

CS 589 Lecture 5: Learning to Rank - Complete Study Guide

Table of Contents

1. Overview and Historical Context
 2. Machine Learning Fundamentals Review
 3. Learning to Rank Approaches
 4. Gradient Boosted Regression Trees (GBRT)
 5. RankNet
 6. LambdaRank and LambdaMART
 7. Deep Learning for Ranking
 8. Practice Problems
 9. Key Exam Topics
-

1. Overview and Historical Context {#overview}

What is Learning to Rank?

Learning to Rank (LTR) is the application of machine learning to build ranking models for information retrieval systems. Instead of manually tuning ranking functions (like BM25 parameters), we use supervised learning to automatically learn the optimal ranking function from data.

Historical Timeline

- **1950s-1970s:** Early IR systems with hand-crafted ranking functions (TF-IDF)
- **1980s:** Introduction of probabilistic ranking principle, learning to rank concepts emerge
- **1990s-2000s:** Web search explosion, need for combining many features
- **2003:** Gradient boosting algorithms
- **2010:** LambdaMART wins Yahoo! Learning to Rank Challenge
- **2016:** XGBoost becomes standard
- **2019+:** Deep learning approaches (BERT, Dense Passage Retrieval)

Why Learning to Rank?

Traditional IR challenges:

- Hand-tuning parameters is difficult
- Hard to combine multiple signals (PageRank, anchor text, click data, etc.)

- Different queries may need different ranking strategies

LTR benefits:

- Automatically learns optimal feature weights
- Can combine 100+ features (modern systems use 500+)
- Adapts to user behavior through training data

2. Machine Learning Fundamentals Review {#ml-fundamentals}

2.1 Linear Regression

Problem: Predict a continuous value \hat{y} from input features x

Model:

$$\hat{y} = w^T x + b$$

Loss Function (Mean Squared Error):

$$L(w,b) = (1/n) \sum_i (y_i - (w^T x_i + b))^2$$

Key limitation: Cannot model non-linear relationships

2.2 Gradient Descent

Optimization Algorithm to minimize loss function $L(\theta)$:

Initialize: θ^0 = random or zero

For $t = 1, 2, \dots, T$:

$$\theta^t = \theta^{t-1} - \lambda \frac{\partial L}{\partial \theta}$$

Where:

- λ = learning rate (step size)
- $\frac{\partial L}{\partial \theta}$ = gradient of loss with respect to parameters

Intuition: Move in the direction of steepest descent to find the minimum.

 **Recommended Video:** [Gradient Descent by 3Blue1Brown](#)

2.3 Decision Trees vs Regression Trees

Decision Tree	Regression Tree
Predicts class labels	Predicts continuous values
Leaf nodes contain class	Leaf nodes contain numeric predictions

Decision Tree	Regression Tree
Uses entropy/Gini for splits	Uses MSE/variance for splits

Regression Tree Splitting Criterion:

For each possible split, minimize the variance:

◀ Split on feature j at value v to minimize:

$$\text{MSE} = \sum(\text{left}) (y_i - \bar{y}_{\text{left}})^2 + \sum(\text{right}) (y_i - \bar{y}_{\text{right}})^2$$

▶

3. Learning to Rank Approaches {#ltr-approaches}

3.1 Problem Formulation

Given:

- Query q
- Set of documents $D = \{d_1, d_2, \dots, d_n\}$
- Feature vectors $x = (x_1, x_2, \dots, x_k)$ for each (query, document) pair
- Relevance labels y (binary or graded)

Goal: Learn a ranking function $f(q, d)$ that produces a score for sorting documents

3.2 Feature Engineering

Common features (modern systems use 500+):

Text Matching Features:

- Cosine similarity
- BM25 score
- Term proximity (minimum query window size)
- Query term coverage

Document Features:

- Document length
- URL length
- Presence of query in URL
- Page loading speed
- Number of images

Link-based Features:

- PageRank
- Number of inbound/outbound links
- Anchor text match

User Features:

- Click-through rate
- Dwell time
- User demographics
- Search history

3.3 LTR Categories

Three main approaches:

1. Pointwise: Treat each document independently

- Classification: relevant vs non-relevant
- Regression: predict relevance score
- Examples: Linear Regression, Logistic Regression

2. Pairwise: Learn from document pairs

- Goal: correctly order pairs
- Examples: RankNet, LambdaRank, LambdaMART

3. Listwise: Optimize on entire ranking list

- Direct optimization of ranking metrics
- Examples: SVM^{MAP}, NDCGBoost

4. Gradient Boosted Regression Trees (GBRT) {#gbrt}

4.1 Core Concept

Key Idea: Build an ensemble of weak learners (regression trees) sequentially, where each tree corrects the errors of the previous ones.

