv W / \sum_{w=1}^{W} \left| \beta_{kw}^{(i)} - \hat{\beta}_{kw}^{\mathrm{MLE}} \right|, \tag{A2}$$

and then the maximization over $\beta_{kw}^{(i)}$ is carried out conditional on $\tilde{\gamma}$.

Marginalizing over γ with an Exponential Prior

Using ideas similar to those described in the previous subsection to show how the value of $\boldsymbol{\beta}_k^{(i)}$ that maximizes

$$p(\boldsymbol{\beta}_k^{(i)}|y_k^{(i)}) \propto \int p(\mathbf{y}_k^{(i)}|\boldsymbol{\beta}_k^{(i)})p(\boldsymbol{\beta}_k^{(i)}|\gamma)p_E(\gamma)\mathrm{d}\gamma \tag{A3}$$

can be calculated via a modified version of the iterative approach of Shevade and Keerthi (2003) in which, at each iteration, a working version, $\tilde{\gamma}$, of γ is defined to be:

$$\tilde{\gamma} \equiv \frac{(W+1)}{\left(a + \sum_{w=1}^{W} \left|\beta_{kw}^{(i)} - \hat{\beta}_{kw}^{\mathrm{MLE}}\right|\right)} \tag{A4}$$

and then the maximization over $\beta_{kw}^{(i)}$ is carried out conditional on γ.

Maximizing over γ with an Exponential Prior

It is also possible to show that values of $\beta_k^{(i)}$ and γ that maximizes

$$p(\boldsymbol{\beta}_k^{(i)}, \gamma|\mathbf{y}_k^{(i)}) \propto p(\mathbf{y}_k^{(i)}|\boldsymbol{\beta}_k^{(i)})p(\boldsymbol{\beta}_k^{(i)}|\gamma)p_E(\gamma) \tag{A5}$$

can be calculated via a modified version of the iterative approach of Shevade and Keerthi (2003) in which, at each iteration, a local conditional maximizer for γ is:

$$\hat{\gamma} \equiv \frac{W}{\left(a + \sum_{w=1}^{W} \left|\beta_{kw}^{(i)} - \hat{\beta}_{kw}^{\mathrm{MLE}}\right|\right)}, \tag{A6}$$

and then the maximization over $\beta_{kw}^{(i)}$ is carried out conditional on $\hat{\gamma}$.

Note that when $a = 0$, the results from this procedure are equivalent to those from the setup where γ is given a Jeffreys prior and then integrated out of the posterior.

References

Adams, Greg D. 1997. Abortion: Evidence of issue evolution. *American Journal of Political Science* 41(3):718–37.

Agresti, Alan. 2002. *Categorical data analysis*. 2nd ed. Hoboken, NJ: Wiley.

Aizawa, Akiko. 2003. An information-theoretic perspective of tf-idf measures. *Information Processing and Management* 39(1):45–65.

Aronson, Peter H., and Peter C. Ordeshook. 1981. Regulation, redistribution, and public choice. *Public Choice* 37(1):69–100.

Baumgartner, Frank R., Suzanna L. DeBoef, and Amber E. Boydstun. 2008. *The decline of the death penalty and the discovery of innocence*. Cambridge: Cambridge University Press.

Black, Duncan. 1958. *The theory of committees and elections*. Cambridge: Cambridge University Press.

Breiman, Leo. 2001. Random forests. *Machine Learning* 45(1):5–32.

Carmines, Edward, and James Stimson. 1990. *Issue evolution*. New York: Princeton University Press.

Cawley, Gavin C., and Nicola L. C. Talbot. 2006. Gene selection in cancer classification using sparse logistic regression with Bayesian regularization. *Bioinformatics* 22(19):2348–55.

Chong, Dennis, and James N. Druckman. 2007. Framing theory. *Annual Review of Political Science* 10:103–26.

Cox, Gary W., and Mathew D. McCubbins. 2005. *Setting the agenda: Responsible party government in the US House of Representatives*. Cambridge: Cambridge University Press.

Diaz, Mercedes Mateo. 2005. *Representing women? Female legislators in West European parliaments*. Colchester, UK: ECPR Press Monographs.

Diermeier, Daniel, Jean-Francois Godbout, Bei Yu, and Stefan Kaufmann. 2007. *Language and ideology in congress*. Chicago, IL: Midwestern Political Science Association.

Emily, Senay, Eli H. Newberger, and Rob Waters. 2004. *From boys to men*. New York: Simon and Schuster.

Ericson, Matthew. 2008. The words they used. *The New York Times,* September 4, http://www.nytimes.com/interactive/2008/09/04/us/politics/20080905_WORDS_GRAPHIC.html (accessed November 1, 2008).

Freund, Yoav, and Robert Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* 55(1):119–39.

Gentzkow, Matthew, and Jesse M. Shapiro. 2006. *What drives media slant? Evidence from U.S. daily newspapers*. University of ChicagoTechnical report.
Gilligan, Carol. 1993. *In a different voice: Psychological theory and women's development*. Cambridge, MA: Harvard University Press.
Hand, David J. 2006. Classifier technology and the illusion of progress. *Statistical Science* 21(1):1–14.
Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2001. *The elements of statistical learning*. New York: Springer.
Hiemstra, Djoerd. 2000. A probabilistic justification for using tfxidf term weighting in information retrieval. *International Journal on Digital Libraries* 3:131–9.
Hillard, Dustin, Stephen Purpura, and John Wilkerson. 2007. Computer-assisted topic classification for mixed-methods social science research. *Journal of Information Technology and Politics* 4(4):31–46.
Hopkins, Daniel, and Gary King. 2007. Extracting systematic social science meaning from text. Chicago, IL: Midwestern Political Science Association.
Jurafsky, Daniel, and James H. Martin. 2000. *Speech and language processing*. Upper Saddle River, NJ: Prentice Hall.
Kleinfeld, Judith, and Andrew Kleinfeld. 2004. Cowboy nation and American character. *Society* 41(3):43–50.
Krippendorff, Klaus. 2004. *Content analysis: An introduction to its methodology.* 2nd ed. New York: Sage.
Lakoff, George. 2004. *Don't think of an elephant! Know your values and frame the debate*. White River Junction, VT: Chelsea Green Publishing.
Laver, Michael, Kenneth Benoit, and John Garry. 2003. Extracting policy positions from political texts using words as data. *American Political Science Review* 97:311–31.
Lowe, Will. 2008. Understanding Wordscores. *Political Analysis* doi:10.1093/pan/mpn004 (accessed October 4, 2008).
Manning, Christopher D., and Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. Cambridge, MA: MIT Press.
Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. 2008. *An introduction to information retrieval*. Cambridge: Cambridge University Press.
Many Eyes (IBM). *Tag Cloud Guide*. http://manyeyes.alphaworks.ibm.com/manyeyes/page/Tag_Cloud.html (accessed July 13, 2008).
McCaffrey, Dawn, and Jennifer Keys. 2008. Competitive framing processes in the abortion debate: Polarization-vilification, frame saving, and frame debunking. *Sociological Quarterly* 41(1):41–61.
McCarty, Nolan, Keith T. Poole, and Howard Rosenthal. 2006. *Polarized America: The dance of ideology and unequal riches*. Cambridge, MA: MIT Press.
Monroe, Burt L., Cheryl L. Monroe, Kevin M. Quinn, Dragomir Radev, Michael H. Crespin, Michael P. Colaresi, Jacob Balazar, and Steven P. Abney. 2006. *United States congressional speech corpus*. State College, PA: Pennsylvania State University.
Monroe, Burt L., and Ko Maeda. 2004. Rhetorical ideal point estimation: Mapping legislative speech. *Society for Political Methodology.* Palo Alto: Stanford University.
Monroe, Burt L., Michael P. Colaresi, and Kevin M. Quinn. *An Animated History of Senate Debate on Defense, 1997–2004*. http://qssi.psu.edu/PartisanDefenseWords.html (accessed November 1, 2008).
Mucciaroni, Gary, and Paul J. Quirk. 2006. *Deliberative choice: Debating public policy in congress*. Chicago, IL: University of Chicago Press.
Nunberg, Geoffrey. 2006. *Talking right: How conservatives turned liberalism into a tax-raising, latte-drinking, sushi-eating, volvo-driving, New York Times-Reading, Body-Piercing, Hollywood-Loving, Left-Wing Freak Show.* New York: Public Affairs.
Poole, Keith T. 2005. *Spatial models of parliamentary voting*. New York, Cambridge: Cambridge University Press.
Poole, Keith T., and Howard Rosenthal. 1997. *Congress: A political-economic history of roll-call voting*. Oxford: Oxford University Press.
Porter, Martin F. 2001. The English (Porter2) stemming algorithm. http://snowball.tartarus.org/algorithms/english/stemmer.html (accessed July 3, 2008).
Quinn, Kevin, Burt L. Monroe, Michael Colaresi, Michael Crespin, and Dragomir Radev. 2006. *How to analyze political attention with minimal assumptions and costs*. Davis, CA: Society for Political Methodology.
Riker, William H. 1986. *The art of political manipulation*. New Haven, CT: Yale University Press.
Rokeach, Milton. 1973. *The nature of human values*. New York: The Free Press.
Sacerdote, Bruce, and Owen Zidar. 2008. Campaigning in poetry: Is there information conveyed in the candidates' choice of words. Technical report, University of Dartmouth.
Schonhardt-Bailey, Cheryl. 2008. The congressional debate on partial-birth abortion: Constitutional gravitas and moral passion. *British Journal of Political Science* 38(3):383–410.

Shevade, S. K., and S. S. Keerthi. 2003. A simple and efficient algorithm for gene selection using sparse logistic regression. *Bioinformatics* 19(17):2246–53.

Sinclair, Barbara. 2006. *Party wars: Polarization and the politics of national policy making*. Norman, OK: University of Oklahoma Press.

Slapin, Jonathan B., and Sven-Oliver Proksch. 2008. A scaling model for estimating time-series party positions from texts. *American Journal of Political Science* 52(3):705–22.

Theriault, Sean M. 2008. *Party polarization in congress*. Cambridge: Cambridge University Press.

Vapnik, Vladimir. 2001. *The nature of statistical learning theory.* New York: Springer-Verlag.

Williams, Melissa S. 1998. *Voice, trust and memory: Marginalized groups and the failings of liberal representation*. Princeton, NJ: Princeton University Press.

Williams, Peter M. 1995. Bayesian regularization and pruning using a Laplace prior. *Neural Computation* 7(1):117–43.

Wordle. 2008. http://wordle.net (accessed July 13, 2008).

Yu, Bei, Stefan Kaufmann, and Daniel Diermeier. 2008. Classifying party affiliation from political speech. *Journal of Information Technology and Politics* 5(1):33–48.

1 *Boolean retrieval*

INFORMATION RETRIEVAL

The meaning of the term *information retrieval* can be very broad. Just getting a credit card out of your wallet so that you can type in the card number is a form of information retrieval. However, as an academic field of study, *information retrieval* might be defined thus:

> Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

As defined in this way, information retrieval used to be an activity that only a few people engaged in: reference librarians, paralegals, and similar professional searchers. Now the world has changed, and hundreds of millions of people engage in information retrieval every day when they use a web search engine or search their email.[1] Information retrieval is fast becoming the dominant form of information access, overtaking traditional database-style searching (the sort that is going on when a clerk says to you: "I'm sorry, I can only look up your order if you can give me your Order ID").

IR can also cover other kinds of data and information problems beyond that specified in the core definition above. The term "unstructured data" refers to data which does not have clear, semantically overt, easy-for-a-computer structure. It is the opposite of structured data, the canonical example of which is a relational database, of the sort companies usually use to maintain product inventories and personnel records. In reality, almost no data are truly "unstructured". This is definitely true of all text data if you count the latent linguistic structure of human languages. But even accepting that the intended notion of structure is overt structure, most text has structure, such as headings and paragraphs and footnotes, which is commonly represented in documents by explicit markup (such as the coding underlying web

1. In modern parlance, the word "search" has tended to replace "(information) retrieval"; the term "search" is quite ambiguous, but in context we use the two synonymously.

pages). IR is also used to facilitate "semistructured" search such as finding a document where the title contains Java and the body contains threading.

The field of information retrieval also covers supporting users in browsing or filtering document collections or further processing a set of retrieved documents. Given a set of documents, clustering is the task of coming up with a good grouping of the documents based on their contents. It is similar to arranging books on a bookshelf according to their topic. Given a set of topics, standing information needs, or other categories (such as suitability of texts for different age groups), classification is the task of deciding which class(es), if any, each of a set of documents belongs to. It is often approached by first manually classifying some documents and then hoping to be able to classify new documents automatically.

Information retrieval systems can also be distinguished by the scale at which they operate, and it is useful to distinguish three prominent scales. In *web search*, the system has to provide search over billions of documents stored on millions of computers. Distinctive issues are needing to gather documents for indexing, being able to build systems that work efficiently at this enormous scale, and handling particular aspects of the web, such as the exploitation of hypertext and not being fooled by site providers manipulating page content in an attempt to boost their search engine rankings, given the commercial importance of the web. We focus on all these issues in Chapters 19–21. At the other extreme is *personal information retrieval*. In the last few years, consumer operating systems have integrated information retrieval (such as Apple's Mac OS X Spotlight or Windows Vista's Instant Search). Email programs usually not only provide search but also text classification: they at least provide a spam (junk mail) filter, and commonly also provide either manual or automatic means for classifying mail so that it can be placed directly into particular folders. Distinctive issues here include handling the broad range of document types on a typical personal computer, and making the search system maintenance free and sufficiently lightweight in terms of startup, processing, and disk space usage that it can run on one machine without annoying its owner. In between is the space of *enterprise, institutional, and domain-specific search*, where retrieval might be provided for collections such as a corporation's internal documents, a database of patents, or research articles on biochemistry. In this case, the documents will typically be stored on centralized file systems and one or a handful of dedicated machines will provide search over the collection. This book contains techniques of value over this whole spectrum, but our coverage of some aspects of parallel and distributed search in web-scale search systems is comparatively light owing to the relatively small published literature on the details of such systems. However, outside of a handful of web search companies, a software developer is most likely to encounter the personal search and enterprise scenarios.

In this chapter we begin with a very simple example of an information retrieval problem, and introduce the idea of a term-document matrix (Section 1.1) and the central inverted index data structure (Section 1.2). We will then examine the Boolean retrieval model and how Boolean queries are processed (Sections 1.3 and 1.4).

1.1 An example information retrieval problem

A fat book which many people own is Shakespeare's Collected Works. Suppose you wanted to determine which plays of Shakespeare contain the words Brutus AND Caesar AND NOT Calpurnia. One way to do that is to start at the beginning and to read through all the text, noting for each play whether it contains Brutus and Caesar and excluding it from consideration if it contains Calpurnia. The simplest form of document retrieval is for a computer to do this sort of linear scan through documents. This process is commonly
GREP referred to as *grepping* through text, after the Unix command `grep`, which performs this process. Grepping through text can be a very effective process, especially given the speed of modern computers, and often allows useful possibilities for wildcard pattern matching through the use of regular expressions. With modern computers, for simple querying of modest collections (the size of Shakespeare's Collected Works is a bit under one million words of text in total), you really need nothing more.

But for many purposes, you do need more:

1. To process large document collections quickly. The amount of online data has grown at least as quickly as the speed of computers, and we would now like to be able to search collections that total in the order of billions to trillions of words.
2. To allow more flexible matching operations. For example, it is impractical to perform the query Romans NEAR countrymen with `grep`, where NEAR might be defined as "within 5 words" or "within the same sentence".
3. To allow ranked retrieval: in many cases you want the best answer to an information need among many documents that contain certain words.

INDEX The way to avoid linearly scanning the texts for each query is to *index* the documents in advance. Let us stick with Shakespeare's Collected Works, and use it to introduce the basics of the Boolean retrieval model. Suppose we record for each document – here a play of Shakespeare's – whether it contains each word out of all the words Shakespeare used (Shakespeare used
INCIDENCE MATRIX about 32,000 different words). The result is a binary term-document *incidence*
TERM *matrix*, as in Figure 1.1. *Terms* are the indexed units (further discussed in Section 2.2); they are usually words, and for the moment you can think of

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

► **Figure 1.1** A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t, and is 0 otherwise.

them as words, but the information retrieval literature normally speaks of terms because some of them, such as perhaps I-9 or Hong Kong are not usually thought of as words. Now, depending on whether we look at the matrix rows or columns, we can have a vector for each term, which shows the documents it appears in, or a vector for each document, showing the terms that occur in it.[2]

To answer the query Brutus AND Caesar AND NOT Calpurnia, we take the vectors for Brutus, Caesar and Calpurnia, complement the last, and then do a bitwise AND:

110100 AND 110111 AND 101111 = 100100

The answers for this query are thus *Antony and Cleopatra* and *Hamlet* (Figure 1.2).

BOOLEAN RETRIEVAL MODEL

The *Boolean retrieval model* is a model for information retrieval in which we can pose any query which is in the form of a Boolean expression of terms, that is, in which terms are combined with the operators AND, OR, and NOT. The model views each document as just a set of words.

DOCUMENT

Let us now consider a more realistic scenario, simultaneously using the opportunity to introduce some terminology and notation. Suppose we have $N = 1$ million documents. By *documents* we mean whatever units we have decided to build a retrieval system over. They might be individual memos or chapters of a book (see Section 2.1.2 (page 20) for further discussion). We will refer to the group of documents over which we perform retrieval as the (document) *collection*. It is sometimes also referred to as a *corpus* (a *body* of texts). Suppose each document is about 1000 words long (2–3 book pages). If

COLLECTION CORPUS

2. Formally, we take the transpose of the matrix to be able to get the terms as column vectors.

Antony and Cleopatra, Act III, Scene ii
Agrippa [Aside to Domitius Enobarbus]: Why, Enobarbus,
When Antony found Julius Caesar dead,
He cried almost to roaring; and he wept
When at Philippi he found Brutus slain.

Hamlet, Act III, Scene ii
Lord Polonius: I did enact Julius Caesar: I was killed i' the
Capitol; Brutus killed me.

► **Figure 1.2** Results from Shakespeare for the query Brutus AND Caesar AND NOT Calpurnia.

we assume an average of 6 bytes per word including spaces and punctuation, then this is a document collection about 6 GB in size. Typically, there might be about $M = 500{,}000$ distinct terms in these documents. There is nothing special about the numbers we have chosen, and they might vary by an order of magnitude or more, but they give us some idea of the dimensions of the kinds of problems we need to handle. We will discuss and model these size assumptions in Section 5.1 (page 86).

