Overview

- Train a model that arranges query results into an ordered list
- Predictors are sample documents encoded as feature matrix, labels are relevance degree
- Rank is based on LambdaMART algorithm

Training with the Pairwise Objective

- LambdaMART is a pairwise model, meaning it compares relevance degree of samples in query group
  - It basically creates pairs of items in a query where one item is more relevant than the other
  - The model learns to order these Paris correctly using a proxy gradient
  - Proxy gradient - an approximation of the gradient that is used to optimize the ranking model without needing an explicit gradient for a ranking-based loss function
- An additional stored array called qid is needed to specify the group of input samples

QID	Label	Features
1	0	x_1
1	1	x_2
1	0	x_3
2	0	x_4
2	1	x_5
2	1	x_6
2	1	x_7

Notice that the samples are sorted based on their query index in a non-decreasing order. In the above example, the first three samples belong to the first query and the next four samples belong to the second. For the sake of simplicity, we will use a synthetic binary learning-to-rank

Position Bias

- User click data is information collected based on which search results people click on
- This can be used to train ranking models like LambdaMART, however this data can be biased for obvious reasons
  - Position bias
- XGB has unbiased LambdaMART algorithm which debiases this data

Loss

- Different LambdaMart objectives based on different metrics
- Examples:
  - Normalized Discounted Cumulative Gain (NDCG), good when not sure about your data.
    - Binary relevance or multi-level relevance
  - Mean average precision (MAP), binary measure used when relevance label is 0 or 1
- LTR (learning to rank) metrics may be more effective but a theoretical lambda loss function may provide insight

Constructing Pairs

- Two main strategies, mean method and took method.
- For mean, XGB samples pairs for each document in a query list.
    - Ex. Given 3 documents and the parameter set to 2, XGB will randomly sample 6 pairs, assuming the labels are different
- Topic constructs around k x |query| number of pairs for each sample at the top k rank positions
    - The number of pairs is an approximation as pairs with the same label are skipped

Obtaining Good Result

- Mean Method: Samples lambdarank_num_pair_per_sample pairs per document in a query list. Good for generalizing with random sampling.
- TopK Method: Focuses on generating pairs for the top-ranked documents. Useful if prioritizing higher-ranked results is needed.

- If data has multi-level relevance: Use rank:ndcg or rank:pairwise.
- If data has binary labels: Choose based on the target metric (MAP, NDCG, etc.), ensuring sufficient effective pairs.

- Effective Pairs:
    - More effective pairs lead to better training.
    - Mean-NDCG (entire query list) generates more effective pairs than NDCG@10 (top 10 results).

- Training Data Size Recommendations:
    - Large Training Data:
        - Use target-matching objective.
        - Prefer the TopK strategy if focusing on high-ranking documents.
    - Small Training Data:
        - Use NDCG or RankNet loss (rank:pairwise).
        - Prefer the Mean strategy for more effective pairs.
- Additional Options:
    - Adjust lambdarank_num_pair_per_sample for more effective pairs as needed.

Distributed Training

- Can scatter query group onto multiple workers
- Theoretically sound but can affect accuracy
- Users do not need to partition data based on query groups

Reproducible Result

- Results should be the same regardless of hardware
- Except when pair method is sent to mean, XGB uses random sampling so results may differ based on platform