Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Information retrieval, extraction and integration

Assignment 3: Lab ML ranking assignment

Authors:

José Antonio Ruiz Heredia

Joseph Tartivel

Álvaro Honrubia Genilloud

Teacher:

David Pérez del Rey

Date:
March 16, 2025

Contents

1 Introduction **2**

2 AdaRank: A Boosting Algorithm for Information Retrieval **2**

3 Construction of the Training Dataset **2**

4 Model Development and Implementation **3**

5 Enhancement of the Dataset **4**

6 Final Model Evaluation **4**

7 Contributions **4**

References **5**

A Appendix **6**

A.1 Relevance Score Formula 6

A.2 Setting Hyperparameters 7

A.3 Adaptive Training with AdaRank 7

1 Introduction

When searching for lab tests, the most relevant results should appear at the top, ensuring efficiency and accuracy in clinical decision-making. To do that, our systems need an accurate ranking of medical information. *Machine learning ranking (MLR)* methods provide a way to optimize this process by learning from examples and refining search rankings based on relevance.

This work explores the use of an *MLR* approach to improve how lab tests are retrieved in response to queries. By building and training a ranking model on a set of documents related to specific medical tests, our aim is to understand how those systems work. The study will also investigate how expanding the dataset in terms of both medical terms and queries can improve the performance of the model.

Beyond developing a functional ranking model, this project highlights the importance of structured search optimization. As medical information grows, ensuring that relevant data are properly ranked becomes more and more important for researchers, clinicians, and healthcare systems.

2 AdaRank: A Boosting Algorithm for Information Retrieval

For this work, we chose to use *AdaRank*, a ranking algorithm that improves search results by learning in multiple steps. Instead of just assigning scores to documents once, it adjusts its focus over time, giving more weight to cases where rankings are incorrect. It builds simple ranking models, known as weak rankers, that fix these mistakes step by step. Each new weak ranker is added to improve the overall ranking, gradually refining the order of results. This method ensures that the most relevant documents consistently move higher in the list, directly improving search quality. [1]

Traditional ranking methods take different approaches but often fail on optimizing the final search results. Pointwise methods treat ranking like a regression problem, assigning scores to documents individually without considering how they relate to others. This means they don't directly focus on getting the best order. Pairwise methods, like *Ranking SVM* and *RankBoost*, compare documents in pairs to decide which one should rank higher, but they still don't fully optimize the overall ranking list. *AdaRank* takes a different approach by looking at the entire list at once. It continuously improves the ranking by learning from errors and refining the results in a way that directly boosts performance metrics like *Normalized Discounted Cumulative Gain (NDCG)*, making it more effective for tasks where ordering is critical. [2] [3]

Since we need to rank *Logical Observation Identifiers Names and Codes (LOINC)* study based on query relevance, *AdaRank's* ability to optimize ranking directly will help us provide better search results. To implement it, we first need to create a training dataset that links queries to studies, assigning relevance scores to each result. Then, we will train the model step by step, letting it learn from mistakes and improve its rankings over time. As we expand our dataset by adding more queries and study terms, *AdaRank* will continue to adapt, ensuring that the system becomes more accurate and effective as it grows.

3 Construction of the Training Dataset

To effectively train our *AdaRank* algorithm, we constructed a structured dataset that maps medical queries to relevant *LOINC* codes, each assigned an appropriate relevance score. This approach aligns with the methodology of the *AdaRank* paper, where each query is associated with multiple documents (*LOINC* codes) and assigned a relevance score.

Our data source was an Excel file containing *LOINC* codes and their associated metadata, organized across multiple sheets. Each sheet corresponded to a specific medical query, such as "glucose in blood" or "bilirubin in plasma." The preprocessing workflow involved several critical steps. We began by text cleaning, involving converting text to lowercase, removing punctuation and special characters, eliminating common stop words, and applying lemmatization to reduce words to their base forms. Additionally, we created a mapping dictionary to expand medical abbreviations, such as '*bld*' to '*blood*' or '*mcnc*' to '*mass concentration*', to standardize terminology and enhance matching accuracy.