4.2 Algorithm

Input: Training data $\{(x_i, y_i)\}$, number of trees M , learning rate η

1. Initialize: $F_0(x) = \operatorname{argmin}_{\gamma} \sum_i L(y_i, \gamma)$
(Usually $F_0(x) = \operatorname{mean}(y)$ for MSE loss)

2. For $m = 1$ to M :

a) Compute residuals (pseudo-residuals):

$$r_{im} = -[\partial L(y_i, F(x_i)) / \partial F(x_i)]|_{F=F_{m-1}}$$

For MSE: $r_{im} = y_i - F_{m-1}(x_i)$

b) Fit regression tree $h_m(x)$ to residuals $\{r_{im}\}$

This creates regions $\{R_{km}\}_{k=1}^K$

c) For each leaf k , compute optimal value:

$$\gamma_{km} = \operatorname{argmin}_{\gamma} \sum_{(x_i \in R_{km})} L(y_i, F_{m-1}(x_i) + \gamma)$$

d) Update model:

$$F_m(x) = F_{m-1}(x) + \eta \sum_k \gamma_{km} \mathbb{1}(x \in R_{km})$$

3. Return: $F_M(x)$

4.3 Step-by-Step Example

Data:

$x_1: (1,1) \rightarrow y=2$
 $x_2: (1,2) \rightarrow y=2$
 $x_3: (2,1) \rightarrow y=2$
 $x_4: (2,2) \rightarrow y=2$
 $x_5: (3,3) \rightarrow y=6$
 $x_6: (3,4) \rightarrow y=6$
 $x_7: (4,3) \rightarrow y=6$
 $x_8: (4,4) \rightarrow y=6$
 $x_9: (5,5) \rightarrow y=5$
 $x_{10}: (5,6) \rightarrow y=5$
 $x_{11}: (6,5) \rightarrow y=5$

Iteration 0:

$$F_0(x) = \operatorname{mean}(y) = (2+2+2+2+6+6+6+6+5+5+5)/11 = 3.818$$

Iteration 1:

Compute residuals:

$$r_1 = 2 - 3.818 = -1.818$$

$$r_2 = 2 - 3.818 = -1.818$$

...

$$r_5 = 6 - 3.818 = 2.182$$


...

$$r_9 = 5 - 3.818 = 1.182$$

Fit regression tree to minimize MSE. Best split: $x_1 < 5.5$

- Left branch: $\text{mean}(r_1 \dots r_8) = \dots$
- Right branch: $\text{mean}(r_9 \dots r_{11}) = \dots$

Continue iteratively...

 **Recommended Video:** [Gradient Boosting by StatQuest](#)

4.4 Key Properties

Advantages:

- Handles non-linear relationships
- Automatically does feature selection
- Robust to outliers
- Works well with mixed data types

Hyperparameters:

- Number of trees M
- Learning rate η (smaller = more trees needed)
- Tree depth (typically 3-10)
- Minimum samples per leaf

5. RankNet {#ranknet}

5.1 Motivation

Problem with regression approach:

- Exact relevance scores are hard to judge consistently
- "Is this document exactly a 3 or a 4?"

Solution: Use pairwise preferences

- "Document A is better than Document B" is easier to judge
- More consistent across annotators and time

5.2 Mathematical Formulation

Model: For document pair (d_i, d_j) , model the probability that d_i is more relevant than d_j :

$$P_{ij} = P(d_i > d_j) = \sigma(s_i - s_j) = 1 / (1 + e^{-(s_i - s_j)})$$

Where:

- $s_i = w^T x_i + b$ is the score for document i
- σ is a scaling parameter

Ground truth label:

$$S_{ij} = \begin{cases} 1 & \text{if } d_i \text{ more relevant than } d_j \\ 0 & \text{if } d_j \text{ more relevant than } d_i \\ 0.5 & \text{if equally relevant} \end{cases}$$

Or equivalently: $S_{ij} = (1 + \text{sign}(y_i - y_j))/2$ where y_i is the relevance label.