AD HOC RETRIEVAL

Our goal is to develop a system to address the *ad hoc retrieval* task. This is the most standard IR task. In it, a system aims to provide documents from within the collection that are relevant to an arbitrary user information need, communicated to the system by means of a one-off, user-initiated query. An

INFORMATION NEED

information need is the topic about which the user desires to know more, and

QUERY

is differentiated from a *query*, which is what the user conveys to the computer in an attempt to communicate the information need. A document is

RELEVANCE

relevant if it is one that the user perceives as containing information of value with respect to their personal information need. Our example above was rather artificial in that the information need was defined in terms of particular words, whereas usually a user is interested in a topic like "pipeline leaks" and would like to find relevant documents regardless of whether they precisely use those words or express the concept with other words such as

EFFECTIVENESS

pipeline rupture. To assess the *effectiveness* of an IR system (i.e., the quality of its search results), a user will usually want to know two key statistics about the system's returned results for a query:

PRECISION

Precision: What fraction of the returned results are relevant to the information need?

RECALL

Recall: What fraction of the relevant documents in the collection were returned by the system?

Detailed discussion of relevance and evaluation measures including precision and recall is found in Chapter 8.

We now cannot build a term-document matrix in a naive way. A 500K × 1M matrix has half-a-trillion 0's and 1's – too many to fit in a computer's memory. But the crucial observation is that the matrix is extremely sparse, that is, it has few non-zero entries. Because each document is 1000 words long, the matrix has no more than one billion 1's, so a minimum of 99.8% of the cells are zero. A much better representation is to record only the things that do occur, that is, the 1 positions.

INVERTED INDEX

This idea is central to the first major concept in information retrieval, the *inverted index*. The name is actually redundant: an index always maps back from terms to the parts of a document where they occur. Nevertheless, *inverted index*, or sometimes *inverted file*, has become the standard term in information retrieval.[3] The basic idea of an inverted index is shown in Figure 1.3.

DICTIONARY
VOCABULARY
LEXICON

We keep a *dictionary* of terms (sometimes also referred to as a *vocabulary* or *lexicon*; in this book, we use *dictionary* for the data structure and *vocabulary* for the set of terms). Then for each term, we have a list that records which documents the term occurs in. Each item in the list – which records that a term appeared in a document (and, later, often, the positions in the document) – is conventionally called a *posting*.[4] The list is then called a *postings list* (or inverted list), and all the postings lists taken together are referred to as the *postings*. The dictionary in Figure 1.3 has been sorted alphabetically and each postings list is sorted by document ID. We will see why this is useful in Section 1.3, below, but later we will also consider alternatives to doing this (Section 7.1.5).

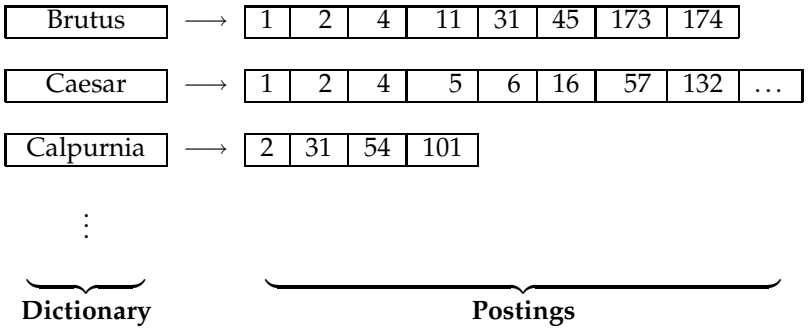
POSTING
POSTINGS LIST
POSTINGS

1.2 A first take at building an inverted index

To gain the speed benefits of indexing at retrieval time, we have to build the index in advance. The major steps in this are:

1. Collect the documents to be indexed:
   Friends, Romans, countrymen. | So let it be with Caesar | ...
2. Tokenize the text, turning each document into a list of tokens:
   Friends | Romans | countrymen | So | ...

3. Some information retrieval researchers prefer the term inverted file, but expressions like index construction and index compression are much more common than inverted file construction and inverted file compression. For consistency, we use (inverted) index throughout this book.
4. In a (non-positional) inverted index, a posting is just a document ID, but it is inherently associated with a term, via the postings list it is placed on; sometimes we will also talk of a (term, docID) pair as a posting.



▶ **Figure 1.3** The two parts of an inverted index. The dictionary is commonly kept in memory, with pointers to each postings list, which is stored on disk.

3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms: friend | roman | countryman | so ...
4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

We will define and discuss the earlier stages of processing, that is, steps 1–3, in Section 2.2 (page 22). Until then you can think of *tokens* and *normalized tokens* as also loosely equivalent to *words*. Here, we assume that the first 3 steps have already been done, and we examine building a basic inverted index by sort-based indexing.

DOCID

Within a document collection, we assume that each document has a unique serial number, known as the document identifier (*docID*). During index construction, we can simply assign successive integers to each new document when it is first encountered. The input to indexing is a list of normalized tokens for each document, which we can equally think of as a list of pairs of term and docID, as in Figure 1.4. The core indexing step is *sorting* this list so that the terms are alphabetical, giving us the representation in the middle column of Figure 1.4. Multiple occurrences of the same term from the same document are then merged.[5] Instances of the same term are then grouped, and the result is split into a *dictionary* and *postings*, as shown in the right column of Figure 1.4. Since a term generally occurs in a number of documents, this data organization already reduces the storage requirements of the index. The dictionary also records some statistics, such as the number of documents which contain each term (the *document frequency*, which is here also the length of each postings list). This information is not vital for a basic Boolean search engine, but it allows us to improve the efficiency of the

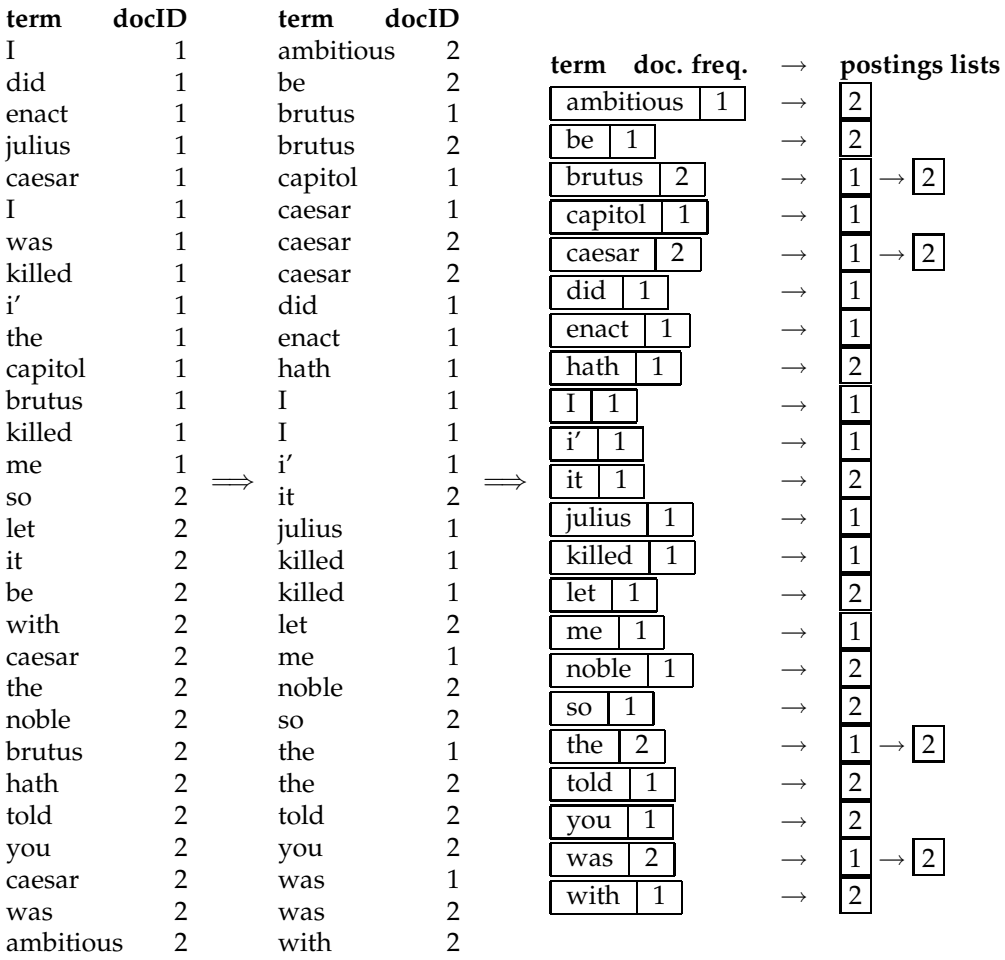
SORTING

DOCUMENT FREQUENCY

5. Unix users can note that these steps are similar to use of the `sort` and then `uniq` commands.

Doc 1
I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

Doc 2
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:



► **Figure 1.4** Building an index by sorting and grouping. The sequence of terms in each document, tagged by their documentID (left) is sorted alphabetically (middle). Instances of the same term are then grouped by word and then by documentID. The terms and documentIDs are then separated out (right). The dictionary stores the terms, and has a pointer to the postings list for each term. It commonly also stores other summary information such as, here, the document frequency of each term. We use this information for improving query time efficiency and, later, for weighting in ranked retrieval models. Each postings list stores the list of documents in which a term occurs, and may store other information such as the term frequency (the frequency of each term in each document) or the position(s) of the term in each document.

search engine at query time, and it is a statistic later used in many ranked retrieval models. The postings are secondarily sorted by docID. This provides the basis for efficient query processing. This inverted index structure is essentially without rivals as the most efficient structure for supporting ad hoc text search.

In the resulting index, we pay for storage of both the dictionary and the postings lists. The latter are much larger, but the dictionary is commonly kept in memory, while postings lists are normally kept on disk, so the size of each is important, and in Chapter 5 we will examine how each can be optimized for storage and access efficiency. What data structure should be used for a postings list? A fixed length array would be wasteful as some words occur in many documents, and others in very few. For an in-memory postings list, two good alternatives are singly linked lists or variable length arrays. Singly linked lists allow cheap insertion of documents into postings lists (following updates, such as when recrawling the web for updated documents), and naturally extend to more advanced indexing strategies such as skip lists (Section 2.3), which require additional pointers. Variable length arrays win in space requirements by avoiding the overhead for pointers and in time requirements because their use of contiguous memory increases speed on modern processors with memory caches. Extra pointers can in practice be encoded into the lists as offsets. If updates are relatively infrequent, variable length arrays will be more compact and faster to traverse. We can also use a hybrid scheme with a linked list of fixed length arrays for each term. When postings lists are stored on disk, they are stored (perhaps compressed) as a contiguous run of postings without explicit pointers (as in Figure 1.3), so as to minimize the size of the postings list and the number of disk seeks to read a postings list into memory.

Exercise 1.1 [⋆]

Draw the inverted index that would be built for the following document collection. (See Figure 1.3 for an example.)

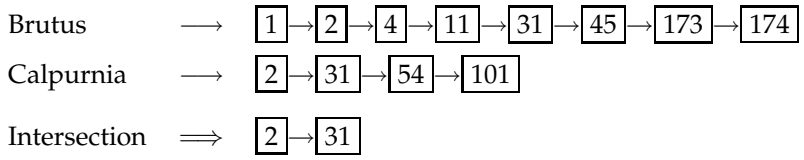
Doc 1	new home sales top forecasts
Doc 2	home sales rise in july
Doc 3	increase in home sales in july
Doc 4	july new home sales rise

Exercise 1.2 [⋆]

Consider these documents:

Doc 1	breakthrough drug for schizophrenia
Doc 2	new schizophrenia drug
Doc 3	new approach for treatment of schizophrenia
Doc 4	new hopes for schizophrenia patients

a. Draw the term-document incidence matrix for this document collection.



► **Figure 1.5** Intersecting the postings lists for Brutus and Calpurnia from Figure 1.3.

b. Draw the inverted index representation for this collection, as in Figure 1.3 (page 7).

Exercise 1.3 [⋆]

For the document collection shown in Exercise 1.2, what are the returned results for these queries:

a. schizophrenia AND drug

b. for AND NOT(drug OR approach)

1.3 Processing Boolean queries

How do we process a query using an inverted index and the basic Boolean retrieval model? Consider processing the *simple conjunctive query*:

SIMPLE CONJUNCTIVE QUERIES

(1.1) Brutus AND Calpurnia

over the inverted index partially shown in Figure 1.3 (page 7). We:

1. Locate Brutus in the Dictionary
2. Retrieve its postings
3. Locate Calpurnia in the Dictionary
4. Retrieve its postings
5. Intersect the two postings lists, as shown in Figure 1.5.

POSTINGS LIST INTERSECTION

POSTINGS MERGE

The *intersection* operation is the crucial one: we need to efficiently intersect postings lists so as to be able to quickly find documents that contain both terms. (This operation is sometimes referred to as *merging* postings lists: this slightly counterintuitive name reflects using the term *merge algorithm* for a general family of algorithms that combine multiple sorted lists by interleaved advancing of pointers through each; here we are merging the lists with a logical AND operation.)

There is a simple and effective method of intersecting postings lists using the merge algorithm (see Figure 1.6): we maintain pointers into both lists

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

▶ **Figure 1.6** Algorithm for the intersection of two postings lists p_1 and p_2.

and walk through the two postings lists simultaneously, in time linear in the total number of postings entries. At each step, we compare the docID pointed to by both pointers. If they are the same, we put that docID in the results list, and advance both pointers. Otherwise we advance the pointer pointing to the smaller docID. If the lengths of the postings lists are x and y, the intersection takes $O(x + y)$ operations. Formally, the complexity of querying is $\Theta(N)$, where N is the number of documents in the collection.[6] Our indexing methods gain us just a constant, not a difference in Θ time complexity compared to a linear scan, but in practice the constant is huge. To use this algorithm, it is crucial that postings be sorted by a single global ordering. Using a numeric sort by docID is one simple way to achieve this.

We can extend the intersection operation to process more complicated queries like:

(1.2) (Brutus OR Caesar) AND NOT Calpurnia

QUERY OPTIMIZATION

Query optimization is the process of selecting how to organize the work of answering a query so that the least total amount of work needs to be done by the system. A major element of this for Boolean queries is the order in which postings lists are accessed. What is the best order for query processing? Consider a query that is an AND of t terms, for instance:

(1.3) Brutus AND Caesar AND Calpurnia

For each of the t terms, we need to get its postings, then AND them together. The standard heuristic is to process terms in order of increasing document

6. The notation $\Theta(\cdot)$ is used to express an asymptotically tight bound on the complexity of an algorithm. Informally, this is often written as $O(\cdot)$, but this notation really expresses an asymptotic upper bound, which need not be tight (Cormen et al. 1990).

```
INTERSECT(⟨t_1, ..., t_n⟩)
1  terms ← SORTBYINCREASINGFREQUENCY(⟨t_1, ..., t_n⟩)
2  result ← postings(first(terms))
3  terms ← rest(terms)
4  while terms ≠ NIL and result ≠ NIL
5  do result ← INTERSECT(result, postings(first(terms)))
6     terms ← rest(terms)
7  return result
```

▶ **Figure 1.7** Algorithm for conjunctive queries that returns the set of documents containing each term in the input list of terms.

frequency: if we start by intersecting the two smallest postings lists, then all intermediate results must be no bigger than the smallest postings list, and we are therefore likely to do the least amount of total work. So, for the postings lists in Figure 1.3 (page 7), we execute the above query as:

(1.4) (Calpurnia AND Brutus) AND Caesar

This is a first justification for keeping the frequency of terms in the dictionary: it allows us to make this ordering decision based on in-memory data before accessing any postings list.

Consider now the optimization of more general queries, such as:

(1.5) (madding OR crowd) AND (ignoble OR strife) AND (killed OR slain)

As before, we will get the frequencies for all terms, and we can then (conservatively) estimate the size of each OR by the sum of the frequencies of its disjuncts. We can then process the query in increasing order of the size of each disjunctive term.

For arbitrary Boolean queries, we have to evaluate and temporarily store the answers for intermediate expressions in a complex expression. However, in many circumstances, either because of the nature of the query language, or just because this is the most common type of query that users submit, a query is purely conjunctive. In this case, rather than viewing merging postings lists as a function with two inputs and a distinct output, it is more efficient to intersect each retrieved postings list with the current intermediate result in memory, where we initialize the intermediate result by loading the postings list of the least frequent term. This algorithm is shown in Figure 1.7. The intersection operation is then asymmetric: the intermediate results list is in memory while the list it is being intersected with is being read from disk. Moreover the intermediate results list is always at least as short as the other list, and in many cases it is orders of magnitude shorter. The postings

intersection can still be done by the algorithm in Figure 1.6, but when the difference between the list lengths is very large, opportunities to use alternative techniques open up. The intersection can be calculated in place by destructively modifying or marking invalid items in the intermediate results list. Or the intersection can be done as a sequence of binary searches in the long postings lists for each posting in the intermediate results list. Another possibility is to store the long postings list as a hashtable, so that membership of an intermediate result item can be calculated in constant rather than linear or log time. However, such alternative techniques are difficult to combine with postings list compression of the sort discussed in Chapter 5. Moreover, standard postings list intersection operations remain necessary when both terms of a query are very common.

? **Exercise 1.4** [⋆]

For the queries below, can we still run through the intersection in time $O(x + y)$, where x and y are the lengths of the postings lists for Brutus and Caesar? If not, what can we achieve?

a. Brutus AND NOT Caesar

b. Brutus OR NOT Caesar

Exercise 1.5 [⋆]

Extend the postings merge algorithm to arbitrary Boolean query formulas. What is its time complexity? For instance, consider:

c. (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

Can we always merge in linear time? Linear in what? Can we do better than this?

Exercise 1.6 [⋆⋆]

We can use distributive laws for AND and OR to rewrite queries.

a. Show how to rewrite the query in Exercise 1.5 into disjunctive normal form using the distributive laws.

b. Would the resulting query be more or less efficiently evaluated than the original form of this query?

c. Is this result true in general or does it depend on the words and the contents of the document collection?

Exercise 1.7 [⋆]

Recommend a query processing order for

d. (tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)

given the following postings list sizes:

Term	Postings size
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

Exercise 1.8 [⋆]

If the query is:

e. friends AND romans AND (NOT countrymen)

how could we use the frequency of countrymen in evaluating the best query evaluation order? In particular, propose a way of handling negation in determining the order of query processing.

Exercise 1.9 [⋆⋆]

For a conjunctive query, is processing postings lists in order of size guaranteed to be optimal? Explain why it is, or give an example where it isn't.

Exercise 1.10 [⋆⋆]

Write out a postings merge algorithm, in the style of Figure 1.6 (page 11), for an x OR y query.

