11 Fusion and Metalearning

In the preceding chapters we have considered a number of core methods for search, categorization, and filtering. Many of these methods have parameters — such as BM25's k_1, v_{title}, v_{body}, b_{title}, and b_{body} — that alter their effectiveness. When one considers all the possible parameter settings of the core methods, along with other design choices such as tokenization and feature selection, one is left with a near-infinite space of methods from which to choose.

This vast space gives rise to an obvious question: Which combination of choices works best? Surely, if we knew the answer to this question, we could fix those choices to come up with the best method, and dispense with the rest. Perhaps surprisingly, it is often possible to improve on the best single method by combining the results of several, without necessarily identifying which is best.

The specific approaches we shall examine are the following:

- *Fusion* or *aggregation*, in which the results returned by several IR methods are combined into one (Section 11.1).
- *Stacking*, in which the best combination of several classifiers is learned, forming a meta-classifier. Section 11.2 considers stacking for adaptive filtering, and Section 11.3 considers stacking for nonadaptive filtering and categorization.
- *Bagging*, or *bootstrap aggregation*, in which the training examples are randomly resampled to generate a set of classifiers whose results are averaged (Section 11.4).
- *Boosting*, in which the training examples are progressively weighted to emphasize misclassified examples thus generating a set of classifiers whose results are combined as a weighted average (Section 11.5).
- *Multicategory ranking* and *categorization*, in which the results of many binary classifiers are combined (Section 11.6).
- *Learning to rank*, in which a ranking function is learned from distinct example rankings (Section 11.7).

Before considering specific illustrative examples, we offer a few general comments. All of the methods considered here aggregate evidence from separate methods into stronger evidence pertinent to the user's information need. Although some items of evidence may be more compelling than others, an appropriate combination of all available evidence, taken as a whole, is in general more compelling than any individual item. This phenomenon is not unique to information retrieval; it is commonly known as the *wisdom of crowds* (Surowiecki, 2004).

The evidence we shall consider takes one of three general forms:

1. *Categorical.* A discrete result indicating that a document or set of documents is relevant or belongs to a particular category. The problem of determining a categorical result is commonly known as *classification.*
2. *Ordinal.* A ranking or score that orders documents by the weight of evidence that they are relevant or belong to some category. The problem of determining an ordinal result is commonly known as *ranking.*
3. *Quantitative.* A calibrated score, typically a probability, indicating the weight of evidence that a document is relevant or belongs to some category. The problem of estimating a calibrated score is commonly known as *regression.*

Regression may be reduced to ranking through the use of a sort, and ranking to classification through the use of a cutoff rank or threshold. Each of these steps loses information. Sometimes the reduction is inherent in the method and therefore unavoidable. The more these losses are avoided, the more compelling the overall weight of evidence.

To illustrate the various methods of combining evidence, we consider the problems that have been addressed in previous chapters: ranked retrieval, topic-oriented filtering, language categorization, and on-line spam filtering. For each problem we take as input the results of the individual methods that we have evaluated for each purpose, and compare the aggregate result against the best individual method. In addition we illustrate learning-to-rank methods using the LETOR 3 data set,[1] a standard benchmark derived from several TREC corpora.

11.1 Search-Result Fusion

The simplest approach to combining evidence that we shall consider is *search-result fusion*, in which the lists of documents returned by several search engines are combined into one. No information other than the lists, and perhaps the ranks or scores of documents within the lists, is used. We investigate three variants of this approach: (1) fixed-cutoff retrieval, in which a fixed number of documents from each list is considered without regard to rank; (2) rank aggregation, in which the rank of each document within each list is considered; (3) score aggregation, in which the rank and score of each document within each list is considered.

To describe search-result fusion, we introduce the following notation. Consider a set of n systems returning document lists $Res_1, Res_2, \ldots, Res_n$ from collection D in response to some query q. We wish to construct a better list Res that aggregates the evidence of the n lists. For each Res_i we define the ranking function $r_i(d)$ to be the position of d in Res_i, or some large number c if d is not in Res_i:

[1] `research.microsoft.com/en-us/um/beijing/projects/letor`

Table 11.1 29 separate runs used for the fusion and learning-to-rank examples in this chapter. Bold numbers indicate the best run(s) in each column.

			TREC45				GOV2			
			1998		1999		2004		2005	
Method	**Stop**	**Stem**	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**
bm25	No	No	0.424	0.178	0.440	0.205	0.471	0.242	0.534	0.277
bm25	No	Yes	0.440	0.199	**0.464**	0.247	**0.500**	0.266	0.600	0.334
bm25	Yes	No	0.424	0.178	0.438	0.205	0.467	0.243	0.538	0.276
bm25	Yes	Yes	0.440	0.199	**0.464**	0.247	**0.500**	0.266	0.592	0.333
bm25_b=0	No	No	0.402	0.177	0.406	0.207	0.418	0.171	0.538	0.207
bm25_notf_b=0	No	No	0.256	0.141	0.224	0.147	0.069	0.050	0.106	0.083
dfr-gb2	No	No	0.426	0.183	0.446	0.216	0.465	0.248	0.550	0.269
dfr-gb2	No	Yes	0.448	**0.204**	0.458	0.253	0.471	0.252	0.584	0.319
dfr-gb2	Yes	No	0.426	0.183	0.446	0.216	0.465	0.248	0.550	0.269
dfr-gb2	Yes	Yes	0.448	**0.204**	0.458	0.253	0.471	0.252	0.584	0.319
lm-dirichlet-1000	No	No	0.450	0.193	0.428	0.226	0.484	0.244	0.580	0.293
lm-dirichlet-1000	No	Yes	**0.464**	**0.204**	0.434	0.262	0.492	0.270	0.600	**0.343**
lm-dirichlet-1000	Yes	No	0.448	0.193	0.430	0.226	0.494	0.247	0.568	0.291
lm-dirichlet-1000	Yes	Yes	0.462	**0.204**	0.436	**0.262**	0.488	**0.272**	**0.602**	0.341
lm-jelinek-0.5	No	No	0.390	0.179	0.432	0.208	0.416	0.211	0.494	0.257
lm-jelinek-0.5	No	Yes	0.406	0.192	0.434	0.248	0.437	0.225	0.522	0.302
lm-jelinek-0.5	Yes	No	0.390	0.179	0.432	0.209	0.414	0.212	0.482	0.253
lm-jelinek-0.5	Yes	Yes	0.406	0.192	0.436	0.249	0.445	0.225	0.508	0.298
lm-unsmoothed	No	No	0.354	0.114	0.396	0.141	0.402	0.171	0.492	0.231
lm-unsmoothed	No	Yes	0.384	0.134	0.416	0.180	0.433	0.196	0.538	0.285
lm-unsmoothed	Yes	No	0.352	0.114	0.394	0.141	0.400	0.172	0.484	0.230
lm-unsmoothed	Yes	Yes	0.384	0.134	0.416	0.180	0.439	0.196	0.518	0.283
prox	No	No	0.396	0.124	0.370	0.146	0.424	0.173	0.560	0.230
prox	No	Yes	0.418	0.139	0.430	0.184	0.453	0.207	0.576	0.283
prox	Yes	No	0.396	0.123	0.370	0.146	0.422	0.173	0.546	0.232
prox	Yes	Yes	0.416	0.139	0.430	0.184	0.447	0.204	0.556	0.282
vsm-lintf-logidf	No	No	0.266	0.106	0.240	0.120	0.298	0.092	0.282	0.097
vsm-logtf-logidf	No	No	0.264	0.126	0.252	0.135	0.120	0.060	0.194	0.092
vsm-logtf-noidf-raw	No	No	0.342	0.132	0.328	0.154	0.400	0.144	0.466	0.151

$$r_i(d) = \begin{cases} k & (Res_i[k] = d) \\ c & (d \notin Res_i) \end{cases} . \tag{11.1}$$

The choice of c is arbitrary, so long as $c > |Res_i|$. Reasonable choices are $c = \infty$, $c = |D|$, $c = \frac{|D|}{2}$ and $c = |Res_i| + 1$. For our examples we use $c = \infty$.

