10 Categorization and Filtering

In this chapter we consider information needs that are longstanding, recurring, or common for a large population of users. Table 10.1 shows short excerpts from 60 Wikipedia articles. Can you identify the language of each excerpt? Figure 10.1 shows an e-mail in-box with 18 messages. Can you identify the 10 spam messages?

In both of these examples the essential information need — to identify the language or to identify spam — is commonly understood and likely to arise in many circumstances. A search engine might need to identify the language of a query so as to provide a relevant response; or a searcher might wish to explicitly specify the language of documents to be retrieved (which may be different from that of the query). An e-mail spam filter must identify spam in order to prevent its delivery. A customer-support "help desk" might need to route incoming e-mail to personnel capable of reading and responding to it.

As a third and final example consider a search-oriented information need statement such as topic 383 from the TREC Filtering Track (shown in Figure 10.2), which concerns medical treatments for mental illness. A search engine, as described in the previous chapters, may be used to find historical documents in a corpus. But the user, perhaps a health-care professional, may wish to remain abreast of developments by being informed on an ongoing basis of any new articles relevant to the query. The essential information need is the same, but in the first case it is transitory, whereas in the second, it is longstanding.

Categorization is the process of labeling documents to satisfy some information need; in our examples the documents might be labeled with a tag indicating language, spam or ham (i.e., not spam), relevance or nonrelevance. *Filtering* is the process of evaluating documents on an ongoing basis according to some standing information need. Generally the outcome of filtering is to deliver the document to zero or more destinations, depending on the information need. A spam filter, for example, will either delete spam or deliver it to a junk folder but deliver ham to the user's in-box. A news filter might deliver articles on mental health to our health-care professional. *Routing* is a synonym for filtering with the connotation that documents are delivered to distinct locations, according to their category. *Selective dissemination of information* (SDI) is another synonym.

The problems of categorization and filtering are similar to those of search. In a sense they are dual formulations of the same problem: Classification and filtering find the categories to which a given document belongs; that is, the longstanding information needs that are met by the document. Search, on the other hand, finds the documents that satisfy a given information need; that is, documents belonging to a particular category, whether relevant or not.

Table 10.1 Snippets from 60 language-specific editions of Wikipedia. Each snippet is the first 50 bytes of the text of an article retrieved via the "random article" link.

18ος αιώνας | 19ος αιώνας |	8052 Novalis is a Main belt asteroid found on Sep
A távolsági jelzőlnozzanatok a látás során t	Artikulu hau ez dator formatu hitzarmenekin bat.
Auzainvilliers é uma comuna francesa na região	Brgudac je selo na sjeveru Istre , u općini Lani
Burung Pacat ekor Biru adalalı salalı satu daripada	Byrteåi er ei elv i Tokke kommune i Telemark . H
Básendar (Bátsandar) voru fyrrum kaupstaður og	Cet article est une ébauche concernant une
Ciklobutan je bezbojni gas koji pripada grupi cik	Coordinates: 53°23′40″N 14°23′30″E
Danang (Da Nang , Đà Nẵng , fransk navn Tour	De Audi S4 is een sportieve versie van de Audi A4
Den Henri Grethen (* 16. Juli 1950 zu Esch-Uelzec	Flavius Constantius (mati 2 Septeınber 421) atau
George Thomas Moore (* 23 de febrero 1871 ,	Gijang-gun là ınột hạt (gun) trực thuộc
Gold Medal är den högsta utmärkelsen från Roy	Guido (-onis) [1] est nomen ınasculinum originis G
Her Majesty je píseň britské hudební skupiny	Hirdskjald hos Harald Hårfager , kendt fra digte
Jelizaveta Petrovna. Portree autor Charles-Amédé	Komz a raer eus filistenerezh evit ober anv en un
Koordinatės : 42°04′N 19°30′E	La strashimirita és un mineral de la classe dels
Maria Theresia Opladen (* 6. April 1948 in Engels	Ministerul Integrării Europene a fost un organ d
NGC 782 je galaxia v súhvezdí Eridanus , ktorú	Ofersæwisc rind is plante and pyrt, þe hātte
Pedr IlI (21 Chwefror , 1728 – 17 Gorffennaf ,	Piec ınartenowski – dokładnie piec Siemensa-Mar
Püstəqasım , Azərbaycan ın Quba rayonunda bi	Questa voce di asteroidi è solo
Rënia heroike e dëshmorëve lsa Kastrati, Sokol	Sun Wukong (geleneksel Çince : 孫悟空 , basi
Suster-vaihe eli Susterian oli luultavasti yläml	USS Gato je ime več plovil Vojne ınornarice ZDA :
Èdè ni ilànà kan pàtàkl tó jé gbòóg	Ĥarkiva Nacia Universitato noine de Vasil Karazin
Більче-Золотецький ландша	Война между родами Токугав
За филма от 1994 вижте Гарван	Курија е село во Општина Не
Уколико сте тражили расу п	מורוני הינה העיד הגדולה בי
مبارک آباد ، روستایی است از	هذه المقالة وسمت عن طريق بو
अनुबंधालु छगु तेल	कृपया या लेखाचा/वि
यह शब्द हिंदी में क	कानूडोस मिऊनिसिपि
জয়গাঁ (ইংরেজি :Jaygaon)	१९६४ (1964) २०भी मदी दा
ஓமான் சுல்த்தானகம	หน้านี้เป็น ชื่อบ
პატარა გუფთა – სოფ	リューディガー・ゲルシュト (Rüdiger
袁武 , GBS , JP (1941年 11月 −), 又	아일랜드 국민당 (Irish Nationalist Party)

Categorization and filtering differ from search in a number of ways. Categories are typically (but not always) specified quite differently from the information need statements or queries associated with search. One would hardly expect the search methods presented in the preceding chapters to give a meaningful response to an information need statement such as "find documents written in Dutch" or "find spam". Even given an amenable information need, such as that of our health-care professional, the longstanding nature of information needs makes it worthwhile to spend more effort communicating the need to the system. One way to do so is to have an expert

Subject	Sender	Recipient
Washington Mutual Urgent Credit/Debit card update	service@wamu.com	emclaug@enron.com
Urgent Humanitarian Relief required for Tsunamis Indonesia	SEHAT CHARITIES	SKean@enron.com
worthy Christian	Khushab	joydish@bareed.alburaq.net
FW: Bin Laden Shoot em up	Dernehl, Ginger	"Shapiro, Richard"
REVISION!	Fresquez, Rick	"Adams, Jacqueline P.", "Bump...
Start Date: 10/25/01; HourAhead hour: 3; <CODESITE>	Schedule Crawler	pete.davis@enron.com
Gift shipment ready KAM.KEISER@enron.com need address	DVDPlayer	KAM.KEISER@enron.com
ERV Notification: (Enron Americas Position Report - 11/21/2001)	Wallace, Cassi	"Abel, Chris", "Allison, John", "...
Rate and Currency Counterparty Exposure as of 12/12/2001	Carrington, Clara	"Moran, Tom", "Shackleton, Sa...
The Internets #1 IRS tax solution. [omzl2]	3myjd9@msn.com	nqeb5e6@msn.com
The Internets #1 IRS tax solution. [omzl2]	3myjd9@msn.com	nqeb5e6@msn.com
The TechNews - Bulletin, April 2005	The TechNews	skean@enron.com
Start Date: 1/31/02; HourAhead hour: 13;	Schedule Crawler	pete.davis@enron.com
Pharrmacy for you	Faith Hightower	kholst
Re:	Quigley, Dutch	"Scott, Susan M."
Windows XP Pro $49.95, Office 2003 $69.95 Win XP	Lynda Hyatt	mmotley@enron.com
3 Days left to Respond	Anthony	kholst@enron.com
佮佸偣儁偱12000媼嗎侐伬偪偨偡佸佮	iouityy	keith.holst@enron.com

Figure 10.1 Ten of these 18 e-mail messages are spam. Can you identify them?

manually specify a number of rules: the vocabulary, grammar, and identifying characteristics of the various languages, spam messages, or mental health treatments. These rules are in effect complex Boolean search queries. With sufficient effort it is possible to define rules to distinguish languages, or spam from ham, or reports on treatments for depression from other articles. But, as just as they are for search, automatic methods for categorization and filtering are more efficient and typically more effective. The automatic approaches we shall consider are *machine learning* methods known as *classifiers.* A classifier learns from examples how to distinguish documents belonging to different categories.

In a sense any search method is a classifier because it aims to distinguish relevant from non-relevant documents. In the same way that a search engine may return either a set of documents or a ranked list, a classifier may be *hard* or *soft.* A hard classifier returns a Boolean result indicating whether or not a document belongs to a category, whereas a soft classifier returns a confidence score. Soft classification is often called *ranking*, and the study of soft classifiers is called *learning to rank.*

Although it is not traditionally considered machine learning, the BM25 ranking method from Chapter 8 meets our definition of a soft classifier, and may be applied with reasonable effectiveness to the filtering and categorization problems introduced in this chapter. In Section 10.1 we develop the example problems outlined previously, showing the role of classification and ranking in their solution. To illustrate this role we use BM25 because it is familiar and yields results that are good enough to serve as baselines for comparison with other methods presented in this chapter and the next. Along the way we develop evaluation measures specific to this purpose.

```
<top>
<num> Number: 383
<title> mental illness drugs
<desc> Description:
Identify drugs used in the treatment of mental illness.
<narr> Narrative:
A relevant document will include the name of a specific
or generic type of drug. Generalities are not relevant.
</top>
```

Figure 10.2 TREC 7 Adhoc and TREC 8 Filtering Topic 383.

In Section 10.2 we treat classification more formally, and in Sections 10.3 through 10.7 we describe a wide variety of learning methods that are useful for categorization and filtering. In Section 10.8, we compare the results from these methods with the BM25 baselines of Section 10.1.

10.1 Detailed Examples

Through a series of graduated examples we illustrate typical filtering and categorization problems, solutions, and evaluation methods. The problems are variants of those posed above: topic-oriented filtering, language categorization, and spam filtering. The solutions illustrated in this section employ the BM25 ranking and relevance feedback methods from Chapter 8. Although BM25 is not typically used for filtering and classification, it provides a convenient starting point for understanding the area as well as a solid baseline for comparison. Evaluation measures are developed in conjunction with the problems and solutions to which they apply. We begin with an example that is similar to the ranked retrieval task for which BM25 was designed, then progress through several variants that highlight differences in the task, approach, and evaluation measures.

10.1.1 Topic-Oriented Batch Filtering

Consider the needs of our health-care professional. The topic format is suitable as input to a search engine, but at the outset of a filtering task the documents to be searched do not yet exist! So there is nothing to do but wait for documents to arrive. If we can afford to wait for a large number of documents to arrive, we can accumulate them into a corpus, index them, and apply a search method such as BM25 to yield a ranked list of the most likely relevant documents, which is then presented to the user. We can then wait for more documents to arrive and repeat the process for each new batch of documents in turn. This approach is known as *batch filtering*. From the user's perspective batch filtering yields a viable solution if enough

Table 10.2 Topic-oriented batch filtering results. Undefined results are represented by a dash.

		First Batch			Second Batch			Avg. (all batches)		
# Batches	n	$\|Rel\|$	AP	P@10	$\|Rel\|$	AP	P@10	$\|Rel\|$	AP	P@10
1000	141	1	1.00	0.1	0	–	0.0	0.07	–	0.01
500	281	1	0.50	0.1	0	–	0.0	0.13	–	0.01
100	1407	3	0.33	0.2	0	–	0.0	0.67	–	0.04
50	2814	3	0.15	0.1	1	0.33	0.1	1.34	–	0.05
10	14070	6	0.06	0.0	8	0.17	0.1	6.70	0.10	0.07
5	28140	14	0.08	0.1	14	0.09	0.1	13.00	0.07	0.10
1	140651	67	0.05	0.1				67.00	0.05	0.10

documents arrive within a suitable time interval. Our health-care professional may be satisfied to see new articles once a week or once a month. Indeed, this may be the preferred mode of delivery. In other circumstances, such as e-mail filtering, the delay associated with batching is probably not acceptable.

To illustrate the batch filtering approach, we assume that topic 383 represents the user's information need, and process a chronological subsequence of the TREC45 corpus using BM25. In particular:

- We use the title field as a query: ⟨ "mental", "illness", "drugs" ⟩.
- Guided by our experience with multilingual search and spam filtering, we use byte 4-grams as tokens (see Section 3.3). For topic 383 stemmed and unstemmed words yield similar results, which we do not report.
- We use the BM25 parameter settings $k_1 = 1.1$ and $b = 0$, effectively ignoring document length. We retain these settings for the subsequent experiments reported in this chapter.
- For this experiment we do not use relevance feedback or pseudo-relevance feedback (Section 8.6).
- We use the *Financial Times* (FT) documents, dated 1993 and later from the TREC45 collection. These documents are chronologically ordered over a period of about two years. There are $N = 140{,}651$ documents, of which 67 are relevant to topic 383. This sequence of documents was used as the evaluation set for the *routing* and *batch filtering* tasks of the TREC 8 Filtering Track.
- To simulate batch filtering with various batch sizes, we group the documents into equal-sized batches of consecutive documents ranging from 141 documents (1000 batches; each the better part of a day's messages) to 140,651 documents (one batch; the full two years' messages). Each batch is indexed as a separate and independent document collection. The batches are presented to the BM25 filter, one batch at a time, in chronological order.

The problem of evaluating filter effectiveness is somewhat different from that of evaluating ranked retrieval. We introduce notions as necessary to evaluate our examples and relate them to the more general evaluation framework.

For the single batch of 140,651 documents, batch filtering is equivalent to ranked retrieval. We could use any of the measures introduced in Section 2.3 and expanded upon in Chapter 12. On the single batch BM25 yields P@10=0.1, P@1000=0.04, and AP=0.053.

When more than one batch is considered, an obvious but unsatisfactory approach is to compute the evaluation measure separately for each batch, then average these results to yield a summary measure. This approach is unsatisfactory for the following reasons:

- When we divide the documents into more batches, the number of documents per batch (n) and the number of relevant documents per batch ($|Rel|$) decrease. When the number of batches is large, many batches will have no relevant documents (i.e., $|Rel| = 0$). Average precision (AP) is undefined if any batch has $|Rel| = 0$. In general any measure that depends on $|Rel|$ will not work for small batches.
- Measures such as precision at k (P@k) are heavily influenced by n, thus rendering it impossible to compare results using different batch sizes.

Table 10.2 illustrates these issues. For batch sizes varying between $n = 141$ (1000 batches) and n =28,140 (5 batches), we show AP and P@10 for the first two batches and the average over all $\lceil N/n \rceil$ batches. For $n = 1$ there is only one batch.

AP is undefined for most batch sizes because many batches contain no relevant documents. Both P@10 and AP (where it is defined) vary with batch size and convey little information about the effectiveness of the method for its intended purpose.

P@k is not a suitable measure for comparing results using different batch sizes because for any given k the number of retrieved documents is proportional to the number of batches. Consider the two extreme cases of 1000 batches and a single batch. If 1000 batches are used, P@10 is the precision achieved when the user is presented 10,000 documents in total, 10 per batch. If a single batch is used, P@10 is the precision achieved when she is presented only 10 documents. A more reasonable approach is to hold the total number of documents presented to the user constant over all batches. This is achieved by evaluating P@$\lfloor \rho n \rfloor$, where $\frac{1}{n} \leq \rho \ll 1$; that is, we assume that the user is presented the top-ranked fraction ρ of each batch of documents and hence this same fraction ρ of all documents in the corpus. For our purposes we choose $\rho = \frac{k}{N}$ where N is the corpus size and k is the total number of documents presented to the user. So regardless of the batch size n, the average of P@$\lfloor \frac{kn}{N} \rfloor$ over all batches is comparable with P@k for the single batch. Table 10.3 shows P@$\lfloor \rho n \rfloor$, which corresponds to k between 10 and 2000. The numbers in each column pertain to the same overall number of retrieved documents. From this table we see that for this example, the average is largely unaffected by batch size so long as the batch size is large enough that $\lfloor \rho n \rfloor > 0$.

The measure P@$\lfloor \rho n \rfloor$ assumes that an equal proportion of documents from each batch is presented to the user. This assumption may be unrealistic because some batches will contain

Table 10.3 Size-specific P@k results for topic-oriented batch filtering.