Exercise 1.11 [⋆⋆]

How should the Boolean query x AND NOT y be handled? Why is naive evaluation of this query normally very expensive? Write out a postings merge algorithm that evaluates this query efficiently.

1.4 The extended Boolean model versus ranked retrieval

RANKED RETRIEVAL MODEL
FREE TEXT QUERIES

The Boolean retrieval model contrasts with *ranked retrieval models* such as the vector space model (Section 6.3), in which users largely use *free text queries*, that is, just typing one or more words rather than using a precise language with operators for building up query expressions, and the system decides which documents best satisfy the query. Despite decades of academic research on the advantages of ranked retrieval, systems implementing the Boolean retrieval model were the main or only search option provided by large commercial information providers for three decades until the early 1990s (approximately the date of arrival of the World Wide Web). However, these systems did not have just the basic Boolean operations (AND, OR, and NOT) which we have presented so far. A strict Boolean expression over terms with an unordered results set is too limited for many of the information needs that people have, and these systems implemented extended Boolean retrieval models by incorporating additional operators such as term proximity operators. A *proximity operator* is a way of specifying that two terms in a query

PROXIMITY OPERATOR

must occur close to each other in a document, where closeness may be measured by limiting the allowed number of intervening words or by reference to a structural unit such as a sentence or paragraph.

Example 1.1: Commercial Boolean searching: Westlaw. Westlaw (http://www.westlaw.com/) is the largest commercial legal search service (in terms of the number of paying subscribers), with over half a million subscribers performing millions of searches a day over tens of terabytes of text data. The service was started in 1975. In 2005, Boolean search (called "Terms and Connectors" by Westlaw) was still the default, and used by a large percentage of users, although ranked free text querying (called "Natural Language" by Westlaw) was added in 1992. Here are some example Boolean queries on Westlaw:

Information need: Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company. *Query:* "trade secret" /s disclos! /s prevent /s employe!

Information need: Requirements for disabled people to be able to access a workplace.
Query: disab! /p access! /s work-site work-place (employment /3 place)

Information need: Cases about a host's responsibility for drunk guests.
Query: host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

Note the long, precise queries and the use of proximity operators, both uncommon in web search. Submitted queries average about ten words in length. Unlike web search conventions, a space between words represents disjunction (the tightest binding operator), & is AND and /s, /p, and /*k* ask for matches in the same sentence, same paragraph or within *k* words respectively. Double quotes give a *phrase search* (consecutive words); see Section 2.4 (page 39). The exclamation mark (!) gives a trailing wildcard query (see Section 3.2, page 51); thus liab! matches all words starting with liab. Additionally work-site matches any of *worksite*, *work-site* or *work site*; see Section 2.2.1 (page 22). Typical expert queries are usually carefully defined and incrementally developed until they obtain what look to be good results to the user.

Many users, particularly professionals, prefer Boolean query models. Boolean queries are precise: a document either matches the query or it does not. This offers the user greater control and transparency over what is retrieved. And some domains, such as legal materials, allow an effective means of document ranking within a Boolean model: Westlaw returns documents in reverse chronological order, which is in practice quite effective. In 2007, the majority of law librarians still seem to recommend terms and connectors for high recall searches, and the majority of legal users think they are getting greater control by using them. However, this does not mean that Boolean queries are more effective for professional searchers. Indeed, experimenting on a Westlaw subcollection, Turtle (1994) found that free text queries produced better results than Boolean queries prepared by Westlaw's own reference librarians for the majority of the information needs in his experiments. A general problem with Boolean search is that using AND operators tends to produce high precision but low recall searches, while using OR operators gives low precision but high recall searches, and it is difficult or impossible to find a satisfactory middle ground.

In this chapter, we have looked at the structure and construction of a basic

inverted index, comprising a dictionary and postings lists. We introduced the Boolean retrieval model, and examined how to do efficient retrieval via linear time merges and simple query optimization. In Chapters 2–7 we will consider in detail richer query models and the sort of augmented index structures that are needed to handle them efficiently. Here we just mention a few of the main additional things we would like to be able to do:

1. We would like to better determine the set of terms in the dictionary and to provide retrieval that is tolerant to spelling mistakes and inconsistent choice of words.

2. It is often useful to search for compounds or phrases that denote a concept such as "operating system". As the Westlaw examples show, we might also wish to do proximity queries such as Gates NEAR Microsoft. To answer such queries, the index has to be augmented to capture the proximities of terms in documents.

3. A Boolean model only records term presence or absence, but often we would like to accumulate evidence, giving more weight to documents that have a term several times as opposed to ones that contain it only once. To be able to do this we need *term frequency* information (the number of times a term occurs in a document) in postings lists.

TERM FREQUENCY

4. Boolean queries just retrieve a set of matching documents, but commonly we wish to have an effective method to order (or "rank") the returned results. This requires having a mechanism for determining a document score which encapsulates how good a match a document is for a query.

With these additional ideas, we will have seen most of the basic technology that supports ad hoc searching over unstructured information. Ad hoc searching over documents has recently conquered the world, powering not only web search engines but the kind of unstructured search that lies behind the large eCommerce websites. Although the main web search engines differ by emphasizing free text querying, most of the basic issues and technologies of indexing and querying remain the same, as we will see in later chapters. Moreover, over time, web search engines have added at least partial implementations of some of the most popular operators from extended Boolean models: phrase search is especially popular and most have a very partial implementation of Boolean operators. Nevertheless, while these options are liked by expert searchers, they are little used by most people and are not the main focus in work on trying to improve web search engine performance.

? **Exercise 1.12** [⋆]

Write a query using Westlaw syntax which would find any of the words professor, teacher, or lecturer in the same sentence as a form of the verb explain.

Exercise 1.13 [⋆]

Try using the Boolean search features on a couple of major web search engines. For instance, choose a word, such as burglar, and submit the queries (i) burglar, (ii) burglar AND burglar, and (iii) burglar OR burglar. Look at the estimated number of results and top hits. Do they make sense in terms of Boolean logic? Often they haven't for major search engines. Can you make sense of what is going on? What about if you try different words? For example, query for (i) knight, (ii) conquer, and then (iii) knight OR conquer. What bound should the number of results from the first two queries place on the third query? Is this bound observed?

1.5 References and further reading

The practical pursuit of computerized information retrieval began in the late 1940s (Cleverdon 1991, Liddy 2005). A great increase in the production of scientific literature, much in the form of less formal technical reports rather than traditional journal articles, coupled with the availability of computers, led to interest in automatic document retrieval. However, in those days, document retrieval was always based on author, title, and keywords; full-text search came much later.

The article of Bush (1945) provided lasting inspiration for the new field:

> "Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and, to coin one at random, 'memex' will do. A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory."

The term *Information Retrieval* was coined by Calvin Mooers in 1948/1950 (Mooers 1950).

In 1958, much newspaper attention was paid to demonstrations at a conference (see Taube and Wooster 1958) of IBM "auto-indexing" machines, based primarily on the work of H. P. Luhn. Commercial interest quickly gravitated towards Boolean retrieval systems, but the early years saw a heady debate over various disparate technologies for retrieval systems. For example Mooers (1961) dissented:

> "It is a common fallacy, underwritten at this date by the investment of several million dollars in a variety of retrieval hardware, that the algebra of George Boole (1847) is the appropriate formalism for retrieval system design. This view is as widely and uncritically accepted as it is wrong."

The observation of AND vs. OR giving you opposite extremes in a precision/recall tradeoff, but not the middle ground comes from (Lee and Fox 1988).

The book (Witten et al. 1999) is the standard reference for an in-depth comparison of the space and time efficiency of the inverted index versus other possible data structures; a more succinct and up-to-date presentation appears in Zobel and Moffat (2006). We further discuss several approaches in Chapter 5.

REGULAR EXPRESSIONS

Friedl (2006) covers the practical usage of *regular expressions* for searching. The underlying computer science appears in (Hopcroft et al. 2000).

2 The term vocabulary and postings lists

Recall the major steps in inverted index construction:

1. Collect the documents to be indexed.
2. Tokenize the text.
3. Do linguistic preprocessing of tokens.
4. Index the documents that each term occurs in.

In this chapter we first briefly mention how the basic unit of a document can be defined and how the character sequence that it comprises is determined (Section 2.1). We then examine in detail some of the substantive linguistic issues of tokenization and linguistic preprocessing, which determine the vocabulary of terms which a system uses (Section 2.2). Tokenization is the process of chopping character streams into tokens, while linguistic preprocessing then deals with building equivalence classes of tokens which are the set of terms that are indexed. Indexing itself is covered in Chapters 1 and 4. Then we return to the implementation of postings lists. In Section 2.3, we examine an extended postings list data structure that supports faster querying, while Section 2.4 covers building postings data structures suitable for handling phrase and proximity queries, of the sort that commonly appear in both extended Boolean models and on the web.

2.1 Document delineation and character sequence decoding

2.1.1 Obtaining the character sequence in a document

Digital documents that are the input to an indexing process are typically bytes in a file or on a web server. The first step of processing is to convert this byte sequence into a linear sequence of characters. For the case of plain English text in ASCII encoding, this is trivial. But often things get much more

complex. The sequence of characters may be encoded by one of various single byte or multibyte encoding schemes, such as Unicode UTF-8, or various national or vendor-specific standards. We need to determine the correct encoding. This can be regarded as a machine learning classification problem, as discussed in Chapter 13,[1] but is often handled by heuristic methods, user selection, or by using provided document metadata. Once the encoding is determined, we decode the byte sequence to a character sequence. We might save the choice of encoding because it gives some evidence about what language the document is written in.

The characters may have to be decoded out of some binary representation like Microsoft Word DOC files and/or a compressed format such as zip files. Again, we must determine the document format, and then an appropriate decoder has to be used. Even for plain text documents, additional decoding may need to be done. In XML documents (Section 10.1, page 197), character entities, such as `&`, need to be decoded to give the correct character, namely `&` for `&`. Finally, the textual part of the document may need to be extracted out of other material that will not be processed. This might be the desired handling for XML files, if the markup is going to be ignored; we would almost certainly want to do this with postscript or PDF files. We will not deal further with these issues in this book, and will assume henceforth that our documents are a list of characters. Commercial products usually need to support a broad range of document types and encodings, since users want things to just work with their data as is. Often, they just think of documents as text inside applications and are not even aware of how it is encoded on disk. This problem is usually solved by licensing a software library that handles decoding document formats and character encodings.

The idea that text is a linear sequence of characters is also called into question by some writing systems, such as Arabic, where text takes on some two dimensional and mixed order characteristics, as shown in Figures 2.1 and 2.2. But, despite some complicated writing system conventions, there is an underlying sequence of sounds being represented and hence an essentially linear structure remains, and this is what is represented in the digital representation of Arabic, as shown in Figure 2.1.

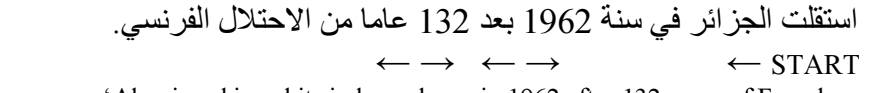
2.1.2 Choosing a document unit

DOCUMENT UNIT

The next phase is to determine what the *document unit* for indexing is. Thus far we have assumed that documents are fixed units for the purposes of indexing. For example, we take each file in a folder as a document. But there

1. A classifier is a function that takes objects of some sort and assigns them to one of a number of distinct classes (see Chapter 13). Usually classification is done by machine learning methods such as probabilistic models, but it can also be done by hand-written rules.

► **Figure 2.1** An example of a vocalized Modern Standard Arabic word. The writing is from right to left and letters undergo complex mutations as they are combined. The representation of short vowels (here, /i/ and /u/) and the final /n/ (nunation) departs from strict linearity by being represented as diacritics above and below letters. Nevertheless, the represented text is still clearly a linear ordering of characters representing sounds. Full vocalization, as here, normally appears only in the Koran and children's books. Day-to-day text is unvocalized (short vowels are not represented but the letter for ā would still appear) or partially vocalized, with short vowels inserted in places where the writer perceives ambiguities. These choices add further complexities to indexing.



► **Figure 2.2** The conceptual linear order of characters is not necessarily the order that you see on the page. In languages that are written right-to-left, such as Hebrew and Arabic, it is quite common to also have left-to-right text interspersed, such as numbers and dollar amounts. With modern Unicode representation concepts, the order of characters in files matches the conceptual order, and the reversal of displayed characters is handled by the rendering system, but this may not be true for documents in older encodings.

are many cases in which you might want to do something different. A traditional Unix (mbox-format) email file stores a sequence of email messages (an email folder) in one file, but you might wish to regard each email message as a separate document. Many email messages now contain attached documents, and you might then want to regard the email message and each contained attachment as separate documents. If an email message has an attached zip file, you might want to decode the zip file and regard each file it contains as a separate document. Going in the opposite direction, various pieces of web software (such as latex2html) take things that you might regard as a single document (e.g., a Powerpoint file or a LaTeX document) and split them into separate HTML pages for each slide or subsection, stored as separate files. In these cases, you might want to combine multiple files into a single document.

INDEXING GRANULARITY

More generally, for very long documents, the issue of indexing *granularity* arises. For a collection of books, it would usually be a bad idea to index an

entire book as a document. A search for Chinese toys might bring up a book that mentions China in the first chapter and toys in the last chapter, but this does not make it relevant to the query. Instead, we may well wish to index each chapter or paragraph as a mini-document. Matches are then more likely to be relevant, and since the documents are smaller it will be much easier for the user to find the relevant passages in the document. But why stop there? We could treat individual sentences as mini-documents. It becomes clear that there is a precision/recall tradeoff here. If the units get too small, we are likely to miss important passages because terms were distributed over several mini-documents, while if units are too large we tend to get spurious matches and the relevant information is hard for the user to find.

The problems with large document units can be alleviated by use of explicit or implicit proximity search (Sections 2.4.2 and 7.2.2), and the tradeoffs in resulting system performance that we are hinting at are discussed in Chapter 8. The issue of index granularity, and in particular a need to simultaneously index documents at multiple levels of granularity, appears prominently in XML retrieval, and is taken up again in Chapter 10. An IR system should be designed to offer choices of granularity. For this choice to be made well, the person who is deploying the system must have a good understanding of the document collection, the users, and their likely information needs and usage patterns. For now, we will henceforth assume that a suitable size document unit has been chosen, together with an appropriate way of dividing or aggregating files, if needed.

2.2 Determining the vocabulary of terms

2.2.1 Tokenization

Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called *tokens*, perhaps at the same time throwing away certain characters, such as punctuation. Here is an example of tokenization:

Input: Friends, Romans, Countrymen, lend me your ears;
Output: | Friends | Romans | Countrymen | lend | me | your | ears |

These tokens are often loosely referred to as terms or words, but it is sometimes important to make a type/token distinction. A *token* is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing. A *type* is the class of all tokens containing the same character sequence. A *term* is a (perhaps normalized) type that is included in the IR system's dictionary. The set of index terms could be entirely distinct from the tokens, for instance, they could be

TOKEN
TYPE
TERM

semantic identifiers in a taxonomy, but in practice in modern IR systems they are strongly related to the tokens in the document. However, rather than being exactly the tokens that appear in the document, they are usually derived from them by various normalization processes which are discussed in Section 2.2.3.[2] For example, if the document to be indexed is *to sleep perchance to dream*, then there are 5 tokens, but only 4 types (since there are 2 instances of *to*). However, if *to* is omitted from the index (as a stop word, see Section 2.2.2 (page 27)), then there will be only 3 terms: *sleep*, *perchance*, and *dream*.

The major question of the tokenization phase is what are the correct tokens to use? In this example, it looks fairly trivial: you chop on whitespace and throw away punctuation characters. This is a starting point, but even for English there are a number of tricky cases. For example, what do you do about the various uses of the apostrophe for possession and contractions?

> Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.

For *O'Neill*, which of the following is the desired tokenization?

neill
oneill
o'neill
o' neill
o neill ?

And for *aren't*, is it:

aren't
arent
are n't
aren t ?

A simple strategy is to just split on all non-alphanumeric characters, but while o neill looks okay, aren t looks intuitively bad. For all of them, the choices determine which Boolean queries will match. A query of neill AND capital will match in three cases but not the other two. In how many cases would a query of o'neill AND capital match? If no preprocessing of a query is done, then it would match in only one of the five cases. For either

2. That is, as defined here, tokens that are not indexed (stop words) are not terms, and if multiple tokens are collapsed together via normalization, they are indexed as one term, under the normalized form. However, we later relax this definition when discussing classification and clustering in Chapters 13–18, where there is no index. In these chapters, we drop the requirement of inclusion in the dictionary. A *term* means a normalized word.

Boolean or free text queries, you always want to do the exact same tokenization of document and query words, generally by processing queries with the same tokenizer. This guarantees that a sequence of characters in a text will always match the same sequence typed in a query.[3]

LANGUAGE IDENTIFICATION

These issues of tokenization are language-specific. It thus requires the language of the document to be known. *Language identification* based on classifiers that use short character subsequences as features is highly effective; most languages have distinctive signature patterns (see page 46 for references).

For most languages and particular domains within them there are unusual specific tokens that we wish to recognize as terms, such as the programming languages C++ and C#, aircraft names like B-52, or a T.V. show name such as M*A*S*H – which is sufficiently integrated into popular culture that you find usages such as *M*A*S*H-style hospitals*. Computer technology has introduced new types of character sequences that a tokenizer should probably tokenize as a single token, including email addresses (jblack@mail.yahoo.com), web URLs (http://stuff.big.com/new/specials.html), numeric IP addresses (142.32.48.231), package tracking numbers (1Z9999W99845399981), and more. One possible solution is to omit from indexing tokens such as monetary amounts, numbers, and URLs, since their presence greatly expands the size of the vocabulary. However, this comes at a large cost in restricting what people can search for. For instance, people might want to search in a bug database for the line number where an error occurs. Items such as the date of an email, which have a clear semantic type, are often indexed separately as document metadata (see Section 6.1, page 110).

HYPHENS

In English, *hyphenation* is used for various purposes ranging from splitting up vowels in words (*co-education*) to joining nouns as names (*Hewlett-Packard*) to a copyediting device to show word grouping (*the hold-him-back-and-drag-him-away maneuver*). It is easy to feel that the first example should be regarded as one token (and is indeed more commonly written as just *coeducation*), the last should be separated into words, and that the middle case is unclear. Handling hyphens automatically can thus be complex: it can either be done as a classification problem, or more commonly by some heuristic rules, such as allowing short hyphenated prefixes on words, but not longer hyphenated forms.