5.3 Loss Function

Cross-entropy loss for pair (i, j) :

$$C_{ij} = -S_{ij} \log(P_{ij}) - (1 - S_{ij}) \log(1 - P_{ij})$$

Substituting P_{ij} :

$$C_{ij} = (1/2)(1 - S_{ij})\sigma(s_i - s_j) + \log(1 + e^{-(s_i - s_j)})$$

Total loss: Sum over all pairs:

$$C = \sum_{ij} C_{ij}$$

5.4 Gradient Computation ★ EXAM CRITICAL

Taking derivative with respect to parameter w_k :

$$\begin{aligned} \partial C_{ij} / \partial s_i &= \sigma(1/2(1 - S_{ij}) - e^{-(s_i - s_j)} / (1 + e^{-(s_i - s_j)})) \\ &= -\partial C_{ij} / \partial s_j \end{aligned}$$

Simplifying:

$$\begin{aligned} \partial C_{ij} / \partial s_i &= \sigma(1/2(1 - S_{ij}) - 1 / (1 + e^{-(s_i - s_j)})) \\ &= \sigma \lambda_{ij} \end{aligned}$$

Where:

$$\lambda_{ij} = (1/2)(1 - S_{ij}) - 1/(1 + e^{(\sigma(s_i - s_j))})$$

By chain rule:

$$\begin{aligned}\partial C_{ij} / \partial w_k &= (\partial C_{ij} / \partial s_i)(\partial s_i / \partial w_k) + (\partial C_{ij} / \partial s_j)(\partial s_j / \partial w_k) \\ &= \lambda_{ij}(\partial s_i / \partial w_k - \partial s_j / \partial w_k)\end{aligned}$$

5.5 From Pairwise to Pointwise ★ EXAM CRITICAL

Key insight: We can express the gradient as a sum over documents:

$$\begin{aligned}\partial \Sigma / \partial w_k &= \sum_{ij} \lambda_{ij}(\partial s_i / \partial w_k - \partial s_j / \partial w_k) \\ &= \sum_i (\sum_{j: i > j} \lambda_{ij} - \sum_{l: l > i} \lambda_{il})(\partial s_i / \partial w_k) \\ &= \sum_i \lambda_i(\partial s_i / \partial w_k)\end{aligned}$$

Where λ_i is the gradient for document i:

$$\lambda_i = \sum_{j: i > j} \lambda_{ij} - \sum_{l: l > i} \lambda_{il}$$

Interpretation:

- λ_i represents the "force" pushing document i up or down in the ranking
- Positive $\lambda_i \rightarrow$ document should be ranked higher
- Negative $\lambda_i \rightarrow$ document should be ranked lower

5.6 Visual Interpretation (Slide 40)

For a ranking with errors:

Ranking: [d₁] [d₂] [d₃] [d₄] ... [d₁₀]

Labels: R R N N ... R

Arrows show λ_i forces:

- Relevant docs (R) get upward arrows
- Non-relevant docs (N) get downward arrows
- Arrow size \propto magnitude of error

Example:

- (d₁) (relevant, position 1): Small upward arrow (already high)
- (d₃) (non-relevant, position 3): Downward arrow (should be lower)
- (d₁₀) (relevant, position 10): Large upward arrow (big error!)

5.7 Implementation Notes

Training:

1. For each query, generate all pairs (d_i, d_j) where $y_i \neq y_j$
2. Compute λ_{ij} for each pair
3. Aggregate to get λ_i for each document
4. Update weights using gradient descent

Prediction:

1. Compute score $s_i = w^T x_i$ for each document
 2. Sort documents by score
-

6. LambdaRank and LambdaMART {#lambdarank}

6.1 Motivation

Problem with RankNet: All pairwise errors are treated equally

Observation: In ranking, some errors are more costly than others!

- Swapping ranks 1 and 2 \rightarrow Big impact on NDCG
- Swapping ranks 99 and 100 \rightarrow Minimal impact

Solution: Weight the gradients (lambdas) by the change in evaluation metric

6.2 LambdaRank

Modified gradient:

$$\lambda_{ij} = -\sigma/(1 + e^{(\sigma(s_i - s_j)))} \cdot |\Delta \text{Metric}|$$

Where $|\Delta \text{Metric}|$ is the absolute change in the evaluation metric (e.g., NDCG) if we swap d_i and d_j .