For score aggregation we assume that each system returns a nonincreasing list $Score_i$, where $Score_i[k]$ is the relevance score associated with $Res_i[k]$. For each $Score_i$, we define the scoring function $s_i(d)$ to be the score of d in Rel_i, or the smallest score if d is not in Rel_i:

$$s_i(d) = \begin{cases} Score_i[k] & (Res_i[k] = d) \\ Score_i[|Score_i|] & (d \notin Res_i) \end{cases} . \tag{11.2}$$

Typically, we assume that $Score_i$ is normalized so that

$$Score_i[|Score_i|] = \min_d\{s_i(d)\} = 0, \qquad Score_i[1] = \max_d\{s_i(d)\} = 1. \tag{11.3}$$

We illustrate result fusion by applying three methods to the results of $n = 29$ variants of the search methods we have considered in previous chapters, as listed in Table 11.1. For each method the table shows P@10 and MAP results over each of the four TREC collections, with the best result indicated in bold font.

11.1.1 Fixed-Cutoff Aggregation

Consider the problem of combining the results in Table 11.1 so as to optimize P@10. For now we shall consider each of the results to be categorical, consisting of a set of 10 documents per topic in no particular order. For our set of 29 example methods, the union of these 29 sets contains somewhere between 10 and 290 documents per topic, from which we wish to identify the 10 most likely to be relevant; that is, to maximize P@10.

An obvious, if hypothetical, approach is to identify exactly those ten documents returned by the "best" system. But which is the best? Perhaps, based on previous evaluation, we are able to guess that BM25 or LMD is the most effective. There is no single method or set of parameter settings that is best for all four corpora. The reader has the luxury of consulting the table as an oracle, so as to come up with an idealized result for comparison: the best individual result. It is important to note that the "best individual result" is an upper bound on the effectiveness that can be achieved by selecting one result; it is an upper bound that cannot reliably be achieved in practice.

A simple and practical approach is to use an "election" strategy to select ten documents from among those "nominated" by the various methods. The study of election strategies is remarkably nuanced, but for now we will simply select the 10 documents that occur most frequently among the 29 results, breaking ties arbitrarily. In effect we are combining several categorical results

Table 11.2 P@10 categorical fusion results for retrieval methods and TREC collections discussed in this book.

	TREC45		GOV2	
	1998	1999	2004	2005
Method	P@10	P@10	P@10	P@10
Best	0.464	0.464	0.500	0.602
Median	0.406	0.430	0.445	0.538
Election	0.452	0.460	0.492	0.554

into a ranked result, and then applying a cutoff (10 documents) to arrive at a new categorical result. The result of our effort is shown as "election" in Table 11.2. For each corpus our simple voting strategy approaches, but does not equal or exceed, the idealized best result.

11.1.2 Rank and Score Aggregation

Consider now the problem of optimizing MAP over the same four collections. Each of the results Res_i is a list from which a ranking r_i is derived. These rankings are combined to yield an aggregate ranking r. The approach is known as *rank aggregation*, *fusion*, or *metasearch*.

Perhaps the simplest and most effective rank-aggregation strategy is *reciprocal rank fusion* (RRF; Cormack et al., 2009). RRF simply sorts the documents according to a naïve scoring formula

$$RRFscore(d) = \sum_i \frac{1}{k + r_i(d)}, \tag{11.4}$$

where $k = 60$ has been found to work well for typical search results. The intuition in choosing this formula is that although highly ranked documents are more important, the importance of lower-ranked documents does not vanish. The harmonic series has these properties. The constant k mitigates the impact of high rankings by rogue methods.

More considered (but not necessarily more effective) methods are based on the theory of elections in which voters express preferences instead of categorical choices. Condorcet fusion (Montague and Aslam, 2002) combines rankings by sorting the documents according to the pairwise relation $r(d_1) < r(d_2)$, which is determined for each (d_1, d_2) by majority vote among the input rankings.

Some other approaches are not pure rank-aggregation methods in that they use score (which is quantitative, albeit arbitrarily calibrated, evidence) in addition to rank. CombMNZ (Lee, 1997), for example, sorts the documents according to the scoring formula

$$CMNZscore(d) = |\{i \mid d \in Res_i\}| \cdot \sum_i s_i(d). \tag{11.5}$$

Table 11.3 Rank-based fusion results for retrieval methods and TREC collections discussed in this book.

	TREC45				GOV2			
	1998		1999		2004		2005	
Method	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**
Best (by MAP)	0.462	0.204	0.434	0.262	0.500	0.272	0.602	0.341
Best (by P@10)	0.464	0.204	0.464	0.247	0.500	0.266	0.602	0.341
RRF (k=60)	0.462	0.215	0.464	0.252	0.543	0.297	0.570	0.352
Condorcet	0.446	0.207	0.462	0.234	0.525	0.281	0.574	0.325
CombMNZ	0.448	0.201	0.448	0.245	0.561	0.270	0.570	0.318

That is, *CMNZscore* multiplies the number of results in which the document occurs by the sum score, which yields a combined score that determines the ranking. Table 11.3 shows the results of these three methods when used to fuse the results of 29 IR methods. For three of the four corpora, RRF improves on the best MAP score. Condorcet also improves, but not by as large a margin. CombMNZ, although competitive with the best MAP scores, does not substantially better them. For comparison with our previous results, we also report P@10, computed by applying a cutoff of 10 to our combined ranking. The P@10 score derived from RRF in all cases exceeds that derived from voting, and is comparable to the best P@10.

Overall, rank and score aggregation methods are valuable because they match and often improve on the best ranking without any knowledge of the relative quality of the individual results. They may be used in conjunction with other approaches described in this chapter, such as *bootstrap aggregation* (Section 11.4).

11.2 Stacking Adaptive Filters

Consider again the problems of combining the categorical or quantitative results of on-line spam filters presented in Table 10.25 (page 369). The categorical result is simply an indication of whether each message, in turn, is spam or ham. We may use a simple majority vote to combine the results; because there is an odd number of filters, we need not concern ourselves with ties. In effect, the vote yields a ranking that is converted to a categorical result by comparing it against a threshold t. We can consider t of the five results to be necessary to label the message as spam. Majority vote performs about as well as the best individual filter (see Table 11.4). In practical terms, $t = 4$ or $t = 5$ might yield more satisfactory results for the intended purpose — filtering spam.

Rank aggregation is not applicable per se to an on-line filter because it requires that we sort the scores. Instead we calibrate the score s_i of document d_i, using past examples to approximate

Table 11.4 Combining on-line spam filters through voting.

Method	Classification Error (%)		
	False Positive Rate	**False Negative Rate**	**Logistic Average**
Best	0.32	0.42	0.37
Majority vote	0.42	0.34	0.37
1 of 5	32.88	0.00	0.61
2 of 5	1.54	0.02	0.54
3 of 5	0.42	0.34	0.37
4 of 5	0.19	0.61	0.34
5 of 5	0.08	1.50	0.36

Table 11.5 Combining on-line spam filters through log-odds averaging and regression.

Method	Classification Error (%)			
	False Positive Rate	**False Negative Rate**	**Logistic Average**	**1-AUC**
Best (by AUC)	0.41	0.47	0.44	0.012
Majority vote	0.42	0.34	0.37	0.095
NB stacking	0.37	0.34	0.36	0.008
LR stacking	0.39	0.27	0.33	0.006

the log-odds that d_i is spam:

$$\text{logOdds}[d_i \in spam] \approx \log \frac{|\{d_{j<i} \in spam \mid s(d_j) \leq s(d_i)\}|}{|\{d_{j<i} \notin spam \mid s(d_j) \geq s(d_i)\}|}. \tag{11.6}$$

These scores may be averaged in the same way as the features of a probabilistic classifier. The use of a (meta-)learning method to combine the results is known as *stacking*. For log-odds averaging, the metalearning method is equivalent to naïve Bayes (Section 10.3.2). Other adaptive metalearning methods, such as gradient descent logistic regression, may be used instead.

Table 11.5 shows spam filtering results for stacking using naïve Bayes and logistic regression metaclassifiers applied to log-odds transformed results. Both naïve Bayes and logistic regression lead to a considerable improvement over the best individual filter.

