Learning to Rank

What does it do?

It is a design experiment for a machine-learning based learning to rank algorithm. It helps us perceive in a detailed level the participating entities and their relationships in learning to rank.

LETOR Problem Definition:

A dataset consisting of a set of queries (Q), a set of documents(D) and their sets of relevance scores(Y).

For each $q_i \in Q$ we have (m) documents from (D) $d_1, d_2, d_3 \ldots\ldots, d_m$.

Goal: To find the optimal ordering $\pi_i \ for \ q_i$ using the function $f(q_i$,D)

Each $d_{i,j}$ has a $y_{i,j}$ for q_i where j = 1 to m. and $y_{i,j}$ is the relevance score for document j in q_i's set of documents. The training set consists of a DataPoint object that represents $d_{i,j}$ where the document id and the label are fields in the object. Training set is $\theta(q_i, \emptyset(d_{i,j}, y_{i,j}))$ where $\emptyset$ represents the datapoint object and θ the composition of the entire training set. The test set consists only $\theta(q_i)$ for which we have to return the $\theta(d_i)$ where θ represents a set from 1 to m for documents and a set 1 to k for queries.

Design Trade-offs:

1. Choosing between Point-wise, Pair-wise and List-wise algorithm
    a. Point-wise methods look at each document for obtaining the rank
    b. Pair-wise methods look at document pairs (d_a, d_b) to identify the ordering between the two $Rank_{d_a} > Rank_{d_b}$.
    c. List-wise methods look at the entire list of documents and try to obtain the optimal ranking for the entire list.

    The list-wise methods are fairly complex as they look at all the documents and hence seems to be a good choice to perform a design experiment on.

Questions

1. What is the difference between Listwise Coordinate Ascent and Pointwise Linear Regression?

    In Pointwise Linear Regression we need to re-order the list to get an optimal ordering based on the given relevance scores to get the optimal ranking for each document. We then fit the linear regression equation to solve for ranking given the optimal ranking and the training data.

    In Listwise Coordinate Ascent for the given features we try to find the optimal ordering for the entire list by computing a score using a linear combination of the weight vector and the feature vector and sorting the list based on the newly computed score. Learning the weight vector is through a process called Coordinate Ascent as we fix coordinates and modify one weight at a time to maximize the objective function that is applied list-wise.

For those from a machine learning background, coordinate ascent is similar to gradient ascent. But instead of finding the gradient of the function we turn the knobs on the parameters to maximize the objective function.

Reaching this understanding was by referencing online resources on related topics. Thanks to the Google Box.

Software Libraries

The below libraries were used to hold the scores and the datapoints and for all list sorting machinery

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map.Entry;
```

In addition to the API given the design has a CoordinateAscent Class

Methods:

1. public void train(List<DataPoint>[] trainingData)
   To do the coordinate Asent
2. public HashMap<DataPoint, Double> sortMap(HashMap<DataPoint, Double> inputMap)
   To sort the scored document list
3. public List<DataPoint> predict(String query, List<String> candidateDocIds)
   To predict a new permutation at test time
4. public static void printResult(List<DataPoint> finalList)
   To print the list by doc ids.

Co-ordinate Ascent VS Evaluation Metrics:

AP – Average Precision

Average Precision is a binary judgement based evaluation metric – It considers only if a document is Relevance Or Not Relevant. Precision is how many of the relevant documents did we rank right. And Average Precision is mean over the total number of relevant documents for each query. The order between the set of relevant documents is not penalized.

With coordinate ascent, when we try to optimize the Average Precision, features that uniquely define the query-document pair will need to be weighted more than irrelevant documents but this difference can be coarse and the algorithm will converge quickly. As it only tries to put the good over the bad and does not care about the ordering among the good.

RR - Reciprocal Rank

Given a list of results with ranks, the reciprocal rank is – the reciprocal of the rank of the first relevant document in the list. If 3rd doc is relevant – then it is 1/3, if 2nd doc is relevant – ½. The objective function when we try to maximize will aim at only getting the most relevant document first. Like AP, it is a binary evaluation metric that differentiates the relevant document form the irrelevant document.

Coordinate Ascent with RR, like with AP will converge quickly. Its focus is to bring the most relevant document to the first rank. Ranks of other relevant documents are ignored. If we would like to get a better result for all the relevant documents over the irrelevant documents then Average Precision is better than RR.

NDCG – Normalized Discounted Cumulative Gain

It is an evaluation measure that allows for varying degrees of relevance judgements not just binary. It penalizes based on the rank of relevant document in the whole list. It pays closer attention to the rank of each document and their relevance.

$$\boldsymbol{DCG} = \sum_{\boldsymbol{i}}^{\boldsymbol{p}} \frac{\boldsymbol{2^{r_i}-1}}{\boldsymbol{log\,(i+1)}}$$

The formula in short penalizes logarithmically for highly relevant documents which come later in the list. In order to obtain a fair comparison between performance across two sets of queries DCG is normalized using IDCG which is the score for the optimal ordering of the list for the given query -document set.

$$\boldsymbol{NDCG} = \frac{\boldsymbol{DCG}}{\boldsymbol{IDCG}}$$

The best score from NDCG is 1.

Coordinate Ascent takes a longer time to converge on NDCG as the objective function keeps pushing the algorithm further up for every document it gets right in the list. Since the permutations for a list can be large the time taken for the algorithm to optimize NDCG maybe longer. There are several techniques to reduce the computational time, some of them are:

1. Perform fixed # of iterations to optimize NDCG – the drawback of this Is that we might not reach a global maximum
2. Incrementally update NDCG rather than computing it fully – a highly math intensive technique was proposed in one of Masters thesis by Wright State University in 2012.

Even though NDCG seems to take more time, it is highly preferred over the other binary evaluation measures.

Coordinate Ascent as an algorithm is very expensive as it performs a sort for every change in the parameter. Optimizing the objective function list wise needs to go through this permute function and cannot be avoided. Since we tune the objective function for the training set closely, this causes us to overfit on the training set and not generalize well for unseen data.