Ranking

Let's see how to design the search ranking model.

We'll cover the following

Objective#

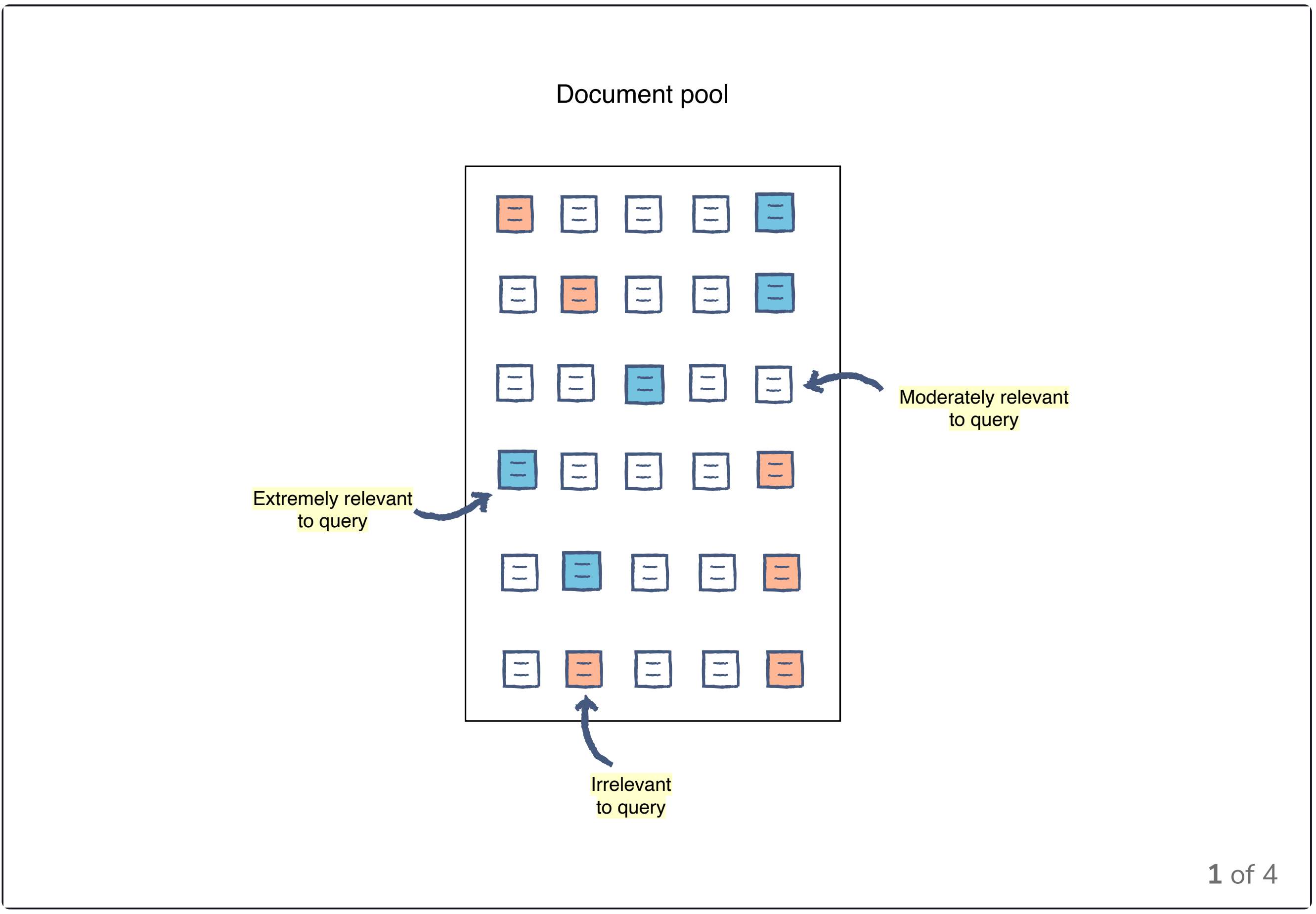
You want to build a model that will display the most relevant results for a searcher's query. Extra emphasis should be applied to the relevance of the top few results since a user generally focuses on them more on the SERP.