Conceptually, splitting on white space can also split what should be regarded as a single token. This occurs most commonly with names (*San Francisco, Los Angeles*) but also with borrowed foreign phrases (*au fait*) and com-

3. For the free text case, this is straightforward. The Boolean case is more complex: this tokenization may produce multiple terms from one query word. This can be handled by combining the terms with an AND or as a phrase query (see Section 2.4, page 39). It is harder for a system to handle the opposite case where the user entered as two terms something that was tokenized together in the document processing.

pounds that are sometimes written as a single word and sometimes space separated (such as *white space* vs. *whitespace*). Other cases with internal spaces that we might wish to regard as a single token include phone numbers ((800) 234-2333) and dates (Mar 11, 1983). Splitting tokens on spaces can cause bad retrieval results, for example, if a search for York University mainly returns documents containing *New York University*. The problems of hyphens and non-separating whitespace can even interact. Advertisements for air fares frequently contain items like *San Francisco-Los Angeles*, where simply doing whitespace splitting would give unfortunate results. In such cases, issues of tokenization interact with handling phrase queries (which we discuss in Section 2.4 (page 39)), particularly if we would like queries for all of *lowercase*, *lower-case* and *lower case* to return the same results. The last two can be handled by splitting on hyphens and using a phrase index. Getting the first case right would depend on knowing that it is sometimes written as two words and also indexing it in this way. One effective strategy in practice, which is used by some Boolean retrieval systems such as Westlaw and Lexis-Nexis (Example 1.1), is to encourage users to enter hyphens wherever they may be possible, and whenever there is a hyphenated form, the system will generalize the query to cover all three of the one word, hyphenated, and two word forms, so that a query for over-eager will search for over-eager OR "over eager" OR overeager. However, this strategy depends on user training, since if you query using either of the other two forms, you get no generalization.

Each new language presents some new issues. For instance, French has a variant use of the apostrophe for a reduced definite article 'the' before a word beginning with a vowel (e.g., *l'ensemble*) and has some uses of the hyphen with postposed clitic pronouns in imperatives and questions (e.g., *donne-moi* 'give me'). Getting the first case correct will affect the correct indexing of a fair percentage of nouns and adjectives: you would want documents mentioning both *l'ensemble* and *un ensemble* to be indexed under *ensemble*. Other languages make the problem harder in new ways. German writes *compound nouns* without spaces (e.g., *Computerlinguistik* 'computational linguistics'; *Lebensversicherungsgesellschaftsangestellter* 'life insurance company employee'). Retrieval systems for German greatly benefit from the use of a *compound-splitter* module, which is usually implemented by seeing if a word can be subdivided into multiple words that appear in a vocabulary. This phenomenon reaches its limit case with major East Asian Languages (e.g., Chinese, Japanese, Korean, and Thai), where text is written without any spaces between words. An example is shown in Figure 2.3. One approach here is to perform *word segmentation* as prior linguistic processing. Methods of word segmentation vary from having a large vocabulary and taking the longest vocabulary match with some heuristics for unknown words to the use of machine learning sequence models, such as hidden Markov models or conditional random fields, trained over hand-segmented words (see the references

COMPOUNDS

COMPOUND-SPLITTER

WORD SEGMENTATION

莎拉波娃现在居住在美国东南部的佛罗里达。今年 4 月 9 日，莎拉波娃在美国第一大城市纽约度过了 1 8 岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

► **Figure 2.3** The standard unsegmented form of Chinese text using the simplified characters of mainland China. There is no whitespace between words, not even between sentences – the apparent space after the Chinese period (。) is just a typographical illusion caused by placing the character on the left side of its square box. The first sentence is just words in Chinese characters with no spaces between them. The second and third sentences include Arabic numerals and punctuation breaking up the Chinese characters.

和尚

► **Figure 2.4** Ambiguities in Chinese word segmentation. The two characters can be treated as one word meaning 'monk' or as a sequence of two words meaning 'and' and 'still'.

a	an	and	are	as	at	be	by	for	from
has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

► **Figure 2.5** A stop list of 25 semantically non-selective words which are common in Reuters-RCV1.

in Section 2.5). Since there are multiple possible segmentations of character sequences (see Figure 2.4), all such methods make mistakes sometimes, and so you are never guaranteed a consistent unique tokenization. The other approach is to abandon word-based indexing and to do all indexing via just short subsequences of characters (character *k*-grams), regardless of whether particular sequences cross word boundaries or not. Three reasons why this approach is appealing are that an individual Chinese character is more like a syllable than a letter and usually has some semantic content, that most words are short (the commonest length is 2 characters), and that, given the lack of standardization of word breaking in the writing system, it is not always clear where word boundaries should be placed anyway. Even in English, some cases of where to put word boundaries are just orthographic conventions – think of *notwithstanding* vs. *not to mention* or *into* vs. *on to* – but people are educated to write the words with consistent use of spaces.

2.2.2 Dropping common terms: stop words

STOP WORDS
COLLECTION FREQUENCY
STOP LIST

Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called *stop words*. The general strategy for determining a stop list is to sort the terms by *collection frequency* (the total number of times each term appears in the document collection), and then to take the most frequent terms, often hand-filtered for their semantic content relative to the domain of the documents being indexed, as a *stop list*, the members of which are then discarded during indexing. An example of a stop list is shown in Figure 2.5. Using a stop list significantly reduces the number of postings that a system has to store; we will present some statistics on this in Chapter 5 (see Table 5.1, page 87). And a lot of the time not indexing stop words does little harm: keyword searches with terms like the and by don't seem very useful. However, this is not true for phrase searches. The phrase query "President of the United States", which contains two stop words, is more precise than President AND "United States". The meaning of flights to London is likely to be lost if the word to is stopped out. A search for Vannevar Bush's article *As we may think* will be difficult if the first three words are stopped out, and the system searches simply for documents containing the word think. Some special query types are disproportionately affected. Some song titles and well known pieces of verse consist entirely of words that are commonly on stop lists (*To be or not to be*, *Let It Be*, *I don't want to be*, ...).

The general trend in IR systems over time has been from standard use of quite large stop lists (200–300 terms) to very small stop lists (7–12 terms) to no stop list whatsoever. Web search engines generally do not use stop lists. Some of the design of modern IR systems has focused precisely on how we can exploit the statistics of language so as to be able to cope with common words in better ways. We will show in Section 5.3 (page 95) how good compression techniques greatly reduce the cost of storing the postings for common words. Section 6.2.1 (page 117) then discusses how standard term weighting leads to very common words having little impact on document rankings. Finally, Section 7.1.5 (page 140) shows how an IR system with impact-sorted indexes can terminate scanning a postings list early when weights get small, and hence common words do not cause a large additional processing cost for the average query, even though postings lists for stop words are very long. So for most modern IR systems, the additional cost of including stop words is not that big – neither in terms of index size nor in terms of query processing time.

Query term	Terms in documents that should be matched
Windows	Windows
windows	Windows, windows, window
window	window, windows

► **Figure 2.6** An example of how asymmetric expansion of query terms can usefully model users' expectations.

2.2.3 Normalization (equivalence classing of terms)

Having broken up our documents (and also our query) into tokens, the easy case is if tokens in the query just match tokens in the token list of the document. However, there are many cases when two character sequences are not quite the same but you would like a match to occur. For instance, if you search for *USA*, you might hope to also match documents containing *U.S.A.*

TOKEN NORMALIZATION
EQUIVALENCE CLASSES

Token normalization is the process of canonicalizing tokens so that matches occur despite superficial differences in the character sequences of the tokens.[4] The most standard way to normalize is to implicitly create *equivalence classes*, which are normally named after one member of the set. For instance, if the tokens *anti-discriminatory* and *antidiscriminatory* are both mapped onto the term antidiscriminatory, in both the document text and queries, then searches for one term will retrieve documents that contain either.

The advantage of just using mapping rules that remove characters like hyphens is that the equivalence classing to be done is implicit, rather than being fully calculated in advance: the terms that happen to become identical as the result of these rules are the equivalence classes. It is only easy to write rules of this sort that remove characters. Since the equivalence classes are implicit, it is not obvious when you might want to add characters. For instance, it would be hard to know to turn *antidiscriminatory* into *anti-discriminatory*.

An alternative to creating equivalence classes is to maintain relations between unnormalized tokens. This method can be extended to hand-constructed lists of synonyms such as *car* and *automobile*, a topic we discuss further in Chapter 9. These term relationships can be achieved in two ways. The usual way is to index unnormalized tokens and to maintain a query expansion list of multiple vocabulary entries to consider for a certain query term. A query term is then effectively a disjunction of several postings lists. The alternative is to perform the expansion during index construction. When the document contains automobile, we index it under car as well (and, usually, also vice-versa). Use of either of these methods is considerably less efficient than equivalence classing, as there are more postings to store and merge. The first

4. It is also often referred to as *term normalization*, but we prefer to reserve the name *term* for the output of the normalization process.

method adds a query expansion dictionary and requires more processing at query time, while the second method requires more space for storing postings. Traditionally, expanding the space required for the postings lists was seen as more disadvantageous, but with modern storage costs, the increased flexibility that comes from distinct postings lists is appealing.

These approaches are more flexible than equivalence classes because the expansion lists can overlap while not being identical. This means there can be an asymmetry in expansion. An example of how such an asymmetry can be exploited is shown in Figure 2.6: if the user enters windows, we wish to allow matches with the capitalized *Windows* operating system, but this is not plausible if the user enters window, even though it is plausible for this query to also match lowercase *windows*.

The best amount of equivalence classing or query expansion to do is a fairly open question. Doing some definitely seems a good idea. But doing a lot can easily have unexpected consequences of broadening queries in unintended ways. For instance, equivalence-classing *U.S.A.* and *USA* to the latter by deleting periods from tokens might at first seem very reasonable, given the prevalent pattern of optional use of periods in acronyms. However, if I put in as my query term *C.A.T.*, I might be rather upset if it matches every appearance of the word *cat* in documents.[5]

Below we present some of the forms of normalization that are commonly employed and how they are implemented. In many cases they seem helpful, but they can also do harm. In fact, you can worry about many details of equivalence classing, but it often turns out that providing processing is done consistently to the query and to documents, the fine details may not have much aggregate effect on performance.

Accents and diacritics. Diacritics on characters in English have a fairly marginal status, and we might well want *cliché* and *cliche* to match, or *naive* and *naïve*. This can be done by normalizing tokens to remove diacritics. In many other languages, diacritics are a regular part of the writing system and distinguish different sounds. Occasionally words are distinguished only by their accents. For instance, in Spanish, *peña* is 'a cliff', while *pena* is 'sorrow'. Nevertheless, the important question is usually not prescriptive or linguistic but is a question of how users are likely to write queries for these words. In many cases, users will enter queries for words without diacritics, whether for reasons of speed, laziness, limited software, or habits born of the days when it was hard to use non-ASCII text on many computer systems. In these cases, it might be best to equate all words to a form without diacritics.

5. At the time we wrote this chapter (Aug. 2005), this was actually the case on Google: the top result for the query *C.A.T.* was a site about cats, the Cat Fanciers Web Site http://www.fanciers.com/.

CASE-FOLDING **Capitalization/case-folding.** A common strategy is to do *case-folding* by reducing all letters to lower case. Often this is a good idea: it will allow instances of *Automobile* at the beginning of a sentence to match with a query of *automobile*. It will also help on a web search engine when most of your users type in *ferrari* when they are interested in a *Ferrari* car. On the other hand, such case folding can equate words that might better be kept apart. Many proper nouns are derived from common nouns and so are distinguished only by case, including companies (*General Motors*, *The Associated Press*), government organizations (*the Fed* vs. *fed*) and person names (*Bush*, *Black*). We already mentioned an example of unintended query expansion with acronyms, which involved not only acronym normalization (*C.A.T.* $\rightarrow$ *CAT*) but also case-folding (*CAT* $\rightarrow$ *cat*).

For English, an alternative to making every token lowercase is to just make some tokens lowercase. The simplest heuristic is to convert to lowercase words at the beginning of a sentence and all words occurring in a title that is all uppercase or in which most or all words are capitalized. These words are usually ordinary words that have been capitalized. Mid-sentence capitalized words are left as capitalized (which is usually correct). This will mostly avoid case-folding in cases where distinctions should be kept apart. The same task can be done more accurately by a machine learning sequence model which uses more features to make the decision of when to case-fold. This is known as *truecasing*. However, trying to get capitalization right in this way probably doesn't help if your users usually use lowercase regardless of the correct case of words. Thus, lowercasing everything often remains the most practical solution.

TRUECASING

Other issues in English. Other possible normalizations are quite idiosyncratic and particular to English. For instance, you might wish to equate *ne'er* and *never* or the British spelling *colour* and the American spelling *color*. Dates, times and similar items come in multiple formats, presenting additional challenges. You might wish to collapse together *3/12/91* and *Mar. 12, 1991*. However, correct processing here is complicated by the fact that in the U.S., *3/12/91* is *Mar. 12, 1991*, whereas in Europe it is *3 Dec 1991*.

Other languages. English has maintained a dominant position on the WWW; approximately 60% of web pages are in English (Gerrand 2007). But that still leaves 40% of the web, and the non-English portion might be expected to grow over time, since less than one third of Internet users and less than 10% of the world's population primarily speak English. And there are signs of change: Sifry (2007) reports that only about one third of blog posts are in English.

Other languages again present distinctive issues in equivalence classing.

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるＭＯＴＴＡＩＮＡＩキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

► **Figure 2.7** Japanese makes use of multiple intermingled writing systems and, like Chinese, does not segment words. The text is mainly Chinese characters with the hiragana syllabary for inflectional endings and function words. The part in latin letters is actually a Japanese expression, but has been taken up as the name of an environmental campaign by 2004 Nobel Peace Prize winner Wangari Maathai. His name is written using the katakana syllabary in the middle of the first line. The first four characters of the final line express a monetary amount that we would want to match with ¥500,000 (500,000 Japanese yen).

The French word for *the* has distinctive forms based not only on the gender (masculine or feminine) and number of the following noun, but also depending on whether the following word begins with a vowel: *le*, *la*, *l'*, *les*. We may well wish to equivalence class these various forms of *the*. German has a convention whereby vowels with an umlaut can be rendered instead as a two vowel digraph. We would want to treat *Schütze* and *Schuetze* as equivalent.

Japanese is a well-known difficult writing system, as illustrated in Figure 2.7. Modern Japanese is standardly an intermingling of multiple alphabets, principally Chinese characters, two syllabaries (hiragana and katakana) and western characters (Latin letters, Arabic numerals, and various symbols). While there are strong conventions and standardization through the education system over the choice of writing system, in many cases the same word can be written with multiple writing systems. For example, a word may be written in katakana for emphasis (somewhat like italics). Or a word may sometimes be written in hiragana and sometimes in Chinese characters. Successful retrieval thus requires complex equivalence classing across the writing systems. In particular, an end user might commonly present a query entirely in hiragana, because it is easier to type, just as Western end users commonly use all lowercase.

Document collections being indexed can include documents from many different languages. Or a single document can easily contain text from multiple languages. For instance, a French email might quote clauses from a contract document written in English. Most commonly, the language is detected and language-particular tokenization and normalization rules are applied at a predetermined granularity, such as whole documents or individual paragraphs, but this still will not correctly deal with cases where language changes occur for brief quotations. When document collections contain mul-

tiple languages, a single index may have to contain terms of several languages. One option is to run a language identification classifier on documents and then to tag terms in the vocabulary for their language. Or this tagging can simply be omitted, since it is relatively rare for the exact same character sequence to be a word in different languages.

When dealing with foreign or complex words, particularly foreign names, the spelling may be unclear or there may be variant transliteration standards giving different spellings (for example, *Chebyshev* and *Tchebycheff* or *Beijing* and *Peking*). One way of dealing with this is to use heuristics to equivalence class or expand terms with phonetic equivalents. The traditional and best known such algorithm is the Soundex algorithm, which we cover in Section 3.4 (page 63).

2.2.4 Stemming and lemmatization

For grammatical reasons, documents are going to use different forms of a word, such as *organize*, *organizes*, and *organizing*. Additionally, there are families of derivationally related words with similar meanings, such as *democracy*, *democratic*, and *democratization*. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set.

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is ⇒ be
car, cars, car's, cars' ⇒ car

The result of this mapping of text will be something like:

the boy's cars are different colors ⇒
the boy car be differ color

STEMMING

LEMMATIZATION

LEMMA

However, the two words differ in their flavor. *Stemming* usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. *Lemmatization* usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*. If confronted with the token *saw*, stemming might return just *s*, whereas lemmatization would attempt to return either *see* or *saw* depending on whether the use of the token was as a verb or a noun. The two may also differ in that stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma.

Linguistic processing for stemming or lemmatization is often done by an additional plug-in component to the indexing process, and a number of such components exist, both commercial and open-source.

PORTER STEMMER

The most common algorithm for stemming English, and one that has repeatedly been shown to be empirically very effective, is *Porter's algorithm* (Porter 1980). The entire algorithm is too long and intricate to present here, but we will indicate its general nature. Porter's algorithm consists of 5 phases of word reductions, applied sequentially. Within each phase there are various conventions to select rules, such as selecting the rule from each rule group that applies to the longest suffix. In the first phase, this convention is used with the following rule group:

(2.1)

Rule			Example		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

Many of the later rules use a concept of the *measure* of a word, which loosely checks the number of syllables to see whether a word is long enough that it is reasonable to regard the matching portion of a rule as a suffix rather than as part of the stem of a word. For example, the rule:

$$(m > 1) \quad \text{EMENT} \quad \rightarrow$$

would map *replacement* to *replac*, but not *cement* to *c*. The official site for the Porter Stemmer is:

http://www.tartarus.org/~martin/PorterStemmer/

Other stemmers exist, including the older, one-pass Lovins stemmer (Lovins 1968), and newer entrants like the Paice/Husk stemmer (Paice 1990); see:

http://www.cs.waikato.ac.nz/~eibe/stemmers/
http://www.comp.lancs.ac.uk/computing/research/stemming/

Figure 2.8 presents an informal comparison of the different behaviors of these stemmers. Stemmers use language-specific rules, but they require less knowledge than a lemmatizer, which needs a complete vocabulary and morphological analysis to correctly lemmatize words. Particular domains may also require special stemming rules. However, the exact stemmed form does not matter, only the equivalence classes it forms.