Now consider the results for topic-oriented filtering from Table 10.23 (page 368). Adaptive stacking methods may be applied directly to the four methods for which adaptive training results are shown. These methods were in fact trained sequentially on the entire data set, but the results are reported only for the test examples. We apply the same methodology as for spam filter stacking: Over the entire data set the scores are converted incrementally to log-odds and either averaged or combined using adaptive logistic regression. The results are reported only for the test examples. Table 11.6 shows that the effect of stacking is consistent

Table 11.6 Stacking results for TREC topic 383, adaptive training. The numbers shown in this table may be compared with the "Adaptive Training" part of Table 10.23 (page 368).

	Historical + Online	
Method	**P@10**	**AP**
Best individual	1.0	0.55
NB stacking	1.0	0.59
LR stacking	1.0	0.60

with that for spam filtering: Both methods show substantial improvement over the best, with logistic regression somewhat better. All but one of the methods whose results are combined are substantially inferior to the best. Nonetheless, they offer evidence that, when harnessed using stacking, improves on the best.

11.3 Stacking Batch Classifiers

Stacking the results of batch-trained filters is problematic because it is difficult to calibrate the scores returned by these filters in terms of log-odds or any other meaningful quantity. Even methods such as logistic regression that directly estimate log-odds, do so only for the training set T. Due to overfitting, it is not at all obvious that these estimates generalize to the test set. Methods such as SVM and decision trees are in even worse shape: It is not at all uncommon for these methods to achieve perfect scores on the training examples; these scores may bear little resemblance to scores on the test examples.

Subject to these caveats, we adapt Equation 11.6 to calibrate the score s_i of document $d_i \in D$ based on the scores of training documents $d_j \in T$:

$$\text{logOdds}[d_i \in spam] \approx \log \frac{|\{d_j \in (T \cap spam) \mid s(d_j) \leq s(d_i)\}|}{|\{d_j \in (T \backslash spam) \mid s(d_j) \geq s(d_i)\}|}. \tag{11.7}$$

Table 11.7 shows the result of stacking the batch training results for all nine methods from Table 10.23. The overall effect is similar to that in our previous examples: Naïve Bayes stacking affords substantial improvement; (batch) logistic regression yields further improvement.

11.3.1 Holdout Validation

Our estimation of log-odds from training results is an example of regression — estimation of a quantitative result from available data. In particular we are estimating the performance of each

Table 11.7 Stacking results for TREC topic 383, batch training. The numbers shown in this table may be compared with the "Historical Training" part of Table 10.23 (page 368).

	Historical Training	
Method	**P@10**	**AP**
Best individual	1.0	0.56
NB stacking	1.0	0.62
LR stacking	1.0	0.65

method in terms of its classification effectiveness as a function of the score $s(d)$. These estimates allow us to construct a better combined ranking method (or classifier).

However, the technique used in the previous section estimates log-odds from $s(d)$, where $d \in T$, and is therefore subject to generalization error. This source of error may be avoided by calculating log-odds (or any other regression quantity) using $s(d)$ from a separate validation set $V \subset D$, where $T \cap V = \emptyset$. That is, the method is trained on T and then used to score (or classify or rank) V.

The most obvious way to construct V is the *holdout* approach: We partition some larger set L of labeled examples, sampled from D, such that $L = T + V$. Assuming that L is an independent and identically distributed (i.i.d.) sample of D, and T is a random subset of L, both T and V may be considered i.i.d. samples of D. Thus our log-odds calculation uses an independent sample of D and is not influenced by the classifier overfitting to T. (However, our regression may overfit to V, but that is a separate matter.) V is known as a *validation set* and, in this particular circumstance, a *holdout set.* The overall method is known as *holdout validation.*

The main limitation of holdout validation derives from the scarcity of labeled examples. In general the size of L is limited by the cost and availability of labels, so there is a direct trade-off between the sizes of T and V. In general a larger T will yield a better base method, whereas a larger V will yield better validation. Each comes at the expense of the other.

A second limitation derives from the fact that the prevalence of positive examples may be different in T and V. Some methods, particularly classification methods that depend on a threshold setting, are very sensitive to prevalence. This limitation is mitigated through the use of *stratified sampling* to create T (and hence V) from L. In stratified sampling, the examples of each category in L are sampled separately so that T (and hence V) has roughly the same proportion of each category as L.

11.3.2 Cross-Validation

Cross-validation is a regression method in which the set of labeled examples L is partitioned into k *splits* (T_1, V_1), (T_2, V_2), ..., (T_k, V_k). The learning method is trained separately on each of the T_i and is used to score each $d \in V_i$. Regression is performed on the combined results, which

Table 11.8 Stacking results for TREC topic 383, using 22-fold stratified cross-validation.

	Historical Training	
Method	**P@10**	**AP**
Best individual	1.0	0.56
NB stacking	1.0	0.64
LR stacking	1.0	0.69

are regarded as one big validation set $V = \bigcup_{i=1}^{k} V_k$. The commonest approach is *k-fold cross-validation*, in which the validation sets are of equal size and pairwise disjoint: $\forall_{i \neq j} V_i \cap V_j = \emptyset$. Each training set consists of the examples not in the validation set: $T_i = L \backslash V_i$. For large k, $T_i \approx L$ and $V = L$. Thus, V is a reasonable approximation of an independent holdout set of the same size. The extreme case of $k = |L|$ is called *leave-one-out* validation.

If stratified sampling is to be used, k is limited by the least prevalent category. In our topic-based filtering example, T contains exactly 22 relevant documents, so we use 22-fold cross-validation with each V_i containing a single relevant document. When not limited by stratification, k may be limited by efficiency considerations because the training effort is proportional to k. Five-fold cross-validation ($k = 5$) and ten-fold cross-validation ($k = 10$) are common. The results in Table 11.8 show that 22-fold cross-validation substantially improves both the average and the stacked fusion results for topic 383.

An important role of cross-validation is in *selection* or *tuning* of learning methods. In this role cross-validation is used to estimate some overall effectiveness measure, and the space of methods and parameter settings is searched to optimize this measure. As we mentioned at the outset of this chapter, and have demonstrated through examples, selecting the single best method or parameter setting does not necessarily yield the best overall result.

11.4 Bagging

When a learning method is trained on the examples from a particular training set T and applied to documents from D, the resulting categorization, ranking, or regression is subject to two kinds of error: *training error* and *generalization error*. Both would be reduced if instead of a single training set T, we were to average the results achieved from training the same method separately on N training sets $T_1, T_2, \ldots, T_N$, where each T_i is an independent sample of D and the same size as T.

The reason is quite straightforward. Let $ideal(d)$ be the score returned by an ideal classifier for a document $d \in D$. A particular method trained on T_i will compute $c_i(d) \approx ideal(d)$. More specifically, $c_i(d) = ideal(d) + E_i$, where E_i is a random variable modeling the error associated

Table 11.9 Bootstrap aggregation (bagging) for decision trees on TREC topic 383. The numbers in this table represent AP values and may be compared with those in Table 10.23 (page 368).

	Number of Bootstrap Samples (N)												
Method	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**
DT (n=2)	0.38	0.22	0.56	0.37	0.52	0.50	0.51	0.51	0.51	0.52	0.52	0.53	0.52
DT (n=8)	0.21	0.33	0.47	0.55	0.59	0.58	0.57	0.58	0.57	0.59	0.58	0.59	0.58
DT (n=256)	0.13	0.44	0.45	0.47	0.55	0.54	0.56	0.55	0.55	0.55	0.55	0.55	0.54

with training on T_i. Now suppose we average the results to form a combined result

$$c(d) \;=\; \frac{1}{N} \cdot \sum_{i=1}^{N} (ideal(d) + E_i) \;=\; ideal(d) + E\,, \tag{11.8}$$

where

$$E \;=\; \sum_{i=1}^{N} \frac{E_i}{N}\,. \tag{11.9}$$

If the E_i are pairwise independent, the variance of E is generally smaller than the variance of any particular E_i. In the particular situation where the $\sigma^2_{E_i}$ are all equal, σ^2_E is a factor of N smaller:

$$\sigma^2_E \;\approx\; \frac{\sigma^2_{E_i}}{N}\,. \tag{11.10}$$

If the T_i are in fact i.i.d. samples of D, then the E_i are independent and have equal variance, so the standard error σ_E is reduced by a factor of $\sqrt{N}$ compared with training on a single T_i.