For NDCG:

$$\lambda_{ij} = -\sigma/(1 + e^{(\sigma(s_i - s_j)))} \cdot |\text{NDCG}(\text{swap } i, j) - \text{NDCG}(\text{current})|$$

Intuition:

- Pairs at top of ranking get larger gradients
- Pairs far down get smaller gradients
- Focus learning on getting the top results right

6.3 NDCG Computation (Review)

DCG (Discounted Cumulative Gain):

$$DCG_k = \sum_{i=1}^k (2^{\text{rel}_i} - 1) / \log_2(i + 1)$$

NDCG (Normalized DCG):

$$NDCG_k = DCG_k / IDC_{G_k}$$

Where IDC_{G_k} is the ideal DCG (using perfect ranking).

Example:

Ranking: [d₁:3, d₂:2, d₃:3, d₄:0, d₅:1]

Ideal: [d₁:3, d₃:3, d₂:2, d₅:1, d₄:0]

$$\begin{aligned} DCG &= (2^3-1)/\log_2(2) + (2^2-1)/\log_2(3) + (2^3-1)/\log_2(4) + \dots \\ &= 7/1 + 3/1.585 + 7/2 + 0/2.322 + 1/2.585 \\ &= 7 + 1.893 + 3.5 + 0 + 0.387 \\ &= 12.78 \end{aligned}$$

$$\begin{aligned} IDC_{G_k} &= (2^3-1)/\log_2(2) + (2^3-1)/\log_2(3) + (2^2-1)/\log_2(4) + \dots \\ &= 7 + 4.416 + 1.5 + 0.431 + 0 \\ &= 13.347 \end{aligned}$$

$$NDCG = 12.78/13.347 = 0.958$$

6.4 LambdaMART ★ EXAM CRITICAL

Combination: LambdaRank + MART (Multiple Additive Regression Trees)

Algorithm:

Input:

- Training data $\{(x_i, y_i)\}$
- Number of trees N
- Learning rate η

1. Initialize $F_0(x) = 0$

2. For $m = 1$ to N :

a) For each query q and documents D_q :

- Compute current scores: $s_i = F_{m-1}(x_i)$
- Compute lambdas: λ_i based on NDCG

b) Fit regression tree h_m to $\{(x_i, \lambda_i)\}$

- Use λ_i as pseudo-residuals
- Tree splits to minimize: $\sum(\lambda_i - h_m(x_i))^2$

c) For each leaf region R_{km} :

- Compute optimal weight: w_k (Newton step)

d) Update: $F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x)$

3. Return $FM(x)$

Key differences from standard GBRT:

- Use λ_i (weighted by NDCG change) instead of simple residuals
- Optimization directly targets ranking metric
- Still benefits from GBRT's ability to handle non-linear relationships

6.5 Comparison of LTR Algorithms

Algorithm	Type	Optimizes	Pros	Cons
Linear Regression	Pointwise	MSE	Simple, interpretable	Can't handle non-linearity
RankNet	Pairwise	Cross-entropy	Learns from pairs	All pairs equal weight
LambdaRank	Pairwise	Metric-weighted	Focuses on top results	More complex
LambdaMART	Pairwise	Metric-weighted + GBRT	State-of-art performance	Computationally expensive
SVM ^{MAP}	Listwise	MAP	Direct metric optimization	Hard to optimize

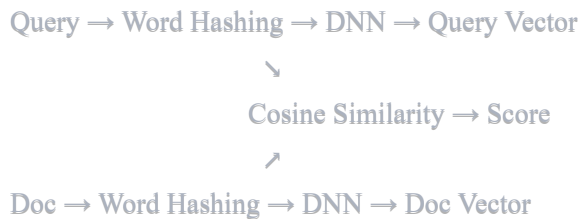
7. Deep Learning for Ranking {#deep-learning}

7.1 Neural Matching Models

Paradigm shift: Instead of hand-crafted features, learn representations

7.2 DSSM (Deep Structured Semantic Model)

Architecture:



Training: Maximize cosine similarity for relevant pairs, minimize for irrelevant

7.3 Match Pyramid

Key Idea: Create a 2D similarity matrix between query and document terms

Process:

1. Embed each word: $w \rightarrow \text{embedding vector}$
2. Create interaction matrix: $M[i,j] = \text{similarity}(q_i, d_j)$
3. Apply CNN to extract patterns
4. Fully connected layers \rightarrow relevance score

Similarity measures:

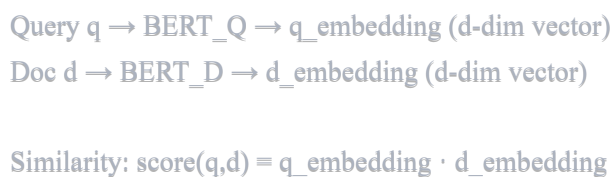
- Cosine similarity
- Dot product
- Indicator (exact match)

7.4 Dense Passage Retrieval (DPR)

Problem: Traditional IR relies on exact term matching. Semantic matching is challenging.

Solution: Encode queries and documents into dense vectors

Architecture:



Training (Contrastive Learning):

Loss function:

$$L(q, d^+, d_1^-, \dots, d_n^-) = -\log\left(\frac{e^{\text{sim}(q,d^+)}}{e^{\text{sim}(q,d^+)} + \sum_i e^{\text{sim}(q,d_i^-)}}\right)$$

Where:

- (d^+) = positive (relevant) passage
- (d^-_i) = negative (non-relevant) passages

Selecting negatives (very important!):

1. **Random negatives:** Sample randomly from corpus
2. **BM25 negatives:** Top-ranked by BM25 but not relevant (hard negatives)
3. **In-batch negatives:** Use other positives in the batch

Results (from paper):

- Adding hard BM25 negatives: +10 points in Top-20 accuracy
- DPR outperforms BM25 on Natural Questions dataset

7.5 Retrieval Process with DPR

Offline (Index building):

```
For each document d in corpus:  
  d_embedding = BERT_D(d)  
  Store in FAISS/Annoy index
```

Online (Query time):

```
1. q_embedding = BERT_Q(q)  
2. Find top-k most similar d_embeddings using approximate nearest neighbor  
3. Return corresponding documents
```

Efficiency: FAISS enables sub-linear search time even with millions of documents

8. Practice Problems {#practice-problems}

Problem 1: BM25 Features for LTR

Given documents and a query, extract features:

Query: "machine learning" **Documents:**

- D1: "Machine learning is a subset of AI" (length: 7)
- D2: "Learning machines learn from data" (length: 5)

Compute:

1. BM25 score for each document

2. Term proximity (minimum window size)

3. Document length normalized score

Solution:

1. **BM25 scores** (assuming IDF values given):

- Need term frequencies, document frequencies, etc.
- This would be computed using BM25 formula from Lecture 2

2. **Term proximity:**

- D1: "Machine" at position 0, "learning" at position 1 \rightarrow window = 2
- D2: "Learning" at position 0, "machines" at position 1 \rightarrow window = 2

3. **Normalized scores:**

- Apply pivoted document length normalization

Problem 2: Regression Tree Building

Data:

Features: (x_1 , x_2)

(1,1) \rightarrow 2

(2,1) \rightarrow 2

(1,2) \rightarrow 3

(2,2) \rightarrow 4

Build a regression tree with depth 1:

Solution:

Try all possible splits:

Split on $x_1 < 1.5$:

- Left: {(1,1) \rightarrow 2, (1,2) \rightarrow 3}, mean = 2.5, variance = 0.25
- Right: {(2,1) \rightarrow 2, (2,2) \rightarrow 4}, mean = 3, variance = 1
- Total MSE = 0.25 + 1 = 1.25

Split on $x_2 < 1.5$:

- Left: {(1,1) \rightarrow 2, (2,1) \rightarrow 2}, mean = 2, variance = 0
- Right: {(1,2) \rightarrow 3, (2,2) \rightarrow 4}, mean = 3.5, variance = 0.25
- Total MSE = 0 + 0.25 = 0.25 \checkmark **Best split**

Tree:

$$\begin{array}{cc} & x_2 < 1.5? \\ / & \backslash \\ 2.0 & 3.5 \end{array}$$

Problem 3: RankNet Gradient Calculation

Given:

- Document pair: d_1 (relevant), d_2 (non-relevant)
- Current scores: $s_1 = 0.3$, $s_2 = 0.7$
- $\sigma = 1$
- Ground truth: $S_{12} = 1$ (d_1 should rank higher)