LEMMATIZER

Rather than using a stemmer, you can use a *lemmatizer*, a tool from Natural Language Processing which does full morphological analysis to accurately identify the lemma for each word. Doing full morphological analysis produces at most very modest benefits for retrieval. It is hard to say more,

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

► **Figure 2.8** A comparison of three stemming algorithms on a sample text.

because either form of normalization tends not to improve English information retrieval performance in aggregate – at least not by very much. While it helps a lot for some queries, it equally hurts performance a lot for others. Stemming increases recall while harming precision. As an example of what can go wrong, note that the Porter stemmer stems all of the following words:

operate operating operates operation operative operatives operational

to oper. However, since *operate* in its various forms is a common verb, we would expect to lose considerable precision on queries such as the following with Porter stemming:

operational AND research
operating AND system
operative AND dentistry

For a case like this, moving to using a lemmatizer would not completely fix the problem because particular inflectional forms are used in particular collocations: a sentence with the words *operate* and *system* is not a good match for the query operating AND system. Getting better value from term normalization depends more on pragmatic issues of word use than on formal issues of linguistic morphology.

The situation is different for languages with much more morphology (such as Spanish, German, and Finnish). Results in the European CLEF evaluations have repeatedly shown quite large gains from the use of stemmers (and compound splitting for languages like German); see the references in Section 2.5.

Exercise 2.1 [⋆]

Are the following statements true or false?

a. In a Boolean retrieval system, stemming never lowers precision.

b. In a Boolean retrieval system, stemming never lowers recall.

c. Stemming increases the size of the vocabulary.

d. Stemming should be invoked at indexing time but not while processing a query.

Exercise 2.2 [⋆]

Suggest what normalized form should be used for these words (including the word itself as a possibility):

a. 'Cos

b. Shi'ite

c. cont'd

d. Hawai'i

e. O'Rourke

Exercise 2.3 [⋆]

The following pairs of words are stemmed to the same form by the Porter stemmer. Which pairs would you argue shouldn't be conflated. Give your reasoning.

a. abandon/abandonment

b. absorbency/absorbent

c. marketing/markets

d. university/universe

e. volume/volumes

Exercise 2.4 [⋆]

For the Porter stemmer rule group shown in (2.1):

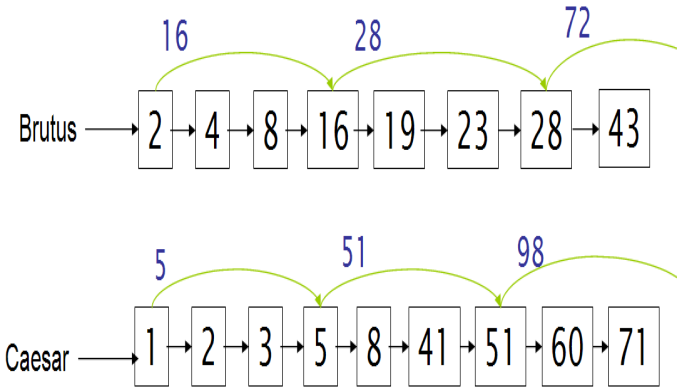
a. What is the purpose of including an identity rule such as SS → SS?

b. Applying just this rule group, what will the following words be stemmed to?

circus *canaries* *boss*

c. What rule should be added to correctly stem *pony*?

d. The stemming for *ponies* and *pony* might seem strange. Does it have a deleterious effect on retrieval? Why or why not?



► **Figure 2.9** Postings lists with skip pointers. The postings intersection can use a skip pointer when the end point is still less than the item on the other list.

2.3 Faster postings list intersection via skip pointers

In the remainder of this chapter, we will discuss extensions to postings list data structures and ways to increase the efficiency of using postings lists. Recall the basic postings list intersection operation from Section 1.3 (page 10): we walk through the two postings lists simultaneously, in time linear in the total number of postings entries. If the list lengths are m and n, the intersection takes $O(m+n)$ operations. Can we do better than this? That is, empirically, can we usually process postings list intersection in sublinear time? We can, if the index isn't changing too fast.

SKIP LIST

One way to do this is to use a *skip list* by augmenting postings lists with skip pointers (at indexing time), as shown in Figure 2.9. Skip pointers are effectively shortcuts that allow us to avoid processing parts of the postings list that will not figure in the search results. The two questions are then where to place skip pointers and how to do efficient merging using skip pointers.

Consider first efficient merging, with Figure 2.9 as an example. Suppose we've stepped through the lists in the figure until we have matched 8 on each list and moved it to the results list. We advance both pointers, giving us 16 on the upper list and 41 on the lower list. The smallest item is then the element 16 on the top list. Rather than simply advancing the upper pointer, we first check the skip list pointer and note that 28 is also less than 41. Hence we can follow the skip list pointer, and then we advance the upper pointer to 28. We thus avoid stepping to 19 and 23 on the upper list. A number of variant versions of postings list intersection with skip pointers is possible depending on when exactly you check the skip pointer. One version is shown

```
INTERSECTWITHSKIPS(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
 9                     then while hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
10                          do p1 ← skip(p1)
11                     else p1 ← next(p1)
12                else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
13                     then while hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
14                          do p2 ← skip(p2)
15                     else p2 ← next(p2)
16  return answer
```

► **Figure 2.10** Postings lists intersection with skip pointers.

in Figure 2.10. Skip pointers will only be available for the original postings lists. For an intermediate result in a complex query, the call $hasSkip(p)$ will always return false. Finally, note that the presence of skip pointers only helps for AND queries, not for OR queries.

Where do we place skips? There is a tradeoff. More skips means shorter skip spans, and that we are more likely to skip. But it also means lots of comparisons to skip pointers, and lots of space storing skip pointers. Fewer skips means few pointer comparisons, but then long skip spans which means that there will be fewer opportunities to skip. A simple heuristic for placing skips, which has been found to work well in practice, is that for a postings list of length P, use $\sqrt{P}$ evenly-spaced skip pointers. This heuristic can be improved upon; it ignores any details of the distribution of query terms.

Building effective skip pointers is easy if an index is relatively static; it is harder if a postings list keeps changing because of updates. A malicious deletion strategy can render skip lists ineffective.

Choosing the optimal encoding for an inverted index is an ever-changing game for the system builder, because it is strongly dependent on underlying computer technologies and their relative speeds and sizes. Traditionally, CPUs were slow, and so highly compressed techniques were not optimal. Now CPUs are fast and disk is slow, so reducing disk postings list size dominates. However, if you're running a search engine with everything in mem-

ory then the equation changes again. We discuss the impact of hardware parameters on index construction time in Section 4.1 (page 68) and the impact of index size on system speed in Chapter 5.

? **Exercise 2.5** [⋆]

Why are skip pointers not useful for queries of the form x OR y?

Exercise 2.6 [⋆]

We have a two-word query. For one term the postings list consists of the following 16 entries:

[4,6,10,12,14,16,18,20,22,32,47,81,120,122,157,180]

and for the other it is the one entry postings list:

[47].

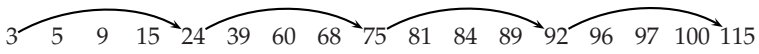
Work out how many comparisons would be done to intersect the two postings lists with the following two strategies. Briefly justify your answers:

a. Using standard postings lists

b. Using postings lists stored with skip pointers, with a skip length of $\sqrt{P}$, as suggested in Section 2.3.

Exercise 2.7 [⋆]

Consider a postings intersection between this postings list, with skip pointers:



and the following intermediate result postings list (which hence has no skip pointers):

3 5 89 95 97 99 100 101

Trace through the postings intersection algorithm in Figure 2.10 (page 37).

a. How often is a skip pointer followed (i.e., p_1 is advanced to $skip(p_1)$)?

b. How many postings comparisons will be made by this algorithm while intersecting the two lists?

c. How many postings comparisons would be made if the postings lists are intersected without the use of skip pointers?

2.4 Positional postings and phrase queries

Many complex or technical concepts and many organization and product names are multiword compounds or phrases. We would like to be able to pose a query such as Stanford University by treating it as a phrase so that a sentence in a document like *The inventor Stanford Ovshinsky never went to university.* is not a match. Most recent search engines support a double quotes syntax ("stanford university") for *phrase queries*, which has proven to be very easily understood and successfully used by users. As many as 10% of web queries are phrase queries, and many more are implicit phrase queries (such as person names), entered without use of double quotes. To be able to support such queries, it is no longer sufficient for postings lists to be simply lists of documents that contain individual terms. In this section we consider two approaches to supporting phrase queries and their combination. A search engine should not only support phrase queries, but implement them efficiently. A related but distinct concept is term proximity weighting, where a document is preferred to the extent that the query terms appear close to each other in the text. This technique is covered in Section 7.2.2 (page 144) in the context of ranked retrieval.

PHRASE QUERIES

2.4.1 Biword indexes

One approach to handling phrases is to consider every pair of consecutive terms in a document as a phrase. For example, the text *Friends, Romans, Countrymen* would generate the *biwords*:

BIWORD INDEX

friends romans
romans countrymen

In this model, we treat each of these biwords as a vocabulary term. Being able to process two-word phrase queries is immediate. Longer phrases can be processed by breaking them down. The query stanford university palo alto can be broken into the Boolean query on biwords:

"stanford university" AND "university palo" AND "palo alto"

This query could be expected to work fairly well in practice, but there can and will be occasional false positives. Without examining the documents, we cannot verify that the documents matching the above Boolean query do actually contain the original 4 word phrase.

Among possible queries, nouns and noun phrases have a special status in describing the concepts people are interested in searching for. But related nouns can often be divided from each other by various function words, in phrases such as *the abolition of slavery* or *renegotiation of the constitution*. These needs can be incorporated into the biword indexing model in the following

way. First, we tokenize the text and perform part-of-speech-tagging.[6] We can then group terms into nouns, including proper nouns, (N) and function words, including articles and prepositions, (X), among other classes. Now deem any string of terms of the form NX*N to be an extended biword. Each such extended biword is made a term in the vocabulary. For example:

renegotiation	of	the	constitution
N	X	X	N

To process a query using such an extended biword index, we need to also parse it into N's and X's, and then segment the query into extended biwords, which can be looked up in the index.

This algorithm does not always work in an intuitively optimal manner when parsing longer queries into Boolean queries. Using the above algorithm, the query

cost overruns on a power plant

is parsed into

"cost overruns" AND "overruns power" AND "power plant"

whereas it might seem a better query to omit the middle biword. Better results can be obtained by using more precise part-of-speech patterns that define which extended biwords should be indexed.

PHRASE INDEX

The concept of a biword index can be extended to longer sequences of words, and if the index includes variable length word sequences, it is generally referred to as a *phrase index*. Indeed, searches for a single term are not naturally handled in a biword index (you would need to scan the dictionary for all biwords containing the term), and so we also need to have an index of single-word terms. While there is always a chance of false positive matches, the chance of a false positive match on indexed phrases of length 3 or more becomes very small indeed. But on the other hand, storing longer phrases has the potential to greatly expand the vocabulary size. Maintaining exhaustive phrase indexes for phrases of length greater than two is a daunting prospect, and even use of an exhaustive biword dictionary greatly expands the size of the vocabulary. However, towards the end of this section we discuss the utility of the strategy of using a partial phrase index in a compound indexing scheme.

6. Part of speech taggers classify words as nouns, verbs, etc. – or, in practice, often as finer-grained classes like "plural proper noun". Many fairly accurate (c. 96% per-tag accuracy) part-of-speech taggers now exist, usually trained by machine learning methods on hand-tagged text. See, for instance, Manning and Schütze (1999, ch. 10).

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

► **Figure 2.11** Positional index example. The word to has a document frequency 993,477, and occurs 6 times in document 1 at positions 7, 18, 33, etc.

2.4.2 Positional indexes

POSITIONAL INDEX

For the reasons given, a biword index is not the standard solution. Rather, a *positional index* is most commonly employed. Here, for each term in the vocabulary, we store postings of the form docID: ⟨position1, position2, . . . ⟩, as shown in Figure 2.11, where each position is a token index in the document. Each posting will also usually record the term frequency, for reasons discussed in Chapter 6.

To process a phrase query, you still need to access the inverted index entries for each distinct term. As before, you would start with the least frequent term and then work to further restrict the list of possible candidates. In the merge operation, the same general technique is used as before, but rather than simply checking that both terms are in a document, you also need to check that their positions of appearance in the document are compatible with the phrase query being evaluated. This requires working out offsets between the words.

Example 2.1: Satisfying phrase queries. Suppose the postings lists for to and be are as in Figure 2.11, and the query is "to be or not to be". The postings lists to access are: to, be, or, not. We will examine intersecting the postings lists for to and be. We first look for documents that contain both terms. Then, we look for places in the lists where there is an occurrence of *be* with a token index one higher than a position of *to*, and then we look for another occurrence of each word with token index 4 higher than the first occurrence. In the above lists, the pattern of occurrences that is a possible match is:

to: ⟨. . . ; 4:⟨. . . ,429,433⟩; . . . ⟩
be: ⟨. . . ; 4:⟨. . . ,430,434⟩; . . . ⟩

```
POSITIONALINTERSECT(p1, p2, k)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then l ← ⟨ ⟩
 5             pp1 ← positions(p1)
 6             pp2 ← positions(p2)
 7             while pp1 ≠ NIL
 8             do while pp2 ≠ NIL
 9                do if |pos(pp1) − pos(pp2)| ≤ k
10                      then ADD(l, pos(pp2))
11                      else if pos(pp2) > pos(pp1)
12                              then break
13                   pp2 ← next(pp2)
14                while l ≠ ⟨ ⟩ and |l[0] − pos(pp1)| > k
15                do DELETE(l[0])
16                for each ps ∈ l
17                do ADD(answer, ⟨docID(p1), pos(pp1), ps⟩)
18                pp1 ← next(pp1)
19             p1 ← next(p1)
20             p2 ← next(p2)
21        else if docID(p1) < docID(p2)
22                then p1 ← next(p1)
23                else p2 ← next(p2)
24  return answer
```

► **Figure 2.12** An algorithm for proximity intersection of postings lists p_1 and p_2. The algorithm finds places where the two terms appear within k words of each other and returns a list of triples giving docID and the term position in p_1 and p_2.

The same general method is applied for within k word proximity searches, of the sort we saw in Example 1.1 (page 15):

employment /3 place

Here, $/k$ means "within k words of (on either side)". Clearly, positional indexes can be used for such queries; biword indexes cannot. We show in Figure 2.12 an algorithm for satisfying within k word proximity searches; it is further discussed in Exercise 2.12.

Positional index size. Adopting a positional index expands required postings storage significantly, even if we compress position values/offsets as we

will discuss in Section 5.3 (page 95). Indeed, moving to a positional index also changes the asymptotic complexity of a postings intersection operation, because the number of items to check is now bounded not by the number of documents but by the total number of tokens in the document collection T. That is, the complexity of a Boolean query is $\Theta(T)$ rather than $\Theta(N)$. However, most applications have little choice but to accept this, since most users now expect to have the functionality of phrase and proximity searches.

Let's examine the space implications of having a positional index. A posting now needs an entry for each occurrence of a term. The index size thus depends on the average document size. The average web page has less than 1000 terms, but documents like SEC stock filings, books, and even some epic poems easily reach 100,000 terms. Consider a term with frequency 1 in 1000 terms on average. The result is that large documents cause an increase of two orders of magnitude in the space required to store the postings list:

Document size	Expected postings	Expected entries in positional posting
1000	1	1
100,000	1	100

While the exact numbers depend on the type of documents and the language being indexed, some rough rules of thumb are to expect a positional index to be 2 to 4 times as large as a non-positional index, and to expect a compressed positional index to be about one third to one half the size of the raw text (after removal of markup, etc.) of the original uncompressed documents. Specific numbers for an example collection are given in Table 5.1 (page 87) and Table 5.6 (page 103).

2.4.3 Combination schemes

The strategies of biword indexes and positional indexes can be fruitfully combined. If users commonly query on particular phrases, such as Michael Jackson, it is quite inefficient to keep merging positional postings lists. A combination strategy uses a phrase index, or just a biword index, for certain queries and uses a positional index for other phrase queries. Good queries to include in the phrase index are ones known to be common based on recent querying behavior. But this is not the only criterion: the most expensive phrase queries to evaluate are ones where the individual words are common but the desired phrase is comparatively rare. Adding *Britney Spears* as a phrase index entry may only give a speedup factor to that query of about 3, since most documents that mention either word are valid results, whereas adding *The Who* as a phrase index entry may speed up that query by a factor of 1000. Hence, having the latter is more desirable, even if it is a relatively less common query.

NEXT WORD INDEX

Williams et al. (2004) evaluate an even more sophisticated scheme which employs indexes of both these sorts and additionally a partial next word index as a halfway house between the first two strategies. For each term, a *next word index* records terms that follow it in a document. They conclude that such a strategy allows a typical mixture of web phrase queries to be completed in one quarter of the time taken by use of a positional index alone, while taking up 26% more space than use of a positional index alone.

Exercise 2.8 [⋆]

Assume a biword index. Give an example of a document which will be returned for a query of New York University but is actually a false positive which should not be returned.

Exercise 2.9 [⋆]

Shown below is a portion of a positional index in the format: term: doc1: ⟨position1, position2, . . . ⟩; doc2: ⟨position1, position2, . . . ⟩; etc.

angels: 2: ⟨36,174,252,651⟩; 4: ⟨12,22,102,432⟩; 7: ⟨17⟩;
fools: 2: ⟨1,17,74,222⟩; 4: ⟨8,78,108,458⟩; 7: ⟨3,13,23,193⟩;
fear: 2: ⟨87,704,722,901⟩; 4: ⟨13,43,113,433⟩; 7: ⟨18,328,528⟩;
in: 2: ⟨3,37,76,444,851⟩; 4: ⟨10,20,110,470,500⟩; 7: ⟨5,15,25,195⟩;
rush: 2: ⟨2,66,194,321,702⟩; 4: ⟨9,69,149,429,569⟩; 7: ⟨4,14,404⟩;
to: 2: ⟨47,86,234,999⟩; 4: ⟨14,24,774,944⟩; 7: ⟨199,319,599,709⟩;
tread: 2: ⟨57,94,333⟩; 4: ⟨15,35,155⟩; 7: ⟨20,320⟩;
where: 2: ⟨67,124,393,1001⟩; 4: ⟨11,41,101,421,431⟩; 7: ⟨16,36,736⟩;

Which document(s) if any match each of the following queries, where each expression within quotes is a phrase query?

a. "fools rush in"

b. "fools rush in" AND "angels fear to tread"

Exercise 2.10 [⋆]

Consider the following fragment of a positional index with the format:

word: document: ⟨position, position, . . .⟩; document: ⟨position, . . .⟩
. . .

Gates: 1: ⟨3⟩; 2: ⟨6⟩; 3: ⟨2,17⟩; 4: ⟨1⟩;
IBM: 4: ⟨3⟩; 7: ⟨14⟩;
Microsoft: 1: ⟨1⟩; 2: ⟨1,21⟩; 3: ⟨3⟩; 5: ⟨16,22,51⟩;

The /*k* operator, word1 /*k* word2 finds occurrences of word1 within *k* words of word2 (on either side), where *k* is a positive integer argument. Thus $k = 1$ demands that word1 be adjacent to word2.

a. Describe the set of documents that satisfy the query Gates /2 Microsoft.

b. Describe each set of values for *k* for which the query Gates /*k* Microsoft returns a different set of documents as the answer.