In general, little is gained from splitting the set of labeled examples L into independent training sets, because training on one large set $T = L$ works just as well, if not better. Instead, a technique known as *the bootstrap* may be used to simulate $T_1, T_2, \ldots, T_N$, each of the same size as L (see Section 12.3.2). The bootstrap uses L as a proxy for D, selecting each member of T_i independently and at random from L. Because elements are selected independently, the same element may be repeated in T_i, and some elements may not appear at all. For any reasonable sample size ($N > 20$) of T_i, the probability that a given element $d \in L$ will occur k times is well approximated by the Poisson formula:

$$\Pr[|\{d \in T_i\}| = k] \;\approx\; \frac{1}{k! \cdot e}\,. \tag{11.11}$$

That is, about $\frac{1}{e} \approx 36.8\%$ of the documents will not appear in any given T_i, $\frac{1}{e} \approx 36.8\%$ will occur once, $\frac{1}{2e} \approx 18.4\%$ twice, and so on. These *bootstrap samples* have sufficient independence that the error variance is reduced. Provided L is large enough to contain a good representation of the sorts of documents in D, and provided the learning method treats duplicate examples as

Table 11.10 Boosting (AdaBoost) results for decision trees on TREC topic 383. The numbers in this table represent AP values and may be compared with those in Tables 10.23 (page 368) and 11.9.

	Number of Boosting Iterations (N)												
Method	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**
DT (n=2)	0.19	0.31	0.38	0.52	0.48	0.50	0.49	0.53	0.54	0.59	0.60	0.61	0.59
DT (n=8)	0.21	0.22	0.27	0.41	0.54	0.59	0.60	0.63	0.61	0.62	0.64	0.64	0.63
DT (n=256)	0.60	0.51	0.54	0.54	0.53	0.49	0.55	0.57	0.56	0.56	0.57	0.57	0.57

separate individuals that happen to have the same properties, bootstrap training sets closely resemble training sets drawn from D instead of L (Efron and Tibshirani, 1993).

The technique of *bootstrap aggregation* or *bagging* averages the results from using bootstrap samples as training sets. Although bagging may be applied to any learning technique, it is most effective and most commonly used for *unstable* learning methods whose result is highly dependent on the particular training examples. Table 11.9 shows the results of bootstrap aggregation for decision tree methods and TREC topic 383. The improvement for decision stumps ($n = 2$) is the most dramatic. Larger trees with $n = 8$ and $n = 256$ leaves show reduced effectiveness for small N but very good effectiveness for $N \geq 32$. Little is gained for $N \geq 128$ because the results appear to approach their asymptote.

11.5 Boosting

Boosting, like bagging, is a method of perturbing the training examples so as to produce an *ensemble* of methods whose results are combined. Boosting systematically manipulates the training data to minimize training error, unlike bagging, which manipulates the training data randomly to minimize generalization error. In both cases the net effect is a better classifier, although the underlying assumptions are quite different.

In boosting, the overall training set T is a random sample of D, as for any supervised learning method. But T is not used directly for training; instead, each $d \in T$ is given a weight $w_d \in \mathbb{R}$ such that $1 = \sum_d w_d$. The base learning method assumes that the prevalence ρ_d of documents like d in D is $\rho_d = w_d$, as opposed to that suggested by the proportion in T (i.e., $\rho_d \approx \frac{1}{|T|}$). For base methods such as decision trees, the assumption is easily accommodated by multiplying the contribution of each d by w_d during classifier construction.

Boosting proceeds by computing a series of k weight vectors $w^{[1]}, w^{[2]}, \ldots w^{[k]}$ and associated classifiers $c^{[1]}, c^{[2]}, \ldots c^{[k]}$ learned using T and the corresponding weights. Initially, $w_d^{[1]} = \frac{1}{|T|}$ for all $d \in T$. In subsequent steps the weight $w_d^{[i+1]}$ is increased (relative to $w_d^{[i]}$) for examples that are incorrectly classified by $c^{[i]}(d)$, normalized so that $1 = \sum_d w_d^{[i+1]}$. The overall classifier is

itself the weighted sum of the first k classifier results:

$$c(d) = \sum_{i=1}^{k} \alpha_i c^{[i]}(d), \tag{11.12}$$

where the weights α_i are chosen to minimize the overall error. Various methods of computing w and α give rise to a family of boosting methods. Perhaps the best known is AdaBoost, which is applied to decision trees; its results for TREC topic 383 are shown in Table 11.10. It has been shown that boosting bears a strong resemblance to logistic regression and other optimization-based methods (Schapire, 2003).

11.6 Multicategory Ranking and Classification

For multicategory ranking and classification problems, we require a score $s(d, q)$ that acts as a surrogate for the probability that document $d \in D$ belongs to category $q \in Q$ (i.e., $\Pr[d \in q]$). For our language categorization example (Section 10.1.4), the categories are the 60 possible languages, so each $q_j \equiv l_j$. For topic-oriented search or filtering, the categories are the separate topics, so each $q_j \equiv t_j$. We shall use language categorization as our principal motivating example, and return later to topic-oriented problems. By "acts as a surrogate for probability" we mean:

- If $s(d, q)$ is categorical, it is correct more often than chance.
- If $s(d, q)$ is ordinal, the rankings implied by it are positively correlated with probability.
- If $s(d, q)$ is quantitative, then for some known function f

$$f(s(d, q)) \approx \Pr[d \in q]. \tag{11.13}$$

Using score as a surrogate for probability is not unique to multicategory ranking. It is the crux of relevance ranking and the fusion methods we have discussed so far. The difference here is the assumption that $s(d, q_j)$ and $s(d, q_k)$ are comparable even when $q_j \neq q_k$:

$$s(d, q_j) < s(d, q_k) \Leftrightarrow \Pr[d \in q_j] < \Pr[d \in q_k]. \tag{11.14}$$

That is, $s(d, q)$ aptly ranks categories within documents, not just documents within categories.

In this section we consider several different ways to compute, and to combine, families of score functions $\{s^{[f]} : D \times Q \to \mathbb{R}\}$ for the purpose of multicategory ranking and classification.

11.6.1 Document Versus Category Scores

In our language categorization example (Section 10.1.4), for each language l we computed a scoring function $s_l(d_i)$ that is a surrogate for $\Pr[d_i \in l]$. To rank the languages that a particular document d may contain, we require $s_d(l_j)$, a surrogate for $\Pr[d \in l_j]$. The method we have used so far simply sets $s_{d_i}(l_j) = s_{l_j}(d_i)$. The assumption underlying this choice is that $s(d_i, l_j) = s_{l_j}(d_i)$ fulfills our previously stated criterion for s: that it is an apt surrogate for $\Pr[d_i \in l_j]$ over all d_i and l_j, not just for some specific language l. Under this assumption the most likely language for a document d is

$$\arg\max_{l_j} \{s(d, l_j)\}. \tag{11.15}$$

Furthermore, the possible languages l_j that d may have are ranked by $s(d, l_j)$.

For logistic regression this assumption is reasonably well justified, because the method estimates probability (as log-odds) directly. For other linear and generalized linear classifiers, the signed distance from the separating hyperplane often yields an amenable, if uncalibrated, score. Nonlinear methods such as nearest neighbor typically use some sort of internal score that may be amenable, such as the proximity to the nearest neighbor or the prevalence of l_j among the k nearest neighbors. Decision trees partition D into discrete sets: A score may be derived from the prevalence of l_j among the partition to which d belongs. A hard classifier yields an extremely coarse estimate; the average of many such estimates using bagging, boosting, or stacking is itself a quantitative estimate.

If $s(d, l)$ is not a good surrogate, it may yield a good document ranking but poor categorization accuracy. This effect may be observed by the many inversions in Table 10.24 (page 369) between the document ranking effectiveness (MAP) and categorization effectiveness (1−error or MRR). In particular, NB yields a MAP result of 0.44 and an MRR of 0.8, whereas DT (n=8) yields superior MAP (0.54) but inferior MRR (0.64). Also, DMC achieves both the best MRR (0.86) and error (20.0%) results (tied with LR), but mediocre MAP (0.74). It is perhaps not surprising, given the assumptions behind the method, that logistic regression shows superior effectiveness for both document and category ranking.