Following the *AdaRank* methodology, we assigned relevance scores to each document-query pair using a hybrid scoring approach. This approach combined traditional keyword matching with semantic similarity via embeddings. We created a query mapping dictionary that defined the expected component and system for each query, assigning higher scores to documents that closely matched these fields. We also utilized a pre-trained biomedical embedding model, *BioBERT-MNLI*, with fallbacks to *all-MiniLM-L6-v2*, to calculate semantic similarity between the query and document fields. This allowed us to capture relationships beyond exact keyword matches. Furthermore, we assigned different weights to various columns based on their importance (see Annex A.1 for detailed calculations). [4] [5]

To ensure consistency across different queries, we normalized the calculated scores to a range between 0 and 1 using min-max normalization. This step is crucial for the *AdaRank* algorithm as it enables fair comparison across different queries and document sets. The resulting training dataset was saved as a *CSV* file with columns including the query, *LOINC* code, name, component, system, property, measurement, and normalized score. This structured dataset provides the foundation for training our *AdaRank* model, allowing it to learn which features contribute most significantly to relevance rankings for medical test searches.

4 Model Development and Implementation

With the training dataset prepared, the next critical phase involved developing and implementing the *AdaRank* model. This process included data preparation, model training, evaluation, and results analysis, all guided by the principles outlined in the *AdaRank* paper.

Initially, the dataset underwent preprocessing to ensure compatibility with the *LightGBM* library, a gradient boosting framework renowned for its efficiency in ranking tasks. This involved encoding categorical features into numerical representations using panda's categorical encoding. Score labels were created by scaling and converting normalized scores into integer labels. The dataset was then divided into training and testing sets with an 75/25 split. In addition, training and testing data were transformed into *LightGBM* Dataset objects, which required specifying the feature matrix, label vector, and group sizes, as *AdaRank* needs the number of documents (*LOINC* codes) per query.

We trained the model using *LightGBM* with the `rank_xendcg` objective, but as it doesn't directly support *AdaRank*, we simulated its boosting and reweighting mechanism, as detailed in Annex A.3.. *AdaRank* creates weak rankers iteratively to minimize a ranking-specific loss, and *LightGBM* handles boosting and loss optimization well. Parameters like `objective` set to `rank_xendcg`, `metric` set to `ndcg` for evaluation, and `boosting_type` set to `gbdt` were configured. We also tuned different parameters such as `num_leaves`, `learning_rate`, `max_depth`, `min_child_samples`, `max_position`, `feature_fraction`, `label_gain`, and regularization terms to prevent overfitting. The model was trained with several boosting rounds and early stopping based on *NDCG* score obtained during training. The functionality of all these parameters is thoroughly explained in Annex A.2. [6]

5 Enhancement of the Dataset

To improve the performance of our *AdaRank* model, we systematically enhanced the dataset by expanding the set of queries, increasing the diversity of search terms, and refining relevance scoring methodologies. This process was carried out in multiple iterations to optimize ranking effectiveness and ensure robustness across different types of medical tests.

Firstly, we expanded our query set beyond initial examples like "*glucose in blood*" or "*bilirubin in plasma*" to include "*calcium in serum*" and "*cells in urine*", making the dataset more representative of real clinical search behaviors.

Secondly, we increased document coverage by retrieving additional relevant documents through *LOINC* dataset searches of different variations of the existing terms like "*bilirubin*", "*cells*", "*leukocytes*" or "*blood*", incorporating partial matches based on related medical terminology rather than relying solely on exact query matches. [7]

Thirdly, we optimized the model's parameters and fine-tuned the training process to align with the specific characteristics and size of the dataset. This involved adjusting hyperparameters, regularization techniques, and learning rates to improve performance and prevent overfitting.