# Batches	n	P@$\lfloor \rho n \rfloor$					
		$\rho = \frac{10}{N}$	$\rho = \frac{20}{N}$	$\rho = \frac{100}{N}$	$\rho = \frac{200}{N}$	$\rho = \frac{1000}{N}$	$\rho = \frac{2000}{N}$
1000	141	–	–	–	–	0.03	0.02
500	281	–	–	–	–	0.04	0.01
100	1407	–	–	0.05	0.04	0.04	0.02
50	2814	–	–	0.07	0.05	0.04	0.01
10	14070	0.1	0.10	0.07	0.06	0.04	0.03
5	28140	0.2	0.10	0.06	0.06	0.04	0.01
1	140651	0.1	0.15	0.07	0.06	0.04	0.03

Table 10.4 Aggregate results for topic-oriented batch filtering.

# Batches	n	P@10	P@20	P@100	P@200	P@1000	P@2000	AP
1000	141	0.2	0.10	0.07	0.08	0.04	0.02	0.058
500	281	0.1	0.15	0.08	0.08	0.04	0.03	0.056
100	1407	0.1	0.15	0.06	0.07	0.04	0.03	0.053
50	2814	0.1	0.15	0.07	0.06	0.04	0.03	0.053
10	14070	0.1	0.15	0.07	0.06	0.04	0.03	0.053
5	28140	0.1	0.15	0.07	0.06	0.04	0.03	0.053
1	140651	0.1	0.15	0.07	0.06	0.04	0.03	0.053

more relevant documents than others. This situation may arise entirely due to chance, or because the documents are chronologically ordered and their makeup will naturally be be influenced by current events. A filtering system that reflects this variable proportion may better meet the user's information needs. Suppose that the user is prepared to examine documents at an average rate of ρ; that is, over time to examine ρN of the N documents processed by the filter. Precision over these ρN documents is not optimized by presenting exactly ρn documents per batch to the user. Precision may be improved if the filter presents to the user more documents from the batches that it deems to contain a larger number of relevant documents.

Because BM25 orders documents ranked by a relevance score s, we may select, rather than the top-ranked $\lfloor \rho n \rfloor$ documents from every batch, those documents with $s > t$ for some fixed threshold t. Overall, a larger value of t will result in fewer documents being returned with higher precision, and vice versa. Therefore, for any particular k there exists a value of t such that about $k = \rho N$ documents have a score $s > t$. For the moment we assume that it is possible to determine in advance the appropriate value of t for any k. Under this assumption we coin the term *aggregate* P@k to characterize the effectiveness of the system under the choice of t that yields k documents in total (see Table 10.4). *Aggregate* AP, also shown, is derived from

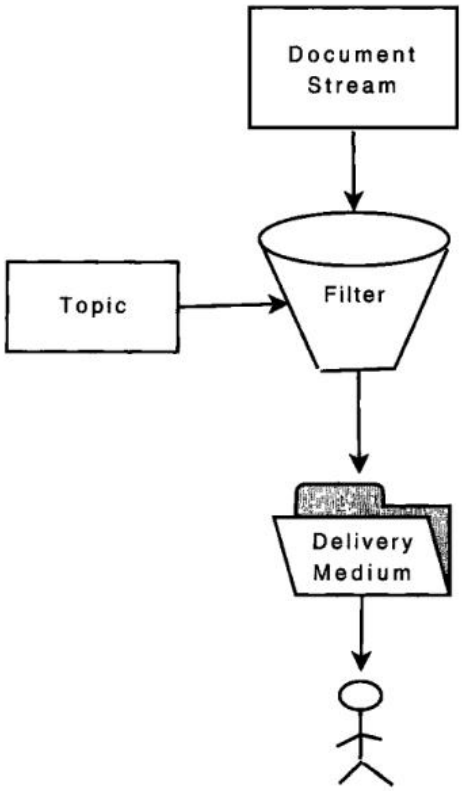

Figure 10.3 Topic-oriented on-line filtering.

aggregate P@k in the normal way. As we would expect, the numbers in each column are nearly identical. The variations are due to slight differences in IDF values from batch to batch.

Aggregate P@k and aggregate AP aptly model the real-world scenario in which the filter may be tuned to a particular user's requirement by setting t as described above. They also model the real-world scenario in which the results of each batch are entered successively into a common priority queue that the user may peruse at will. Aggregate P@k is the precision of the top k elements of the queue once all batches are processed, whereas aggregate AP measures the effectiveness of the final ranking.

10.1.2 On-Line Filtering

On-line filtering is analogous to batch filtering with batch size $n = 1$. As depicted in Figure 10.3, an on-line filter acts immediately on each message in sequence rather than on batches of messages. Messages deemed likely to be relevant are delivered to the user; those deemed unlikely to be relevant are discarded. The delivery medium may be a text message system or an e-mail system, or a result archive that the user consults when convenient. The possible media form a continuum largely characterized by immediacy, storage capacity, and access capability.

Table 10.5 Aggregate results for topic-oriented batch versus on-line filtering. The first row is repeated from Table 10.4.

Method	P@10	P@20	P@100	P@200	P@1000	P@2000	AP
Single batch	0.1	0.15	0.07	0.06	0.04	0.03	0.053
On-line	0.2	0.15	0.05	0.05	0.03	0.02	0.041

Like batch filtering, on-line filtering requires a trade-off between precision and recall. In general it is possible to increase recall at the expense of precision by delivering more messages to the user. Nowadays even the most immediate media (e.g., text messaging) have large storage capacity and sophisticated access capabilities, so the user's capacity to deal with the messages is more likely to be a constraint than that of the medium. To that end it is advantageous, where possible, to *prioritize* messages delivered to the medium, which in effect acts as a *priority queue*. As with ranked retrieval, the user tacitly determines the trade-off by examining some number of the highest-priority documents.

In the event that prioritization is impossible, it is necessary to determine a threshold t such that an appropriate number of documents is delivered. We may set t so that the rate of delivery is ρ (as described in the previous section) or to optimize some function (e.g., F_β; page 68) that captures the trade-off between precision and recall.

The relevance score of a ranked retrieval method like BM25 is a suitable indicator of priority, provided the score can be computed on a per-document basis. In this regard BM25 is not entirely amenable, because it depends on the term count N_t, whose computation assumes a large collection of documents. That said, we may apply BM25 to the near-vacuous collection consisting of only the document to be filtered, and come up with aggregate results that are not dramatically inferior to those for batch filtering, as shown in Table 10.5. In any event both batch and on-line filtering are improved substantially by the use of historical examples, as detailed in the following section.

10.1.3 Learning from Historical Examples

Generally, when a filter is deployed, we have available historical examples of documents from the same source or a similar source. Recall that the filter in our running example is to be deployed on *Financial Times* documents dated 1993 or later. The TREC45 collection also contains 69,508 *Financial Times* documents dated 1992 or earlier that can serve as historical examples.

We know from previous TREC evaluation efforts that 22 of these historical documents are relevant to topic 383. But this information would not necessarily be known at the time of filter deployment. We first consider using the historical documents without this relevance information, and then how this information, if available, may be harnessed to dramatically improve effectiveness.

Table 10.6 The effect of historical collection statistics on aggregate results for topic-oriented on-line filtering. The first two rows are repeated from Table 10.5. The third row represents a filter that uses historical collection statistics to compute IDF values. The last row shows the effect of augmenting the query with 20 terms extracted from labeled training examples through BM25 relevance feedback prior to filter deployment.

Method	P@10	P@20	P@100	P@200	P@1000	P@2000	AP
Single batch	0.1	0.15	0.07	0.06	0.04	0.03	0.053
On-line no history	0.2	0.15	0.05	0.05	0.03	0.02	0.041
On-line historical IDF	0.1	0.15	0.07	0.06	0.04	0.03	0.054
On-line historical IDF + training	1.0	0.95	0.39	0.24	0.06	0.03	0.555

Historical collection statistics

Perhaps the simplest way to use historical documents is to increase the effective batch size for batch filtering. For any given batch the historical documents are combined with those of the batch and the entire collection is ranked. Then the historical documents are removed from the ranked list, thus yielding a rank and a score for each document in the batch to be filtered. The principal effect of this approach is to yield better collection statistics (IDF in particular) and hence a more accurate relevance score. A secondary effect is that rather than returning a fixed number k of documents per batch, we can return only those that appear among the top $k' > k$ in the combined ranked list. If the batch contains more high scoring documents, more documents will be returned. For a suitable choice of k' the system will return the same number of documents overall while achieving better precision.

The use of historical documents effectively transforms the on-line filtering problem into a batch problem in which the only work to be done is to compute the score of each filtered document and, perhaps, its relative rank within the historical collection. For this purpose the index structures detailed in previous chapters are overkill. All we need to compute BM25 and similar scores is a dictionary indicating the document count N_t for each term and, if a rank score is desired, an ordered list of the historical documents' scores. To compute the relevance score we simply fetch the relevant N_t values and apply the formula; to compute rank we apply a binary search to the list of historical scores.

The third row in Table 10.6 shows the effect of using historical collection statistics on the aggregate effectiveness of our on-line filter. The aggregate result is nearly identical to that achieved by our batch filter.

Historical training examples

Because the information need associated with a filter is ongoing, it is likely that examples of relevant documents are already known or can be found without too much effort. These examples are known as *training examples* or *labeled data*. In machine learning using labeled data to build a classifier is known as *supervised learning*. In this book we have already described one simple method of supervised learning: the BM25 relevance feedback approach (Section 8.6).

As we mentioned previously, there are 22 relevant documents in our historical *Financial Times* data. We used BM25 relevance feedback to select the best $m = 20$ terms from these documents, which were used to augment the 3 terms from the topic. Using these terms, the BM25 scoring formula was applied to each document in turn as an on-line filter.

The aggregate results of this method of supervised learning are shown in the bottom row of Table 10.6. The first and last columns of results are not misprints: For this specific task the use of historical training examples yields roughly a tenfold improvement in both P@10 and AP.

10.1.4 Language Categorization

From the 60 language-specific editions of Wikipedia in Table 10.7, we fetched 8012 articles using the "random article" link and extracted from each the first 50 bytes of the main text (shown in Table 10.1 on page 311). We consider the dual problems of categorization and filtering:

- *Categorization*: Given a document d and a set of possible categories, identify the category (or categories) to which d belongs. Categorization may be reduced to *category ranking*, which ranks the categories according to how likely they are to contain the document.
- *Filtering*: Given a particular category c, identify documents that belong to c. Filtering may be reduced to *document ranking*, which ranks documents according to how likely they are to belong to the category.

For the purposes of our example the documents are snippets from Wikipedia articles and the categories are languages. By definition each document in our example belongs to exactly one category, determined by the Wikipedia edition from which it was extracted. In other situations it may make sense to consider a document as belonging to several categories; for example, a document might contain several languages or a language might have several overlapping dialects. When working with full documents, language identification is relatively straightforward (McNamee, 2005), but we deliberately limit ourselves to short snippets for the purposes of this example.

Our language categorization problem may arise when we wish to identify the language of a message for routing purposes or to organize documents into language-specific collections. Neither of these problems is readily expressed as a query that can be processed by a search engine. Instead, we rely exclusively on training examples to communicate the information need. To this end we arbitrarily consider 4000 of the snippets to be historical, and the remaining 4012 snippets as the ones to be categorized or filtered. In machine-learning terms, the historical examples form a *training set* and the others form a *test set*. The overall objective is to categorize or filter a set of examples much larger than the test set, which should be viewed as a *sample* for the purpose of estimating classifier effectiveness on the larger set.

Filtering and document ranking can be done with the methods we have introduced for topic-oriented filtering with historical examples. We consider each language to be a separate topic and apply BM25 ranking with relevance feedback (Section 8.6). For this example we treat each

Table 10.7 Tags identifying 60 language-specific Wikipedia editions.

Tag	Language	Tag	Language	Tag	Language
ang	Anglo-Saxon	fi	Finnish	nn	Nynorsk
ar	Arabic	fr	French	no	Bokmål
az	Azeri	he	Hebrew	pa	Punjabi
bg	Bulgarian	hi	Hindi	pl	Polish
bn	Bengali	hr	Croatian	pt	Portuguese
bpy	Bishnupriya	hu	Hungarian	ro	Romanian
br	Breton	id	Indonesian	ru	Russian
bs	Bosnian	is	Icelandic	simple	Simple English
ca	Catalan	it	Italian	sk	Slovak
cs	Czech	ja	Japanese	sl	Slovenian
cy	Welsh	ka	Georgian	sq	Albanian
da	Danish	ko	Korean	sr	Serbian
de	German	la	Latin	sv	Swedish
el	Greek	lb	Luxembourgish	ta	Tamil
en	English	lt	Lithuanian	th	Thai
eo	Esperanto	mk	Macedonian	tr	Turkish
es	Spanish	mr	Marathi	uk	Ukrainian
et	Estonian	ms	Malay	vi	Vietnamese
eu	Basque	new	Nepal Bhasa	yo	Yoruba
fa	Persian	nl	Dutch	zh	Chinese

of the training snippets in a topic language as a "relevant document". For a given language we create a query from the training snippets by treating each byte 4-gram as a query term.

Table 10.8 shows the per-language AP results that are achieved by applying BM25 to each of the 60 filtering problems, as well as the overall MAP. By usual search standards MAP=0.78 would indicate extremely high effectiveness, but it tells us little about how effectively (or indeed how) BM25 addresses the categorization problem.

To address the categorization problem we must rank languages instead of documents, and we must compute this ranking for every document. To do so we use the relevance scores from the document ranking. Given a document d and a language l, let $s(d, l)$ be the relevance score for d in the document ranking for language l. Now consider two languages l_1 and l_2. We assume that $s(d, l_1) > s(d, l_2)$ indicates that d is more likely to be written in language l_1 than in l_2. Under this assumption $s(d, l)$ is also the relevance score for l in the category ranking for document d.

It is infeasible to show all 4012 category rankings. Table 10.9 shows one example: the complete ranking of languages for a snippet drawn from the German Wikipedia. We see that the correct category ($l_1 =$ de) is ranked first and other categories that are apparently similar — the Germanic languages — are ranked ahead of those that are dissimilar, such as Asian languages.

Table 10.8 AP results for 60 language-specific filtering problems, using BM25 with terms selected from 4000 training examples.

Tag	AP	Tag	AP	Tag	AP
ang	0.91	ar	0.99	az	0.53
bg	0.67	bn	0.84	bpy	0.81
br	0.94	bs	0.48	ca	0.70
cs	0.67	cy	0.95	da	0.40
de	0.76	el	1.00	en	0.59
eo	0.72	es	0.67	et	0.45
eu	0.82	fa	0.95	fi	0.85
fr	0.79	he	1.00	hi	0.72
hr	0.47	hu	0.75	id	0.67
is	0.93	it	0.79	ja	0.94
ka	0.99	ko	1.00	la	0.79
lb	0.97	lt	0.82	mk	0.81
mr	0.86	ms	0.68	new	0.91
nl	0.84	nn	0.63	no	0.45
pa	0.96	pl	0.83	pt	0.83
ro	0.92	ru	0.75	simple	0.54
sk	0.75	sl	0.59	sq	0.60
sr	0.68	sv	0.76	ta	0.97
th	1.00	tr	0.82	uk	0.84
vi	0.96	yo	0.63	zh	0.87
			MAP: 0.78		

Strictly applying our problem definition, there is exactly one relevant result: the correct category. All others are nonrelevant.

Table 10.10 lists overall summary measures for the 4012 language rankings. In this table P@1 is more commonly called *accuracy*: the proportion of documents for which the correct category was ranked first. MAP is equal to the measure known as *mean reciprocal rank* (MRR) because there is exactly one relevant result per ranking (see page 409). For a single topic, reciprocal rank is defined as $\text{RR} = \frac{1}{r}$, where r is the rank of the first (and only) relevant result. Because $\text{AP} = \text{P@}r = \frac{1}{r}$, in this case RR = AP and MRR = MAP.

The summaries shown in the table are expressed as *micro-averages* and as *macro-averages*, defined as follows:

- *Micro-average*: A summary measure over all documents, without regard to category.
- *Macro-average*: The average of summary measures computed for each category.

Table 10.9 Language ranking for the German snippet shown at the top of the table (correct tag: de). $s(d, l)$ denotes the BM25 score for the given language.