When categorical or ordinal values of $s(d, l)$ are considered, a category ranking may not be uniquely defined. That is, arg max may be ill-defined and as there may be ties in the ranking according to $s(d, l_j)$. In this event a tie breaking strategy is required. If ties are rare, random tie breaking might be acceptable. Or ties might be resolved in favor of the category known to have highest prevalence, a technique that optimizes micro-averaged (but not macro-averaged) error rate. More generally, ties may be resolved by deferring to a secondary surrogate measure $s^{[2]}(d, l)$. For the specific case of resolving by prevalence, $s^{[2]}(d, l) = prev(l)$.

The use of a secondary surrogate measure is a special case of a still more general approach: combining several $s^{[f]}$ to form an aggregate scoring function. Instances of this approach are the following:

Table 11.11 Document rank fusion results for language categorization. The numbers in this table may be compared with those in Table 10.24 (page 369).

	Dual Ranking	Categorization	
Method	**MAP**	**Error (%)**	**MRR**
Best (category ranking)	0.82	20.0	0.86
Best (document ranking)	0.83	20.0	0.80
RRF	0.83	18.9	0.87
NB stacking	0.84	19.2	0.87
LR stacking	0.82	18.6	0.87
Bagging (DT; n=8, N=1024)	0.77	25.9	0.82
Boosting (DT; n=8, N=1024)	0.75	35.7	0.69

- *Document rank fusion.* For each l, combine the results of k document scoring methods $s_l^{[1]}, s_l^{[2]}, \ldots, s_l^{[k]}$ to form an overall score $s_l(d)$; set the category scoring method $s_d(l) = s_l(d)$.
- *Category rank fusion.* For each d, combine the results of k category ranking methods $s_d^{[1]}, s_d^{[2]}, \ldots, s_d^{[k]}$ to form an overall score $s_d(l)$, which is used to rank the categories.
- *Multicategory methods.* Consider d and l in combination in order to compute $s(d, l)$.

11.6.2 Document Versus Category Rank Fusion

Table 11.11 shows the results of five methods of combining the nine language-specific document ranking methods whose results are listed in Table 10.24 (page 369). For comparison the results of the best individual methods are shown as well, as measured by categorization effectiveness and by document ranking effectiveness. RRF is applied exactly as for topic-oriented retrieval, in order to combine the nine separate document rankings into one. The score from RRF is then used to rank the languages for each document. Three of the base ranking methods (NB, DMC, and gradient descent LR) are on-line methods, so log-odds was estimated using Equation 11.6 (page 382). The other six are batch methods, so log-odds was estimated using Equation 11.7. The results were averaged to yield the result labeled "NB stacking" and combined using logistic regression on the training examples to yield the result labeled "LR stacking". Bagging and boosting were used to build ensembles of 1024 decision trees (n=8). The other document ranking methods played no role in the bagging and boosting results.

Table 11.12 shows the result of category rank fusion, which combines the various scores on a per-document basis. The fusion methods are a bit different because the languages to be ranked are not partitioned into training and test sets. Thus it is not immediately apparent how to convert the scores into probability estimates. Instead we directly apply the RRF and Condorcet rank aggregation methods, which require no estimate, as well as logistic regression to the scores

Table 11.12 Category rank fusion results for language categorization.

Method	Categorization	
	Error (%)	MRR
Reciprocal rank fusion (RRF)	27.0	0.80
Condorcet	19.5	0.87
Logistic regression (LR)	18.0	0.88

returned by the various methods. RRF performs relatively poorly, while Condorcet improves on the best base method. Logistic regression yields a further improvement. The logistic regression method is a special case of learning to rank, which is detailed in Section 11.7.

11.6.3 Multicategory Methods

A number of binary classification, ranking, or regression methods may be adapted to solve the category ranking problem directly; we call these methods *multicategory*. In the literature they are often characterized as *multinomial* or *polytomous*, as opposed to *binomial* or *dichotomous*.

One versus rest

The approach we have presented so far is first to solve the document ranking problem, then combine the results to solve the category ranking problem. This approach is generally known as *one versus rest* because it combines the results of n binary classification problems, each of which distinguishes one category from the rest. We have so far assumed that a soft classification result is an indication of confidence, and have chosen the category that is indicated with highest confidence among the n one versus rest results.

For hard binary classifiers the one versus rest strategy is problematic. It may be that exactly one of the classifiers yields a positive result, in which case the category is determined. But if no result is positive, or if many results are positive, some sort of tie breaking scheme is necessary.

One versus one

An alternative, *one versus one*, is to reduce the problem to $n(n-1)/2$ binary problems, one for each pair of categories. The $n(n-1)/2$ results are combined in an *election* to select the category for a particular document or to rank the categories. A wide variety of election strategies have been studied; a comprehensive discussion is beyond the scope of this book. Perhaps the simplest is to count the number of wins (i.e., positive results) for each category and to rank by this number. The possibility of a tie still exists, but only in the event that the binary results are inconsistent. Various *runoff* strategies can be used to resolve these ties. In many circumstances the tie may be resolved by repeating the election among only the tied categories. The confidence of soft classifiers may also be taken into account.

Multicategory logistic regression

Logistic regression is particularly easy to adapt to multicategory regression, and hence to ranking. Recall that (binomial) logistic regression estimates

$$\text{logOdds}[x] = \log\left(\frac{\Pr[x]}{\Pr[\bar{x}]}\right) = \log\left(\frac{\Pr[x]}{1-\Pr[x]}\right), \tag{11.16}$$

where x is some binary quantity of interest, and $\bar{x}$ is its complement.

In our language categorization example we used logistic regression for multicategory ranking in the following way. For each category $q_i (1 \leq i \leq n)$, let x_i denote the membership $d \in q_i$ of some document d in the category. Compute $l_i \approx \text{logOdds}[x_i]$ separately for each category using logistic regression. For the purpose of ranking, this method works reasonably well because l_i is a reasonable indication of the strength of the evidence that $d \in q_i$. It is also possible to derive a probability estimate:

$$p_i = \text{logit}^{-1}(l_i) = \frac{1}{1+e^{-l_i}} \approx \Pr[x_i], \tag{11.17}$$

but this estimate is unsatisfactory when the categories are exclusive (as they are in our language example). It fails to incorporate the constraint that the probabilities must sum to 1:

$$1 = \sum_{i=1}^{n} \Pr[x_i]. \tag{11.18}$$

Post-hoc attempts to fit this constraint yield poor estimates. We mention two solely to recommend that they be avoided. One is to normalize the p_i so that they sum to 1 as per Equation 11.18; the other is to estimate all but one of the p_i using logistic regression and then solve for the nth using Equation 11.18.

A better approach is to use logistic regression to estimate the *ratio* between pairs of probabilities

$$r_{ij} \approx \frac{\Pr[x_i]}{\Pr[x_j]} \tag{11.19}$$

and to compute p_i so as to satisfy both 11.18 and 11.19. This approach requires that we compute only $n-1$ of the $\frac{n(n-1)}{2}$ possible r_{ij} that might be estimated; the rest are redundant. The choice of which r_{ij} to compute is arbitrary, so long as they form a basis from which all r_{ij} may be derived.

A standard choice is the *baseline category logit model*. In this model we arbitrarily deem x_1 to be the baseline and compute r_{i1} for all i. For $i = 1$, by definition

$$r_{11} = 1. \tag{11.20}$$

Table 11.13 Results for one versus rest and multicategory language categorization.