Exercise 2.11 [⋆⋆]

Consider the general procedure for merging two positional postings lists for a given document, to determine the document positions where a document satisfies a $/k$ clause (in general there can be multiple positions at which each term occurs in a single document). We begin with a pointer to the position of occurrence of each term and move each pointer along the list of occurrences in the document, checking as we do so whether we have a hit for $/k$. Each move of either pointer counts as a step. Let L denote the total number of occurrences of the two terms in the document. What is the big-O complexity of the merge procedure, if we wish to have postings including positions in the result?

Exercise 2.12 [⋆⋆]

Consider the adaptation of the basic algorithm for intersection of two postings lists (Figure 1.6, page 11) to the one in Figure 2.12 (page 42), which handles proximity queries. A naive algorithm for this operation could be $O(PL_{\text{max}}^2)$, where P is the sum of the lengths of the postings lists (i.e., the sum of document frequencies) and L_{max} is the maximum length of a document (in tokens).

a. Go through this algorithm carefully and explain how it works.

b. What is the complexity of this algorithm? Justify your answer carefully.

c. For certain queries and data distributions, would another algorithm be more efficient? What complexity does it have?

Exercise 2.13 [⋆⋆]

Suppose we wish to use a postings intersection procedure to determine simply the list of documents that satisfy a $/k$ clause, rather than returning the list of positions, as in Figure 2.12 (page 42). For simplicity, assume $k \geq 2$. Let L denote the total number of occurrences of the two terms in the document collection (i.e., the sum of their collection frequencies). Which of the following is true? Justify your answer.

a. The merge can be accomplished in a number of steps linear in L and independent of k, and we can ensure that each pointer moves only to the right.

b. The merge can be accomplished in a number of steps linear in L and independent of k, but a pointer may be forced to move non-monotonically (i.e., to sometimes back up)

c. The merge can require kL steps in some cases.

Exercise 2.14 [⋆⋆]

How could an IR system combine use of a positional index and use of stop words? What is the potential problem, and how could it be handled?

2.5 References and further reading

EAST ASIAN LANGUAGES

Exhaustive discussion of the character-level processing of East Asian languages can be found in Lunde (1998). Character bigram indexes are perhaps the most standard approach to indexing Chinese, although some systems use word segmentation. Due to differences in the language and writing system, word segmentation is most usual for Japanese (Luk and Kwok 2002, Kishida

et al. 2005). The structure of a character k-gram index over unsegmented text differs from that in Section 3.2.2 (page 54): there the k-gram dictionary points to postings lists of entries in the regular dictionary, whereas here it points directly to document postings lists. For further discussion of Chinese word segmentation, see Sproat et al. (1996), Sproat and Emerson (2003), Tseng et al. (2005), and Gao et al. (2005).

Lita et al. (2003) present a method for truecasing. Natural language processing work on computational morphology is presented in (Sproat 1992, Beesley and Karttunen 2003).

Language identification was perhaps first explored in cryptography; for example, Konheim (1981) presents a character-level k-gram language identification algorithm. While other methods such as looking for particular distinctive function words and letter combinations have been used, with the advent of widespread digital text, many people have explored the character n-gram technique, and found it to be highly successful (Beesley 1998, Dunning 1994, Cavnar and Trenkle 1994). Written language identification is regarded as a fairly easy problem, while spoken language identification remains more difficult; see Hughes et al. (2006) for a recent survey.

Experiments on and discussion of the positive and negative impact of stemming in English can be found in the following works: Salton (1989), Harman (1991), Krovetz (1995), Hull (1996). Hollink et al. (2004) provide detailed results for the effectiveness of language-specific methods on 8 European languages. In terms of percent change in mean average precision (see page 159) over a baseline system, diacritic removal gains up to 23% (being especially helpful for Finnish, French, and Swedish). Stemming helped markedly for Finnish (30% improvement) and Spanish (10% improvement), but for most languages, including English, the gain from stemming was in the range 0–5%, and results from a lemmatizer were poorer still. Compound splitting gained 25% for Swedish and 15% for German, but only 4% for Dutch. Rather than language-particular methods, indexing character k-grams (as we suggested for Chinese) could often give as good or better results: using within-word character 4-grams rather than words gave gains of 37% in Finnish, 27% in Swedish, and 20% in German, while even being slightly positive for other languages, such as Dutch, Spanish, and English. Tomlinson (2003) presents broadly similar results. Bar-Ilan and Gutman (2005) suggest that, at the time of their study (2003), the major commercial web search engines suffered from lacking decent language-particular processing; for example, a query on www.google.fr for l'électricité did not separate off the article *l'* but only matched pages with precisely this string of article+noun.

SKIP LIST

The classic presentation of skip pointers for IR can be found in Moffat and Zobel (1996). Extended techniques are discussed in Boldi and Vigna (2005). The main paper in the algorithms literature is Pugh (1990), which uses multilevel skip pointers to give expected $O(\log P)$ list access (the same expected

efficiency as using a tree data structure) with less implementational complexity. In practice, the effectiveness of using skip pointers depends on various system parameters. Moffat and Zobel (1996) report conjunctive queries running about five times faster with the use of skip pointers, but Bahle et al. (2002, p. 217) report that, with modern CPUs, using skip lists instead slows down search because it expands the size of the postings list (i.e., disk I/O dominates performance). In contrast, Strohman and Croft (2007) again show good performance gains from skipping, in a system architecture designed to optimize for the large memory spaces and multiple cores of recent CPUs.

Johnson et al. (2006) report that 11.7% of all queries in two 2002 web query logs contained phrase queries, though Kammenhuber et al. (2006) report only 3% phrase queries for a different data set. Silverstein et al. (1999) note that many queries without explicit phrase operators are actually implicit phrase searches.

6 *Scoring, term weighting and the vector space model*

Thus far we have dealt with indexes that support Boolean queries: a document either matches or does not match a query. In the case of large document collections, the resulting number of matching documents can far exceed the number a human user could possibly sift through. Accordingly, it is essential for a search engine to rank-order the documents matching a query. To do this, the search engine computes, for each matching document, a score with respect to the query at hand. In this chapter we initiate the study of assigning a score to a (query, document) pair. This chapter consists of three main ideas.

1. We introduce parametric and zone indexes in Section 6.1, which serve two purposes. First, they allow us to index and retrieve documents by metadata such as the language in which a document is written. Second, they give us a simple means for scoring (and thereby ranking) documents in response to a query.

2. Next, in Section 6.2 we develop the idea of weighting the importance of a term in a document, based on the statistics of occurrence of the term.

3. In Section 6.3 we show that by viewing each document as a vector of such weights, we can compute a score between a query and each document. This view is known as vector space scoring.

Section 6.4 develops several variants of term-weighting for the vector space model. Chapter 7 develops computational aspects of vector space scoring, and related topics.

As we develop these ideas, the notion of a query will assume multiple nuances. In Section 6.1 we consider queries in which specific query terms occur in specified regions of a matching document. Beginning Section 6.2 we will in fact relax the requirement of matching specific regions of a document; instead, we will look at so-called free text queries that simply consist of query terms with no specification on their relative order, importance or where in a document they should be found. The bulk of our study of scoring will be in this latter notion of a query being such a set of terms.

6.1 Parametric and zone indexes

We have thus far viewed a document as a sequence of terms. In fact, most documents have additional structure. Digital documents generally encode, in machine-recognizable form, certain *metadata* associated with each document. By metadata, we mean specific forms of data about a document, such as its author(s), title and date of publication. This metadata would generally include *fields* such as the date of creation and the format of the document, as well the author and possibly the title of the document. The possible values of a field should be thought of as finite – for instance, the set of all dates of authorship.

METADATA

FIELD

Consider queries of the form "find documents authored by William Shakespeare in 1601, containing the phrase alas poor Yorick". Query processing then consists as usual of postings intersections, except that we may merge postings from standard inverted as well as *parametric indexes*. There is one parametric index for each field (say, date of creation); it allows us to select only the documents matching a date specified in the query. Figure 6.1 illustrates the user's view of such a parametric search. Some of the fields may assume ordered values, such as dates; in the example query above, the year 1601 is one such field value. The search engine may support querying ranges on such ordered values; to this end, a structure like a B-tree may be used for the field's dictionary.

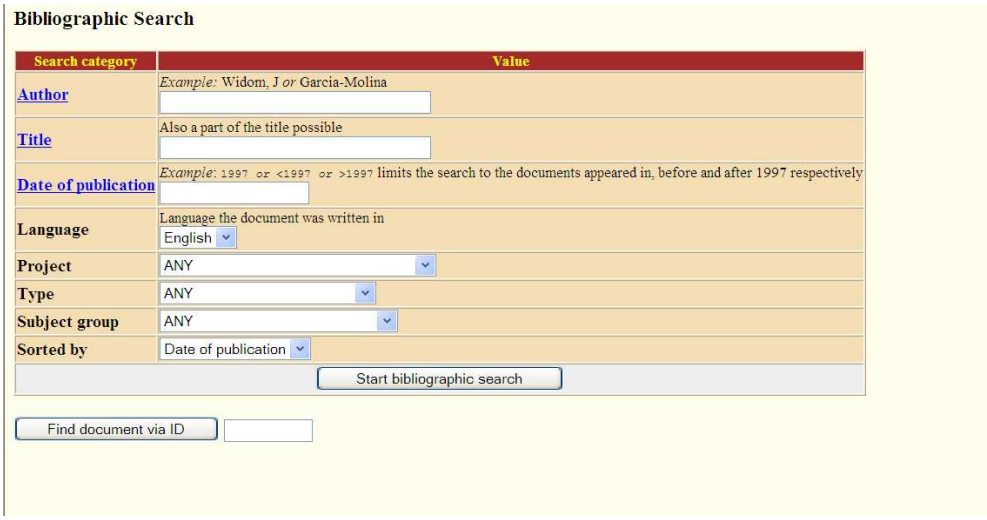
PARAMETRIC INDEX

ZONE

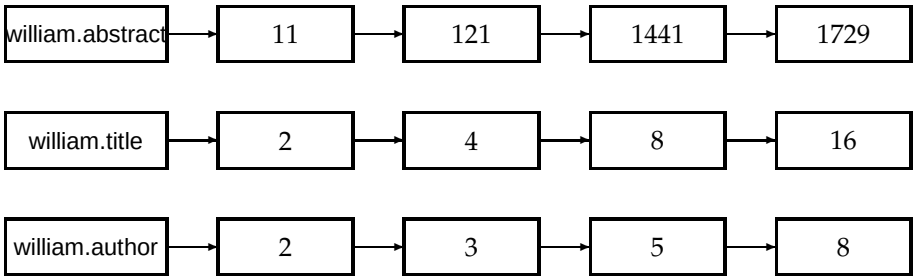
Zones are similar to fields, except the contents of a zone can be arbitrary free text. Whereas a field may take on a relatively small set of values, a zone can be thought of as an arbitrary, unbounded amount of text. For instance, document titles and abstracts are generally treated as zones. We may build a separate inverted index for each zone of a document, to support queries such as "find documents with merchant in the title and william in the author list and the phrase gentle rain in the body". This has the effect of building an index that looks like Figure 6.2. Whereas the dictionary for a parametric index comes from a fixed vocabulary (the set of languages, or the set of dates), the dictionary for a zone index must structure whatever vocabulary stems from the text of that zone.

In fact, we can reduce the size of the dictionary by encoding the zone in which a term occurs in the postings. In Figure 6.3 for instance, we show how occurrences of william in the title and author zones of various documents are encoded. Such an encoding is useful when the size of the dictionary is a concern (because we require the dictionary to fit in main memory). But there is another important reason why the encoding of Figure 6.3 is useful: the efficient computation of scores using a technique we will call *weighted zone scoring*.

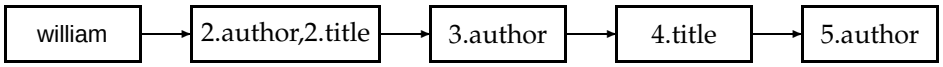
WEIGHTED ZONE SCORING



► **Figure 6.1** Parametric search. In this example we have a collection with fields allowing us to select publications by zones such as Author and fields such as Language.



► **Figure 6.2** Basic zone index ; zones are encoded as extensions of dictionary entries.



► **Figure 6.3** Zone index in which the zone is encoded in the postings rather than the dictionary.

6.1.1 Weighted zone scoring

Thus far in Section 6.1 we have focused on retrieving documents based on Boolean queries on fields and zones. We now turn to a second application of zones and fields.

Given a Boolean query q and a document d, weighted zone scoring assigns to the pair (q, d) a score in the interval $[0, 1]$, by computing a linear combination of *zone scores*, where each zone of the document contributes a Boolean value. More specifically, consider a set of documents each of which has ℓ zones. Let $g_1, \ldots, g_\ell \in [0, 1]$ such that $\sum_{i=1}^{\ell} g_i = 1$. For $1 \leq i \leq \ell$, let s_i be the Boolean score denoting a match (or absence thereof) between q and the ith zone. For instance, the Boolean score from a zone could be 1 if all the query term(s) occur in that zone, and zero otherwise; indeed, it could be any Boolean function that maps the presence of query terms in a zone to $0, 1$. Then, the weighted zone score is defined to be

$$\sum_{i=1}^{\ell} g_i s_i. \tag{6.1}$$

RANKED BOOLEAN RETRIEVAL

Weighted zone scoring is sometimes referred to also as *ranked Boolean retrieval*.

Example 6.1: Consider the query shakespeare in a collection in which each document has three zones: *author*, *title* and *body*. The Boolean score function for a zone takes on the value 1 if the query term shakespeare is present in the zone, and zero otherwise. Weighted zone scoring in such a collection would require three weights g_1, g_2 and g_3, respectively corresponding to the *author*, *title* and *body* zones. Suppose we set $g_1 = 0.2, g_2 = 0.3$ and $g_3 = 0.5$ (so that the three weights add up to 1); this corresponds to an application in which a match in the *author* zone is least important to the overall score, the *title* zone somewhat more, and the *body* contributes even more.

Thus if the term shakespeare were to appear in the *title* and *body* zones but not the *author* zone of a document, the score of this document would be 0.8.

How do we implement the computation of weighted zone scores? A simple approach would be to compute the score for each document in turn, adding in all the contributions from the various zones. However, we now show how we may compute weighted zone scores directly from inverted indexes. The algorithm of Figure 6.4 treats the case when the query q is a two-term query consisting of query terms q_1 and q_2, and the Boolean function is AND: 1 if both query terms are present in a zone and 0 otherwise. Following the description of the algorithm, we describe the extension to more complex queries and Boolean functions.

The reader may have noticed the close similarity between this algorithm and that in Figure 1.6. Indeed, they represent the same postings traversal, except that instead of merely adding a document to the set of results for

```
ZONESCORE(q1, q2)
 1  float scores[N] = [0]
 2  constant g[ℓ]
 3  p1 ← postings(q1)
 4  p2 ← postings(q2)
 5  // scores[] is an array with a score entry for each document, initialized to zero.
 6  //p1 and p2 are initialized to point to the beginning of their respective postings.
 7  //Assume g[] is initialized to the respective zone weights.
 8  while p1 ≠ NIL and p2 ≠ NIL
 9  do if docID(p1) = docID(p2)
10        then scores[docID(p1)] ← WEIGHTEDZONE(p1, p2, g)
11             p1 ← next(p1)
12             p2 ← next(p2)
13        else if docID(p1) < docID(p2)
14                then p1 ← next(p1)
15                else p2 ← next(p2)
16  return scores
```

► **Figure 6.4** Algorithm for computing the weighted zone score from two postings lists. Function WEIGHTEDZONE (not shown here) is assumed to compute the inner loop of Equation 6.1.

ACCUMULATOR

a Boolean AND query, we now compute a score for each such document. Some literature refers to the array scores[] above as a set of *accumulators*. The reason for this will be clear as we consider more complex Boolean functions than the AND; thus we may assign a non-zero score to a document even if it does not contain all query terms.

6.1.2 Learning weights

How do we determine the weights g_i for weighted zone scoring? These weights could be specified by an expert (or, in principle, the user); but increasingly, these weights are "learned" using training examples that have been judged editorially. This latter methodology falls under a general class of approaches to scoring and ranking in information retrieval, known as *machine-learned relevance*. We provide a brief introduction to this topic here because weighted zone scoring presents a clean setting for introducing it; a complete development demands an understanding of machine learning and is deferred to Chapter 15.

MACHINE-LEARNED RELEVANCE

1. We are provided with a set of *training examples*, each of which is a tuple consisting of a query q and a document d, together with a relevance

judgment for d on q. In the simplest form, each relevance judgments is either *Relevant* or *Non-relevant*. More sophisticated implementations of the methodology make use of more nuanced judgments.

2. The weights g_i are then "learned" from these examples, in order that the learned scores approximate the relevance judgments in the training examples.

For weighted zone scoring, the process may be viewed as learning a linear function of the Boolean match scores contributed by the various zones. The expensive component of this methodology is the labor-intensive assembly of user-generated relevance judgments from which to learn the weights, especially in a collection that changes frequently (such as the Web). We now detail a simple example that illustrates how we can reduce the problem of learning the weights g_i to a simple optimization problem.

We now consider a simple case of weighted zone scoring, where each document has a *title* zone and a *body* zone. Given a query q and a document d, we use the given Boolean match function to compute Boolean variables $s_T(d,q)$ and $s_B(d,q)$, depending on whether the title (respectively, body) zone of d matches query q. For instance, the algorithm in Figure 6.4 uses an AND of the query terms for this Boolean function. We will compute a score between 0 and 1 for each (document, query) pair using $s_T(d,q)$ and $s_B(d,q)$ by using a constant $g \in [0,1]$, as follows:

$$score(d,q) = g \cdot s_T(d,q) + (1-g)s_B(d,q). \tag{6.2}$$

We now describe how to determine the constant g from a set of *training examples*, each of which is a triple of the form $\Phi_j = (d_j, q_j, r(d_j,q_j))$. In each training example, a given training document d_j and a given training query q_j are assessed by a human editor who delivers a relevance judgment $r(d_j,q_j)$ that is either *Relevant* or *Non-relevant*. This is illustrated in Figure 6.5, where seven training examples are shown.