Werner Haase (* 2. August 1900 in Köthen (Anhalt					
Tag	$s(d, l)$	**Tag**	$s(d, l)$	**Tag**	$s(d, l)$
de	58.2	lb	35.2	da	33.6
no	29.1	en	28.4	tr	19.4
sk	18.5	hu	17.5	bs	16.4
la	14.7	cs	14.5	et	13.2
fi	13.0	es	12.7	cy	12.0
is	11.8	sl	10.6	simple	10.4
nl	9.3	pl	7.3	yo	7.1
sv	6.0	sq	5.9	ru	5.0
ro	4.6	ang	4.5	lt	3.9
el	3.4	eo	3.3	az	2.4
sr	2.4	nn	2.4	ms	1.7
fr	1.4	zh	0.0	vi	0.0
uk	0.0	th	0.0	ta	0.0
pt	0.0	pa	0.0	new	0.0
mr	0.0	mk	0.0	ko	0.0
ka	0.0	ja	0.0	it	0.0
id	0.0	hr	0.0	hi	0.0
he	0.0	fa	0.0	eu	0.0
ca	0.0	br	0.0	bpy	0.0
bn	0.0	bg	0.0	ar	0.0
AP (RR): 1.00					

For this example the difference between micro-average and macro-average is small because there are roughly the same number of examples of each language in the test. The difference may be profound when the number of examples differs or when the effectiveness of the ranking varies from category to category. To illustrate the difference, we extrapolate the summary measures from our test set to the entire Wikipedia.

Table 10.11 shows, for each Wikipedia edition, the fraction of all articles in the edition and the MRR summary for those articles. The editions vary substantially in size, and the largest edition (English) contains vastly more articles than any other but achieves a mediocre MRR score of 0.71. The net result is that for the test set, micro-averaged MRR is lower than macro-averaged MRR (0.82 versus 0.86). In other applications, such as spam filtering, the difference is even more important. Overall, a macro-average is independent of the proportion of documents

Table 10.10 Summary measures for language ranking over 4012 snippets. P@1 is equivalent to accuracy: the total fraction of correctly classified snippets. MAP is equivalent to MRR, since the problem statement defines exactly one "relevant" language for each snippet.

	Accuracy (P@1)	Error (1-accuracy)	MRR (MAP)
Micro-average	0.79	0.21	0.860
Macro-average	0.79	0.21	0.857

Table 10.11 Language categorization results extrapolated over the contents of 60 Wikipedia editions. The columns labeled "%" contain the fraction of all Wikipedia articles that belong to the given edition. Some editions (e.g., English) contain far more articles than others (e.g., Icelandic).

Tag	%	MRR	Tag	%	MRR	Tag	%	MRR
ang	0.01	0.90	ar	0.81	0.99	az	0.19	0.91
bg	0.60	0.87	bn	0.16	0.89	bpy	0.20	0.84
br	0.21	0.89	bs	0.22	0.69	ca	1.45	0.80
cs	1.05	0.77	cy	0.19	0.95	da	0.90	0.65
de	7.52	0.88	el	0.35	1.00	en	23.95	0.71
eo	0.95	0.80	es	3.91	0.78	et	0.52	0.73
eu	0.32	0.87	fa	0.50	0.97	fi	1.69	0.88
fr	6.66	0.88	he	0.77	1.00	hi	0.25	0.93
hr	0.49	0.58	hu	1.04	0.87	id	0.86	0.68
is	0.21	0.94	it	4.71	0.81	ja	4.88	0.94
ka	0.25	0.99	ko	0.80	1.00	la	0.23	0.80
lb	0.22	0.99	lt	0.72	0.91	mk	0.24	0.88
mr	0.19	0.81	ms	0.32	0.83	new	0.42	0.85
nl	4.47	0.89	nn	0.40	0.71	no	1.81	0.57
pa	0.01	0.96	pl	5.03	0.89	pt	3.98	0.90
ro	1.04	0.93	ru	3.20	0.81	simple	0.49	0.79
sk	0.90	0.79	sl	0.63	0.78	sq	0.19	0.88
sr	0.62	0.70	sv	2.63	0.85	ta	0.15	0.97
th	0.38	1.00	tr	1.07	0.85	uk	1.21	0.88
vi	0.68	0.96	yo	0.05	0.93	zh	2.10	0.91
MRR: 0.82/0.86 (micro-average/macro-average)								

in each category, whereas a micro-average is weighted by that proportion. To extrapolate micro-average from one collection to another, it is necessary to know both the proportions and the class-specific summary scores. The distinction between micro-averaging and macro-averaging is specific to categorization: MAP and other summary measures, when applied to search results, are invariably macro-averages.

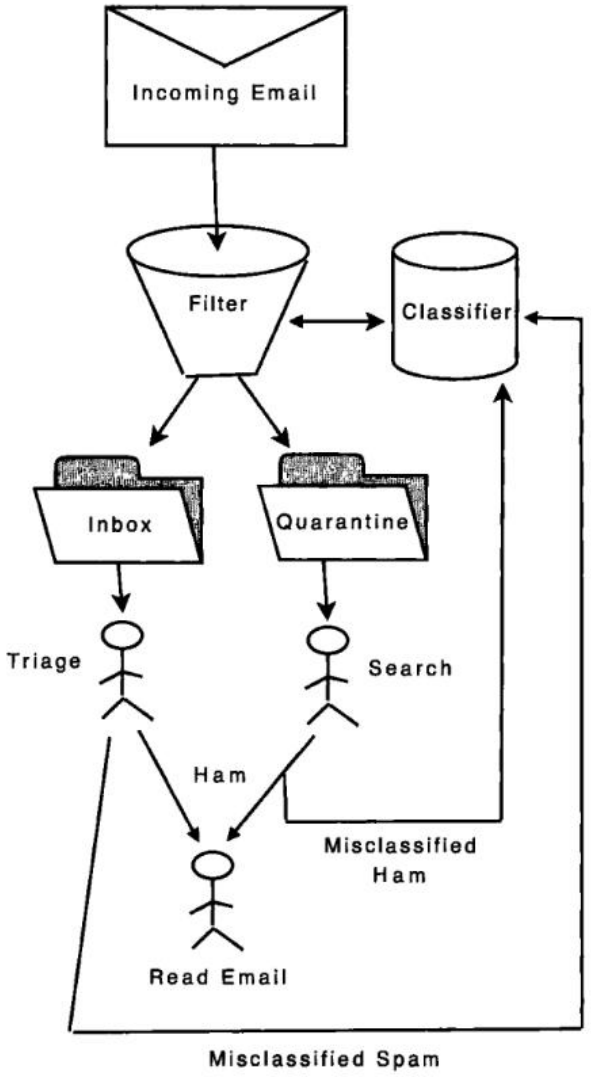

Figure 10.4 On-line spam filter deployment.

10.1.5 On-Line Adaptive Spam Filtering

Figure 10.4 shows the context of the spam filtering problem that we address in our third and final example. The filter receives a stream of e-mail messages and must deliver each, in turn, to the user's in-box or to a quarantine file. The messages in the in-box are read by the user in chronological order, whereas the quarantine file is occasionally searched to find ham (i.e., non-spam) messages that may have been placed there incorrectly by the filter. When, through either of these processes, the user notices spam in the in-box or ham in the quarantine, feedback is given to the filter. For this application it is useful to consider the in-box to be a simple queue and the quarantine a priority queue (that is, a dynamically updated ranked list).

Table 10.12 Command-line interface for the TREC Spam Filter Evaluation Toolkit. For evaluation within the toolkits framework a filter must implement these five operations, which are invoked in sequence by the toolkit.

Command	Filter Action
`initialize`	Create a filter to process a new stream of messages.
`classify` *file*	Return a category and priority score for the message in *file*.
`train spam` *file*	Use *file* as a historical example of spam.
`train ham` *file*	Use *file* as a historical example of ham.
`finalize`	Shut down the filter.

We use the TREC 2005 Public Spam Corpus[1], a sequence of 92,180 messages, of which 39,399 are spam and 78,798 are ham. Rather than splitting the sequence into training and test sets, we employ *on-line feedback* in which an *ideal* user is assumed to detect and report all errors. Under this assumption every message may be used as a historical training example immediately after it is delivered: Those reported as errors are assumed to belong to the category indicated by the user, and those not reported as errors are assumed to have been correctly categorized by the filter. Although the assumption of an ideal user is unrealistic, evaluation results based on it serve as a well-defined basis for comparison.

The on-line adaptive filtering scenario is implemented by the TREC Spam Filter Evaluation Toolkit, a test harness that, along with test corpora and sample filters, may be downloaded from the Web.[2] Although the toolkit uses categories that happen to be named `spam` and `ham`, it works equally well for any *on-line binary categorization* task. All of the examples presented in this chapter were prepared using the toolkit. For evaluation using the toolkit the filter must implement the five command-line operations detailed in Table 10.12.

As an illustrative example we again use BM25, applying it to build an on-line adaptive spam filter. Again we emphasize that this is not a typical application of BM25, but it does provide a solid baseline for the techniques we discuss later in the chapter. Spam filtering is an example of *binary categorization*, for which there are exactly two exclusive categories: each document, by definition, belongs to one category or the other. In contrast to our previous examples, in this one there are neither topic statements nor historical examples. The training examples are derived exclusively from on-line feedback. At the outset the filter has no information on which to base its decision, but as more messages are processed, it learns the characteristics of messages that distinguish the categories. The rate and manner in which the filter improves with training is known as the *learning curve*.

Our baseline approach is the same as for language categorization. We apply BM25 with relevance feedback separately to each of the document ranking problems: identifying spam

[1] `trec.nist.gov/data`

[2] `plg.uwaterloo.ca/~gvcormac/jig`

Table 10.13 Effectiveness measures for the document ranking problem associated with spam filtering. The three rows show the effectiveness of document ranking using separate spam and ham relevance scores, and their difference.

Relevance Score	P@10	P@20	P@100	P@200	P@1000	P@2000	AP
$s(d, \texttt{spam})$	1	1	1	1	1	1	0.9918
$s(d, \texttt{ham})$	1	1	1	1	1	1	0.9906
$s(d, \texttt{spam}) - s(d, \texttt{ham})$	1	1	1	1	1	1	0.9927

Table 10.14 IR-oriented effectiveness measures for the primary categorization problem in an on-line spam filter.

Class	Accuracy (P@1)	Error (%)	Precision	Recall	F_1
Spam	0.99989	0.011	0.80308	0.99989	0.89074
Ham	0.67149	32.860	0.99977	0.67149	0.80339
Micro-average	0.85954	14.055	0.85945	0.85954	0.85954
Macro-average	0.83569	16.431	–	–	0.84707
Logit average	0.99260	0.740	–	–	0.85233

and identifying ham, yielding two scores for each document d: $s(d, \texttt{spam})$ and $s(d, \texttt{ham})$. The category ranking for d is determined by the difference $s(d) = s(d, \texttt{spam}) - s(d, \texttt{ham})$. If $s(d) > 0$, the message is deemed to be spam; otherwise it is ham.

As before, query terms are byte 4-grams taken from messages labeled as spam or ham. As the filtering progresses, we create and maintain separate queries for ham and spam. As feedback is provided, new terms are added to the appropriate queries and IDF values are updated. Initially both queries are empty, yielding scores of 0, but they quickly grow as messages are seen. In contrast to the previous experiments we do not restrict the number of feedback terms added to each query, but allow both queries to grow without limits.

The score $s(d)$ also serves as a priority for the purpose of on-line ranking. Evaluation measures for the the document ranking problem, shown in Table 10.13, fail to shed much light on the intended purpose of our method: spam filtering. Broadly speaking, this is because the purpose of search is to identify documents of interest and effectiveness measures such as P@k and AP reward systems that find the documents that are easiest to categorize. Such measures tell us little about how well the filter works for difficult-to-categorize documents, the crux of spam filter effectiveness.

Evaluation measures for the primary categorization problem in an online spam filter are shown in Table 10.14. Because there are only two categories, reciprocal rank is redundant and not reported. For computing the filter's F_1 score we assumed that spam constitutes the relevant class. This is a common assumption in the literature. Whenever you encounter recall or precision numbers for which it is not obvious whether they refer to ham or spam, you should assume that the author treats spam as the relevant class.

As you see from Table 10.14, the accuracy (P@1) for spam messages is so high that it is difficult to interpret due to the large number of leading nines. Error (1-P@1) expressed as a percentage is more convenient and intuitive. In assessing filter effectiveness it is quite easy to observe that an error rate of, say, 0.1% is ten times better than an error rate of 1%. The observation that an accuracy of 0.999 is ten times better than an accuracy of 0.990 requires mental arithmetic and counting nines. Returning to our example, the spam error rate is 0.01%, meaning that about 1 in 10,000 spam messages is delivered to the user's in-box. On the other hand, the ham error rate is 32.9%, meaning that about a third of the user's good messages are delivered to quarantine. The user is unlikely to find this behavior acceptable; but if the situation were reversed, she might. If the ham error rate were 0.01% and the spam error rate were 32.9%, the filter would deliver the overwhelming majority of ham to the in-box while eliminating two-thirds of the spam.

Summary statistics, whether micro-averages or macro-averages, fail to consider the category-specific consequences of errors. In addition, micro-averages are highly sensitive to the overall proportion of spam. Suppose, for example, that the amount of spam were to double while the amount of ham — and the filter's spam and ham error rates – remained constant. The micro-averaged error rate would fall from 14% to 8.9%. Did the filter improve? Of course not. Micro-averaging measures the *prevalence* of spam as much as any property of the filter.

The logit average (LAM) is a macro-average under the logit transformation (see Equation 10.38):

$$lam(x, y) = \text{logit}^{-1}\left(\frac{\text{logit}(x) + \text{logit}(y)}{2}\right). \tag{10.1}$$

Intuitively, this average interprets the scores as probabilities and combines them according to the weight of evidence they afford. Under the logit average the difference between scores of 0.1 and 0.01 has the same impact as the difference between scores of 0.9 and 0.99. In a sense each of these pairs represents an order of magnitude difference in effectiveness. On the other hand, the difference between 0.5 and 0.51 has little impact. Cormack (2008) gives a more formal derivation of LAM based on *odds ratios* (Section 10.2.1).

Precision and recall, along with other summary measures based on them, are difficult to interpret for filtering tasks and also are strongly influenced by prevalence. For spam filtering we must arbitrarily deem either spam or ham to be "relevant" in order to compute precision and recall. The results in Table 10.14 show precision and recall under the common assumption that spam is "relevant". Note that micro-averaged precision and recall are redundant; both are equal to accuracy. It can easily be shown that this equality will hold whenever the categories are exclusive — as is the case for our spam filtering and language categorization examples. Macro-averaging precision and recall separately makes no sense, but it is possible to compute the macro-average of a summary recall/precision measure such as F_1, also shown in Table 10.14. We have difficulty offering a sensible interpretation for precision, recall, or F_1 in this context, and suggest they be avoided as evaluation measures for filtering tasks.

Table 10.15 The trade-off between spam and ham error rates as a function of the threshold setting.

Threshold t	Spam Error	Ham Error	Macro-Avg. Error	Logit Avg. Error	Macro-Avg. F_1
2156	56.8%	0.0%	28.4%	1.1%	0.66
1708	36.2%	0.1%	18.2%	2.3%	0.79
1295	10.0%	0.4%	5.2%	2.0%	0.94
1045	2.2%	1.0%	1.6%	1.5%	0.98
931	1.0%	2.8%	1.9%	1.7%	0.98
713	0.1%	10.0%	5.1%	1.0%	0.95
-180	0.0%	34.9%	17.4%	0.7%	0.84

10.1.6 Threshold Choice for Binary Categorization

When a ranking method such as BM25 is used for binary categorization, the choice of the threshold t involves a direct trade-off between the two category-specific error rates. In our spam filtering example we somewhat arbitrarily chose to label messages with $s(d) > 0$ as spam and the others as ham. We could have used any t instead of 0 and labeled messages with $s(d) > t$ as spam and the rest as ham. The error rates for several values of t are shown in Table 10.15.