	One versus rest		Multicategory	
Method	**Error (%)**	**MRR**	**Error (%)**	**MRR**
LR	20.0	0.86		
SVM (C=0.01)	20.1	0.85	29.9	0.79
SVM (C=1)	20.0	0.86	31.8	0.74
SVM (C=100)	20.0	0.86	22.8	0.84
SVM (C=10000)	20.0	0.86	19.7	0.86
Boosted stumps (N=1024)	46.4	0.63	32.2	0.76

For $i > 1$, we first note that it is possible to reformulate r_{ij} as the odds of x_i over x_j when all other categories are eliminated:

$$r_{ij} \approx \frac{\Pr[x_i]}{\Pr[x_j]} = \text{Odds}[x_i \,|\, x_i \text{ or } x_j]\,. \tag{11.21}$$

Therefore,

$$r_{\mathbf{1}j} = e^{l_{i\mathbf{1}}} \; (i > 1)\,, \tag{11.22}$$

where

$$l_{i\mathbf{1}} \approx \text{logOdds}[x_i \,|\, x_{\mathbf{1}} \text{ or } x_i]\,. \tag{11.23}$$

That is, $l_{i\mathbf{1}}$ is calculated by applying logistic regression to training documents d with categories $q_{\mathbf{1}}$ or q_j, but not the rest: $d \in (q_{\mathbf{1}} \cup q_j) \cap T$. For $i, j > 1$,

$$r_{ij} = \frac{r_{i\mathbf{1}}}{r_{\mathbf{11}}} \cdot \frac{r_{\mathbf{11}}}{r_{j\mathbf{1}}} = \frac{r_{i\mathbf{1}}}{r_{j\mathbf{1}}} \; (i, j > 1)\,. \tag{11.24}$$

Given $\{r_{i\mathbf{1}}\}$, the computation of p_i is straightforward:

$$p_i = \frac{r_{i\mathbf{1}}}{\sum_{i=\mathbf{1}}^{n} r_{i\mathbf{1}}}\,. \tag{11.25}$$

For reasons beyond the scope of this exposition, the calculation of p_i yields the same result regardless of which category $q_{\mathbf{1}}$ is chosen as the baseline (Agresti, 2007).

Multicategory SVM

Recall that a (binary) support vector machine finds a separating hyperplane between two classes, thus balancing a trade-off between the size of the margin and the number (and severity) of misclassified points. A multicategory SVM (Crammer and Singer, 2002) constructs n hyperplanes as for document ranking, but considers all margins and all misclassified points in a common optimization problem. That is, it trades off maximizing the size of the smallest of the n margins

with the overall number (and severity) of misclassified points. Thus, it more directly addresses categorization error, in contrast to logistic regression, which is concerned with the probability distribution over all classes.

Multicategory boosting

A similar approach may be applied to boosting. Instead of computing w and α separately for each category, multicategory boosting optimizes the training error over all categories (Schapire and Singer, 2000).

Table 11.13 shows the results of one versus rest and multicategory methods for TREC topic 383. Overall, it is difficult to conclude from these results that any one of them yields a startling benefit. Multicategory SVM (SVM-Multiclass[2]) is worse except for the extreme setting of C=10,000, in which case it is marginally better. Multicategory boosting (Boostexter[3]) shows a definite improvement, but over a very weak baseline.

11.7 Learning to Rank

The term *learning to rank* refers to the problem of estimating the correct order

$$\prec\colon\ R \times R$$

among a set of objects R. In contrast, classification and regression estimate some ideal function on elements of R:

$$ideal:\ R \to \{true, false\} \quad \text{or} \quad ideal:\ R \to \mathbb{R}.$$

Under this broad characterization we have already considered many examples of learning to rank. For instance, when $R \subseteq D$, we have document ranking with respect to some assumed notion of relevance; for example, binary relevance to some particular query q:

$$d_1 \prec d_2 \ =_{def}\ (d_1 \notin rel_q \text{ and } d_2 \in rel_q)\,. \tag{11.26}$$

For document ranking, the learner l takes as input a training set $T \subset D$ and a labeling function $label : T \to \{rel, non\}$, and produces a scoring function $s : D \to \mathbb{R}$. The result is recast in terms of learning to rank by defining its output to be $\precsim\colon D \times D$, which approximates the specified correct order $\prec$:

$$d_1 \precsim d_2 \ =_{def}\ s(d_1) < s(d_2)\,. \tag{11.27}$$

[2] svmlight.joachims.org/svm_multiclass.html

[3] www.cs.princeton.edu/~schapire/boostexter.html

It is important to note that $\prec$ is typically a partial order. There may be pairs of documents $d_1 \neq d_2$ such that $d_1 \not\prec d_2$ and $d_2 \not\prec d_1$. For Equation 11.26 this situation arises whenever either $d_1, d_2 \in rel_q$ or $d_1, d_2 \notin rel_q$. The estimate $\precsim$ may also be a partial order. However, ranked IR evaluation measures such as MAP and P@k assume that it is total. If it is not, because $s(d_1) = s(d_2)$ for some $d_1 \neq d_2$, evaluation software such as `trec_eval` (Section 2.3.2) arbitrarily assumes either $d_1 \prec d_2$ or $d_2 \prec d_1$.

11.7.1 What Is Learning to Rank?

In the literature the phrase "learning to rank" suggests a more general learning problem than our introductory example, in which q and D are fixed and $\prec$ is binary relevance. Although there is no commonly accepted definition of what is or is not learning to rank, the term connotes several of the following aspects:

- Metalearning, where the feature representation $x^{[d]}$ of d is the result of applying many different feature engineering and ranking formulae to q and d.
- A more general characterization of $\prec$, as opposed to binary relevance (Equation 11.26). For example, *graded relevance* assumes an ordered set of k relevance categories $rel_q^{[1]}, \ldots, rel_q^{[k]}$, where $rel_q^{[1]}$ is the least relevant to q and $rel_q^{[k]}$ is the most relevant (see Section 12.5.1). Under graded relevance,

$$d_1 \prec d_2 \;\Leftrightarrow\; d_1 \in rel_q^{[i]} \text{ and } d_2 \in rel_q^{[j]}\,(i < j)\,. \tag{11.28}$$

- The problem of learning a family of rankings, where $\prec_q$ and $\precsim_q$ are both parameterized by $q \in Q$.
- Query-limited training labels, where $d_1 \prec_q d_2$ is known only for $q \in Q_T$ and Q_T is a set of training queries, a very small subset of Q.
- Pairwise training, where $d_1 \prec_q d_2$ is known only for $(q, d_1, d_2) \subset Q_T \times D_T \times D_T$ and $D_T \subseteq D$ is a set of training documents.
- A standard example of learning to rank for IR, illustrated by the LETOR test data sets.[4] In LETOR the training set T consists of a number of triples $(q, x^{[d,q]}, r) \in Q_T \times \mathbb{R}^k \times \mathbb{Z}$, where $Q_T \subset Q$ is a set of training queries, $x^{[d,q]} = s_1(d,q), s_2(d,q), \ldots, s_k(d,q)$ is the feature representation derived by applying k different scoring functions to d with respect to q , and r represents $\prec$:

$$\forall_{q \in Q_T} : r_1 < r_2 \;\Leftrightarrow\; (q, d_1) \prec_q (q, d_2)\,. \tag{11.29}$$

[4] `research.microsoft.com/en-us/um/beijing/projects/letor`

- A standard example of using *clickthrough data* (Joachims et al., 2005) to derive pairwise, uncertain training examples. Suppose that a search engine presents a ranked list Res to the user in response to query q. Suppose also that the user inspects the kth-ranked document $d_k = Res[k]$ but does not inspect a higher-ranked document $d_j = Res[j]$, where $j < k$. From this information we infer that, on balance of probability,

$$d_j \prec_q d_k \,. \tag{11.30}$$

  This inference is uncertain in that it may be wrong, and incomplete in that $d_1 \prec_q d_2$ is defined for only a subset of $(d_1, d_2) \in D_T \times D_T$.

11.7.2 Learning-to-Rank Methods

Methods for learning to rank may be generally characterized as *pointwise*, *pairwise*, or *listwise*, depending on how they are trained (Cao et al., 2007). Pointwise methods reduce the problem to scoring each (d, q) according to the probability that document d is relevant to query q. Logistic regression directly estimates this probability (as log-odds) and maximizes the overall likelihood of the labeled examples. Other methods, such as linear regression, ranking perceptrons, and neural networks, generally give a higher score to (d, q) when d is labeled as relevant to q and a lower score when d is labeled as nonrelevant. A support vector machine seeks to find a hyperplane that separates relevant (d, q) examples from nonrelevant ones.

