Learning to Order Things

William W. Cohen Robert E. Schapire Yoram Singer
AT&T Labs, 180 Park Ave., Florham Park, NJ 07932
{wcohen,schapire,singer}@research.att.com

Abstract

There are many applications in which it is desirable to order rather than classify instances. Here we consider the problem of learning how to order, given feedback in the form of preference judgments, i.e., statements to the effect that one instance should be ranked ahead of another. We outline a two-stage approach in which one first learns by conventional means a *preference function*, of the form PREF(u, v), which indicates whether it is advisable to rank u before v. New instances are then ordered so as to maximize agreements with the learned preference function. We show that the problem of finding the ordering that agrees best with a preference function is NP-complete, even under very restrictive assumptions. Nevertheless, we describe a simple greedy algorithm that is guaranteed to find a good approximation. We then discuss an on-line learning algorithm, based on the "Hedge" algorithm, for finding a good linear combination of ranking "experts." We use the ordering algorithm combined with the on-line learning algorithm to find a combination of "search experts," each of which is a domain-specific query expansion strategy for a WWW search engine, and present experimental results that demonstrate the merits of our approach.

1 Introduction

Most previous work in inductive learning has concentrated on learning to classify. However, there are many applications in which it is desirable to order rather than classify instances. An example might be a personalized email filter that gives a priority ordering to unread mail. Here we will consider the problem of learning how to construct such orderings, given feedback in the form of *preference judgments*, i.e., statements that one instance should be ranked ahead of another.

Such orderings could be constructed based on a learned classifier or regression model, and in fact often are. For instance, it is common practice in information retrieval to rank documents according to their estimated probability of relevance to a query based on a learned classifier for the concept "relevant document." An advantage of learning orderings directly is that preference judgments can be much easier to obtain than the labels required for classification learning.

For instance, in the email application mentioned above, one approach might be to rank messages according to their estimated probability of membership in the class of "urgent" messages, or by some numerical estimate of urgency obtained by regression. Suppose, however, that a user is presented with an ordered list of email messages, and elects to read the third message first. Given this election, it is not necessarily the case that message three is urgent, nor is there sufficient information to estimate any numerical urgency measures; however, it seems quite reasonable to infer that message three should have been ranked ahead of the others. Thus, in this setting, obtaining preference information may be easier and more natural than obtaining the information needed for classification or regression.

In the remainder of this paper, we will investigate the following two-stage approach to learning how to order. In stage one, we learn a *preference function*, a two-argument function PREF(u, v) which returns a numerical measure of how certain it is that u should be ranked before v. In stage two, we use the learned preference function to order a set of new instances U; to accomplish this, we evaluate the learned function PREF(u, v) on all pairs of instances $u, v \in U$, and choose an ordering of U that agrees, as much as possible, with these pairwise preference judgments. This general approach is novel; for related work in various fields see, for instance, references [2, 3, 1, 7, 10].

As we will see, given an appropriate feature set, learning a preference function can be reduced to a fairly conventional classification learning problem. On the other hand, finding a total order that agrees best with a preference function is NP-complete. Nevertheless, we show that there is an efficient greedy algorithm that always finds a good approximation to the best ordering. After presenting these results on the complexity of ordering instances using a preference function, we then describe a specific algorithm for learning a preference function. The algorithm is an on-line weight allocation algorithm, much like the weighted majority algorithm [9] and Winnow [8], and, more directly, Freund and Schapire's [4] "Hedge" algorithm. We then present some experimental results in which this algorithm is used to combine the results of several "search experts," each of which is a domain-specific query expansion strategy for a WWW search engine.

2 Preliminaries

Let X be a set of instances (possibly infinite). A *preference function* PREF is a binary function PREF : $X \times X \rightarrow [0, 1]$. A value of PREF$(u, v)$ which is close to 1 or 0 is interpreted as a strong recommendation that u should be ranked before v. A value close to $1/2$ is interpreted as an abstention from making a recommendation. As noted above, the hypothesis of our learning system will be a preference function, and new instances will be ranked so as to agree as much as possible with the preferences predicted by this hypothesis.

In standard classification learning, a hypothesis is constructed by combining primitive features. Similarly, in this paper, a preference function will be a combination of other preference functions. In particular, we will typically assume the availability of a set of N primitive preference functions $R_1, \ldots, R_N$. These can then be combined in the usual ways, e.g., with a boolean or linear combination of their values; we will be especially interested in the latter combination method.