As we can see, the threshold values in themselves are quite meaningless, but larger values of t yield lower ham error rates at the expense of higher spam error rates, and vice versa. A *receiver operating characteristic* (ROC) *curve* consists of all points (x, y) where x is the spam error and $1 - y$ is the ham error (that is, y is the ham accuracy) for some threshold t. The ROC curve thus provides a geometric characterization of filter effectiveness, independent of any particular threshold setting, in the same way that a recall-precision curve characterizes ranked retrieval effectiveness, independent of any particular cutoff.

Plotted on a linear scale (Figure 10.5), the ROC curve isn't much use as a visual indication of filter performance: It follows the x and y axes too closely. For this purpose the logit-transformed representation shown in Figure 10.6 is preferred. From this curve one can readily determine what spam error would result if were t adjusted to achieve some particular ham error or vice versa. One can also use the curve to compare filters. If one curve is superior to another, it indicates better effectiveness across the board. If the curves cross, which one is better depends on the relative importance of spam and ham errors. For comparison we juxtapose our BM25 results with those achieved by the popular SpamAssassin[3] filter on the same messages (version 3.02 with default parameters). We see that BM25 is superior to SpamAssassin for much of the range, but at very low ham error rates the curves cross and SpamAssassin is slightly better.

The *area under the ROC curve* (AUC or ROCA) is a number between 0 and 1 that may be used as a threshold-independent indicator of filter performance. A filter with a superior ROC

[3] `spamassassin.apache.org`

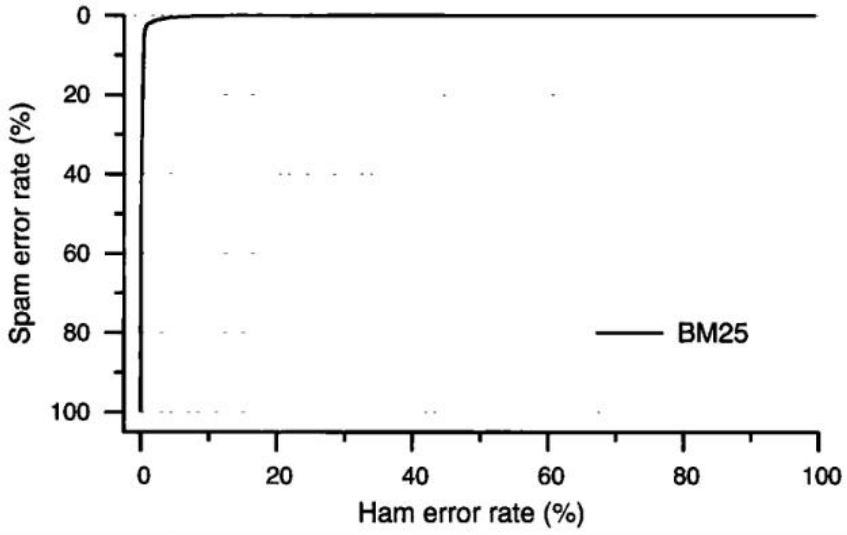

Figure 10.5 Linear-scale ROC curve for our BM25-based spam filter.

curve will tend to have a larger AUC. In addition to this geometric interpretation AUC has a probabilistic interpretation. It is the probability that a random spam message will have a higher score than a random ham message:

$$\text{AUC} = \Pr[s(d_1) > s(d_2) \mid d_1 \in \texttt{spam},\ d_2 \in \texttt{ham}]. \tag{10.2}$$

Our BM25 filter has AUC = 0.9981. For ease of interpretation, it is convenient to state the area *above* the ROC curve, 1 − AUC, as a percentage: 1 − AUC = 0.1896%. SpamAssassin has 1 − AUC = 0.5163%.

In the event that the threshold is fixed prior to evaluation, the ROC curve consists of exactly one point (x, y), so AUC is undefined. In this event the logistic average LAM offers a summary measure that is much less sensitive to threshold setting than others, as shown in Table 10.15. Micro- or macro-averaged error and F_1 are much more strongly influenced by threshold setting — to the extent that they measure little else.

If the filter is trained on historical data, it is a reasonably straightforward process to try several values of t to determine the one that achieves the desired trade-off between spam error and ham error. In an on-line setting with feedback, t may be adjusted dynamically by incrementing it by a small amount if the ham error is too high and decrementing it otherwise. But it is not easy to formalize "desired trade-off", and in many circumstances the choice of threshold is best determined by the user, either explicitly or implicitly. The user interface may solicit threshold adjustment (e.g., a slider from "show fewer" to "show more" or a set of buttons labeled "strict filtering", "moderate filtering", "none"). Or it may require the user to specify an acceptable error rate for one of the categories, or to specify some number of documents per unit time, or some fraction of the incoming documents that should be delivered. These quantities are easily estimated; t can be adjusted automatically to maintain them over time at the specified level.

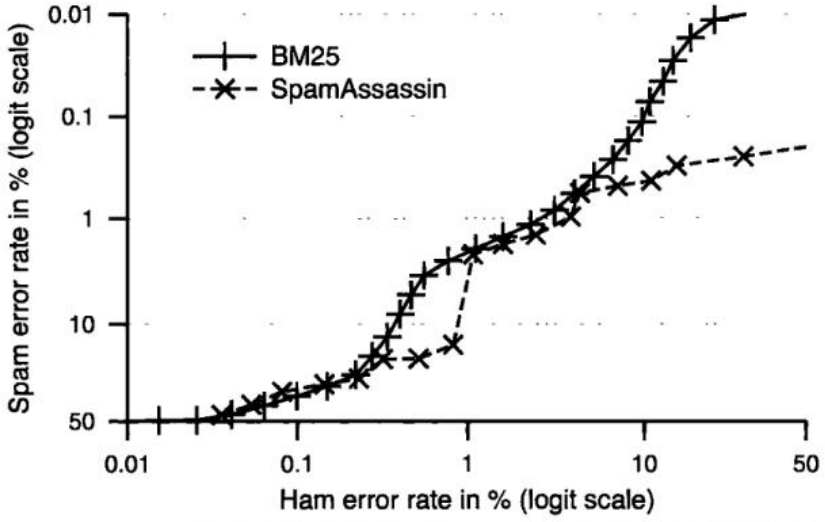

Figure 10.6 Logit-scale ROC curve for SpamAssassin and our BM25-based spam filter.

10.2 Classification

Having used BM25 to illustrate the nature of classification and filtering through detailed examples, we now examine other methods for solving the same types of problems. At the core of filtering and categorization is *classification*. The most fundamental classification problem is that of *binary classification*, the process of determining whether or not an object has some particular property P. A *binary classifier* is an automated procedure for binary classification. Throughout our development of binary classification methods we shall rely heavily on the example of spam filtering, in which the objects are e-mail messages, the property P is "being spam", and the classifier is the spam filter. Later we consider the application of the methods to our other examples, where the objects are documents and the property is relevance, or a particular language such as "Ukrainian".

The problem of classification long predates computer science. Much of the early work in classical statistics concerned binary classification problems such as identifying bad batches of beer for quality control at a brewery. From these efforts arose some arcane terminology that remains in common use. Then, as now, any real classifier will necessarily be imperfect. A *type 1 error* occurs when the classifier determines the object to have the property when it does not (identifies a good batch of beer as bad). A *type 2 error* occurs when the classifier fails to determine that the object has the property when it does (fails to detect a bad batch). The consequences of these errors depend on the intended purpose of classification. For beer the purpose of classification is presumably to avoid shipping bad beer to customers. If type 1 errors are frequent, profits are lost due to waste; if type 2 errors are frequent, the consequences are less tangible but potentially more severe, including lost goodwill and compromised public health.

Table 10.16 Contingency table for diagnostic test results.

		Property or Condition	
		absent	present
Test result	negative	tn	fn
	positive	fp	tp

The study of classification was further advanced through *signal detection theory*, invented in conjunction with radar. Signal detection theory has been widely adopted in medical diagnostic testing and other fields. Practically everyone is familiar with the use of diagnostic tests in this context. For example, a positive pregnancy test is taken as evidence (but not proof) that a person is pregnant. A *positive* test result is evidence for some condition; a *negative* test result is evidence against it. A *true positive* (tp) is a positive result that is correct, whereas a *false positive* (fp) is a positive result that is incorrect. Similarly, a *true negative* (tn) is a negative test result that is correct, and a *false negative* is a negative result that is incorrect. That is, false positives and false negatives are exactly type 1 and type 2 errors, respectively. Table 10.16 illustrates the four possibilities. It is known as the *contingency table* for diagnostic test results.

The effectiveness of a diagnostic test is typically characterized by its *sensitivity* and *specificity*, also known as *true positive rate* (tpr) and *true negative rate* (tnr). If we take the elements of Table 10.16 to represent the frequencies of the four possibilities for a particular test, we have

$$sensitivity = tpr = \frac{tp}{tp + fn}, \qquad specificity = tnr = \frac{tn}{tn + fp}. \tag{10.3}$$

Because sensitivity and specificity approach 1 for good tests, it is more convenient to work with the complementary error rates, *false negative rate* (fnr) and *false positive rate* (fpr):

$$fnr = 1 - sensitivity = \frac{fn}{tp + fn}, \qquad fpr = 1 - specificity = \frac{fp}{tn + fp}. \tag{10.4}$$

In pregnancy testing, the condition of interest is pregnancy. The false negative rate is the fraction of pregnant individuals for which the test is negative; the false positive rate is the fraction of nonpregnant individuals for which the test is positive.

False negative rates and false positive rates are very commonly misinterpreted through specious reasoning known as the *prosecutor's fallacy*. Suppose we administer a pregnancy test that has $fpr = 1\%$ and $fnr = 10\%$ to an individual getting a positive result. The prosecutor's fallacy asserts that the probability that the individual is pregnant is $1 - fpr = 99\%$. The absurdity of this conclusion is obvious when we consider the case of administering the test to a man. The probability of a man being pregnant is vanishingly small, regardless of what the test says. By far the most likely explanation is that the test result is a false positive. On the other hand, if the individual is a fertile woman displaying symptoms of pregnancy, the most likely explanation

is that the test result is a true positive, with probability much higher than 99%. In order to quantify the consideration of such prior probabilities, we introduce the notions of odds, odds ratios, and likelihood ratios.

10.2.1 Odds and Odds Ratios

The manipulation of probabilities and conditional probabilities may be simplified considerably by recasting them as *odds* and *odds ratios* (OR), respectively. Given an event e conditioned on x, we define

$$\text{Odds}[e] = \frac{\Pr[e]}{\Pr[\overline{e}]}, \tag{10.5}$$

$$\text{OR}(e, x) = \frac{\text{Odds}[e \,|\, x]}{\text{Odds}[e]}. \tag{10.6}$$

From the definition it follows directly that

$$\text{Odds}[e \,|\, x] = \text{Odds}[e] \cdot \text{OR}(e, x). \tag{10.7}$$

By applying Bayes' rule to $\Pr[e \,|\, x]$ and to $\Pr[\overline{e} \,|\, x]$ in Equation 10.7, we arrive at

$$\text{OR}(e, x) = \frac{\frac{\Pr[e|x]}{\Pr[\overline{e}|x]}}{\frac{\Pr[e]}{\Pr[\overline{e}]}} = \frac{\frac{\Pr[x|e]\cdot\Pr[e]\cdot\Pr[x]}{\Pr[x|\overline{e}]\cdot\Pr[\overline{e}]\cdot\Pr[x]}}{\frac{\Pr[e]}{\Pr[\overline{e}]}} = \frac{\Pr[x \,|\, e]}{\Pr[x \,|\, \overline{e}]}. \tag{10.8}$$

In this form the odds ratio is commonly known the *likelihood ratio* (LR) for x given e:

$$\text{LR}(e, x) = \frac{\Pr[x \,|\, e]}{\Pr[x \,|\, \overline{e}]}, \tag{10.9}$$

where $\Pr[x \,|\, e]$ is the likelihood of x given e, and $\Pr[x \,|\, \overline{e}]$ is the likelihood of x given $\overline{e}$.

Odds are easily interpreted and easily estimated by counting "wins" and "losses". If a sports team wins 50 games and loses 30, we estimate its odds of winning to be $\frac{50}{30} \approx 1.67$. An odds ratio may be interpreted as how much the odds would change if x were true. For example, if acquiring a new player, Mika, would improve our team's odds of winning to $\frac{51}{29} \approx 1.76$, we say that the odds ratio is

$$\text{OR}(\text{winning, Mika}) = \frac{\frac{51}{29}}{\frac{50}{30}} \approx 1.06. \tag{10.10}$$

In general, if $\text{OR}(e, x) > 1$, we say that x improves the odds of e; if $\text{OR}(e, x) < 1$, then x diminishes the odds. Now consider the effect of adding a different player, Jenson, where $\text{OR}(\text{winning, Jenson}) \approx 1.12$. This evidence indicates that Jenson would be the better acquisition. But what if it were possible to acquire both players? What would the value of

OR(winning, Mika *and* Jenson) be? The *naïve Bayes* assumption asserts that

$$\text{OR}(e, x \text{ and } y) = \text{OR}(e, x) \cdot \text{OR}(e, y) . \tag{10.11}$$

Under this assumption

$$\text{OR}(\text{winning, Mika } \textit{and} \text{ Jenson}) \approx 1.18 . \tag{10.12}$$

But in many circumstances this assumption is unrealistic. If Mika and Jenson play the same position on the team, for example, goalkeeper, there may be little cumulative benefit in hiring both. And if there were a personality conflict between Mika and Jenson, the overall effect might even be to diminish the odds of winning.

Now consider pregnancy testing. We wish to compute the odds of pregnancy, given a positive test result:

$$\text{Odds}[\text{pregnant} \mid \text{positive}] = \text{Odds}[\text{pregnant}] \cdot \text{OR}(\text{pregnant, positive}) . \tag{10.13}$$

From Equation 10.8 we have

$$\text{OR}(\text{pregnant, positive}) = \text{LR}(\text{pregnant, positive}) = \frac{tpr}{fpr} = \frac{90\%}{1\%} = 90 . \tag{10.14}$$

The *prior odds*, Odds[pregnant], depend entirely on evidence other than the test. If about 80% of women requesting a pregnancy test are in fact pregnant, we have

$$\text{Odds}[\text{pregnant} \mid \text{positive}] = \frac{80\%}{20\%} \cdot \frac{90\%}{1\%} = 360 \tag{10.15}$$

and thus

$$\Pr[\text{pregnant} \mid \text{positive}] = \frac{360}{1 + 360} \approx 99.7\% . \tag{10.16}$$

On the other hand, if the test were negative,

$$\text{Odds}[\text{pregnant} \mid \text{negative}] = \frac{80\%}{20\%} \cdot \frac{fnr}{tnr} = \frac{80\%}{20\%} \cdot \frac{10\%}{99\%} \approx 0.404 \tag{10.17}$$

and thus

$$\Pr[\text{pregnant} \mid \text{negative}] \approx \frac{0.404}{1 + 0.404} = 28.8\% . \tag{10.18}$$

In such a situation a second test would be in order.

10.2.2 Building Classifiers

For categorization and filtering applications, we wish to build binary classifiers automatically from examples of documents that do and do not belong to specific categories. When there are

two exclusive categories (e.g., ham and spam), a binary classifier is constructed to identify membership in one of the categories. A positive result indicates membership in that category, and a negative result indicates membership in the other. When the categories are nonexclusive, we may construct a separate binary classifier to identify each. For $n > 2$ categories it is necessary either to combine several binary classifiers or to build an n-way classifier (see Section 11.6). In this section we consider the problem of constructing a binary classifier, taking into account its role in categorization and filtering.