The pointwise SVM method illustrates a possible shortcoming in pointwise methods. Although by definition we have

$$(d_1, q_1) \not\prec (d_2, q_2) \quad (\text{for } q_1 \neq q_2)\,, \tag{11.31}$$

the SVM will attempt to separate all relevant (d_1, q_1) from nonrelevant (d_2, q_2) even when $q_1 \neq q_2$. A *ranking SVM* considers only the cases in which $q_1 = q_2$ in optimizing both margin and training error. In effect the ranking SVM builds $|Q_T|$ distinct separating hyperplanes, one for each $q \in Q_T$. Other pointwise methods may be adapted to pairwise training in a similar manner.

The listwise approach considers the ordering of the entire set $D_T \times \{q\}$ separately for each $q \in Q_T$. A possible advantage of this approach is that it is easy to place more weight on the highest-ranked documents in each D_T, to emphasize precision or any other aspect of retrieval effectiveness. On the other hand, there are $|D_T|!$ possible permutations of D_T, so finding a compact and generalizable representation of the best one presents a challenge.

11.7.3 What to Optimize?

Every learning method optimizes some objective function (minimizes some loss function) subject to a number of constraints. In learning to rank, perhaps the most obvious loss function is the

number of inversions; pairs for which the learned and ideal orders are opposite:

$$inv = |\{(d_1, q_1), (d_2, q_2) \mid (d_1, q_1) \prec (d_2, q_2) \text{ and } s(d_2, q_2) < s(d_1, q_1)\}| . \tag{11.32}$$

When $\prec$ is binary relevance, minimizing inv is equivalent to minimizing AUC, the area under the ROC curve. When $\prec$ is a total order, minimizing inv is equivalent to maximizing Kendall's τ correlation:

$$\tau = 2 \cdot (1 - inv) . \tag{11.33}$$

Minimizing inv may not maximize retrieval effectiveness, which typically emphasizes precision over recall. Measures such as MAP, P@k, and nDCG are all improved when inversions among top-ranked documents are given higher weight than the rest. In general it is not feasible to optimize these measures directly, since they are discontinuous and nonconvex. A number of methods use continuous, convex surrogate loss functions that asymptotically bound these measures.

11.7.4 Learning to Rank for Categorization

The methods we employed for category rank fusion (Section 11.6) are examples of learning to rank, albeit with the senses of D and Q reversed from the presentation above. We wish to compute $\precsim: (D \times L) \times (D \times L)$ that nearly satisfies

$$(d, l') \prec (d, l) \Leftrightarrow d \text{ has language } l \neq l' . \tag{11.34}$$

Our two measures of "nearly" are accuracy (1−error) and MRR. The features representing each (d, l) comprise

$$x^{[d,l]} = (r, s_1(d, l), s_2(d, l), \ldots s_k(d, l)) , \tag{11.35}$$

where

$$r = \begin{cases} 1 & (d \text{ has language } l) \\ 0 & (d \text{ has language } l' \neq l) \end{cases} \tag{11.36}$$

$$s_m(d, l) = \text{the score for } d \text{ and } l \text{ from base method } m . \tag{11.37}$$

In Table 11.12 the results of RRF and Condorcet are fixed rules rather than learning methods, but they compute the same result as logistic regression, a pointwise learning method.

To employ a pairwise or listwise method, it is necessary to distinguish the particular document d in the feature representation of (d, l), so that the subsets with a common d are known:

$$x^{[d_i,l]} = (r, i, s_1(d_i, l), s_2(d_i, l), \ldots, s_k(d_i, l)) . \tag{11.38}$$

Table 11.14 repeats the LR pointwise result and shows the result of ranking SVM (SVM^{light}) applied to the language category ranking problem, for two values of the regularization parameter C. For both settings, ranking SVM yields slightly better categorization results than pointwise logistic regression.

Table 11.14 Ranking SVM results for language categorization.

Method	Error (%)	MRR
LR (pointwise)	18.0	0.88
Rank SVM (C=0.01)	17.3	0.88
Rank SVM (C=0.1)	17.6	0.88

Table 11.15 Logistic regression learning-to-rank results, trained on one year's runs and tested on the next year's topics.

Corpus	TREC45		GOV2	
Train	1998		2004	
Test	1999		2005	
Measure	P@10	MAP	P@10	MAP
RRF	0.464	0.252	0.570	0.352
LR	0.446	0.266	0.588	0.309
Rank SVM (C=0.02)	0.420	0.234	0.556	0.268

11.7.5 Learning for Ranked IR

Table 11.3 on page 381 shows that fixed-combination methods effectively combine the ranked results from various IR methods. Because exactly the same methods were employed for all four experiments, we explore the possibility of using learning-to-rank methods to come up with a better combination method. To this end we identify the TREC45 1998 and GOV2 2004 runs and qrels as training examples and labels, and the TREC45 1999 and GOV2 2005 runs as the corresponding test examples. Each example (d, q_j) was represented as a query vector

$$x^{[d,q_j]} = (r, j, s_1(d, q_j), s_2(d, q_j), \ldots, s_k(d, q_j)), \tag{11.39}$$

where $k = 29$ is the number of methods. We applied (pointwise) logistic regression and ranking SVM to the problem.

Table 11.15 shows the results of logistic regression and ranking SVM compared with the baseline of reciprocal rank fusion. It is difficult to argue that the LR result is substantively different from that of the baseline. Rank SVM (SVM-Rank[5]) appears to be inferior, and requires roughly four days of CPU time to process the GOV2 data set, as opposed to a few seconds for RRF and a few minutes for LR.

[5] `www.cs.cornell.edu/people/tj/svm_light/svm_rank.html`

Table 11.16 Learning-to-rank results for 583,850 document-query pairs in the LETOR 3 corpus. The P@10 and MAP scores are the averages of the scores for 680 topics over the seven LETOR 3 data sets. RRF, Condorcet, and CombMNZ fuse the results of the separate learning-to-rank methods.

Method	P@10	MAP
ListNet (Cao et al., 2007)	0.1853	0.5846
LR (gr. desc.)	0.1821	0.5837
AdaRank-MAP (Xu and Li, 2007)	0.1789	0.5778
RankSVM (Joachims, 2002)	0.1811	0.5737
LR (batch)	0.1780	0.5715
RankBoost (Freund et al., 2003)	0.1836	0.5622
RRF	0.1902	0.6051
Condorcet	**0.1907**	0.5917
CombMNZ	0.1893	**0.6107**

11.7.6 The LETOR Data Set

LETOR, a benchmark data set for learning to rank (Liu et al., 2007), is a test collection for evaluating learning-to-rank methods. At the time of writing, version 3 of the LETOR data set was current. LETOR 3 consists of seven data sets, each containing examples of the form shown in Equation 11.39. The examples are divided into five standard splits for the purpose of five-fold cross-validation. Six of the data sets use selected documents from the TREC GOV (not GOV2) collection and topics from the TREC 2003 and TREC 2004 Web Track tasks. One of the data sets uses selected documents and topics from the OHSUMED collection. For the TREC documents relevance is binary, and there are 64 features representing different features of the query and the document. For the OHSUMED documents relevance is ternary (not relevant, relevant, very relevant) and there are 45 content-based features. Standard evaluation tools are supplied with the corpus.

Included with the LETOR data set are raw results Res_i and $Score_i$ for several state-of-the-art learning-to-rank methods:

- ListNet (Cao et al., 2007) uses gradient descent to optimize a listwise objective function.
- AdaRank-MAP (Xu and Li, 2007) uses AdaBoost with a surrogate objective function specifically intended to optimize average precision.
- RankBoost (Freund et al., 2003) uses AdaBoost to minimize inversions between pairwise training examples and the resulting ranked list.
- RankSVM is the same SVM^{light} implementation we used for language categorization, which minimizes inversions between pairwise training examples.

For each of the 680 topics there are up to 1000 documents. Although the methods were trained separately on the seven data sets, we averaged the 680 per-topic P@10 and AP scores to yield the summary results shown in Table 11.16. We also included results from the LR (batch) pointwise method used in the previous examples. LR (gr. desc.) uses the gradient descent pairwise LR method with an objective function that gives increased weight to inversions involving highly ranked nonrelevant documents.