Compute: λ_{12} and the direction of update

Solution:

$$\begin{aligned} \lambda_{12} &= 1/2(1 - S_{12}) - 1/(1 + e^{\sigma(s_1 - s_2)}) \\ &= 1/2(1 - 1) - 1/(1 + e^{1(0.3 - 0.7)}) \\ &= 0 - 1/(1 + e^{-0.4}) \\ &= -1/(1 + 0.67) \\ &= -1/1.67 \\ &= -0.60 \end{aligned}$$

Interpretation:

- Negative λ_{12} means d_1 should be pushed up (increase s_1)
- And d_2 should be pushed down (decrease s_2)
- This makes sense: currently $s_2 > s_1$ (wrong order), so we need to flip them

Problem 4: NDCG Calculation

Ranking: [$d_1:2$, $d_2:3$, $d_3:1$, $d_4:0$, $d_5:2$]

Compute NDCG@5:

Solution:

$$DCG@5 = \sum_{i=1}^5 (2^{\text{rel}_i} - 1) / \log_2(i+1)$$

$$= (2^2-1)/\log_2 2 + (2^3-1)/\log_2 3 + (2^1-1)/\log_2 4 + (2^0-1)/\log_2 5 + (2^2-1)/\log_2 6$$

$$= 3/1 + 7/1.585 + 1/2 + 0/2.322 + 3/2.585$$

$$= 3 + 4.42 + 0.5 + 0 + 1.16$$

$$= 9.08$$

$$IDCG@5 \text{ (ideal: [3,2,2,1,0])}$$

$$= (2^3-1)/\log_2 2 + (2^2-1)/\log_2 3 + (2^2-1)/\log_2 4 + (2^1-1)/\log_2 5 + 0$$

$$= 7 + 1.89 + 1.5 + 0.43 + 0$$

$$= 10.82$$

$$NDCG@5 = 9.08/10.82 = 0.839$$

Problem 5: Dense Passage Retrieval

Given:

- Query embedding: $q = [0.5, 0.3, 0.2]$
- Passage embeddings:
 - $p_1 = [0.6, 0.2, 0.2]$
 - $p_2 = [0.1, 0.8, 0.1]$
 - $p_3 = [0.5, 0.3, 0.2]$

Rank passages by similarity:

Solution:

Dot product similarity:

$$\text{sim}(q, p_1) = 0.5 \times 0.6 + 0.3 \times 0.2 + 0.2 \times 0.2 = 0.3 + 0.06 + 0.04 = 0.40$$

$$\text{sim}(q, p_2) = 0.5 \times 0.1 + 0.3 \times 0.8 + 0.2 \times 0.1 = 0.05 + 0.24 + 0.02 = 0.31$$

$$\text{sim}(q, p_3) = 0.5 \times 0.5 + 0.3 \times 0.3 + 0.2 \times 0.2 = 0.25 + 0.09 + 0.04 = 0.38$$

Ranking: $p_1 > p_3 > p_2$

9. Key Exam Topics {#exam-topics}

9.1 Must Know Concepts

✓ Feature Engineering

- Types of features for ranking
- Why combining features is important
- Understand query-dependent vs query-independent features

✓ LTR Categories

- Pointwise: treat each doc independently
- Pairwise: learn from pairs
- Listwise: optimize ranking metrics
- Pros/cons of each

✓ GBRT

- Difference from single regression tree
- What are residuals and why we fit trees to them
- How to compute predictions (sum of all trees)
- Time complexity of building splits: $O(n^2)$ per feature

✓ RankNet

- Probability formulation: $P(d_i > d_j)$
- Cross-entropy loss
- **KEY:** Gradient derivation (pairwise to pointwise)
- Interpretation of λ_i (slide 40)
- Why relative judgments are easier than absolute

✓ LambdaRank

- Why weight by metric change
- How to compute $|\Delta \text{NDCG}|$
- When to use LambdaRank vs RankNet

✓ LambdaMART

- Combination of LambdaRank + MART
- Use λ_i as residuals in GBRT
- State-of-the-art traditional ML method

✓ Deep Learning

- DSSM architecture
- Match Pyramid concept (2D interaction matrix)
- Dense Passage Retrieval:
 - Contrastive learning
 - Importance of hard negatives