Classification may be formalized as a learner L that builds a classifier c, where c renders either a positive/negative result or a continuous score. Given a collection of documents D, we denote by $P \subset D$ the subset of documents having some property of interest. For convenience we denote by $\overline{P} = D \backslash P$ the complementary subset from which the property is absent. The key challenge is to build a classifier — a concrete function that for any document d accurately answers the question "Given d, is $d \in P$?" An ideal classifier would be a total function

$$isp: \ D \to \{pos, neg\} \tag{10.19}$$

such that $isp(d) = pos$ if and only if $d \in P$. No such ideal classifier exists; instead, we build an approximation $c \approx isp$. Hard and soft classifiers employ different notions of approximation. A hard classifier

$$c: \ D \to \{pos, neg\} \tag{10.20}$$

approximates isp to the extent that $c(d) = isp(d)$ for most $d \in D$. A soft classifier

$$c: \ D \to \mathbb{R} \tag{10.21}$$

approximates isp to the extent that $c(d) > c(d')$ for most $(d, d') \in P \times \overline{P}$. A hard classifier c_h may be defined in terms of a soft classifier c_s and a fixed threshold t:

$$c_h(d) = \begin{cases} pos & (c_s(d) > t) \\ neg & (c_s(d) \leq t) \end{cases} . \tag{10.22}$$

A naïve utility measure for the effectiveness of a hard classifier is

$$accuracy \ = \ 1 - error \ = \ \frac{|\{d | c_h(d) \ = \ isp(d)\}|}{|D|} . \tag{10.23}$$

Accuracy and *error* are commonly reported and commonly optimized in filter construction, notwithstanding their dependence on prevalence. As a pair we use the error measures

$$fpr \ = \ \frac{\left|\overline{P} \cap \{d | c(d) = pos\}\right|}{\left|\overline{P}\right|}, \quad fnr \ = \ \frac{|P \cap \{d | c(d) = neg\}|}{|P|} . \tag{10.24}$$

For a soft classifier, we use the cost measure

$$1 - \text{AUC} = \frac{\left|P \times \overline{P} \cap \{(d,d') \,|\, \text{c}(d) < \text{c}(d')\}\right| + \frac{1}{2}\left|P \times \overline{P} \cap \{(d,d') \,|\, \text{c}(d) = \text{c}(d')\}\right|}{|P| \cdot |\overline{P}|}. \quad (10.25)$$

For filtering it is useful to regard c as a fixed formula with a hidden parameter specifying a learned *profile*; that is, $\text{c}(d)$ is shorthand for $\text{c}(\mathit{profile}, d)$ where *profile* is created by a learner L.

10.2.3 Learning Modes

The learner L constructs the *profile* from evidence presented to it. The manner in which the evidence is presented depends on the *learning mode*.

- **Supervised learning** is a common mode for machine-learning classifiers. The learner's input (T, label) consists of a set $T \subseteq D$ of training examples and a function $\mathit{label} : T \to \{\mathit{pos}, \mathit{neg}\}$ that approximates *isp* over the subdomain T. The function *label* is typically handcrafted by human adjudicators. Under the assumption that T is an independent and identically distributed (i.i.d.) sample of D and that $\mathit{label}(d) = \mathit{isp}(d)$ for all $d \in T$, the learner induces a *profile* that optimizes c for some utility function, typically *accuracy*. Supervised learning, along with its associated assumptions, is so common that it is often assumed without question. But it can be exceedingly difficult to obtain a sample — especially an i.i.d. sample — of documents to be classified. Indeed, many members of D (the ones we are interested in classifying) exist only in the future and are therefore simply impossible to sample. Constructing *label* is sufficiently onerous and error-prone that the assumption of its existence may be questionable. One should not assume that optimizing accuracy yields the classifier most suitable for its intended purpose.
- **Semi-supervised learning** assumes input (T, S, label) where $T \subseteq D$, $S \subset T$, and $\mathit{label} : S \to \{\mathit{pos}, \mathit{neg}\}$; that is, *label* is defined for only a subset of the training examples. Semi-supervised learning accommodates the fact that obtaining sample documents may be considerably easier than labeling them. Like supervised learning, it assumes that T is an *i.i.d.* sample of D. This assumption may allow the learner to learn more about the distribution of D from the unlabeled examples in $T \backslash S$. A vacuous semi-supervised learner is simply the supervised learner with input (S, S, label); this special case provides a convenient baseline with which semi-supervised learners may be compared.
- **Transductive learning**, like semi-supervised learning, uses labeled and unlabeled examples (T, S, label) as before. The difference is that the unlabeled examples include all of the test examples; that is, $T = D$. So the classifier $\text{c} : T \to \{\mathit{true}, \mathit{false}\}$ or $\text{c} : T \to \mathbb{R}$ applies only to the elements of T. Classical information retrieval is an example of transductive learning, as the entire corpus is used for document statistics, and the set of documents classified as relevant is a subset of the corpus.

- **Unsupervised learning** assumes no *label* function at all; that is, the input is simply a set $T \subseteq D$ and is rarely used directly to construct a classifier. Unsupervised learning methods may nevertheless be used in conjunction with others; for example, *clustering* methods may be used to find groups of similar messages under the assumption that each member of a group belongs to the same class.
- In **on-line learning** there is no a priori division between training and test examples. The examples to be classified form a sequence $S = d_1, d_2, \ldots, d_n$. When each document d_k is classified, all previous documents $\{d_{i<k}\}$ are available as training examples. Every example is first used to test the classifier and subsequently used for training. An on-line classifier, but not necessarily an efficient or effective one, may be implemented by constructing a new batch classifier for each d_k, using $T_k = \{d_i \mid i < k\}$ as the training set. If every document is labeled, we have supervised on-line learning; otherwise it is semisupervised or unsupervised. This approach has two principal shortcomings:

  1. The nature of the examples may change considerably over time, a phenomenon known as *concept drift*. The proposed method is unable to model this phenomenon because it ignores the temporal cues inherent in the sequencing of the training examples.
  2. If we use every available training example to construct a new classifier c_k from scratch to classify each d_k, the overall running time over S will be quadratic or worse. When building c_k there are $k-1$ training examples, and examining each takes time proportional to k. A lower bound on the time to create all c_k is therefore

$$\sum_{i=k}^{n} k - 1 \quad \in \Omega(n^2)\,. \tag{10.26}$$

- **Incremental learning** may be used to reduce the overall cost of classifying a sequence. An incremental learner efficiently constructs c_{k+1} from the hidden $profile_k$ underlying the classifier c_k constructed for d_k, without necessarily examining all of the examples in T_k. The amenability of the learner to efficient incremental construction is an important criterion in the choice of a method for on-line filtering. Incremental learning may be approximated by using a non-incremental learner that uses batches and a sliding window.
- **Active learning** allows the classifier to request labels for a limited number of the unlabeled training examples. A filter may, for example, solicit the user to label particular messages and classify the rest based on these examples. The prototypical method for active learning is *uncertainty sampling* (Lewis and Catlett, 1994), in which a soft classifier is applied to each unlabeled example and labels are requested for those examples whose classifier result is closest to the threshold t.

10.2.4 Feature Engineering

Although we have characterized the classifier's domain D as a set of documents, few classifiers operate directly on the textual representation. Instead, each document is typically represented as a collection of *features* derived from the text or from extrinsic information related to it, such as the time of arrival of an e-mail message. The process of defining and extracting features likely to be useful to the classifier — called *feature engineering* — has a profound impact on overall filter effectiveness. The reader should be skeptical of published results (good or bad) for any particular learning method that fail to note the method of feature representation.

A document d is typically represented as a vector of n features $x^{[d]} = \langle x_1^{[d]}, x_2^{[d]}, \ldots, x_n^{[d]} \rangle$, where each $x_i^{[d]}$ is a real or discrete value quantifying some evidence pertaining to the message that might be useful to the classifier. Each message is thus represented as a point in an n-dimensional *feature space*. The most obvious features are simple statistics computed from the tokens and terms detailed in Chapter 3.

The issues involved in tokenization are generally the same for classification as for search, with a few differences in emphasis. In filtering and categorization applications, documents are processed in sequence rather than fetched from a large corpus, so index construction is unnecessary. It is possible to use far more features for classification than might be appropriate for a search engine, which would have to retrieve a postings list for each. Features such as character and word n-grams (see Section 3.3) are practical and have been found useful in a number of applications. *Sparse bigrams* — pairs of tokens separated by k or fewer tokens — increase the dimension of the feature space but have been used to good effect in filtering (Siefkes et al., 2004).

Some learners nevertheless perform poorly, either in terms of efficiency or of classification performance, for large n. Consider the problem of learning a classifier from training set T where $|T| \ll n$. In such a case it is almost certainly possible to solve a set of simultaneous equations so as to build a perfect classifier for T. But that classifier would have poor performance over D as a whole, a flaw known as *overfitting*. All classifiers exhibit some *generalization error* in that they perform worse on D than on T. One of the primary differences among methods is their ability to minimize such error.

Feature selection is commonly used to reduce n, and hence the dimensionality of the feature space X. More generally, *dimensionality reduction* techniques may be used to project or transform X to a space of smaller dimension. There are two principal purposes for dimensionality reduction: It may be required to achieve reasonable time or space efficiency and it may be required to reduce generalization error.

Historically, feature selection or dimensionality reduction has been considered a separate preprocessing step done prior to building the classifier. In light of modern advances in classifier construction, this view is no longer appropriate. There is a strong interaction between feature selection and the classifier, so the two should be considered together. There is nothing in our definition of c that prevents it from projecting its input onto a smaller space as part of its internal working. Some classifiers, such as the naïve Bayes and decision tree methods we shall consider, do this quite aggressively. Some of the best classifiers, such as logistic regression and support

vector machines, do no explicit dimensionality reduction, handling the issues of efficiency and generalization error by other means. We are not saying that feature selection and dimensionality reduction have no use; rather we are saying that they must be considered in the context of a particular application and classification method, and in many circumstances are unnecessary.

Feature selection is the most direct method of dimensionality reduction; stopping (Section 3.1.3) is a trivial example. Many statistical techniques have been proposed and used to identify the most *important* features; the rest are eliminated. Stemming (Section 3.1.2) is a simple example of dimensionality reduction in which multiple features are conflated into one. Hashing has also been used successfully to reduce very large spaces by arbitrarily combining dimensions whose tokens hash to the same value. More sophisticated methods, such as principal component analysis, use linear algebra to transform the entire space to one of smaller dimension.

Some feature engineering choices may be difficult to reconcile with incremental classifier construction, and hence with efficient on-line adaptive filtering. Previously unseen features may be encountered at any point, thus expanding the dimensionality of the space. The set of known values for a particular feature may grow as new examples are learned. Global statistics such as IDF must be recomputed when used because the addition of a single document changes the IDF for every term it contains. This change may dramatically alter the effective scores of previously filtered documents. Statistical feature selection and dimensionality reduction are problematic in an on-line environment because the statistics are ever-changing.

A common error in evaluating classifiers is to perform feature selection or dimensionality reduction based on the training and test documents. This approach is simply wrong; it effectively communicates information about the test examples to the classifier's profile.

10.3 Probabilistic Classifiers

A probabilistic classifier computes an estimate $p^{[d]} \approx \Pr[d \in P \,|\, x^{[d]}]$ of the probability that a given document d, represented by $x^{[d]} = \langle x_1^{[d]}, x_2^{[d]}, \ldots, x_n^{[d]} \rangle$, has property P or equivalently, is contained in the set P of documents with that property. This estimate may be used directly as a soft classifier

$$c(d) = p^{[d]} \tag{10.27}$$

or, in combination with some threshold $0 < t < 1$, as a hard classifier

$$\mathrm{c}(d) = \begin{cases} pos & (p^{[d]} > t) \\ neg & (p^{[d]} \leq t) \end{cases}. \tag{10.28}$$

Probabilistic classification involves two steps:

1. Estimating $p_i^{[x]} \approx \Pr[d \in P \,|\, x_i^{[d]}]$ for each $x_i^{[d]}$ in $x^{[d]}$.
2. Combining the estimates to yield $p^{[d]} \approx \Pr[d \in P \,|\, x^{[d]}]$.

10.3.1 Probability Estimates

Features may be either discrete or continuous. A discrete feature (also called a *categorical feature*) can assume one of a finite number of possible values. For a continuous feature the set of possible values is infinite. Different methods are used to estimate probabilities for categorical and for continuous features.

Probability estimates from categorical features

We first consider the special case of a *binary* feature that has exactly two possible values, 0 and 1. Although the values of a binary feature $x_i^{[d]} : \{0, 1\}$ have no intrinsic meaning, $x_i^{[d]} = 1$ often represents the presence of some token in d, whereas $x_i^{[d]} = 0$ indicates its absence. The odds that d is in P given $x_i^{[d]} = k$ may be estimated as the ratio of positive to negative examples in the training set T:

$$\text{Odds}[d \in P \,|\, x_i^{[d]} = k] \;\approx\; \frac{\left|\left\{d \in T \,|\, x_i^{[d]} = k\right\} \cap P\right|}{\left|\left\{d \in T \,|\, x_i^{[d]} = k\right\} \cap \overline{P}\right|}. \tag{10.29}$$

For example, consider the hypothetical situation in which the word "money" occurs in 100 spam messages and 5 ham messages. The odds that a particular message d containing "money" is spam may be estimated as $\text{Odds}[d \in P \,|\, x_i^{[d]} = 1] \approx \frac{100}{5} = \frac{20}{1}$. The same estimate, expressed as a probability, is $\Pr[d \in P \,|\, x_i^{[d]} = 1] \approx \frac{20}{1+20} = 0.952$. The value $k = 1$ is not particularly special; assuming that T consists of, say, 1000 spam and 1000 ham messages, we may deduce that 900 spam messages and 995 ham messages have $x_i = 0$, so the odds of a message not containing "money" being spam are $\text{Odds}[d \in P \,|\, x_i^{[d]} = 0] = \frac{900}{995} \approx 0.9$, that is, nearly even odds. Intuitively, the nonoccurrence of "money" contributes little to solving the filtering problem; for this reason filters usually ignore the information obtainable from the absence of a token.

The ratio $\frac{a}{b}$ of the number of positive to negative training examples is a good odds estimate if these numbers, a and b, are sufficiently large. If they are small, the estimates will be unreliable due to chance, and if either or both are 0, the resulting estimates are $\frac{0}{1}$, $\frac{1}{0}$, or $\frac{0}{0}$, none of which is a sensible estimate. A simple way to mitigate this problem is smoothing: We add small positive constants γ and ϵ to the numerator and denominator, respectively; that is, we use $\frac{a+\gamma}{b+\epsilon}$ as the odds estimate. In the case of $a = b = 0$ this yields an estimate of $\frac{\gamma}{\epsilon}$, whereas for large a and b the estimate is indistinguishable from $\frac{a}{b}$. Typically $\gamma = \epsilon = 1$.

We note that this derivation may be extended to non-binary categorical features, where k may take on values other than 0 or 1. Odds are then estimated separately for each k.

Probability estimates from continuous features

If $x_i^{[d]}$ is a real-valued feature, a direct way to estimate a probability is to transform its value to a binary value $b_i^{[d]} : \{0, 1\}$ by comparing it against a threshold t

$$b_i^{[d]} = \begin{cases} 1 & x_i^{]d]} > t \\ 0 & x_i^{]d]} \leq t \end{cases} \tag{10.30}$$

and to estimate $\text{Odds}[d \in P \,|\, b_i^{[d]} = k]$ as described in the previous section. Although a real value, like a discrete value, has no intrinsic meaning, features may be engineered so that a larger value of $x_i^{[d]}$ indicates higher odds that $d \in P$. In other words, $x_i^{]d]}$ is itself a soft classifier and $b_i^{]d]}$ is the corresponding hard classifier. An n-ary categorical value $b_i^{[d]} \in \{0, 1, \ldots, n-1\}$ may be computed using n *bins* delimited by $n-1$ threshold values:

$$b_i^{]d]} = \begin{cases} n-1 & t_{n-1} < x_i^{]d]} \\ \cdots & \cdots \\ 1 & t_1 < x_i^{]d]} \leq t_2 \\ 0 & x_i^{[d]} \leq t_1 \end{cases}, \tag{10.31}$$

thus effecting a piecewise approximation of the odds implied by the continuous value $x_i^{[d]}$.

The principal drawback of this approach is that as n increases, the number of documents with $b_i^{[d]} = k$ for any particular k decreases, and the odds estimates become less reliable. An alternative approach is to define a transformation $f : \mathbb{R} \to \mathbb{R}$ such that

$$f(k) \approx \text{Odds}[d \in P \,|\, x_i^{]d]} = k]\,. \tag{10.32}$$

Parametric models may be used instead of simple counting to estimate the distributions of $x_i^{]d \in P]}$ and $x_i^{]d \in \overline{P}]}$. For example, if a Gaussian[4] (normal) distribution is assumed, the four parameters $\mu_{(i,P)}$, $\sigma_{(i,P)}$, $\mu_{(i,\overline{P})}$, $\sigma_{(i,\overline{P})}$ — the means and standard deviations for $x_i^{]d \in P]}$ and $x_i^{]d \in \overline{P}]}$ — fully characterize the distributions. Given these parameters, we may compute the likelihood ratio

$$\text{LR}(d \in P,\ x_i^{]d]} = k) \approx \frac{g(\mu_{(i,P)}, \sigma_{(i,P)}, k)}{g(\mu_{(i,\overline{P})}, \sigma_{(i,\overline{P})}, k)}, \tag{10.33}$$

where g is the probability density function of the Gaussian distribution. By equations 10.7 and 10.8, we have

$$\text{Odds}[d \in P \,|\, x_i^{[d]} = k] \approx \frac{N_P}{N_{\overline{P}}} \cdot \frac{g(\mu_{(i,P)}, \sigma_{(i,P)}, k)}{g(\mu_{(i,\overline{P})}, \sigma_{(i,\overline{P})}, k)}, \tag{10.34}$$

where N_P and $N_{\overline{P}}$ are the number of positive and negative examples in T.

