Ref:https://towardsdatascience.com/ranking-basics-pointwise-pairwise-listwise-cd5318f86e1b/

https://yjoonjang.medium.com/re-ranker-%EB%AA%A8%EB%8D%B8-%ED%95%99%EC%8A%B5%EC%9D%84-%EC%9C%84%ED%95%9C-%EB%8B%A4%EC%96%91%ED%95%9C-loss%EB%A5%BC-%EC%95%8C%EC%95%84%EB%B3%B4%EC%9E%90-feat-learning-to-rank-listnetloss-lambdaloss-1a18b9697efb

3 types of LTR

- Pointwise Approach
- Pairwise Approach
- Listwise Apporach

Pointwise Approach

> we treat the ranking problem as a simple classification task. For each query-document pair, we assign a target label that indicates the relevance of the document to the query. For example:
> - Label 1 if the document is relevant.
> - Label 0 if the document is not relevant.

Concept: This method treats each individual item (document) as an independent data point. For a given query, the model predicts a relevance score for **each item in isolation**. The ranking is then determined by sorting these predicted scores.

- **How it Works:** It transforms the ranking task into a standard classification (e.g., relevant/not relevant) or regression (e.g., predicting a relevance score from 0-5) problem.

- **Pros:**
- **Simplicity:** Easiest to implement, as it leverages well-established machine learning algorithms for classification and regression.
- **Scalability:** Highly scalable and suitable for large datasets, as each item's prediction is independent.

- **Cons:**
- **Ignores Relative Order:** Does not directly optimize for the relative order of items within a list. It predicts individual relevance, not the list's overall quality.

- **Mismatch with Ranking Metrics:** May not directly optimize for common ranking metrics (e.g., NDCG, MAP) that consider the entire list's structure.

Example:

- "On a scale of 1 to 5, what is the relevance score of this document?" (Regression)
- "Is this document relevant or not relevant?" (Binary Classification)

Models:
Almost any standard classification or regression model can be used.
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees
- Random Forests
- General Neural Networks (e.g., Multi-Layer Perceptrons)

Pairwise Approach

> focus on **pairs of documents** for the same query and try to predict which one is more relevant. This helps incorporate the context of comparison between documents.

Concept: The pairwise approach transforms the ranking problem into a **binary classification task** where the model learns to predict the **relative order** between two items. Instead of predicting an absolute relevance score for a single item, it determines which of two given items is "better" or "more relevant" than the other.

- **How it Works:** It converts the ranking problem into a binary classification task: given two items, predict which one should appear higher in the ranked list. The final ranking is then derived by aggregating these pairwise preferences.

- **Pros:**
- **Directly Learns Relative Order:** More aligned with the nature of ranking, as it explicitly models the preference between items.
- **Robustness:** Less sensitive to the absolute relevance scores of individual items, focusing instead on their relative positions.

- **Cons:**
- **Data Explosion:** Can lead to a quadratic increase in training data pairs for each query, making training computationally intensive for large lists.
- **Transitivity Issues:** Ensuring consistent transitive preferences (if A > B and B > C, then A > C) can be challenging.
- **Still Not Global:** While better than Pointwise, it still doesn't directly optimize the quality of the *entire* ranked list

Structure of pairwise:

- ***q1,(d1,d2)***→label: 1(indicating ***d1*** is more relevant than ***d2***)
- ***q1,(d2,d3)***→label: 0 (indicating ***d2*** is less relevant than ***d3***)
- ***q1,(d1,d3)***→label: 1 (indicating ***d1*** is more relevant than ***d3***)
- ***q2,(d4,d5)***→label: 0(indicating ***d4*** is less relevant than ***d5***)

Pairwise Approach - Models

Specific models designed for pairwise comparisons, often leveraging neural networks or SVMs.
- **RankNet**: A prominent neural network-based model that takes two items as input, computes their individual scores through a shared network, and then uses a sigmoid function to predict the probability that one item is preferred over the other. The loss function optimizes this probability.

- **RankSVM**: An SVM-based approach that aims to find a hyperplane that separates preferred items from less preferred ones in a feature space.
- Other neural network architectures adapted for comparative learning.

- Example:
- (Document A, Document B) - "Document A is preferred over Document B"
- (Document C, Document D) - "Document D is preferred over Document C"
- This often comes from explicit user feedback (e.g., A was clicked more than B when both were shown), implicit feedback (e.g., A was shown above B and clicked), or expert judgments.

Listwise Approach

> to optimize the entire list of documents based on their relevance to a query. Instead of treating individual documents separately, the focus is on the order in which they appear in the list.

Concept: This is the most holistic approach, where the model considers and optimizes the **entire list of items** for a given query as a single unit.

- **How it Works:** It directly optimizes for a global ranking metric (like NDCG or MAP) that evaluates the quality of the entire ranked list. The model learns to produce an optimal ordering of all relevant items.

- **Pros:**
- **Direct Optimization of Ranking Metrics:** Most closely aligns the training objective with the ultimate goal of producing high-quality ranked lists.
- **Considers Inter-item Relationships:** Can inherently capture dependencies and interactions among items within the list.

- **Cons:**
- **Complexity:** Most complex to implement, often requiring specialized algorithms and neural network architectures.
- **Computational Cost:** Training and inference can be computationally expensive as the entire list needs to be processed together.
- **Data Requirements:** Requires more sophisticated data labeling and preparation, as the "ground truth" is a ranked list rather than individual scores or pairs

Listwise Ranking: ListNet and LambdaRank

NDCG(Normalized Discounted Cumulative Gain)

- It **rewards highly relevant items** appearing at the top of the list.
- It **penalizes relevant items** appearing lower down (discounting their value).
- It **normalizes the score** (typically between 0 and 1) to allow for fair comparison across different queries or lists.

Models:

These models are typically more complex and often involve sophisticated neural networks or ensemble methods specifically designed to handle sequences or permutations.

- **ListNet**: A neural network that learns a ranking function by optimizing a loss function that compares the predicted ranking with the ground-truth ranking. It often uses a probabilistic approach to model permutations.

- **ListMLE** (List Maximum Likelihood Estimation): Aims to maximize the likelihood of observing the true permutation of items.
- **LambdaRank / LambdaMART: These are highly effective and widely used listwise algorithms (LambdaRank is the loss function used by LambdaMART). Instead of directly optimizing a ranking metric, they optimize a "lambda" value for each document, which represents the change in a ranking metric (like NDCG) if that document's rank were swapped. This makes the optimization process more efficient and directly relevant to ranking metrics.
- **Deep Learning Models for Sequence-to-Sequence Ranking**: Modern approaches using Transformer networks or other sequence models can be adapted to take a set of items and output a ranked sequence.