The fusion methods RRF, Condorcet, and CombMNZ combine the results of the six learning-to-rank methods, improving both P@10 and MAP effectiveness. At the time of writing, no learning-to-rank method surpasses these fusion results. Furthermore, statistical analysis fails to show any significant difference among the six individual methods (Cormack et al., 2009).

11.8 Further Reading

Belkin et al. (1995) report early efforts to combine search results generated by different queries, both manually and automatically derived from a TREC topic. A family of methods for score combination — of which CombMNZ is one — is presented. These methods are investigated in a number of subsequent studies and are found to be effective in combining a variety of retrieval results (e.g., Lee, 1997). Montague and Aslam (2002) survey and evaluate score combination and election methods, and propose Condorcet as the method of choice. Voorhees et al. (1995) consider test-collection fusion in which the same query is applied to disjoint collections of documents. The term *metasearch* is usually taken to mean the combination of results from autonomous search engines, as surveyed by Meng et al. (2002).

Vogt and Cottrell (1999) investigate the use of linear regression to combine search results based on training examples. Stacked generalization (stacking), proposed by Wolpert (1992), is a standard technique in machine learning. The particular method of log-odds transformation and gradient descent used in our examples is due to Lynam and Cormack (2006). Bootstrap aggregation (bagging; Breiman, 1996) and boosting (Schapire, 2003), like stacking, are staples of machine learning. Most machine learning textbooks (e.g., Hastie et al., 2009) treat *ensemble* methods such as stacking, bagging, and boosting as major topics.

Multicategory logistic regression, like logistic regression itself, is a standard technique for general data analysis (Hosmer and Lemeshow, 2000). Crammer and Singer (2002) describe practical algorithms for multiclass support vector machines. Schapire and Singer (2000) describe the application of boosting to multicategory problems.

Learning to rank has been the subject of workshops at NIPS 2005, SIGIR 2007, and SIGIR 2008. Even so, a precise definition of what is meant by the term remains elusive. Seminal papers by Burges et al. (2005) and Joachims et al. (2005) tacitly define the problem to be one of minimizing inversions between learned and target rankings: Burges et al. use gradient descent in a method known as Ranknet, whereas Joachims et al. use support vector machines.

Li et al. (2007) frame learning to rank as a multiple category classification problem in which each graded relevance level corresponds to a category. Herbrich et al. (2000) and Freund et al. (2003) describe pairwise methods. Xu and Li (2007) describe the AdaRank-MAP listwise method; Burges et al. (2006) descibe the LambdaRank listwise method. Svore and Burges (2009) demonstrate that LambdaRank can outperform BM25 using an equivalent feature set. Yilmaz and Robertson (2010) discuss the use of IR evaluation measures as optimization targets for learning to rank. The LETOR data set (Liu et al., 2007) provides a standard benchmark for evaluating learning-to-rank methods. Liu (2009) surveys current approaches for learning to rank. The sofia-ml package by D. Sculley provides a new suite of fast incremental algorithms for machine learning.[6]

11.9 Exercises

Exercise 11.1 Download a TREC test collection and one or more search engines. Run various configurations of the search engines on the topics, and combine their results using RRF, CombMNZ, and Condorcet. Compare the results.

Exercise 11.2 Download the TREC Spam Filter Evaluation Toolkit, and run several of the sample filters on the sample corpus. Combine the results using voting, naïve Bayes, and logistic regression. Evaluate the results.

Exercise 11.3 Use RRF, CombMNZ, and Condorcet to combine the results of your spam filter runs. Does this experiment aptly model the use of a spam filter? Do these methods improve on voting, naïve Bayes, and logistic regression?

Exercise 11.4 Download the LETOR 3 data set. Apply one or more learning-to-rank methods (e.g., SVM^{light}) to the data set and compare your results against the published baselines.

Exercise 11.5 Apply straightforward logistic regression (e.g., LibLinear) to the LETOR 3 data set, ignoring the `qid` field. Compare the results with those achieved by the learning-to-rank methods.

11.10 Bibliography

Agresti, A. (2007). *An Introduction to Categorical Data Analysis* (2nd ed.). New York: Wiley-Interscience.

[6] `code.google.com/p/sofia-ml`

Belkin, N., Kantor, P., Fox, E., and Shaw, J. (1995). Combining the evidence of multiple query representations for information retrieval. *Information Processing & Management*, 31(3):431–448.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Burges, C. J. C., Ragno, R., and Le, Q. V. (2006). Learning to rank with nonsmooth cost functions. In *Proceedings of the 20th Annual Conference on Neural Information Processing Systems*, pages 193–200. Vancouver, Canada.

Burges, C. J. C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 89–96. Bonn, Germany.

Cao, Z., Qin, T., Liu, T. Y., Tsai, M. F., and Li, H. (2007). Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 129–136. Corvalis, Oregon.

Cormack, G. V., Clarke, C. L. A., and Büttcher, S. (2009). Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 758–759. Boston, Massachusetts.

Crammer, K., and Singer, Y. (2002). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292.

Efron, B., and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Boca Raton, Florida: Chapman & Hall/CRC.

Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). *The Elements of Statistical Learning* (2nd ed.). Berlin, Germany: Springer.

Herbrich, R., Graepel, T., and Obermayer, K. (2000). Large margin rank boundaries for ordinal regression. In Bartlett, P. J., Schölkopf, B., Schuurmans, D., and Smola, A. J., editors, *Advances in Large Margin Classifiers*, chapter 7, pages 115–132. Cambridge, Massachusetts: MIT Press.

Hosmer, D. W., and Lemeshow, S. (2000). *Applied Logistic Regression* (2nd ed.). New York: Wiley-Interscience.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142. Edmonton, Canada.

Joachims, T., Granka, L., Pan, B., Hembrooke, H., and Gay, G. (2005). Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 154–161. Salvador, Brazil.

Lee, J.H. (1997). Analyses of multiple evidence combination. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–276.

Li, P., Burges, C., and Wu, Q. (2007). McRank: Learning to rank using multiple classification and gradient boosting. In *Proceedings of the 21st Annual Conference on Neural Information Processing Systems*, pages 897–904. Vancouver, Canada.

Liu, T.Y. (2009). Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331.

Liu, T.Y., Xu, J., Qin, T., Xiong, W., and Li, H. (2007). LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, pages 481–490. Amsterdam, The Netherlands.

Lynam, T.R., and Cormack, G.V. (2006). On-line spam filter fusion. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 123–130. Seattle, Washington.

Meng, W., Yu, C., and Liu, K.L. (2002). Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89.

Montague, M., and Aslam, J.A. (2002). Condorcet fusion for improved retrieval. In *Proceedings of the 11th International Conference on Information and Knowledge Management*, pages 538–548. McLean, Virginia.

Schapire, R. (2003). The boosting approach to machine learning: An overview. In Denison, D.D., Hansen, M.H., Holmes, C.C., Mallick, B., and Yu, B., editors, *Nonlinear Estimation and Classification*, volume 171 of *Lecture Notes in Statistics*, pages 149–172. Berlin, Germany: Springer.

Schapire, R., and Singer, Y. (2000). BoosTexter: A boosting-based system for text categorization. *Machine learning*, 39(2):135–168.

Surowiecki, J. (2004). *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations.* New York: Doubleday.

Svore, K.M., and Burges, C.J. (2009). A machine learning approach for improved BM25 retrieval. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 1811–1814. Hong Kong, China.

Vogt, C., and Cottrell, G. (1999). Fusion via a linear combination of scores. *Information Retrieval*, 1(3):151–173.

Voorhees, E.M., Gupta, N.K., and Johnson-Laird, B. (1995). Learning collection fusion strategies. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 172–179. Seattle, Washington.

Wolpert, D.H. (1992). Stacked generalization. *Neural Networks*, 5:241–259.

Xu, J., and Li, H. (2007). Adarank: A boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 391–398. Amsterdam, The Netherlands.

Yilmaz, E., and Robertson, S. (2010). On the choice of effectiveness measures for learning to rank. *Information Retrieval*.