[4] See Section 12.3.2 for a discussion of the Gaussian distribution: $g(\mu, \sigma^2, x) = \varphi_{\mu,\sigma^2}(x)$.

Table 10.17 Example from the TREC 2005 Public Spam Corpus.

Message tag	True Class	head:enron	body:enron
016/201	spam	12	0
033/101	spam	11	0
050/001	spam	10	0
066/186	ham	7	24
083/101	ham	21	0
083/101	ham	21	0
100/001	ham	27	4
133/101	spam	12	17
148/013	ham	22	5
166/201	ham	13	23
183/101	spam	11	0
200/001	spam	14	4
216/201	ham	25	2
233/101	spam	13	20
250/001	ham	5	0
266/201	spam	12	0
283/101	spam	13	0
300/001	spam	11	22

An example

We illustrate the methods described above by using two features derived from the TREC 2005 Public Spam Corpus (Cormack and Lynam, 2005). Our features are chosen to harness the knowledge that all messages in the corpus were delivered to individuals at one particular organization and that the organization's name (Enron in this instance) might have different prevalence in spam and ham messages. Our two features simply count the number of occurrences of the character sequence "enron" in the header and in the body of each message after conversion to lowercase. The notation `head : enron` indicates the number of instances of "enron" in the header, and `body : enron` indicates the number in the body. Table 10.17 presents these attributes for 18 messages selected from the corpus, 10 of which are spam and 8 of which are ham.

As a binary feature indicating the presence of "enron" in the respective message components, this information is of limited use. The token "enron" occurs in *every* header, and therefore its presence yields no information beyond the ratio of spam to ham in the sample (i.e., a likelihood ratio of 1). The token occurs in the bodies of 4 spam and 5 ham messages, yielding a 4:5 estimate of the odds that a message whose body contains "enron" is spam. It does not occur in the bodies of 6 spam and 3 ham message, yielding a 2:1 odds estimate for such messages.

Table 10.18 Sample versus gold standard estimates for the example shown in Table 10.17.

	Training Data		Gold Standard	
Feature f	**Frequency**	**Pr[spam $\mid f$]**	**Frequency**	**Pr[spam $\mid f$]**
head : enron $\neq 0$	1.0	0.56	0.9999	0.57
head : enron $= 0$	0.0	0.50	0.0001	0.00
body : enron $\neq 0$	0.5	0.44	0.62	0.45
body : enron $= 0$	0.5	0.67	0.38	0.77
Overall	1.0	0.56	1.00	0.57

Table 10.19 Discrete range feature estimates for the example shown in Table 10.17.

	Training Data			Gold Standard	
Feature f	**Frequency**	$\gamma = \epsilon = 0$	$\gamma = \epsilon = 1$	**Frequency**	**Pr[spam $\mid f$]**
$0 \leq$ head : enron < 10	0.11	0.00	0.25	0.18	0.05
$10 \leq$ head : enron < 20	0.61	0.91	0.85	0.74	0.75
$20 \leq$ head : enron < 30	0.28	0.00	0.14	0.05	0.19

Table 10.18 compares these estimates (recast as probabilities) with our "gold standard" best estimate of the *true* probability — computed over the entire corpus.

Table 10.19 shows the result of splitting the values of **head:enron** into three discrete ranges: $[0, 9]$, $[10, 19]$, and $[20, 30]$. Probability estimates derived from the training data are shown for two choices of smoothing parameters: $\gamma = \epsilon = 0$ and $\gamma = \epsilon = 1$ (see page 340 for a reminder of how this kind of smoothing works). Values in the center range ($10 \leq$ **head : enron** < 20) clearly predict spam, whereas extreme values indicate ham.

Table 10.20 shows the predictions made for each possible value of **head:enron** assuming a Gaussian distribution with parameters computed from the sample: $\mu_P = 11.9$, $\sigma_P = 1.2$, $\mu_{\overline{P}} = 17.6$, $\sigma_{\overline{P}} = 8.3$. The model aptly estimates $\Pr[d \in P \mid \textbf{head : enron} = k]$ for small values of k but dramatically underestimates it for larger values. These larger values are fairly rare, thus mitigating the effect of the underestimate; and even the underestimates will yield a correct classification more often than not. Nevertheless, there is plenty of room to improve the model.

10.3.2 Combining Probability Estimates

We wish to estimate

$$p^{[d]} \approx \Pr[d \in P \mid x^{[d]}], \tag{10.35}$$

given separate estimates for the individual features of d:

$$p_i^{[d]} \approx \Pr[d \in P \mid x_i^{[d]}] \quad (\text{for } 1 \leq i \leq n). \tag{10.36}$$

Table 10.20 Sample versus gold standard spam estimates under the Gaussian model for the example shown in Table 10.17.

	Training Data		Gold Standard	
k	**Frequency**	**Pr[spam | head : enron = k]**	**Frequency**	**Pr[spam | head : enron = k]**
5	0.06	0.0000	0.00	0.0000
6	0.00	0.0000	0.01	0.0705
7	0.06	0.0017	0.08	0.0000
8	0.00	0.0311	0.05	0.0409
9	0.00	0.2315	0.03	0.1767
10	0.06	0.5880	0.07	0.6191
11	0.17	0.7735	0.28	0.8366
12	0.17	0.8049	0.19	0.7343
13	0.17	0.7158	0.09	0.7838
14	0.06	0.4371	0.04	0.7269
15	0.00	0.1079	0.02	0.6321
16	0.00	0.0094	0.01	0.4687
17	0.00	0.0004	0.01	0.4162
18	0.00	0.0000	0.01	0.4838
19	0.00	0.0000	0.01	0.3539
20	0.00	0.0000	0.01	0.5745
21	0.11	0.0000	0.01	0.4236
22	0.06	0.0000	0.01	0.4008
23	0.00	0.0000	0.00	0.5281
24	0.00	0.0000	0.00	0.1026
25	0.06	0.0000	0.02	0.0114
26	0.00	0.0000	0.00	0.0629
27	0.06	0.0000	0.00	0.0026

For convenience, we compute instead the corresponding log-odds estimate

$$l^{[d]} \approx \text{logOdds}[d \in P \,|\, x^{[d]}], \tag{10.37}$$

where

$$l^{[d]} = \text{logit}(p^{[d]}) = \log \frac{p^{[d]}}{1 - p^{[d]}}, \qquad p^{[d]} = \text{logit}^{-1}(l^{[d]}) = \frac{1}{1 + e^{-l^{[d]}}}. \tag{10.38}$$

We also define

$$l_i^{[d]} \approx \text{logOdds}[d \in P \,|\, x_i^{[d]}] \quad (\text{for } 1 \leq i \leq n). \tag{10.39}$$

We consider the special cases $n = 0 \ldots 2$, as well as their generalization to $n > 2$:

- $n = 0$ denotes the empty vector, so the estimate, which we denote l_0, reduces to

$$l^{[d]} = l_0^{[d]} = \frac{|P \cap T| + \gamma}{|\overline{P} \cap T| + \epsilon} \approx \text{logOdds}[d \in P] \tag{10.40}$$

  (where γ and ϵ are smoothing parameters, as before).

- $n = 1$ is also trivial; we have

$$l^{[d]} = l_1^{[d]} \approx \text{logOdds}[d \in P \,|\, x_1^{[d]}]\,. \tag{10.41}$$

- $n = 2$ is more problematic; there is no general method of combining $l_1^{[d]}$ and $l_2^{[d]}$ into a common estimate without considering the conditional dependence of $x_1^{[d]}$ and $x_2^{[d]}$. From Equations 10.7 and 10.40 we have

$$\log \text{OR}(d \in P, x_1^{[d]}) \approx l_1^{[d]} - l_0^{[d]}\,, \tag{10.42}$$

$$\log \text{OR}(d \in P, x_2^{[d]}) \approx l_2^{[d]} - l_0^{[d]}\,. \tag{10.43}$$

  Under the naïve Bayes assumption (Equation 10.11) we have

$$\log \text{OR}(d \in P, x^{[d]}) = \log \text{OR}(d \in P, x_1^{[d]} \textit{ and } x_2^{[d]}) \approx l_1^{[d]} - l_0^{[d]} + l_2^{[d]} - l_0^{[d]} \tag{10.44}$$

  and therefore

$$l^{[d]} = -l_0^{[d]} + l_1^{[d]} + l_2^{[d]} \approx \text{logOdds}[d \in P \,|\, x^{[d]}]\,. \tag{10.45}$$

  The naïve Bayes assumption seldom holds in practice; the word "sildenafil", for example, is far more likely to be found in e-mail messages — spam and ham alike — that also contain the word "Viagra". Invalid assumptions aside, naïve Bayes classifiers are commonly used because they are simple and perform adequately as hard classifiers with a probability threshold $t = 0.5$, even if their probability estimates are far from accurate (Domingos and Pazzani, 1997).

  A contrasting assumption is that x_1 and x_2 are dependent; for example, that the presence of both "sildenafil" and "Viagra" is an indicator of spam; but whether one, or the other, or both occur in a particular message is of no consequence. In short, a message containing "sildenafil" *and* "Viagra" is no more and no less likely to be spam than a message containing either term alone. Under this assumption l_1 and l_2 may be averaged because they are both estimates of the same quantity and differ only in estimation error:

$$l^{[d]} = \frac{l_1^{[d]} + l_2^{[d]}}{2}\,. \tag{10.46}$$

Table 10.21 Combining probability estimates using log-odds averaging and Naïve Bayes.

		$\Pr[\text{spam} \mid f_1, f_2]$		
Feature f_1	**Feature f_2**	**Log-Odds Avg.**	**Naïve Bayes**	**Gold Standard**
$0 \leq$ head : enron < 10	body : enron $= 0$	0.45	0.36	0.14
$0 \leq$ head : enron < 10	body : enron > 0	0.33	0.16	0.03
$10 \leq$ head : enron < 20	body : enron $= 0$	0.77	0.90	0.86
$10 \leq$ head : enron < 20	body : enron > 0	0.66	0.76	0.65
$20 \leq$ head : enron < 30	body : enron $= 0$	0.36	0.21	0.40
$20 \leq$ head : enron < 30	body : enron > 0	0.25	0.08	0.12

For both naïve Bayes and log-odds averaging the general solution for $n \geq 2$ computes $l^{[d]}$ as a linear combination of the individual $l_i^{[d]}$:

$$l^{[d]} = \sum_{i=0}^{n} \beta_i \cdot l_i^{[d]}. \tag{10.47}$$

For naïve Bayes,

$$\beta_i = \begin{cases} 1-n & (i=0) \\ 1 & (i>0) \end{cases}, \tag{10.48}$$

for log-odds averaging,

$$\beta_i = \begin{cases} 0 & (i=0) \\ \frac{1}{n} & (i>0) \end{cases}. \tag{10.49}$$

Table 10.21 compares the two methods, using all combinations of values for the two discrete features in our running example. We see that averaging tends to yield conservative results closer to $p_0 = 0.55$, whereas those due to naïve Bayes are more extreme. For some examples, averaging appears to yield the better estimate; for others, naïve Bayes does.

Many other choices of β_i are possible. One might, for example, average the naïve Bayes and log-odds averaging estimates, which would yield a different linear combination that reflects partial conditional dependence among the various $x_i^{[d]}$. Or one might choose weights other than $\frac{1}{n}$ for specific β_i, depending on the accuracy of the corresponding $l_i^{[d]}$.

Logistic regression is a method that computes β_i so as to maximize *likelihood*, given a set of labeled training examples. Likelihood is simply the combined probability of the examples under the assumption that the estimate $p^{[d]} = \frac{1}{1+e^{-l^{[d]}}} = \Pr[d \in P \mid x^{[d]}]$; that is,

$$\textit{likelihood} = \prod_{d \in T \cap P} p^{[d]} \cdot \prod_{d \in T \cap \overline{P}} 1 - p^{[d]}. \tag{10.50}$$

Inputs:

Set $T \subset D$ of training examples $d \in T$, represented by $x^{[d]} = \langle 1, x_1^{[d]}, x_2^{[d]}, \ldots, x_n^{[d]} \rangle$

Labeling $label : T \rightarrow \{0, 1\}$

Smoothing parameters γ, ϵ

Output:

$\beta \cdot x^{[d]}$ is the naïve Bayes estimate of logOdds$[label(d) = 1]$, where $\beta = \langle \beta_0, \ldots, \beta_n \rangle$

```
1  p ← a ← ⟨0, ... 0)
2  for d ∈ T do
3    for i ∈ [0..n] do
4      if x_i^[d] = 1 then
5        if label(i) = 1 then p_i ← p_i + 1 else  a_i ← a_i + 1
6  β_0 ← logit((p_0+γ)/(a_0+ε))
7  for i ← 1 to n do
8    β_i ← logit((p_i+γ)/(a_i+ε)) − β_0
```

Figure 10.7 Naïve Bayes classifier construction.

Because logistic regression solves for β_i, it is not essential that the separate $l_i^{[d]}$ be calibrated as log-odds estimates. In particular it is unnecessary to transform categorical features for the purpose of logistic regression. Instead, a feature $x_i^{[d]} : \{k_1, k_2, \ldots, k_m\}$ may be interpreted as m distinct binary features $x_{i1}^{[d]}, x_{i2}^{[d]}, \ldots, x_{im}^{[d]}$ where

$$x_{ij}^{[d]} = \begin{cases} 1 & (x_i^{[d]} = k_j) \\ 0 & (x_i^{[d]} \neq k_j) \end{cases} .$$

Commonly $x_i^{[d]}$ is binary-valued and $x_{i0}^{[d]}$ is discarded for the reasons stated in Section 10.3.1, so $x_i^{[d]}$ is effectively replaced by $x_{i1}^{[d]}$. It is also unnecessary to transform continuous features, so long as they are proportional (or nearly proportional) to log-odds.

10.3.3 Practical Considerations

The choice of feature representation and combining method may have a dramatic effect on the simplicity and efficiency of the resulting classifier, particularly for on-line deployment. We have previously mentioned that feature transformations such as TF-IDF and statistical feature selection are difficult to reconcile with adaptive classifiers. Probability-based interpretations that model global distributions entail similar difficulties. For this purpose discrete features that may be derived from individual messages, independent of others in the training set, are more amenable to on-line settings. Even for batch filtering, these simple feature representations usually work as well as or better than more complex and less adaptive ones based on global statistics.

Inputs:

Set $T \subset D$ of training examples $d \in T$, represented by $x^{[d]} = \langle 1, x_1^{[d]}, x_2^{[d]}, \ldots, x_n^{[d]} \rangle$

Labeling $label : T \rightarrow \{0, 1\}$

Rate parameter δ

Output:

$\beta \cdot x^{[d]}$ is the maximum likelihood estimate of $\Pr[label(d) = 1]$ over $d \in T$

```
1  β ← ⟨0, ..., 0⟩
2  loop until convergence:
3    for d ∈ T do
4      p ← 1 / (1 + e^{−β·x^[d]})
5      β ← β + (label(d) − p) · δ · x^[d]
```

Figure 10.8 Logistic regression using the gradient descent method.

Provided amenable features are used, a naïve Bayes classifier is very easy to implement and there is little difference between batch and on-line versions. The batch version is shown in Figure 10.7. It simply counts the number of occurrences of each feature in $T \cap P$ and $T \cap \overline{P}$ and then computes the log-odds coefficients from them. An equivalent on-line adaptive version (not shown) simply combines the counting and coefficient calculations. Incremental discovery of new features is easily accomplished by setting the initial count of each new feature to 0 when it first occurs in a document.