For each training example Φ_j we have Boolean values $s_T(d_j,q_j)$ and $s_B(d_j,q_j)$ that we use to compute a score from (6.2)

$$score(d_j,q_j) = g \cdot s_T(d_j,q_j) + (1-g)s_B(d_j,q_j). \tag{6.3}$$

We now compare this computed score to the human relevance judgment for the same document-query pair (d_j,q_j); to this end, we will quantize each *Relevant* judgment as a 1 and each *Non-relevant* judgment as a 0. Suppose that we define the error of the scoring function with weight g as

$$\varepsilon(g,\Phi_j) = (r(d_j,q_j) - score(d_j,q_j))^2,$$

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Non-relevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Non-relevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3191	driver	1	0	Non-relevant

► **Figure 6.5** An illustration of training examples.

s_T	s_B	Score
0	0	0
0	1	$1-g$
1	0	g
1	1	1

► **Figure 6.6** The four possible combinations of s_T and s_B.

where we have quantized the editorial relevance judgment $r(d_j, q_j)$ to 0 or 1. Then, the total error of a set of training examples is given by

$$\sum_j \varepsilon(g, \Phi_j). \tag{6.4}$$

The problem of learning the constant g from the given training examples then reduces to picking the value of g that minimizes the total error in (6.4).

Picking the best value of g in (6.4) in the formulation of Section 6.1.3 reduces to the problem of minimizing a quadratic function of g over the interval $[0, 1]$. This reduction is detailed in Section 6.1.3.

6.1.3 The optimal weight g

We begin by noting that for any training example Φ_j for which $s_T(d_j, q_j) = 0$ and $s_B(d_j, q_j) = 1$, the score computed by Equation (6.2) is $1 - g$. In similar fashion, we may write down the score computed by Equation (6.2) for the three other possible combinations of $s_T(d_j, q_j)$ and $s_B(d_j, q_j)$; this is summarized in Figure 6.6.

Let n_{01r} (respectively, n_{01n}) denote the number of training examples for which $s_T(d_j, q_j) = 0$ and $s_B(d_j, q_j) = 1$ and the editorial judgment is *Relevant* (respectively, *Non-relevant*). Then the contribution to the total error in Equation (6.4) from training examples for which $s_T(d_j, q_j) = 0$ and $s_B(d_j, q_j) = 1$

is

$$[1-(1-g)]^2 n_{01r} + [0-(1-g)]^2 n_{01n}.$$

By writing in similar fashion the error contributions from training examples of the other three combinations of values for $s_T(d_j, q_j)$ and $s_B(d_j, q_j)$ (and extending the notation in the obvious manner), the total error corresponding to Equation (6.4) is

$$(n_{01r} + n_{10n})g^2 + (n_{10r} + n_{01n})(1-g)^2 + n_{00r} + n_{11n}. \tag{6.5}$$

By differentiating Equation (6.5) with respect to g and setting the result to zero, it follows that the optimal value of g is

$$\frac{n_{10r} + n_{01n}}{n_{10r} + n_{10n} + n_{01r} + n_{01n}}. \tag{6.6}$$

Exercise 6.1

When using weighted zone scoring, is it necessary for all zones to use the same Boolean match function?

Exercise 6.2

In Example 6.1 above with weights $g_1 = 0.2, g_2 = 0.31$ and $g_3 = 0.49$, what are all the distinct score values a document may get?

Exercise 6.3

Rewrite the algorithm in Figure 6.4 to the case of more than two query terms.

Exercise 6.4

Write pseudocode for the function WeightedZone for the case of two postings lists in Figure 6.4.

Exercise 6.5

Apply Equation 6.6 to the sample training set in Figure 6.5 to estimate the best value of g for this sample.

Exercise 6.6

For the value of g estimated in Exercise 6.5, compute the weighted zone score for each (query, document) example. How do these scores relate to the relevance judgments in Figure 6.5 (quantized to 0/1)?

Exercise 6.7

Why does the expression for g in (6.6) not involve training examples in which $s_T(d_t, q_t)$ and $s_B(d_t, q_t)$ have the same value?

6.2 Term frequency and weighting

Thus far, scoring has hinged on whether or not a query term is present in a zone within a document. We take the next logical step: a document or zone that mentions a query term more often has more to do with that query and therefore should receive a higher score. To motivate this, we recall the notion of a free text query introduced in Section 1.4: a query in which the terms of the query are typed freeform into the search interface, without any connecting search operators (such as Boolean operators). This query style, which is extremely popular on the web, views the query as simply a set of words. A plausible scoring mechanism then is to compute a score that is the sum, over the query terms, of the match scores between each query term and the document.

Towards this end, we assign to each term in a document a *weight* for that term, that depends on the number of occurrences of the term in the document. We would like to compute a score between a query term t and a document d, based on the weight of t in d. The simplest approach is to assign the weight to be equal to the number of occurrences of term t in document d.

TERM FREQUENCY

This weighting scheme is referred to as *term frequency* and is denoted $\text{tf}_{t,d}$, with the subscripts denoting the term and the document in order.

For a document d, the set of weights determined by the tf weights above (or indeed any weighting function that maps the number of occurrences of t in d to a positive real value) may be viewed as a quantitative digest of that

BAG OF WORDS

document. In this view of a document, known in the literature as the *bag of words model*, the exact ordering of the terms in a document is ignored but the number of occurrences of each term is material (in contrast to Boolean retrieval). We only retain information on the number of occurrences of each term. Thus, the document "Mary is quicker than John" is, in this view, identical to the document "John is quicker than Mary". Nevertheless, it seems intuitive that two documents with similar bag of words representations are similar in content. We will develop this intuition further in Section 6.3.

Before doing so we first study the question: are all words in a document equally important? Clearly not; in Section 2.2.2 (page 27) we looked at the idea of *stop words* – words that we decide not to index at all, and therefore do not contribute in any way to retrieval and scoring.

6.2.1 Inverse document frequency

Raw term frequency as above suffers from a critical problem: all terms are considered equally important when it comes to assessing relevancy on a query. In fact certain terms have little or no discriminating power in determining relevance. For instance, a collection of documents on the auto industry is likely to have the term auto in almost every document. To this

Word	cf	df
try	10422	8760
insurance	10440	3997

► **Figure 6.7** Collection frequency (cf) and document frequency (df) behave differently, as in this example from the Reuters collection.

end, we introduce a mechanism for attenuating the effect of terms that occur too often in the collection to be meaningful for relevance determination. An immediate idea is to scale down the term weights of terms with high *collection frequency*, defined to be the total number of occurrences of a term in the collection. The idea would be to reduce the tf weight of a term by a factor that grows with its collection frequency.

DOCUMENT FREQUENCY

Instead, it is more commonplace to use for this purpose the *document frequency* df_t, defined to be the number of documents in the collection that contain a term t. This is because in trying to discriminate between documents for the purpose of scoring it is better to use a document-level statistic (such as the number of documents containing a term) than to use a collection-wide statistic for the term. The reason to prefer df to cf is illustrated in Figure 6.7, where a simple example shows that collection frequency (cf) and document frequency (df) can behave rather differently. In particular, the cf values for both try and insurance are roughly equal, but their df values differ significantly. Intuitively, we want the few documents that contain insurance to get a higher boost for a query on insurance than the many documents containing try get from a query on try.

How is the document frequency df of a term used to scale its weight? Denoting as usual the total number of documents in a collection by N, we define the *inverse document frequency* (idf) of a term t as follows:

INVERSE DOCUMENT FREQUENCY

$$\text{idf}_t = \log \frac{N}{\text{df}_t}. \tag{6.7}$$

Thus the idf of a rare term is high, whereas the idf of a frequent term is likely to be low. Figure 6.8 gives an example of idf's in the Reuters collection of 806,791 documents; in this example logarithms are to the base 10. In fact, as we will see in Exercise 6.12, the precise base of the logarithm is not material to ranking. We will give on page 227 a justification of the particular form in Equation (6.7).

6.2.2 Tf-idf weighting

We now combine the definitions of term frequency and inverse document frequency, to produce a composite weight for each term in each document.

term	df_t	idf_t
car	18,165	1.65
auto	6723	2.08
insurance	19,241	1.62
best	25,235	1.5

► **Figure 6.8** Example of idf values. Here we give the idf's of terms with various frequencies in the Reuters collection of 806,791 documents.

TF-IDF

The *tf-idf* weighting scheme assigns to term t a weight in document d given by

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t. \tag{6.8}$$

In other words, $\text{tf-idf}_{t,d}$ assigns to term t a weight in document d that is

1. highest when t occurs many times within a small number of documents (thus lending high discriminating power to those documents);
2. lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal);
3. lowest when the term occurs in virtually all documents.

DOCUMENT VECTOR

At this point, we may view each document as a *vector* with one component corresponding to each term in the dictionary, together with a weight for each component that is given by (6.8). For dictionary terms that do not occur in a document, this weight is zero. This vector form will prove to be crucial to scoring and ranking; we will develop these ideas in Section 6.3. As a first step, we introduce the *overlap score measure*: the score of a document d is the sum, over all query terms, of the number of times each of the query terms occurs in d. We can refine this idea so that we add up not the number of occurrences of each query term t in d, but instead the tf-idf weight of each term in d.

$$\text{Score}(q, d) = \sum_{t \in q} \text{tf-idf}_{t,d}. \tag{6.9}$$

In Section 6.3 we will develop a more rigorous form of Equation (6.9).

?

Exercise 6.8

Why is the idf of a term always finite?

Exercise 6.9

What is the idf of a term that occurs in every document? Compare this with the use of stop word lists.

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

► **Figure 6.9** Table of tf values for Exercise 6.10.

Exercise 6.10

Consider the table of term frequencies for 3 documents denoted Doc1, Doc2, Doc3 in Figure 6.9. Compute the tf-idf weights for the terms car, auto, insurance, best, for each document, using the idf values from Figure 6.8.

Exercise 6.11

Can the tf-idf weight of a term in a document exceed 1?

Exercise 6.12

How does the base of the logarithm in (6.7) affect the score calculation in (6.9)? How does the base of the logarithm affect the relative scores of two documents on a given query?

Exercise 6.13

If the logarithm in (6.7) is computed base 2, suggest a simple approximation to the idf of a term.

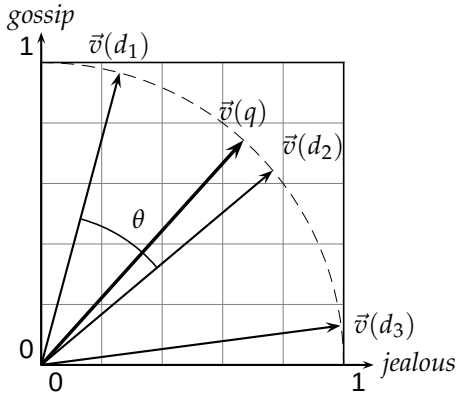
6.3 The vector space model for scoring

In Section 6.2 (page 117) we developed the notion of a document vector that captures the relative importance of the terms in a document. The representation of a set of documents as vectors in a common vector space is known as the *vector space model* and is fundamental to a host of information retrieval operations ranging from scoring documents on a query, document classification and document clustering. We first develop the basic ideas underlying vector space scoring; a pivotal step in this development is the view (Section 6.3.2) of queries as vectors in the same vector space as the document collection.

VECTOR SPACE MODEL

6.3.1 Dot products

We denote by $\vec{V}(d)$ the vector derived from document d, with one component in the vector for each dictionary term. Unless otherwise specified, the reader may assume that the components are computed using the tf-idf weighting scheme, although the particular weighting scheme is immaterial to the discussion that follows. The set of documents in a collection then may be viewed as a set of vectors in a vector space, in which there is one axis for



► **Figure 6.10** Cosine similarity illustrated. $\text{sim}(d_1, d_2) = \cos\theta$.

each term. This representation loses the relative ordering of the terms in each document; recall our example from Section 6.2 (page 117), where we pointed out that the documents *Mary is quicker than John* and *John is quicker than Mary* are identical in such a *bag of words* representation.

How do we quantify the similarity between two documents in this vector space? A first attempt might consider the magnitude of the vector difference between two document vectors. This measure suffers from a drawback: two documents with very similar content can have a significant vector difference simply because one is much longer than the other. Thus the relative distributions of terms may be identical in the two documents, but the absolute term frequencies of one may be far larger.

To compensate for the effect of document length, the standard way of quantifying the similarity between two documents d_1 and d_2 is to compute the *cosine similarity* of their vector representations $\vec{V}(d_1)$ and $\vec{V}(d_2)$

COSINE SIMILARITY

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}, \tag{6.10}$$

where the numerator represents the *dot product* (also known as the *inner product*) of the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$, while the denominator is the product of their *Euclidean lengths*. The dot product $\vec{x} \cdot \vec{y}$ of two vectors is defined as $\sum_{i=1}^{M} x_i y_i$. Let $\vec{V}(d)$ denote the document vector for d, with M components $\vec{V}_1(d) \ldots \vec{V}_M(d)$. The Euclidean length of d is defined to be $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$.

DOT PRODUCT

EUCLIDEAN LENGTH

The effect of the denominator of Equation (6.10) is thus to *length-normalize* the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$ to unit vectors $\vec{v}(d_1) = \vec{V}(d_1)/|\vec{V}(d_1)|$ and

LENGTH-NORMALIZATION

	Doc1	Doc2	Doc3
car	0.88	0.09	0.58
auto	0.10	0.71	0
insurance	0	0.71	0.70
best	0.46	0	0.41

► **Figure 6.11** Euclidean normalized tf values for documents in Figure 6.9.

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6

► **Figure 6.12** Term frequencies in three novels. The novels are Austen's *Sense and Sensibility, Pride and Prejudice* and Brontë's *Wuthering Heights.*

$\vec{v}(d_2) = \vec{V}(d_2)/|\vec{V}(d_2)|$. We can then rewrite (6.10) as

$$\text{sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2). \tag{6.11}$$

Example 6.2: Consider the documents in Figure 6.9. We now apply Euclidean normalization to the tf values from the table, for each of the three documents in the table. The quantity $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$ has the values 30.56, 46.84 and 41.30 respectively for Doc1, Doc2 and Doc3. The resulting Euclidean normalized tf values for these documents are shown in Figure 6.11.

Thus, (6.11) can be viewed as the dot product of the normalized versions of the two document vectors. This measure is the cosine of the angle θ between the two vectors, shown in Figure 6.10. What use is the similarity measure $\text{sim}(d_1, d_2)$? Given a document d (potentially one of the d_i in the collection), consider searching for the documents in the collection most similar to d. Such a search is useful in a system where a user may identify a document and seek others like it – a feature available in the results lists of search engines as a *more like this* feature. We reduce the problem of finding the document(s) most similar to d to that of finding the d_i with the highest dot products (sim values) $\vec{v}(d) \cdot \vec{v}(d_i)$. We could do this by computing the dot products between $\vec{v}(d)$ and each of $\vec{v}(d_1), \ldots, \vec{v}(d_N)$, then picking off the highest resulting sim values.

Example 6.3: Figure 6.12 shows the number of occurrences of three terms (affection, jealous and gossip) in each of the following three novels: Jane Austen's *Sense and Sensibility* (SaS) and *Pride and Prejudice* (PaP) and Emily Brontë's *Wuthering Heights* (WH).

term	SaS	PaP	WH
affection	0.996	0.993	0.847
jealous	0.087	0.120	0.466
gossip	0.017	0	0.254

► **Figure 6.13** Term vectors for the three novels of Figure 6.12. These are based on raw term frequency only and are normalized as if these were the only terms in the collection. (Since affection and jealous occur in all three documents, their tf-idf weight would be 0 in most formulations.)

Of course, there are many other terms occurring in each of these novels. In this example we represent each of these novels as a unit vector in three dimensions, corresponding to these three terms (only); we use raw term frequencies here, with no idf multiplier. The resulting weights are as shown in Figure 6.13.

Now consider the cosine similarities between pairs of the resulting three-dimensional vectors. A simple computation shows that sim($\vec{v}$(SAS), $\vec{v}$(PAP)) is 0.999, whereas sim($\vec{v}$(SAS), $\vec{v}$(WH)) is 0.888; thus, the two books authored by Austen (SaS and PaP) are considerably closer to each other than to Brontë's *Wuthering Heights*. In fact, the similarity between the first two is almost perfect (when restricted to the three terms we consider). Here we have considered tf weights, but we could of course use other term weight functions.

TERM-DOCUMENT MATRIX

Viewing a collection of N documents as a collection of vectors leads to a natural view of a collection as a *term-document matrix*: this is an $M \times N$ matrix whose rows represent the M terms (dimensions) of the N columns, each of which corresponds to a document. As always, the terms being indexed could be stemmed before indexing; for instance, jealous and jealousy would under stemming be considered as a single dimension. This matrix view will prove to be useful in Chapter 18.

6.3.2 Queries as vectors

There is a far more compelling reason to represent documents as vectors: we can also view a *query* as a vector. Consider the query $q =$ jealous gossip. This query turns into the unit vector $\vec{v}(q) = (0, 0.707, 0.707)$ on the three coordinates of Figures 6.12 and 6.13. The key idea now: to assign to each document d a score equal to the dot product

$$\vec{v}(q) \cdot \vec{v}(d).$$

In the example of Figure 6.13, *Wuthering Heights* is the top-scoring document for this query with a score of 0.509, with *Pride and Prejudice* a distant second with a score of 0.085, and *Sense and Sensibility* last with a score of 0.074. This simple example is somewhat misleading: the number of dimen-

sions in practice will be far larger than three: it will equal the vocabulary size M.

To summarize, by viewing a query as a "bag of words", we are able to treat it as a very short document. As a consequence, we can use the cosine similarity between the query vector and a document vector as a measure of the score of the document for that query. The resulting scores can then be used to select the top-scoring documents for a query. Thus we have

$$\text{score}(q,d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}. \tag{6.12}$$

A document may have a high cosine score for a query even if it does not contain all query terms. Note that the preceding discussion does not hinge on any specific weighting of terms in the document vector, although for the present we may think of them as either tf or tf-idf weights. In fact, a number of weighting schemes are possible for query as well as document vectors, as illustrated in Example 6.4 and developed further in Section 6.4.

Computing the cosine similarities between the query vector and each document vector in the collection, sorting the resulting scores and selecting the top K documents can be expensive — a single similarity computation can entail a dot product in tens of thousands of dimensions, demanding tens of thousands of arithmetic operations. In Section 7.1 we study how to use an inverted index for this purpose, followed by a series of heuristics for improving on this.