It is convenient to assume that the R_i's are well-formed in certain ways. To this end, we introduce a special kind of preference function called a *rank ordering*. Let S be a totally ordered set[1] with '$>$' as the comparison operator. An *ordering function into* S is a function $f : X \rightarrow S$. The function f induces the preference function R_f, defined as

$$R_f(u, v) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } f(u) > f(v) \\ 0 & \text{if } f(u) < f(v) \\ \frac{1}{2} & \text{otherwise.} \end{cases}$$

We call R_f a *rank ordering for* X *into* S. If $R_f(u, v) = 1$, then we say that u is preferred to v, or u is ranked higher than v.

It is sometimes convenient to allow an ordering function to "abstain" and not give a preference for a pair u, v. Let ϕ be a special symbol not in S, and let f be a function into $S \cup \{\phi\}$. We will interpret the mapping $f(u) = \phi$ to mean that u is "unranked," and let $R_f(u, v) = \frac{1}{2}$ if either u or v is unranked.

To give concrete examples of rank ordering, imagine learning to order documents based on the words that they contain. To model this, let X be the set of all documents in a repository,

[1]That is, for all pairs of distinct elements $s_1, s_2 \in S$, either $s_1 < s_2$ or $s_1 > s_2$.

and for N words $w_1, \ldots, w_N$, let $f_i(u)$ be the number of occurrences of w_i in u. Then R_{f_i} will prefer u to v whenever w_i occurs more often in u than v. As a second example, consider a meta-search application in which the goal is to combine the rankings of several WWW search engines. For N search engines $e_1, \ldots, e_N$, one might define f_i so that R_{f_i} prefers u to v whenever u is ranked ahead of v in the list L_i produced by the corresponding search engine. To do this, one could let $f_i(u) = -k$ for the document u appearing in the k-th position in the list L_i, and let $f_i(u) = \phi$ for any document not appearing in L_i.

3 Ordering instances with a preference function

We now consider the complexity of finding the total order that agrees best with a learned preference function. To analyze this, we must first quantify the notion of agreement between a preference function PREF and an ordering. One natural notion is the following: Let X be a set, PREF be a preference function, and let ρ be a total ordering of X, expressed again as an ordering function (i.e., $\rho(u) > \rho(v)$ iff u precedes v in the order). We define AGREE(ρ, PREF) to be the sum of PREF(u, v) over all pairs u, v such that u is ranked ahead of v by ρ:

$$\text{AGREE}(\rho, \text{PREF}) = \sum_{u,v:\rho(u)>\rho(v)} \text{PREF}(u, v). \qquad (1)$$

Ideally, one would like to find a ρ that maximizes AGREE(ρ, PREF). This general optimization problem is of little interest since in practice, there are many constraints imposed by learning: for instance PREF must be in some restricted class of functions, and will generally be a combination of relatively well-behaved preference functions R_i. A more interesting question is whether the problem remains hard under such constraints.

The theorem below gives such a result, showing that the problem is NP-complete even if PREF is restricted to be a linear combination of rank orderings. This holds even if all the rank orderings map into a set S with only three elements, one of which may or may not be ϕ. (Clearly, if S consists of more than three elements then the problem is still hard.)

Theorem 1 *The following decision problem is NP-complete:*

Input: *A rational number κ; a set X; a set S with $|S| \geq 3$; a collection of N ordering functions $f_i : X \to S$; and a preference function* PREF *defined as* $\text{PREF}(u, v) = \sum_{i=1}^{N} w_i R_{f_i}(u, v)$ *where* $\mathbf{w} = (w_1, \ldots, w_N)$ *is a weight vector in* $[0, 1]^N$ *with* $\sum_{i=1}^{N} w_i = 1$.

Question: *Does there exist a total order ρ such that* AGREE(ρ, PREF) $\geq \kappa$*?*

The proof (omitted) is by reduction from CYCLIC-ORDERING [5, 6].

Although this problem is hard when $|S| \geq 3$, it becomes tractable for linear combinations of rank orderings into a set S of size two. In brief, suppose one is given X, S and PREF as in Theorem 1, save that S is a two-element set, which we assume without loss of generality to be $S = \{0, 1\}$. Now define $\rho(u) = \sum_i w_i f_i(u)$. It can be shown that the total order defined by ρ maximizes AGREE(ρ, PREF). (In case of a tie, $\rho(u) = \rho(v)$ for distinct u and v, ρ defines only a partial order. The claim still holds in this case for any total order which is consistent with this partial order.) Of course, when $|S| = 2$, the rank orderings are really only binary classifiers. The fact that this special case is tractable underscores the fact that manipulating orderings can be computationally more difficult than performing the corresponding operations on binary classifiers.