6 Final Model Evaluation

After implementing the dataset enhancements, we conducted evaluation experiments to measure the effectiveness of the *AdaRank* model in ranking medical test results accurately. The evaluation was performed using key ranking metrics, including *NDCG*, Mean *Squared Error (MSE)*, *R-squared (R^2)*, and *Spearman's Rank Correlation.*

We observed the following improvements across different dataset versions:

- **Basic Dataset (Initial Version)**: MSE: 0.1642, R^2: -2.5187, Spearman's Rank Correlation: 0.7265, NDCG: 0.9086.
- **First Enhanced Dataset (3 Basic Queries + "*Calcium in Serum*")**: MSE: 0.0479, R^2: -1.9010, Spearman's Rank Correlation: 0.4700, **NDCG: 0.8533**.
- **Second Enhanced Dataset** (*"Bilirubin"*, *"Glucose"*, *"Leukocytes"*, and *"Calcium"* Queries Added): MSE: 0.0461, R^2: -0.8984, Spearman's Rank Correlation: 0.6024, **NDCG: 0.9421**.
- **Third Enhanced Dataset** (*"Blood"* and *"Serum or Plasma"* Queries Added): MSE: 0.0252, R^2: -0.4765, Spearman's Rank Correlation: 0.4983, **NDCG: 0.9398**.
- **Fourth Enhanced Dataset** (*"Cells in Urine"* Query Added): MSE: 0.0450, R^2: -1.4383, Spearman's Rank Correlation: 0.4323, **NDCG: 0.9448**.
- **Final Model Performance** (*"Cells"* and *"Urine"* Queries Added): MSE: 0.0191, R^2: -0.6009, Spearman's Rank Correlation: 0.4615, **NDCG: 0.9517**.

7 Contributions

This work was divided mainly by *Documentation search* and *Score Calculation* by **Jospeh Tartivel**, *Code Implementation*, *Datasets Experimentation* and *Documentation of the process* by **José Ruiz**, and *Report* and *Slides* by **Álvaro Honrubia**.

References

[1] Jun Xu, Hang Li "AdaRank: A Boosting Algorithm for Information Retrieval" https://dl.acm.org/doi/10.1145/1277741.1277809

[2] Fredric C. Gey "Inferring Probability of Relevance Using the Method of Logistic Regression" https://dl.acm.org/doi/pdf/10.5555/188490.188560

[3] Thorsten Joachims "Optimizing Search Engines using Clickthrough Data" https://dl.acm.org/doi/10.1145/775047.775067

[4] Deka, Pritam and Jurek-Loughrey, Anna and others "Evidence Extraction to Validate Medical Claims in Fake News Detection" https://huggingface.co/pritamdeka/BioBERT-mnli-snli-scinli-scitail-mednli-stsb

[5] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, Ming Zhou "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers" https://arxiv.org/abs/2002.10957

[6] LightGBM. *LightGBM Dataset Documentation.* https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.Dataset.html

[7] LOINC. *LOINC Search - Logical Observation Identifiers Names and Codes.* https://loinc.org/search/

A Appendix

A.1 Relevance Score Formula

The total relevance score is the sum of a traditional keyword matching score and an embedding-based semantic similarity score:

$$\text{Score}(q,d) = \text{Score}_{\text{traditional}}(q,d) + \text{Score}_{\text{embedding}}(q,d)$$

Where:

- q represents the query
- d represents a document (*LOINC* code entry)

The traditional score is calculated as:

$$\text{Score}_{\text{traditional}}(q,d) = \sum_{f \in \{component, system\}} \omega_f \cdot \text{Match}_f(q,d)$$

Where:

- ω_f is the weight for field f (e.g., 6.0 for component, 3.0 for system)
- $\text{Match}_f(q,d)$ equals:
  - Component
    - if there's an exact match between query term and component field:
      $$\text{weight}_{\text{component}} \cdot \text{weight}_{\text{component}}$$
    - if the query term is contained within the component field:
      $$\text{weight}_{\text{component}} \cdot \frac{\text{weight}_{\text{component}}}{2}$$
    - 0 if there's no match
  - System
    - if there's an exact match between query term and system field:
      $$\text{weight}_{\text{system}} \cdot \text{weight}_{\text{system}}$$
    - if the query term is contained within the system field:
      $$\text{weight}_{\text{system}} \cdot \frac{\text{weight}_{\text{system}}}{2}$$
    - 0 if there's no match

The embedding score is calculated as:

$$\text{Score}_{\text{embedding}}(q,d) = \sum_{f \in F} \omega_f \cdot \frac{\cos(\vec{q}, \vec{d_f}) + 1}{2} \cdot 5$$