Although the naïve Bayes method makes an adequate hard classifier, it is a horrible soft classifier because it overestimates the combined effect of the individual features. The more features considered, the larger the overestimate. To mitigate this effect we may select a fixed number of terms per document. For some fixed value m only the m largest and m smallest values of $x_i^{[d]}$ are used in computing $x^{[d]}$. This selection process tends to normalize the magnitude of the overestimate among documents, and among positive and negative features.

Logistic regression is traditionally viewed as a batch algorithm, but a simple *gradient descent* method yields an implementation that is simpler and more effective than naïve Bayes and that may be used in on-line or batch scenarios. Gradient descent methods find the local minimum of a function by taking a step along the direction of the negative gradient at each iteration. The batch version is presented in Figure 10.8. The simplest incremental version, instead of iterating until convergence, performs exactly one gradient step for each incoming document. It is also possible to maintain an incremental history of training examples seen so far and to train on them. For example, when training a new positive example, one could also train a negative example selected at random, and so on. The net effect is to balance the number of positive and negative examples, which may yield better classification.

In many circumstances the space efficiency of gradient descent makes it attractive for very large problems. However, if the number of training examples is large, more time-efficient algorithms are available.

The relatively slow convergence of the gradient descent method may be an advantage in combating overfitting. Logistic regression, after all, can solve a system of simultaneous equations, and hence overfit, when the number of features exceeds the number of examples. Through a suitable choice of the learning rate δ, the gradient descent method avoids this pitfall. Batch methods, on the other hand, use a technique known as *regularization* to avoid overfitting. Regularization involves not only maximizing the likelihood with respect to the training examples but also minimizing the magnitude of β because large coefficients tend to indicate overfitting. The trade-off between maximizing likelihood and minimizing $|\beta|$ is specified by a regularization parameter, often denoted as C.

All major statistical and mathematical software packages implement batch logistic regression, which is a standard tool for scientific research. For classification notable implementations include Weka (Witten and Frank, 2005), LR-TRIRLS (Komarek and Moore, 2003), and LibLinear.[5]

10.4 Linear Classifiers

A linear classifier views the feature vector $x^{[d]}$ for a message d as a point in an n-dimensional space, where n is the number of features. The classifier consists of a vector of coefficients $\beta = \langle \beta_1, \beta_2, \ldots, \beta_n \rangle$ and a threshold t. The equation $\beta \cdot x = t$ defines a hyperplane that divides the space into half-spaces. All points on one side of the hyperplane ($\beta \cdot x^{[d]} > t$) are classified as positive and the ones on the other side ($\beta \cdot x^{[d]} \leq t$) are classified as negative. $\beta \cdot x = t$ is a *separating hyperplane* if $\forall_{d \in P}$: $\beta \cdot x^{[d]} > t$ and $\forall_{d \in \overline{P}}$: $\beta \cdot x^{[d]} \leq t$. A set of messages is said to be *linearly separable* if there exists a separating hyperplane for the set. The log-odds formulation of the probabilistic classifier developed in the previous section is an example of a linear classifier. In this section we describe a geometric interpretation and several construction methods.

For convenience we limit our illustrations to the case of $n = 2$; it should be kept in mind that typical filtering applications involve more features and hence more dimensions. Figure I0.9 shows the vector space representation for the I8 messages in our running example (Table I0.17). The x-axis corresponds to the `head:enron` feature transformed using the Gaussian model (Table I0.20). The y-axis corresponds to the `body:enron` feature represented as a simple count. The diagonal line is a separating hyperplane because all positive examples fall on one side and all negative examples on the other. Thus it is a perfect classifier — at least for the sample data.

Figure I0.I0 shows that the same line is not a separating hyperplane for a larger sample from the same source; indeed, none exists. Still, most positive examples lie on the *spam* side of the line and most negative ones lie on the other side. Thus the line is a reasonable classifier. But is it the *best* linear classifier within this vector space? And how may it be chosen using only the training data? The answer depends on the definition of *best*.

[5] `www.csie.ntu.edu.tw/~cjlin/liblinear`

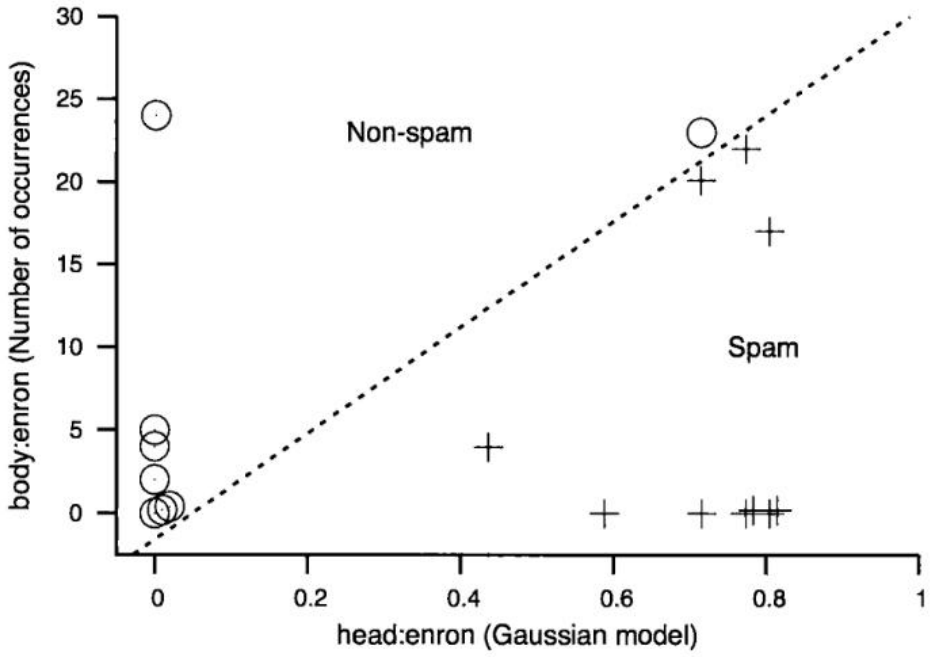

Figure 10.9 A linearly separable sample along with one possible separating hyperplane.

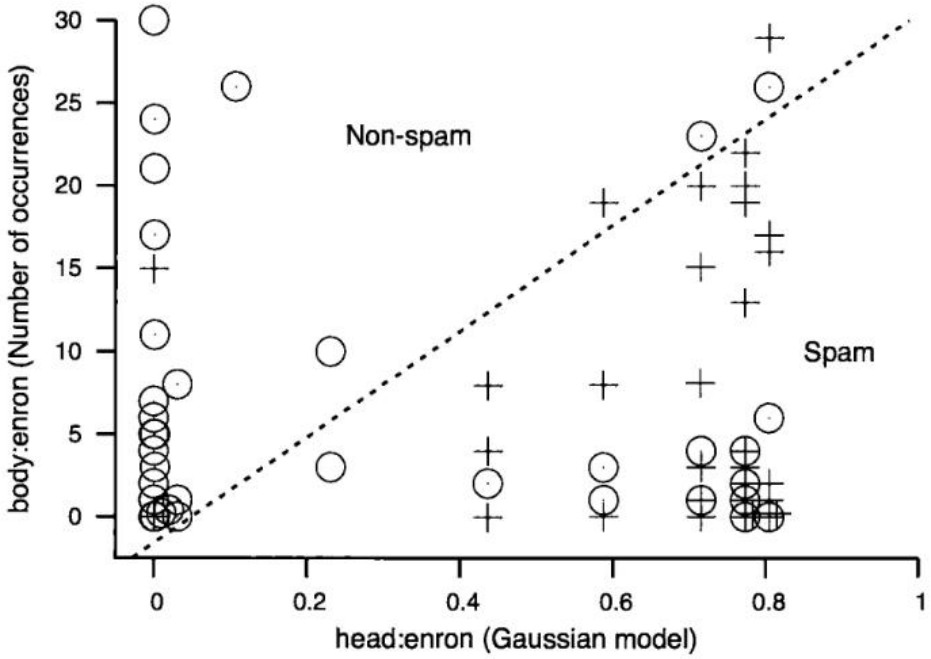

Figure 10.10 A larger linearly inseparable sample.

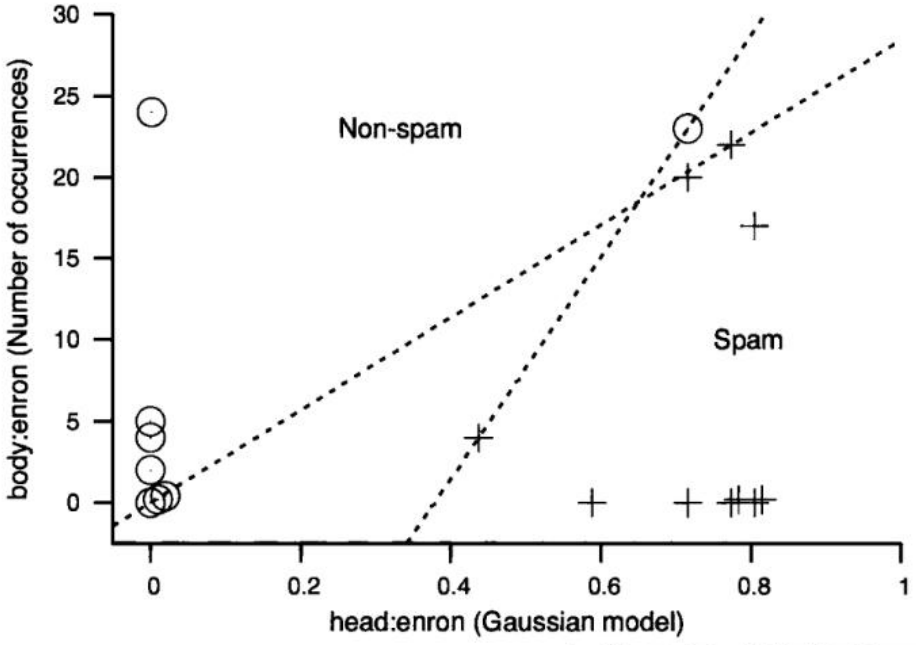

Figure 10.11 A linearly separable sample with two separating hyperplanes. Which one is "better"?

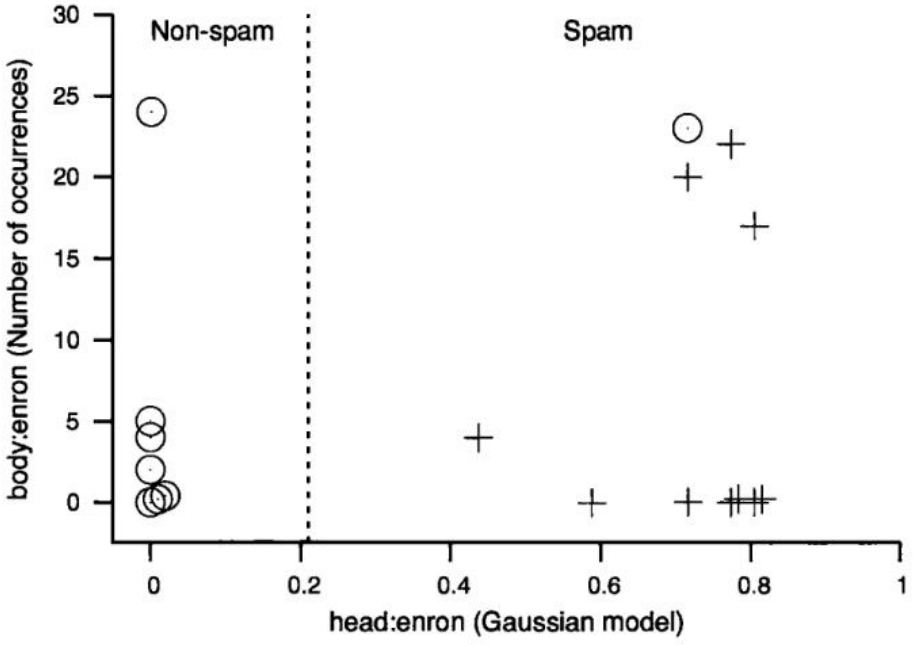

Figure 10.12 Ignoring one point.

Inputs:
Set $T \subset D$ of training examples $d \in T$, represented by $x^{[d]} = \langle 1, x_1^{[d]}, x_2^{[d]}, \ldots, x_n^{[d]} \rangle$
Labeling $label : T \rightarrow \{-1, 1\}$
Output:
if linearly separable, β such that $\beta \cdot x^{[d]} > 0$ if and only if $label(d) = 1$
else fails to terminate

```
1  β ← ⟨0, ..., 0⟩
2  while ∃_{d∈T} : β · x^[d] · label(d) < 0 do
3      β ← β + x^[d] · label(d)
```

Figure 10.13 The perceptron learning algorithm.

If the points are linearly separable, there are in general an infinite number of separating hyperplanes. Any positive linear combination of the extreme curves shown in Figure 10.11 will itself separate positive from negative. It is not apparent that the best classifier is a separating hyperplane, even if one exists. If one were to assume that the ham $\langle 0.72, 23 \rangle$ were an outlier — perhaps a mistake in the training data — one might reasonably choose the vertical separator in Figure 10.12, which reflects the assumption that the second feature has no real effect. Hindsight (i.e., Figure 10.10) tells us intuitively that our original separator was more appropriate. However, we are concerned here with justifying the choice based on the training data alone. Figures 10.9 and 10.12 represent two competing views of what constitutes the best classifier:

- One that correctly classifies all training examples while maximizing the distance from the nearest example to the hyperplane (Figure 10.9)
- One that allows one or more examples to be misclassified while increasing the distance to the rest (Figure 10.12).

10.4.1 Perceptron Algorithm

The perceptron algorithm (Figure 10.13) iteratively finds a separating hyperplane — any separating hyperplane, if one exists — starting with a weight vector β that is incremented or decremented for every example on the wrong side of the hyperplane specified by β. The algorithm ignores correctly classified examples. If the examples are linearly separable, the algorithm converges in a finite number of steps; otherwise it fails to terminate. For practical purposes it is sufficient to stop training after some time, under the assumption that a good, if not optimal in any sense, classifier has been found. The perceptron is attractive for filtering because it is simple, incremental, and adaptive.

The *margin perceptron* algorithm increments β for examples that are near but on the correct side of the hyperplane, and also for examples that are on the wrong side. *Margin* is defined to be the distance to the nearest example in Euclidean space. The margin perceptron (Figure 10.14; see Sculley et al. (2006) for details) adds a margin parameter τ so as to bias the method to prefer higher margin separators. Where the standard perceptron would stop, the margin perceptron

Inputs:

Set $T \subset D$ of training examples $d \in T$, represented by $x^{[d]} = \langle 1, x_1^{[d]}, x_2^{[d]}, \ldots, x_n^{[d]} \rangle$
Labeling $label : T \rightarrow \{-1, 1\}$
Margin parameter τ

Output:

if linearly separable, β such that $\beta \cdot x^{[d]} \geq \tau$ if $label(d) = 1$, $\beta \cdot x^{[d]} \leq -\tau$ if $label\langle d) = -1$
else fails to terminate

```
1  β ←⟨0, ..., 0⟩
2  while ∃_{d∈T} : β · x^[d] · label⟨d) < τ do
3      β ← β + x^[d] · label(d)
```

Figure 10.14 Margin perceptron.

continues to adjust the hyperplane until a margin of $\frac{\tau}{|\beta|}$ is achieved. Note that $|\beta|$ grows with each step, effectively reducing the margin until a suitable hyperplane is found. There is no guarantee that the largest possible margin ⟨i.e., the smallest possible $|\beta|$⟩ is found, but for sufficiently large τ the margin perceptron usually finds a reasonable approximation.

10.4.2 Support Vector Machines

A support vector machine ⟨SVM⟩ directly computes the separating hyperplane that maximizes the margin or distance to the nearest example points. Several points will be at the same distance; these points are known as the *support vectors* and the resulting classifier is a linear combination of these vectors — all other points may be ignored. Thus the SVM would prefer the solution in Figure 10.9 over the ones in Figure 10.11 with support vectors of $\langle 0, 0 \rangle$, $\langle 0.72, 23 \rangle$ on the ham side and $\langle 0.72, 20 \rangle$ on the spam side.