Theorem 1 implies that we are unlikely to find an efficient algorithm that finds the optimal total order for a weighted combination of rank orderings. Fortunately, there do exist efficient algorithms for finding an *approximately* optimal total order. Figure 1 summarizes a greedy

```
**Algorithm Order-By-Preferences**
**Inputs:** an instance set $X$; a preference function PREF
**Output:** an approximately optimal ordering function $\hat{\rho}$
let $V = X$
for each $v \in V$ do $\pi(v) = \sum_{u \in V} \text{PREF}(v,u) - \sum_{u \in V} \text{PREF}(u,v)$
while $V$ is non-empty do
    let $t = \arg\max_{u \in V} \pi(u)$
    let $\hat{\rho}(t) = |V|$
    $V = V - \{t\}$
    for each $v \in V$ do $\pi(v) = \pi(v) + \text{PREF}(t,v) - \text{PREF}(v,t)$
endwhile
```

Figure 1: A greedy ordering algorithm

algorithm that produces a good approximation to the best total order, as we will shortly demonstrate. The algorithm is easiest to describe by thinking of PREF as a directed weighted graph where, initially, the set of vertices V is equal to the set of instances X, and each edge $u \to v$ has weight PREF(u, v). We assign to each vertex $v \in V$ a *potential* value $\pi(v)$, which is the weighted sum of the outgoing edges *minus* the weighted sum of the ingoing edges. That is, $\pi(v) = \sum_{u \in V} \text{PREF}(v,u) - \sum_{u \in V} \text{PREF}(u,v)$. The greedy algorithm then picks some node t that has maximum potential, and assigns it a rank by setting $\hat{\rho}(t) = |V|$, effectively ordering it ahead of all the remaining nodes. This node, together with all incident edges, is then deleted from the graph, and the potential values π of the remaining vertices are updated appropriately. This process is repeated until the graph is empty; notice that nodes removed in subsequent iterations will have progressively smaller and smaller ranks.

The next theorem shows that this greedy algorithm comes within a factor of two of optimal. Furthermore, it is relatively simple to show that the approximation factor of 2 is tight.

Theorem 2 *Let* OPT(PREF) *be the weighted agreement achieved by an optimal total order for the preference function* PREF *and let* APPROX(PREF) *be the weighted agreement achieved by the greedy algorithm. Then* APPROX(PREF) $\geq \frac{1}{2}$OPT(PREF).

4 Learning a good weight vector

In this section, we look at the problem of learning a good linear combination of a set of preference functions. Specifically, we assume access to a set of *ranking experts* which provide us with preference functions R_i of a set of instances. The problem, then, is to learn a preference function of the form PREF$(u, v) = \sum_{i=1}^{N} w_i R_i(u, v)$. We adopt the on-line learning framework first studied by Littlestone [8] in which the weight w_i assigned to each ranking expert R_i is updated incrementally.

Learning is assumed to take place in a sequence of rounds. On the t-th round, the learning algorithm is provided with a set X^t of instances to be ranked and to a set of N preference functions R_i^t of these instances. The learner may compute $R_i^t(u, v)$ for any and all preference functions R_i^t and pairs $u, v \in X^t$ before producing a final ordering ρ_t of X^t. Finally, the learner receives feedback from the environment. We assume that the feedback is an arbitrary set of assertions of the form "u should be preferred to v." That is, formally we regard the feedback on the t-th round as a set F^t of pairs (u, v) indicating such preferences.

The algorithm we propose for this problem is based on the "weighted majority algorithm" [9] and, more directly, on the "Hedge" algorithm [4]. We define the *loss* of a preference function

```
Allocate Weights for Ranking Experts
Parameters:
$\beta \in [0,1]$, initial weight vector $\mathbf{w}^1 \in [0,1]^N$ with $\sum_{i=1}^N w_i^1 = 1$
$N$ ranking experts, number of rounds $T$
Do for $t = 1, 2, \ldots, T$
    1. Receive a set of elements $X^t$ and preference functions $R_1^t, \ldots, R_N^t$.
    2. Use algorithm Order-By-Preferences to compute ordering function $\hat{\rho}_t$ which approximates $\mathrm{PREF}_t(u,v) = \sum_{i=1}^N w_i R_i^t(u,v)$.
    3. Order $X^t$ using $\hat{\rho}_t$.
    4. Receive feedback $F^t$ from the user.
    5. Evaluate losses $\mathrm{Loss}(R_i^t, F^t)$ as defined in Eq. (2).
    6. Set the new weight vector $w_i^{t+1} = w_i^t \cdot \beta^{\mathrm{Loss}(R_i^t, F^t)} / Z_t$ where $Z_t$ is a normalization constant, chosen so that $\sum_{i=1}^N w_i^{t+1} = 1$.
```

Figure 2: The on-line weight allocation algorithm.

R with respect to the user's feedback F as

$$\mathrm{Loss}(R,F) \stackrel{\mathrm{def}}{=} \frac{\sum_{(u,v)\in F}(1 - R(u,v))}{|F|}. \tag{2}$$

This loss has a natural probabilistic interpretation. If R is viewed as a randomized prediction algorithm that predicts that u will precede v with probability $R(u,v)$, then $\mathrm{Loss}(R,F)$ is the probability of R disagreeing with the feedback on a pair (u,v) chosen uniformly at random from F.

We now can use the Hedge algorithm almost verbatim, as shown in Figure 2. The algorithm maintains a positive weight vector whose value at time t is denoted by $\mathbf{w}^t = (w_1^t, \ldots, w_N^t)$. If there is no prior knowledge about the ranking experts, we set all initial weights to be equal so that $w_i^1 = 1/N$. The weight vector $\mathbf{w}^t$ is used to combine the preference functions of the different experts to obtain the preference function $\mathrm{PREF}_t = \sum_{i=1}^N w_i^t R_i^t$. This, in turn, is converted into an ordering $\hat{\rho}_t$ on the current set of elements X^t using the method described in Section 3. After receiving feedback F^t, the loss for each preference function $\mathrm{Loss}(R_i^t, F^t)$ is evaluated as in Eq. (2) and the weight vector $\mathbf{w}^t$ is updated using the multiplicative rule $w_i^{t+1} = w_i^t \cdot \beta^{Loss(R_i^t, F^t)} / Z_t$ where $\beta \in [0,1]$ is a parameter, and Z_t is a normalization constant, chosen so that the weights sum to one after the update. Thus, based on the feedback, the weights of the ranking experts are adjusted so that experts producing preference functions with relatively large agreement with the feedback are promoted.

We will briefly sketch the theoretical rationale behind this algorithm. Freund and Schapire [4] prove general results about Hedge which can be applied directly to this loss function. Their results imply almost immediately a bound on the cumulative loss of the preference function PREF_t in terms of the loss of the best ranking expert, specifically

$$\sum_{t=1}^T \mathrm{Loss}(\mathrm{PREF}_t, F^t) \le a_\beta \min_i \sum_{t=1}^T \mathrm{Loss}(R_i^t, F^t) + c_\beta \ln N$$

where $a_\beta = \ln(1/\beta)/(1-\beta)$ and $c_\beta = 1/(1-\beta)$. Thus, if one of the ranking experts has low loss, then so will the combined preference function PREF_t.

However, we are not interested in the loss of PREF_t (since it is not an ordering), but rather in the performance of the actual ordering $\hat{\rho}_t$ computed by the learning algorithm. Fortunately, the losses of these can be related using a kind of triangle inequality. It can be shown that, for any PREF, F and ρ:

$$\text{Loss}(R_\rho, F) \le \frac{\text{DISAGREE}(\rho, \text{PREF})}{|F|} + \text{Loss}(\text{PREF}, F) \tag{3}$$

where, similar to Eq. (1), $\text{DISAGREE}(\rho, \text{PREF}) = \sum_{u,v:\rho(u)>\rho(v)}(1 - \text{PREF}(u, v))$. Not surprisingly, maximizing AGREE is equivalent to minimizing DISAGREE.

So, in sum, we use the greedy algorithm of Section 3 to minimize (approximately) the first term on the right hand side of Eq. (3), and we use the learning algorithm Hedge to minimize the second term.

5 Experimental results for metasearch

We now present some experiments in learning to combine the results of several WWW searches. We note that this problem exhibits many facets that require a general approach such as ours. For instance, approaches that learn to combine similarity scores are not applicable since the similarity scores of WWW search engines are often unavailable.

We chose to simulate the problem of learning a domain-specific search engine. As test cases we picked two fairly narrow classes of queries—retrieving the home pages of machine learning researchers (ML), and retrieving the home pages of universities (UNIV). We obtained a listing of machine learning researchers, identified by name and affiliated institution, together with their home pages, and a similar list for universities, identified by name and (sometimes) geographical location. Each entry on a list was viewed as a query, with the associated URL the sole relevant document.

We then constructed a series of special-purpose "search experts" for each domain. These were implemented as query expansion methods which converted a name, affiliation pair (or a name, location pair) to a likely-seeming Altavista query. For example, one expert for the ML domain was to search for all the words in the person's name plus the words "machine" and "learning," and to further enforce a strict requirement that the person's last name appear. Overall we defined 16 search experts for the ML domain and 22 for the UNIV domain. Each search expert returned the top 30 ranked documents. In the ML domain there were 210 searches for which at least one search expert returned the named home page; for the UNIV domain, there were 290 such searches.

For each query t, we first constructed the set X^t consisting of all documents returned by all of the expanded queries defined by the search experts. Next, each search expert i computed a preference function R_i^t. We chose these to be rank orderings defined with respect to an ordering function f_i^t in the natural way: We assigned a rank of $f_i^t = 30$ to the first listed document, $f_i = 29$ to the second-listed document, and so on, finally assigning a rank of $f_i = 0$ to every document not retrieved by the expanded query associated with expert i.

To encode feedback, we considered two schemes. In the first we simulated complete relevance feedback—that is, for each query, we constructed feedback in which the sole relevant document was preferred to all other documents. In the second, we simulated the sort of feedback that could be collected from "click data," i.e., from observing a user's interactions with a metasearch system. For each query, after presenting a ranked list of documents, we noted the rank of the one relevant document. We then constructed a feedback ranking in which the relevant document is preferred to all preceding documents. This would correspond to observing which link the user actually followed, and making the assumption that this link was preferred to previous links.

To evaluate the expected performance of a fully-trained system on novel queries in this domain, we employed leave-one-out testing. For each query q, we removed q from the

	ML Domain				University Domain			
	Top 1	Top 10	Top 30	Av. rank	Top 1	Top 10	Top 30	Av. rank
Learned System (Full Feedback)	114	185	198	4.9	111	225	253	7.8
Learned System ("Click Data")	93	185	198	4.9	87	229	259	7.8
Naive	89	165	176	7.7	79	157	191	14.4
Best (Top 1)	**119**	170	184	6.7	**112**	221	247	8.2
Best (Top 10)	114	**182**	190	5.3	111	**223**	249	8.0
Best (Top 30)	97	181	**194**	5.6	111	223	**249**	8.0
Best (Av. Rank)	114	182	190	**5.3**	111	223	249	**8.0**

Table 1: Comparison of learned systems and individual search queries

query set, and recorded the rank of q after training (with $\beta = 0.5$) on the remaining queries. For click data feedback, we recorded the median rank over 100 randomly chosen permutations of the training queries.

We the computed an approximation to average rank by artificially assigning a rank of 31 to every document that was either unranked, or ranked above rank 30. (The latter case is to be fair to the learned system, which is the only one for which a rank greater than 30 is possible.) A summary of these results is given in Table 1, together with some additional data on "top-k performance"—the number of times the correct homepage appears at rank no higher than k. In the table we give the top-k performance (for three values of k) and average rank for several ranking systems: the two learned systems, the naive query (the person or university's name), and the single search expert that performed best with respect to each performance measure. The table illustrates the robustness of the learned systems, which are nearly always competitive with the best expert for every performance measure listed; the only exception is that the system trained on click data trails the best expert in top-k performance for small values of k. It is also worth noting that in both domains, the naive query (simply the person or university's name) is not very effective. Even with the weaker click data feedback, the learned system achieves a 36% decrease in average rank over the naive query in the ML domain, and a 46% decrease in the UNIV domain.

To summarize the experiments, on these domains, the learned system not only performs much better than naive search strategies; it also consistently performs at least as well as, and perhaps slightly better than, any single domain-specific search expert. Furthermore, the performance of the learned system is almost as good with the weaker "click data" training as with complete relevance feedback.

References

[1] D.S. Hochbaum (Ed.). *Approximation Algorithms for NP-hard problems.* PWS Publishing Company, 1997.

[2] O. Etzioni, S. Hanks, T. Jiang, R. M. Karp, O. Madani, and O. Waarts. Efficient information gathering on the internet. In *37th Ann. Symp. on Foundations of Computer Science*, 1996.

[3] P.C Fishburn. *The Theory of Social Choice.* Princeton University Press, Princeton, NJ, 1973.

[4] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 1997.

[5] Z. Galil and N. Megido. Cyclic ordering is NP-complete. *Theor. Comp. Sci.*, 5:179–182, 1977.

[6] M.R. Gary and D.S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-completeness.* W. H. Freeman and Company, New York, 1979.

[7] P.B. Kantor. Decision level data fusion for routing of documents in the TREC3 context: a best case analysis of worste case results. In *TREC-3*, 1994.

[8] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 1988.

[9] N. Littlestone and M.K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[10] K.E. Lochbaum and L.A. Streeter. Comparing and combining the effectiveness of latent semantic indexing and the ordinary vector space model for information retrieval. *Information processing and management*, 25(6):665–676, 1989.