Where:

- F is the set of all fields in the document
- ω_f is the weight for field f

- $\vec{q}$ is the embedding vector of the query
- $\vec{d_f}$ is the embedding vector of field f in document d
- $\cos(\vec{q}, \vec{d_f})$ is the cosine similarity between the query embedding and field embedding
- The term $\frac{\cos(\vec{q},\vec{d_f})+1}{2}$ normalizes the cosine similarity from [-1,1] to [0,1]
- The multiplier 5 scales the normalized similarity to match the scale of the traditional score

After calculating the total score, it's normalized to [0,1] using min-max normalization:

$$\text{Score}_{\text{normalized}}(q, d) = \frac{\text{Score}(q, d) - \text{Score}_{\text{min}}(q)}{\text{Score}_{\text{max}}(q) - \text{Score}_{\text{min}}(q)}$$

Where:

- $\text{Score}_{\text{min}}(q)$ is the minimum score across all documents for query q
- $\text{Score}_{\text{max}}(q)$ is the maximum score across all documents for query q

A.2 Setting Hyperparameters

The model is built using LightGBM, which is a gradient boosting framework that utilizes decision trees. The training process for the model is controlled by setting several hyperparameters.

- **objective:** This specifies the type of learning task. In this case, `rank_xendcg` is used, which is a ranking objective specifically suited for ranking tasks.
- **metric:** This defines the evaluation metric to be used during training. The `ndcg` metric is employed here, which is widely used for ranking tasks as it evaluates the relevance of items in a ranked list.
- **boosting_type:** This defines the type of boosting method used in the training process. Here, `gbdt` (Gradient Boosting Decision Trees) is selected, which is the standard boosting method in LightGBM.
- **num_leaves:** The number of leaves in each tree. This parameter affects the model's ability to capture interactions in the data.
- **learning_rate:** The learning rate determines the step size for each iteration.
- **max_depth:** This defines the maximum depth of the trees.
- **verbosity:** This controls the amount of output produced during training.
- **lambda_l1, lambda_l2:** These parameters control the L1 and L2 regularization terms.
- **colsample_bytree:** This parameter controls the fraction of features to be used for each tree.
- **label_gain:** This parameter assigns higher weight to the higher-ranking samples.
- **n_estimators:** This sets a high number of boosting rounds to ensure better performance.

A.3 Adaptive Training with AdaRank

In this model, we simulate the *AdaRank* boosting mechanism using *LightGBM*. Although LightGBM does not directly support *AdaRank*, we modify the model's training process to adaptively adjust the weights of misclassified samples in each iteration, similar to the *AdaRank* algorithm.

- The model is trained for multiple iterations, denoted as `n_iterations`.
- In each iteration, the model predicts the output on the training data:

$$y_{\text{pred}} = \text{model.predict}(X_{\text{train}})$$

- The prediction error for each data point is calculated by taking the absolute difference between the true labels (y_{train}) and the predicted values:

$$\text{errors} = |y_{\text{train}} - y_{\text{pred}}|$$

- The weights of the samples are updated based on the errors. The weight for each sample is increased if the prediction error is high, forcing the model to focus more on harder-to-predict cases. The update rule is:
$$\text{weights}* = 1 + \left(\frac{\text{errors}}{\text{errors.std}() + 1e - 6}\right) * 0.001$$
- The weights are then transformed using the log1p function, which applies a logarithmic transformation to stabilize the range of the weights and prevent extreme values from dominating:
$$\text{weights} = \text{np.log1p(weights)}$$
- Finally, the weights are normalized to ensure they fall within a consistent range:
$$\text{weights} = \frac{\text{weights} - \text{weights.min}()}{\text{weights.max}() - \text{weights.min}()}$$
  This normalization ensures that the weights remain in a standard range, preventing numerical instability and ensuring fair weighting across all samples.
- The updated weights are then applied to the training data to reweight the samples for the next iteration:
$$\text{train\_data} = \text{lgb.Dataset}(X_{\text{train}}, \text{label} = y_{\text{train}}, \text{group} = q_{\text{train}}, \text{weight} = \text{weights})$$