Example 6.4: We now consider the query best car insurance on a fictitious collection with $N = 1{,}000{,}000$ documents where the document frequencies of auto, best, car and insurance are respectively 5000, 50000, 10000 and 1000.

term	query				document			product
	tf	df	idf	$w_{t,q}$	tf	wf	$w_{t,d}$	
auto	0	5000	2.3	0	1	1	0.41	0
best	1	50000	1.3	1.3	0	0	0	0
car	1	10000	2.0	2.0	1	1	0.41	0.82
insurance	1	1000	3.0	3.0	2	2	0.82	2.46

In this example the weight of a term in the query is simply the idf (and zero for a term not in the query, such as auto); this is reflected in the column header $w_{t,q}$ (the entry for auto is zero because the query does not contain the termauto). For documents, we use tf weighting with no use of idf but with Euclidean normalization. The former is shown under the column headed wf, while the latter is shown under the column headed $w_{t,d}$. Invoking (6.9) now gives a net score of $0 + 0 + 0.82 + 2.46 = 3.28$.

6.3.3 Computing vector scores

In a typical setting we have a collection of documents each represented by a vector, a free text query represented by a vector, and a positive integer K. We

```
COSINESCORE(q)
 1  float Scores[N] = 0
 2  Initialize Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5     for each pair(d, tf_{t,d}) in postings list
 6     do Scores[d] += wf_{t,d} × w_{t,q}
 7  Read the array Length[d]
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

► **Figure 6.14** The basic algorithm for computing vector space scores.

seek the K documents of the collection with the highest vector space scores on the given query. We now initiate the study of determining the K documents with the highest vector space scores for a query. Typically, we seek these K top documents in ordered by decreasing score; for instance many search engines use $K = 10$ to retrieve and rank-order the first page of the ten best results. Here we give the basic algorithm for this computation; we develop a fuller treatment of efficient techniques and approximations in Chapter 7.

Figure 6.14 gives the basic algorithm for computing vector space scores. The array Length holds the lengths (normalization factors) for each of the N documents, whereas the array Scores holds the scores for each of the documents. When the scores are finally computed in Step 9, all that remains in Step 10 is to pick off the K documents with the highest scores.

The outermost loop beginning Step 3 repeats the updating of Scores, iterating over each query term t in turn. In Step 5 we calculate the weight in the query vector for term t. Steps 6-8 update the score of each document by adding in the contribution from term t. This process of adding in contributions one query term at a time is sometimes known as *term-at-a-time* scoring or accumulation, and the N elements of the array *Scores* are therefore known as *accumulators*. For this purpose, it would appear necessary to store, with each postings entry, the weight $\text{wf}_{t,d}$ of term t in document d (we have thus far used either tf or tf-idf for this weight, but leave open the possibility of other functions to be developed in Section 6.4). In fact this is wasteful, since storing this weight may require a floating point number. Two ideas help alleviate this space problem. First, if we are using inverse document frequency, we need not precompute idf_t; it suffices to store N/df_t at the head of the postings for t. Second, we store the term frequency $\text{tf}_{t,d}$ for each postings entry. Finally, Step 12 extracts the top K scores – this requires a priority queue

TERM-AT-A-TIME

ACCUMULATOR

data structure, often implemented using a heap. Such a heap takes no more than $2N$ comparisons to construct, following which each of the K top scores can be extracted from the heap at a cost of $O(\log N)$ comparisons.

Note that the general algorithm of Figure 6.14 does not prescribe a specific implementation of how we traverse the postings lists of the various query terms; we may traverse them one term at a time as in the loop beginning at Step 3, or we could in fact traverse them concurrently as in Figure 1.6. In such a concurrent postings traversal we compute the scores of one document at a time, so that it is sometimes called *document-at-a-time* scoring. We will say more about this in Section 7.1.5.

DOCUMENT-AT-A-TIME

?

Exercise 6.14

If we were to stem jealous and jealousy to a common stem before setting up the vector space, detail how the definitions of tf and idf should be modified.

Exercise 6.15

Recall the tf-idf weights computed in Exercise 6.10. Compute the Euclidean normalized document vectors for each of the documents, where each vector has four components, one for each of the four terms.

Exercise 6.16

Verify that the sum of the squares of the components of each of the document vectors in Exercise 6.15 is 1 (to within rounding error). Why is this the case?

Exercise 6.17

With term weights as computed in Exercise 6.15, rank the three documents by computed score for the query car insurance, for each of the following cases of term weighting in the query:

1. The weight of a term is 1 if present in the query, 0 otherwise.
2. Euclidean normalized idf.

6.4 Variant tf-idf functions

For assigning a weight for each term in each document, a number of alternatives to tf and tf-idf have been considered. We discuss some of the principal ones here; a more complete development is deferred to Chapter 11. We will summarize these alternatives in Section 6.4.3 (page 128).

6.4.1 Sublinear tf scaling

It seems unlikely that twenty occurrences of a term in a document truly carry twenty times the significance of a single occurrence. Accordingly, there has been considerable research into variants of term frequency that go beyond counting the number of occurrences of a term. A common modification is

to use instead the logarithm of the term frequency, which assigns a weight given by

$$\text{wf}_{t,d} = \begin{cases} 1 + \log \text{tf}_{t,d} & \text{if } \text{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{6.13}$$

In this form, we may replace tf by some other function wf as in (6.13), to obtain:

$$\text{wf-idf}_{t,d} = \text{wf}_{t,d} \times \text{idf}_t. \tag{6.14}$$

Equation (6.9) can then be modified by replacing tf-idf by wf-idf as defined in (6.14).

6.4.2 Maximum tf normalization

One well-studied technique is to normalize the tf weights of all terms occurring in a document by the maximum tf in that document. For each document d, let $\text{tf}_{\max}(d) = \max_{\tau \in d} \text{tf}_{\tau,d}$, where τ ranges over all terms in d. Then, we compute a normalized term frequency for each term t in document d by

$$\text{ntf}_{t,d} = a + (1-a)\frac{\text{tf}_{t,d}}{\text{tf}_{\max}(d)}, \tag{6.15}$$

SMOOTHING

where a is a value between 0 and 1 and is generally set to 0.4, although some early work used the value 0.5. The term a in (6.15) is a *smoothing* term whose role is to damp the contribution of the second term – which may be viewed as a scaling down of tf by the largest tf value in d. We will encounter smoothing further in Chapter 13 when discussing classification; the basic idea is to avoid a large swing in $\text{ntf}_{t,d}$ from modest changes in $\text{tf}_{t,d}$ (say from 1 to 2). The main idea of maximum tf normalization is to mitigate the following anomaly: we observe higher term frequencies in longer documents, merely because longer documents tend to repeat the same words over and over again. To appreciate this, consider the following extreme example: supposed we were to take a document d and create a new document d' by simply appending a copy of d to itself. While d' should be no more relevant to any query than d is, the use of (6.9) would assign it twice as high a score as d. Replacing $\text{tf-idf}_{t,d}$ in (6.9) by $\text{ntf-idf}_{t,d}$ eliminates the anomaly in this example. Maximum tf normalization does suffer from the following issues:

1. The method is unstable in the following sense: a change in the stop word list can dramatically alter term weightings (and therefore ranking). Thus, it is hard to tune.
2. A document may contain an outlier term with an unusually large number of occurrences of that term, not representative of the content of that document.

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1+\log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5+\frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0,\log\frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases}1 & \text{if } \mathrm{tf}_{t,d}>0\\ 0 & \text{otherwise}\end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha<1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

► **Figure 6.15** SMART notation for tf-idf variants. Here *CharLength* is the number of characters in the document.

3. More generally, a document in which the most frequent term appears roughly as often as many other terms should be treated differently from one with a more skewed distribution.

6.4.3 Document and query weighting schemes

Equation (6.12) is fundamental to information retrieval systems that use any form of vector space scoring. Variations from one vector space scoring method to another hinge on the specific choices of weights in the vectors $\vec{V}(d)$ and $\vec{V}(q)$. Figure 6.15 lists some of the principal weighting schemes in use for each of $\vec{V}(d)$ and $\vec{V}(q)$, together with a mnemonic for representing a specific combination of weights; this system of mnemonics is sometimes called SMART notation, following the authors of an early text retrieval system. The mnemonic for representing a combination of weights takes the form *ddd.qqq* where the first triplet gives the term weighting of the document vector, while the second triplet gives the weighting in the query vector. The first letter in each triplet specifies the term frequency component of the weighting, the second the document frequency component, and the third the form of normalization used. It is quite common to apply different normalization functions to $\vec{V}(d)$ and $\vec{V}(q)$. For example, a very standard weighting scheme is *lnc.ltc*, where the document vector has log-weighted term frequency, no idf (for both effectiveness and efficiency reasons), and cosine normalization, while the query vector uses log-weighted term frequency, idf weighting, and cosine normalization.

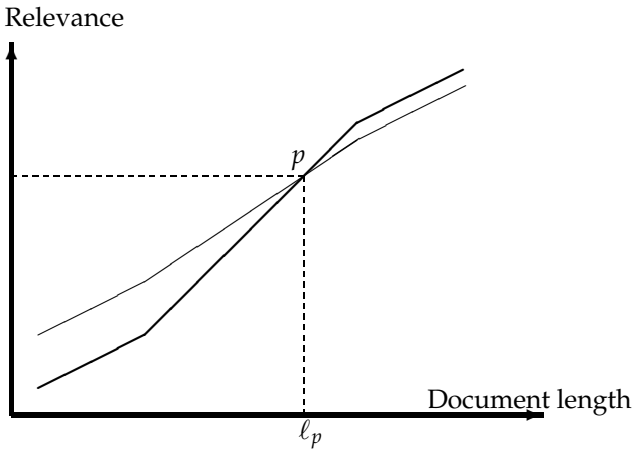
6.4.4 Pivoted normalized document length

In Section 6.3.1 we normalized each document vector by the Euclidean length of the vector, so that all document vectors turned into unit vectors. In doing so, we eliminated all information on the length of the original document; this masks some subtleties about longer documents. First, longer documents will – as a result of containing more terms – have higher tf values. Second, longer documents contain more distinct terms. These factors can conspire to raise the scores of longer documents, which (at least for some information needs) is unnatural. Longer documents can broadly be lumped into two categories: (1) *verbose* documents that essentially repeat the same content – in these, the length of the document does not alter the relative weights of different terms; (2) documents covering multiple different topics, in which the search terms probably match small segments of the document but not all of it – in this case, the relative weights of terms are quite different from a single short document that matches the query terms. Compensating for this phenomenon is a form of document length normalization that is independent of term and document frequencies. To this end, we introduce a form of normalizing the vector representations of documents in the collection, so that the resulting "normalized" documents are not necessarily of unit length. Then, when we compute the dot product score between a (unit) query vector and such a normalized document, the score is skewed to account for the effect of document length on relevance. This form of compensation for document length is known as *pivoted document length normalization*.

PIVOTED DOCUMENT LENGTH NORMALIZATION

Consider a document collection together with an ensemble of queries for that collection. Suppose that we were given, for each query q and for each document d, a Boolean judgment of whether or not d is relevant to the query q; in Chapter 8 we will see how to procure such a set of relevance judgments for a query ensemble and a document collection. Given this set of relevance judgments, we may compute a *probability of relevance* as a function of document length, averaged over all queries in the ensemble. The resulting plot may look like the curve drawn in thick lines in Figure 6.16. To compute this curve, we bucket documents by length and compute the fraction of relevant documents in each bucket, then plot this fraction against the median document length of each bucket. (Thus even though the "curve" in Figure 6.16 appears to be continuous, it is in fact a histogram of discrete buckets of document length.)

On the other hand, the curve in thin lines shows what might happen with the same documents and query ensemble if we were to use relevance as prescribed by cosine normalization Equation (6.12) – thus, cosine normalization has a tendency to distort the computed relevance vis-à-vis the true relevance, at the expense of longer documents. The thin and thick curves crossover at a point p corresponding to document length ℓ_p, which we refer to as the *pivot*



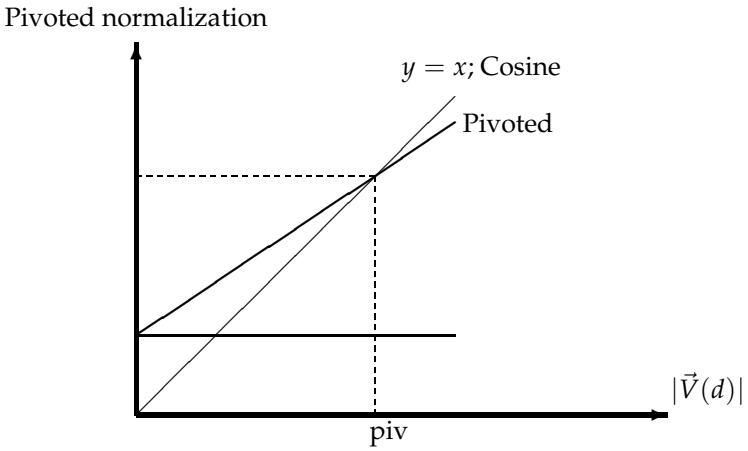
► **Figure 6.16** Pivoted document length normalization.

length; dashed lines mark this point on the $x-$ and $y-$ axes. The idea of pivoted document length normalization would then be to "rotate" the cosine normalization curve counter-clockwise about p so that it more closely matches thick line representing the relevance vs. document length curve. As mentioned at the beginning of this section, we do so by using in Equation (6.12) a normalization factor for each document vector $\vec{V}(d)$ that is not the Euclidean length of that vector, but instead one that is larger than the Euclidean length for documents of length less than ℓ_p, and smaller for longer documents.

To this end, we first note that the normalizing term for $\vec{V}(d)$ in the denominator of Equation (6.12) is its Euclidean length, denoted $|\vec{V}(d)|$. In the simplest implementation of pivoted document length normalization, we use a normalization factor in the denominator that is linear in $|\vec{V}(d)|$, but one of slope < 1 as in Figure 6.17. In this figure, the $x-$ axis represents $|\vec{V}(d)|$, while the $y-$axis represents possible normalization factors we can use. The thin line $y = x$ depicts the use of cosine normalization. Notice the following aspects of the thick line representing pivoted length normalization:

1. It is linear in the document length and has the form

$$a|\vec{V}(d)| + (1-a)\text{piv}, \tag{6.16}$$



► **Figure 6.17** Implementing pivoted document length normalization by linear scaling.

where piv is the cosine normalization value at which the two curves intersect.

2. Its slope is $a < 1$ and (3) it crosses the $y = x$ line at piv.

It has been argued that in practice, Equation (6.16) is well approximated by

$$au_d + (1 - a)\text{piv},$$

where u_d is the number of unique terms in document d.

Of course, pivoted document length normalization is not appropriate for all applications. For instance, in a collection of answers to frequently asked questions (say, at a customer service website), relevance may have little to do with document length. In other cases the dependency may be more complex than can be accounted for by a simple linear pivoted normalization. In such cases, document length can be used as a feature in the machine learning based scoring approach of Section 6.1.2.

?

Exercise 6.18

EUCLIDEAN DISTANCE

One measure of the similarity of two vectors is the *Euclidean distance* (or L_2 distance) between them:

$$|\vec{x} - \vec{y}| = \sqrt{\sum_{i=1}^{M}(x_i - y_i)^2}$$

	query					document			
word	tf	wf	df	idf	q_i = wf-idf	tf	wf	d_i = normalized wf	$q_i \cdot d_i$
digital			10,000						
video			100,000						
cameras			50,000						

► **Table 6.1** Cosine computation for Exercise 6.19.

Given a query q and documents $d_1, d_2, \ldots$, we may rank the documents d_i in order of increasing Euclidean distance from q. Show that if q and the d_i are all normalized to unit vectors, then the rank ordering produced by Euclidean distance is identical to that produced by cosine similarities.

Exercise 6.19

Compute the vector space similarity between the query "digital cameras" and the document "digital cameras and video cameras" by filling out the empty columns in Table 6.1. Assume N = 10,000,000, logarithmic term weighting (wf columns) for query and document, idf weighting for the query only and cosine normalization for the document only. Treat and as a stop word. Enter term counts in the tf columns. What is the final similarity score?

Exercise 6.20

Show that for the query affection, the relative ordering of the scores of the three documents in Figure 6.13 is the reverse of the ordering of the scores for the query jealous gossip.

Exercise 6.21

In turning a query into a unit vector in Figure 6.13, we assigned equal weights to each of the query terms. What other principled approaches are plausible?

Exercise 6.22

Consider the case of a query term that is not in the set of M indexed terms; thus our standard construction of the query vector results in $\vec{V}(q)$ not being in the vector space created from the collection. How would one adapt the vector space representation to handle this case?

Exercise 6.23

Refer to the tf and idf values for four terms and three documents in Exercise 6.10. Compute the two top scoring documents on the query best car insurance for each of the following weighing schemes: (i) `nnn.atc`; (ii) `ntc.atc`.

Exercise 6.24

Suppose that the word coyote does not occur in the collection used in Exercises 6.10 and 6.23. How would one compute `ntc.atc` scores for the query coyote insurance?

6.5 References and further reading

Chapter 7 develops the computational aspects of vector space scoring. Luhn (1957; 1958) describes some of the earliest reported applications of term weighting. His paper dwells on the importance of medium frequency terms (terms that are neither too commonplace nor too rare) and may be thought of as anticipating tf-idf and related weighting schemes. Spärck Jones (1972) builds on this intuition through detailed experiments showing the use of inverse document frequency in term weighting. A series of extensions and theoretical justifications of idf are due to Salton and Buckley (1987) Robertson and Jones (1976), Croft and Harper (1979) and Papineni (2001). Robertson maintains a web page (http://www.soi.city.ac.uk/~ser/idf.html) containing the history of idf, including soft copies of early papers that predated electronic versions of journal article. Singhal et al. (1996a) develop pivoted document length normalization. Probabilistic language models (Chapter 11) develop weighting techniques that are more nuanced than tf-idf; the reader will find this development in Section 11.4.3.

We observed that by assigning a weight for each term in a document, a document may be viewed as a vector of term weights, one for each term in the collection. The SMART information retrieval system at Cornell (Salton 1971b) due to Salton and colleagues was perhaps the first to view a document as a vector of weights. The basic computation of cosine scores as described in Section 6.3.3 is due to Zobel and Moffat (2006). The two query evaluation strategies term-at-a-time and document-at-a-time are discussed by Turtle and Flood (1995).

The SMART notation for tf-idf term weighting schemes in Figure 6.15 is presented in (Salton and Buckley 1988, Singhal et al. 1995; 1996b). Not all versions of the notation are consistent; we most closely follow (Singhal et al. 1996b). A more detailed and exhaustive notation was developed in Moffat and Zobel (1998), considering a larger palette of schemes for term and document frequency weighting. Beyond the notation, Moffat and Zobel (1998) sought to set up a space of feasible weighting functions through which hill-climbing approaches could be used to begin with weighting schemes that performed well, then make local improvements to identify the best combinations. However, they report that such hill-climbing methods failed to lead to any conclusions on the best weighting schemes.