> 📝 **Learning to Rank (LTR)**: A class of techniques that applies supervised machine learning (ML) to solve ranking problems. The pointwise and pairwise techniques that we will apply fall under this class.

Stage-wise approach#

As discussed in the architectural components section, the number of documents matching a single query can be very large. So, for a large scale search engine, it makes sense to adopt a multi-layer funnel approach. The top layer of the funnel looks at a large number of documents and uses simpler and faster algorithms for ranking. The bottom layer ranks a small number of documents with complex machine-learned models.

Let's assume here that you will use two layers for ranking with one-hundred thousand and five-hundred documents ranked at each layer, as shown below. Though, the choice of layers and documents ranked on each layer depends highly on the available capacity.



The configuration shown above assumes that the first stage will receive one-hundred thousand relevant documents from the document s or component. You then reduce this number to five-hundred after ranking in this layer, ensuring that the topmost relevant results are forwarded to the second stage (also referred to as the recall of the documents).

It will then be the responsibility of the second stage to rank the documents such that topmost relevant results are placed in the correct order (also referred to as the precision of the documents).

> 📝 First stage model will focus on the **recall** of the top five to ten relevant documents in the first five-hundred results while the second stage will ensure **precision** of the top five to ten relevant documents.

Stage 1#

As we try to limit documents in this stage from a large set to a relatively smaller set, it's important that we don't miss out on highly relevant documents for the query from the smaller set. So, this layer needs to ensure that the top relevant documents are forwarded to stage 2. This can be achieved with the **pointwise approach**, which has been discussed previously. The problem can be approximated with the binary classification of results as relevant or irrelevant.

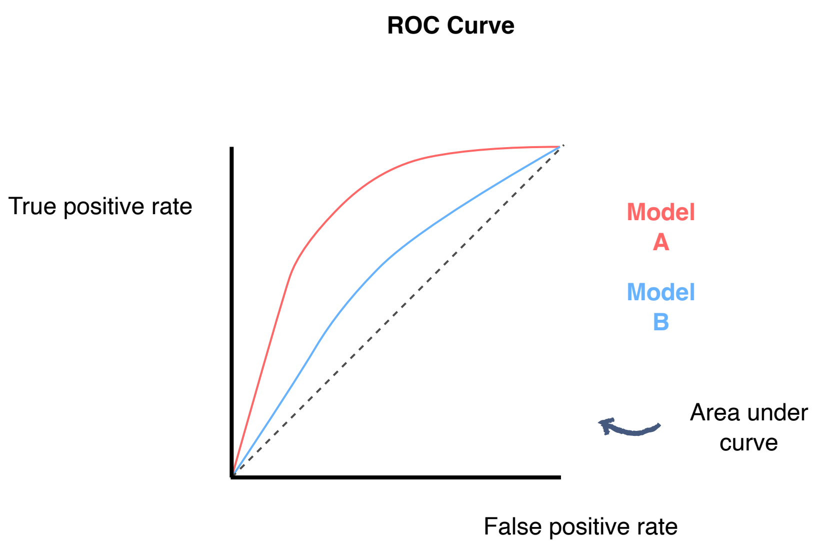
Logistic regression#

A relatively less complex linear algorithm, like logistic regression or small MART(Multiple additive regression trees) model, is well suited for scoring a large set of documents. The ability to score each document extremely quickly (microseconds or less) for the fairly large document pool at this stage is super critical.

Analyzing performance#

To analyze the performance of our model, we will look at the *area under curve* (AUC) of receiver operating characteristics curves or ROC curves.

For instance, if we train two models A and B on different feature sets, AUC will help us determine which model performs better.