In the case of nonseparable data or of separable data in which a few points dramatically affect the solution ⟨e.g., Figure 10.12 or point $\langle 0.72, 23 \rangle$ in our training data⟩, it may be desirable to relax the requirement that all training data be correctly classified. Contemporary SVM formulations implement a trade-off between maximizing the margin and mimimizing the magnitude of training errors. The trade-off parameter C determines the relative weight of the second objective to the first. $C = 0$ specifies the pure SVM detailed in the previous paragraph; $C = 1$ gives the objectives balanced weight and is a typical default value; $C = 100$ gives the second objective substantial weight and has been found to be appropriate for spam filtering ⟨see Drucker et al. ⟨1999⟩ or Sculley et al. ⟨2006⟩⟩.

A number of software packages implementing SVMs are available, including Weka,[6] SVM-light,[7] and LibSVM[8]. Sculley and Wachman ⟨2007⟩ describe a gradient method for efficient incremental on-line filtering using SVMs.

[6] `www.cs.waikato.ac.nz/ml/weka`

[7] `svmlight.joachims.org`

[8] `www.csie.ntu.edu.tw/~cjlin/libsvm`

10.5 Similarity-Based Classifiers

In this section we consider similarity-based classifiers, which harness the assumption that similar documents are more likely to belong to the same category than dissimilar ones. For classification it is necessary to formalize the notion of similarity as a function $sim : D \times D \rightarrow \mathbb{R}$, where $sim(d_1, d_2) > sim(d_3, d_4)$ means that in some sense d_1 and d_2 are more similar to one another than are d_3 and d_4. Perhaps the most familiar example is the vector space model for information retrieval from Chapter 2, where sim is the cosine formula (Equation 2.12):

$$sim(d_1, d_2) = \frac{|x^{[d_1]} \cdot x^{[d_2]}|}{|x^{[d_1]}| \cdot |x^{[d_2]}|} . \tag{10.51}$$

For topic-oriented filtering without historical examples, documents may be prioritized by similarity to the query q, exactly as for ranked retrieval, yielding a soft classifier:

$$c(d) = sim(d, q) . \tag{10.52}$$

If historical examples are available, we may compute the similarity of d to any or all of them, and combine the resulting evidence to classify d. We formalize this approach by defining a new similarity function $Sim : D \times 2^D \rightarrow \mathbb{R}$, where $Sim(d, D')$ indicates the similarity of d to the documents in a set $D' \subset D$. Various similarity-based classifiers are distinguished by how they define sim and by how Sim is derived from sim.

10.5.1 Rocchio's Method

Rocchio's method (Rocchio, 1971) defines

$$Sim(d, D') = sim(d, d'), \quad \text{where } x^{[d']} = \frac{1}{|D'|} \sum_{d \in D'} x^{[d]} . \tag{10.53}$$

That is, D' is represented by a hypothetical surrogate document d' whose feature vector is the centroid of all members of D'. In its simplest form the Rocchio classifier uses only positive training examples:

$$c(d) = Sim(d, T \cap P) . \tag{10.54}$$

For categorization, where positive and negative examples are available, a better classifier is derived from the difference

$$c(d) = Sim(d, T \cap P) - Sim(d, T \cap \overline{P}) . \tag{10.55}$$

Rocchio's method has been used extensively for relevance feedback in the vector space model. Like the cosine measure on which it is based, Rocchio's method is essentially obsolete for this purpose; methods like BM25 relevance feedback work better. It is not difficult to see that

Rocchio's method is in fact a linear classifier, but it is not as effective as the other classifiers presented here.

Our application of BM25 to language categorization and spam filtering (Section 10.1) may be considered a variant of Rocchio's method in which *Sim* is defined in terms of the BM25 relevance feedback formula instead of the cosine measure.

10.5.2 Memory-Based Methods

Memory-based methods, also known as *case-based* methods, use the training examples themselves as the classifier profile. These examples are searched as necessary to classify a document or set of documents. Perhaps the simplest memory-based method is *nearest neighbor* (NN), which is traditionally taken to be a hard classifier:

$$c_h(d) = label\left(\arg\max_{d' \in T} sim(d, d')\right). \tag{10.56}$$

If *sim* happens to be one of the ranking methods we have considered for standard search tasks, the nearest neighbor classifier may be implemented efficiently by indexing the training examples and using a search engine to retrieve the most similar document. Otherwise it may be necessary to enumerate $sim(d, d')$ for every $d' \in T$.

More generally, we may compute a soft nearest neighbor classifier and derive the hard classifier from it:

$$Sim(d, D') = \max_{d' \in D'} sim(d, d') \tag{10.57}$$

$$c_s(d) = Sim(d, T \cap P) - Sim(d, T \cap \overline{P}) \tag{10.58}$$

$$c_h(d) = \begin{cases} pos & (c_s(d) > 0) \\ neg & (c_s(d) \leq 0) \end{cases}. \tag{10.59}$$

An efficient implementation of c_s involves the construction of separate search indices for T_{pos} and T_{neg}.

A simple variant is *k-nearest neighbor* (kNN), in which the k documents most similar to d are considered, for some fixed k. Most commonly a hard classifier is then defined as the majority vote among the most similar k documents.

10.6 Generalized Linear Models

As we have noted previously, there is no firm distinction between feature engineering and classifier construction. Through a suitable choice of the feature representation $x^{[d]}$ one can

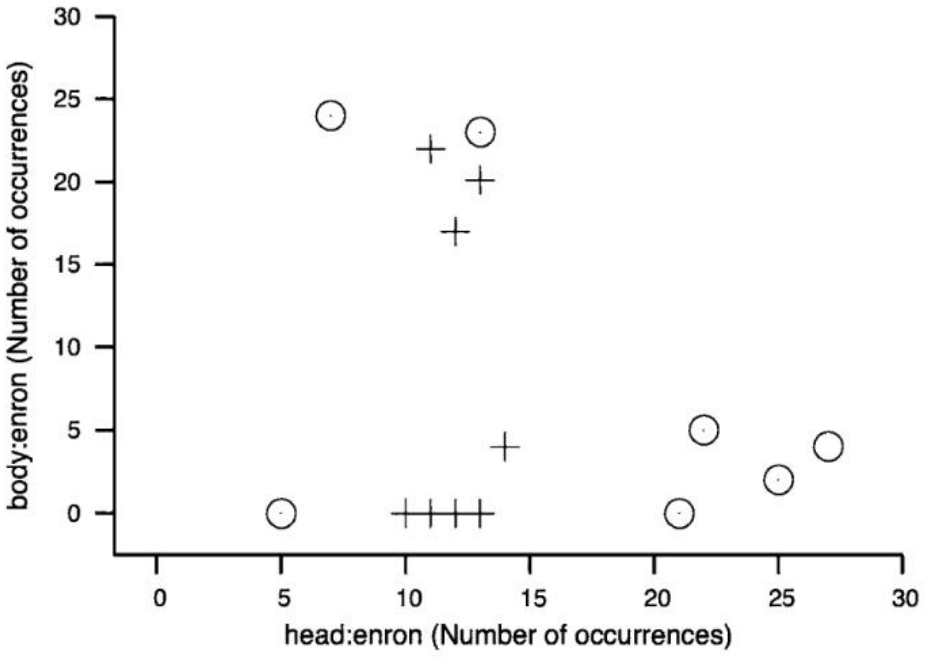

Figure 10.15 Untransformed features.

transform essentially any classification problem into one that may be solved by a linear classifier. The examples we have presented so far rely on such feature engineering:

- For probabilistic classifiers we applied the logit transformation, known as a *link function* or *transfer function*, to make the problem amenable to linear classification.
- For our linear classification example we applied a Gaussian transformation to one of the dimensions.

Figure 10.15 shows the linear classification example without transformation. The feature representation is the raw term frequency for both dimensions. In this exposition we distinguish raw and transformed representations using the following notation:

- $x^{[d]}$ is the *raw* representation of d. For this example $x^{[d]}$ is the vector of the frequencies of the two terms. In general, $x^{[d]}$ is a representation that is in some sense a straightforward representation of d.
- $X^{[d]} = \varphi(x^{[d]})$ is the *transformed* representation of d, where φ is a mapping function. X is a vector space whose dimensionality need not be the same as x. It may be smaller, in which case the transformation is an example of *dimensionality reduction*. Or it may be considerably larger, even infinite. In this second situation *kernel methods* may be used to construct linear classifiers over X without having to calculate $X^{[d]}$. Perceptrons and SVMs are readily implemented as kernel methods.

Using Equation 10.34, the mapping for our example is

$$X^{[d]} = \varphi(x^{[d]}) = \left\langle \frac{N_P}{N_{\overline{P}}} \cdot \frac{g(\mu_{(1,P)}, \sigma_{(1,P)}, x_1^{[d]})}{g(\mu_{(1,\overline{P})}, \sigma_{(1,\overline{P})}, x_1^{[d]})}, x_2^{[d]} \right\rangle . \tag{10.60}$$

Note that the four parameters $\mu_{(1,P)}, \sigma_{(1,P)}, \mu_{(1,\overline{P})}, \sigma_{(1,\overline{P})}$ are estimated from the training examples as part of the feature engineering process. An alternative is to guess several values for these parameters and to see which one works best, a process known as *parameter selection* or, more colloquially, *tuning*. One may guess φ as well; the overall process of guessing φ and its parameters is known as *model selection*. In evaluating classifiers whose construction involves parameter or model selection, it is essential that the evaluation (i.e., test) data not be consulted. A common approach is to subdivide the training examples into separate training and *validation* sets (as detailed in Chapter 11). For our immediate purpose we assume that φ (including any implicit parameters on which it depends) is fixed.

For expository purposes we consider a second mapping φ^+ on the same data that augments the dimensionality of x, to form a three-dimensional space:

$$\varphi^+(x^{[d]}) = \left\langle x_1^{[d]}, x_2^{[d]}, \frac{N_P}{N_{\overline{P}}} \cdot \frac{g(\mu_{(1,P)}, \sigma_{(1,P)}, x_1^{[d]})}{g(\mu_{(1,\overline{P})}, \sigma_{(1,\overline{P})}, x_1^{[d]})} \right\rangle . \tag{10.61}$$

This mapping may be visualized as lifting the positive examples from the plane, along the new third dimension, while depressing the negative examples. In this new three-dimensional space the examples are linearly separable. Furthermore, the first dimension is unnecessary because the examples are still linearly separable when we reduce the space to two dimensions by eliminating it:

$$\varphi^-(X^{[d]}) = \left\langle X_3^{[d]}, X_2^{[d]} \right\rangle \tag{10.62}$$

Our original mapping φ is simply the composition of φ^+ and φ^-:

$$\varphi(x^{[d]}) = \varphi^-(\varphi^+(x^{[d]})) . \tag{10.63}$$

10.6.1 Kernel Methods

A linear classifier may be reformulated as a similarity-based classifier where

$$sim(d_1, d_2) = X^{[d_1]} \cdot X^{[d_2]} . \tag{10.64}$$

Consider the case in which the feature space x is linearly separable. The perceptron algorithm may be used to compute a weight vector β such that

$$c(d) = \beta \cdot X^{[d]} \qquad c(d \in P) > 0 \qquad c(d \in \overline{P}) < 0 . \tag{10.65}$$

Inputs:

Set $T \subset D$ of training examples $d \in T$, represented arbitrarily
Labeling $label : T \rightarrow \{-1, 1\}$
Similarity (kernel) function $sim : D \times D \rightarrow \mathbb{R}$,
where $sim(d_1, d_2) = X^{[d_1]} \cdot X^{[d_2]}$ in virtual feature space X

Output:

if linearly separable,
α such that $c(d) > 0$ if and only if $label(d) = 1$, where $c(d) = \sum_{d' \in T} \alpha_{d'}\ sim(d', d)$
else fails to terminate

```
1  α ← ⟨0, ..., 0⟩
2  while ∃_{d∈T} : c(d) · label(d) < 0 do
3      α_d ← α_d + label(d)
```

Figure 10.16 Kernel perceptron learning algorithm.

From Figure 10.13 we see that β must be a linear combination of positive and negative training examples in which the positive and negative examples have nonnegative and nonpositive coefficients, respectively:

$$\beta = \sum_{d \in T} \alpha_i X^{[d]} \qquad \alpha_{d \in P} \geq 0 \qquad \alpha_{d \in \overline{P}} \leq 0\,. \tag{10.66}$$

Here α is a vector of weights, one for each document in T. Combining Equations 10.64, 10.65, and 10.66, we have the *dual formulation* of the classifier:

$$c(d) \;=\; (\sum_{d' \in T} \alpha_{d'} \cdot x^{[d']}) \cdot x^{[d]} \;=\; \sum_{d' \in T} \alpha_{d'} \cdot (x^{[d']} \cdot x^{[d]}) \;=\; \sum_{d' \in T} \alpha_{d'} \cdot sim(d, d')\,. \tag{10.67}$$

This dual formulation does not reference $X^{[d]}$ except within the definition of *sim*. The *kernel perceptron* algorithm (figure 10.16) uses this representation and computes α instead of β, replacing the update rule

$$\beta \leftarrow \beta + X^{[d]} \cdot label(d) \quad \text{with} \quad \alpha_d \leftarrow \alpha_d + label(d)\,. \tag{10.68}$$

Thus the kernel perceptron manipulates *sim* and α, but never $X^{[d]}$ or β. Other methods, including the margin perceptron and SVM, may be formulated as kernel methods.

The so-called *kernel trick* is to implement $sim(d_1, d_2)$ without reference to X, thus avoiding its computation altogether. So long as Equation 10.64 holds, any implementation will do. Such an implementation is called a *kernel function*. The kernel trick enables us to use a vast or unlimited number of virtual features, which would not otherwise be possible.

Consider, for example, a set of documents $d \in D$ whose raw feature representation $x^{[d]}$ consists of n term frequencies. Rather than using the traditional cosine rule that directly yields a linear classifier, let us define similarity to be the number of terms that two documents share with

identical frequencies:

$$sim(d_1, d_2) = |\{i \in [1, n] \,|\, x_i^{[d_1]} = x_i^{[d_2]}\}| . \tag{10.69}$$

In general there is an unbounded number of possible values for $f = x_i^{[d]}$, but we may enumerate them by defining a mapping π from combinations of (f, i) to $\mathbb{N}$:

$$\pi(f, i) = i + n \cdot f . \tag{10.70}$$

Given π, the virtual feature representation $X^{[d]}$ is defined by

$$X_{\pi(f,i)}^{[d]} = \begin{cases} 1 & (x_i^{[d]} = f) \\ 0 & (x_i^{[d]} \neq f) \end{cases} . \tag{10.71}$$

In this space we have $sim(d_1, d_2) = X^{[d_1]} \cdot X^{[d_2]}$, as required. But *sim* is implemented by enumerating $x^{[d_1]}$ and $x^{[d_2]}$, not $X^{[d_1]}$ and $X^{[d_2]}$.

The design of feature spaces and kernel functions is limited only by the imagination. Standard examples include string kernels, polynomial kernels, and radial basis function (Gaussian) kernels. SVM packages commonly support a variety of kernels, including user-defined kernels in which *sim* is specified in a programming language.

10.7 Information-Theoretic Models

A *model* is an estimate of the probability of some event derived from available evidence. A meteorologist might use evidence such as humidity and barometric pressure to predict an 80% chance of rain tomorrow. In this example the event is rain and the meteorologist's model estimates $\Pr[\text{rain}] \approx 0.8$ and $\Pr[\text{not rain}] \approx 0.2$.

In Chapter 6 we considered several models for data compression under the simplifying assumption that the model $\mathcal{M}$ predicting each symbol s from some alphabet $\mathcal{S}$ was exact:

$$\forall_{s \in \mathcal{S}} \Pr[s_i] = \mathcal{M}(s_i) . \tag{10.72}$$

Here we make the opposite assumption: that any real model $\mathcal{M}$ is necessarily inexact and that $\mathcal{M}$ is better than some other model $\mathcal{M}'$ if it is closer to the true probability. The task of determining which of $\mathcal{M}$ or $\mathcal{M}'$ is better presents two challenges:

- The *true probability* is intangible, and cannot be used as a gold standard for comparison.
- The notion of "closer to" has many possible interpretations.

Assuming these challenges are met and we have a method to determine which is the better model, we can construct a classifier in one of two ways: