1. Summary

This paper provides an overview for 3 related ranking algorithms: RankNet, LambdaRank and LambdaMART.

RankNet maps input features in n-dimensional space to real numbers (score *s*), and the function should be differentiable. For a pair of features (from URLs) *i* and *j*, learned probability *P* that one should be ranked higher than another is expressed as the function of difference of their scores. And the cost *C* could be expressed in the form of information entropy. (By converting *P* to *S*, the form cost *C* becomes symmetric.) The learning idea of using gradient descent to update model parameters *w* is applied for not only RankNet but also the other two algorithms. In order to obtain the gradient of cost *C* with respect to (w.r.t.) model parameters *w*, the gradient of cost *C* w.r.t. model score *s* is required. And in RankNet cost *C* is usually calculated followed by the derivation of the gradient, which usually faces difficulties of sorting in most IR measures (?).

From RankNet to LambdaRank, a key observation to speed up is that the cost is not needed while the gradient of cost *c* w.r.t score *s* is enough. (It can be *directly* written down without derivation.) And "lambda" which originates from factoring RankNet is such gradient. Given a *set* of pairs *I*, "lambda" could be factored out from the summation of *pairwise* contributions to update of model weights being viewed as summation of *identical* contributions affected by *all others*. lambda for a URL could be interpreted as arrows (or forces) provided by all other URLs to push it up/down per relative relevance to the query. Unlike stochastic gradient descent (update for each pair of URLs) used in RankNet, LambdaRank uses (mini) batch gradient descent (update for each URL summing up its contribution from all pairs?), which reduces the training time complexity from quadratic to linear in the number of queries. Additionally, experiments show that incorporating some IR measures (e.g., Normalized Discounted Cumulative Gain (NDCG)) can produce better/general results.

Multiple Additive Regression Trees (MART) is combined with LambdaRank to generate LambdaMART. According to the paper, MART can be viewed as performing gradient descent using a *set* of least squares *regression* trees in the function space. Given a n-dimensional vector as input feature, each single tree maps it to a real number and the overall output is a linear combination of each score. Each tree models the gradient of the cost w.r.t the model score, and a *new* tree is added to the *ensemble* with a step size (w/ leaf value gamma_{*km*} to be learned for each leaf *k* in each tree *m*, where Newton approximation taking first and second derivatives of cost w.r.t. model score can help).

To implement LambdaMART, MART is used where the gradient is the lambda gradient in LambdaRank. And the Newton step for updating leaf value gamma also needs appropriate specification. Starting from an *initial* base model (tree), adaption is performed iteratively for remaining (*N*-1) trees by: 1) calculating lambda gradient and the gradient of lambda gradient w.r.t. linear combination of *previous* model scores for each training sample, 2) assigning leaf values gamma based on Newton step taking in calculated parameters, and 3) update the overall model from first (*m*-1) trees to first *m* trees. Furthermore, the paper compares the way LambdaRank and LambdaMART update parameters. The former updates all weights after each query is examined, while the later updates part of parameters (leaf node) but using all data. It implies that LambdaMART is capable to increase overall utility ("inversed" cost) by decreasing the utility for some queries.

2. Comments

As the paper says, it provides a self-contained, detailed and complete description of RankNet, LambdaRank and LambdaMART. The logic flow is clear: background knowledge is discussed with appropriate level of details before starting a new topic, notations are carefully and mostly consistent, examples and figures gives good intuition for abstract concepts and algorithms. Additionally, main and supplemental materials are distinguished by colored blocks, which is very reader-friendly. On the other hand, it might be less confusing that the notation convention being mentioned in the paper. Is Einstein notation applied in the paper?

3. Questions

(1) At the end of Section 7, it is mentioned that "…LambdaMART is able to choose splits and leaf values that may decrease the utility for some queries, as long as the overall utility increases". I guess it implies that LambdaMART could explore "more widely" in result space and is less likely to get stuck in "local optima". Does LambdaRank also has this capability? Relevantly, at the end of Section 4.1, it is mentioned that "…and for a given url, all the lambdas ate incremented…" Does it mean all lambdas are always monotonously increased for each iteration so that it cannot decrease some lambdas for potential better overall results?

(2) For each of three discussed ranking algorithms, gradient descent plays an important role in model training. I was curious about the strategies of choosing reasonable values of learning rates for different algorithms. How can people pick up a learning rate for fast and convergent iteration? Moreover, what is the actual applications of different algorithms? Are there any uniqueness/limitations (e.g., generalization ability, storage requirement, thread safety) for each of them?