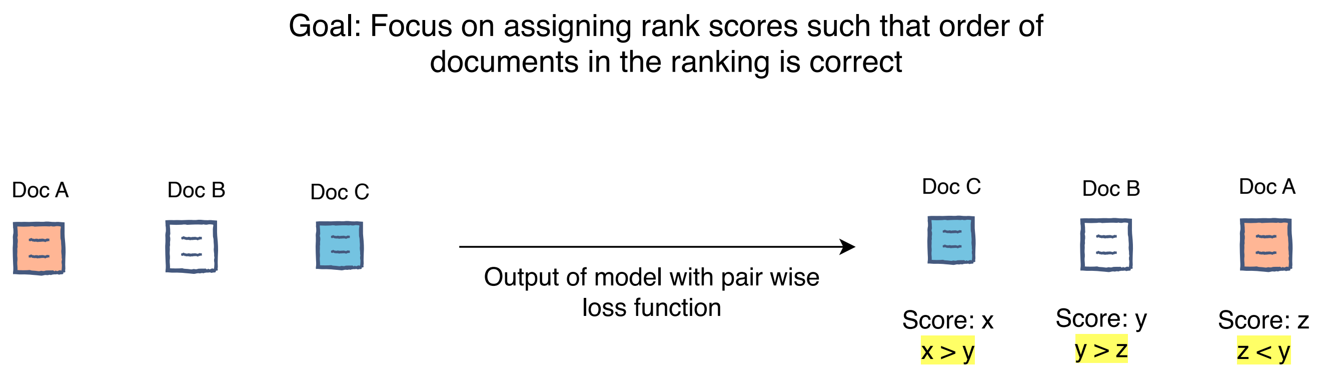
Model A is superior than Model B

We can observe that Model A is better than Model B because the area under its curve is greater.

Stage 2#

As discussed previously, the main objective of our stage 2 model is to find the optimized rank order.

This is achieved by changing the objective function from a single pointwise objective (click, session success) to a pairwise objective. Pairwise optimization for learning to rank means that the model is not trying to minimize the classification error but rather trying to get as many pairs of documents in the right order as possible.


Goal: Focus on assigning rank scores such that order of documents in the ranking is correct
Doc A
Doc B
Doc C
Output of model with pair wise loss function
Doc C
Doc B
Doc A
Score: x
x > y
Score: y
y > z
Score: z
z < y

Let’s look at two **pairwise learning algorithms**.

LambdaMART#

LambdaMART is a variation of MART where we change the objective to improve pairwise ranking, as explained above. Tree-based algorithms are generally able to generalize effectively using a moderate set of training data. Therefore, if your training data is limited to a few million examples, this definitely will be the best choice to use in pairwise ranking in the second stage.

If we are optimizing for offline NDCG (based on human-rated data), then this is definitely one of the best options.

LambdaRank#

LambdaRank is a neural network-based approach utilizing pairwise loss to rank the documents. Neural network-based models are relatively slower (given the large number of computations based on width and depth) and need more training data. So training data size and capacity are key questions before selecting this modeling approach. The online training data generation method for the pairwise approach can generate ranking examples for a popular search engine in abundance. So, that can be one option to generate ample pairwise data.

Your training data contains pairs of documents (*i* , *j*), where *i* ranks higher than *j*. Let's look at the Lambda rank model's learning process. Suppose we have to rank two documents *i* and *j* for a given query. We feed their corresponding feature vectors x_i and x_j to the model, and it gives us their relevance scores, i.e., s_i and s_j. The model should compute these scores (s_i and s_j) such that the probability of document *i* being ranked higher than document *j* is close to that of the ground truth. The optimization function tries to minimize the inversions in the ranking.

Both LambdaMART and LambdaRank are very well explained in this paper.

> 📝 We can calculate the NDCG score of the ranked results to compare the performance of different models.