- Why BM25 negatives help
- FAISS for efficient retrieval

9.2 Important Formulas

BM25 (from Lecture 2):

$$\text{score}(D, Q) = \sum_{t \in Q} \text{IDF}(t) \cdot (f(t, D) \cdot (k_1 + 1)) / (f(t, D) + k_1 \cdot (1 - b + b \cdot |D| / \text{avgdl}))$$

NDCG:

$$\begin{aligned} \text{NDCG}@k &= \text{DCG}@k / \text{IDCG}@k \\ \text{DCG}@k &= \sum_{i=1}^k (2^{\text{rel}_i} - 1) / \log_2(i+1) \end{aligned}$$

RankNet Probability:

$$P(d_i > d_j) = \sigma(s_i - s_j) = 1 / (1 + e^{-(s_i - s_j)})$$

RankNet Loss:

$$C_{ij} = 1/2(1 - S_{ij})\sigma(s_i - s_j) + \log(1 + e^{-(s_i - s_j)})$$

Lambda:

$$\lambda_{ij} = \partial C_{ij} / \partial s_i = \sigma(1/2(1 - S_{ij}) - 1/(1 + e^{-(s_i - s_j)}))$$

Dot Product Similarity:

$$\text{sim}(q, d) = q^T d = \sum_i q_i d_i$$

9.3 Common Exam Question Types

Type 1: Feature extraction

- Given query and documents, compute features
- BM25 score, term proximity, document length

Type 2: Build regression tree

- Given small dataset, manually build tree with depth 1
- Show split criterion calculations

Type 3: GBRT iterations

- Given initial model F_0 and data

- Compute residuals
- Describe next tree

Type 4: RankNet gradient

- Given document pair and scores
- Compute λ_{ij}
- Interpret direction of update

Type 5: NDCG calculation

- Given ranking and relevance labels
- Compute DCG, IDCG, NDCG

Type 6: Dense retrieval

- Given query and document embeddings
- Compute similarities
- Rank documents

Type 7: Conceptual questions

- Why pairwise better than pointwise?
- Why hard negatives important in DPR?
- When to use LambdaMART vs deep learning?

9.4 Comparison Table for Quick Review

Approach	Pros	Cons	Use When
Linear Regression	Fast, interpretable	Can't capture non-linearity	Baseline, simple features
GBRT	Handles non-linearity, feature selection	Slower to train	Medium-sized datasets
RankNet	Learns from pairs, more stable labels	Equal weight to all pairs	Standard pairwise ranking
LambdaMART	State-of-art traditional ML	Complex, slower	Production systems (pre-deep learning)
DPR	Semantic matching, end-to-end	Needs lots of data, expensive	Large-scale modern systems

10. Additional Resources

Stanford IR Book

- **Chapter 15:** Support Vector Machines and Machine Learning on Documents
- **Chapter 20:** Flat Clustering (for EM algorithm background)

Video Lectures

- Gradient Boosting - StatQuest
- Neural Networks - 3Blue1Brown
- BERT Paper Explained

Papers (Optional Deep Dive)

- Burges et al. (2010): "From RankNet to LambdaRank to LambdaMART: An Overview"
- Karpukhin et al. (2020): "Dense Passage Retrieval for Open-Domain Question Answering"
- Friedman (1999): "Greedy Function Approximation: A Gradient Boosting Machine"

Tools and Libraries

- **XGBoost:** <https://github.com/dmlc/xgboost>
 - **RankLib:** <https://sourceforge.net/p/lemur/wiki/RankLib/>
 - **Sentence Transformers** (for DPR): <https://www.sbert.net/>
 - **FAISS:** <https://github.com/facebookresearch/faiss>
-

Summary Checklist

Before the exam, make sure you can:

- ☐ Explain why learning to rank is better than hand-tuned formulas
- ☐ List and categorize common ranking features
- ☐ Distinguish pointwise, pairwise, and listwise approaches
- ☐ Build a simple regression tree by hand
- ☐ Explain gradient boosting residual fitting
- ☐ Derive RankNet gradient (pairwise \rightarrow pointwise)
- ☐ Interpret λ_i as forces on documents
- ☐ Explain why LambdaRank weights by metric change
- ☐ Calculate NDCG given a ranking
- ☐ Explain DPR training with contrastive loss
- ☐ Describe why hard negatives matter
- ☐ Compare traditional ML vs deep learning for ranking

Good luck with your exam! 🚀