Alex Ferroni, Luca Ilardi, Gabriele Rinaldi

BIOMEDICAL INFORMATICS

Machine Learning Ranking: Pairwise Approach

Ranking SVM for Learning to Rank in Biomedical Information Retrieval

1. Introduction

1.1 Objective

The objective of this project is to understand and explain the process of Machine Learned Ranking (MLR), also known as Learning to Rank (LTR), applied to information retrieval in the biomedical domain.
The project implements a simplified pipeline inspired by Joachims' 2002 paper *"Optimizing Search Engines using Clickthrough Data"*, using Ranking SVM as the learning approach.

1.2 Problem Setup

We initially work with a small dataset of biomedical terms derived from the LOINC database. Each document corresponds to a LOINC code, described by textual fields such as long_common_name, component, system, property.

We initially consider only three example queries on the same set of LOINC terms: "glucose in blood", "bilirubin in plasma", "white blood cells count".

The goal is to rank the LOINC documents in order of relevance to each query, using a specific kind of data-driven model: the pairwise document ranking approach based on clicks and SVMs described in the chosen paper.

2. Choice of MLR Approach: Ranking SVM

We use the Ranking SVM, a **pairwise** Learning to Rank method proposed by Joachims.
The key idea is that instead of predicting absolute relevance scores and thus building a full ranking, the model instead learns preferences between pairs of documents for a given query.

If document A should rank above document B, the model learns a function that assigns a higher score to A. Mathematically, each document is associated with a feature vector x. The model learns a weight vector w so that:

$$\mathbf{w} \cdot \mathbf{x}_i > \mathbf{w} \cdot \mathbf{x}_j$$

whenever document *i* is preferred over document *j*.
This converts the ranking problem into a **binary classification** task using difference vectors (x_i - x_j).

This approach is conceptually simple and widely used in academic work, and it can be implemented with standard SVM classifiers. Additionally, it gives the advantage of making it easy to understand the relationship between ranking and classification.

3. Data and Preprocessing

3.1 Dataset Overview

Each sheet represents one query, each document includes several LOINC attributes that describe laboratory tests through the following set of columns based on the LOINC nomenclature: loinc_num, long_common_name, component, system, and property.

3.2 Feature Extraction and Baseline Ranking

To prepare data for the Learning to Rank (LTR) model, a feature vector was developed for each document, relative to its query. This process utilized the generate_feature_ranking.py script and encompassed the following key steps:

1. **Code Normalization:** LOINC shorthand (e.g., bld, plas, ser) was converted to full English words (blood, plasma, serum) to bridge the semantic gap between natural language queries and medical codes.
2. **Feature Computation:** After normalization, textual and structural features were extracted to capture the relationship between the query and each LOINC field. These features are categorized as:
    - **Similarity Features:** These quantify the textual resemblance between the query and various LOINC fields. For instance, `component_similarity` calculates the TF-IDF cosine similarity between the query and the document's component field, crucial for matching queries like "glucose in blood" to a component like "Glucose."
    - **Query Term Count Features:** These measure direct word overlap, indicating the number of query tokens present in a given field (e.g., `component_query_terms_count`). They are useful when TF-IDF similarities are low but exact matches exist.
    - **Baseline Ranking Features:** A `baseline_similarity` score (e.g., the average of other similarity measures) is computed to provide an initial relevance score and create an initial ranked list before model training.

The `rank_position` for each document was also saved, which is important for simulating user clicks in a later step (but not for direct model training).

These features collectively form the feature vector for each document, which is subsequently used to compute the pairwise difference vectors required for SVM training.

4. Generating Training Signals

4.1 Simulated Clicks

Script: `simulate_clicks.py`

To generate preference signals in the absence of real user data, click behavior is simulated using a simple probabilistic model. This model dictates that the probability of a click decreases with rank, following the formula: P(click) = C / (rank + 1).
A new boolean column, "clicked," is added to each file.**Output:** The `data/click_logs/` directory will contain the ranked lists, now enriched with simulated click information.

5. Creating Preference Pairs

5.1 Conceptual Preference Generation

Script: `create_pair.py`

This script generates training pairs based on a rule derived from Joachims' paper: if a user clicks on document *i* but skips a higher-ranked document *j*, then document *i* is preferred over document *j*. For each query, the script compares clicked and non-clicked documents to form these training pairs. Each pair is structured as (preferred_doc_id, not_preferred_doc_id, label=1).
Output:
A single CSV file, `data/conceptual_preference_pairs.csv`, containing the following columns: query, preferred_doc_id, not_preferred_doc_id, and label.

6. Model Implementation

The training file (`train.py`) begins by preparing the feature vectors for the SVM. This necessitates loading and filtering the data to select only the relevant numerical features, discarding unneeded metadata and textual columns before generating the normalized difference vectors.
After removing irrelevant data, the dataset was split into training and testing sets to evaluate the model's performance on unseen data.
At this point the application of a normalization was useful for the correct functioning of the model (SVM) that is particularly sensitive to the scales, paying attention to avoid the “data leakage” (apply the *.fit* method only on the training data).

The normalization used is: $z = (x - \mu)/\sigma$
- z is the new feature scaled
- x is the old feature
- μ is the mean from all the samples of the feature
- σ is the standard deviation from all the samples of the feature

The training process uses the generated normalized difference vectors derived from the preferred and not preferred document pairs to train the SVM:

- $Z_{ij} = Z_i - Z_j$ preferred pair (label +1)
- $Z_{ji} = Z_j - Z_i$ not preferred pair (label -1)

Creating both the preferred and non-preferred pairs allows the SVM model to learn to maximize the margin of separation between the two classes of document pairs, effectively distinguishing preferred items from non-preferred ones.
Now our dataset is ready to train the SVM. The formula of the loss function of the SVM is the following:

$$J(w) = \frac{1}{2}||w||^2 + C\sum_i \xi_i$$

- w represent the weights
- ξ_i represent the penalty due to the margin violation of the i-th example.
- C is a user-defined hyperparameter.
    - A small C (e.g., 0.1 - 1) allows the model to be more tolerant of margin violations.
    - A large C (e.g., 10 - 100) makes the model more strict regarding margin violations, which can increase the risk of overfitting.

Our implementation utilizes a GridSearch approach to determine the optimal 'C' value for our task, and also the kernel is set to linear (means find a hyperplane that can separate linearly the features). Cross-validation (KFold = 5) was also employed to maximize data generalization.

7. Final Ranking

The `ranker.py` file demonstrates how the model (SVM), trained on preferences extracted from simulated clicks, re-ranks a set of documents whose initial features were calculated based on a given query (from feature_ranking folder). The final ranking produced by the model replaces the starting ranking, integrating the preferences learned from the clicks.

8. Dataset Expansion

8.1 Increasing Terms and Queries

The objective was to expand the dataset to include both more queries and terms.
This helps simulate a larger and more realistic environment for ranking, thus training a more robust model.Steps:

1. Start with the original three queries.
2. Create new queries by combining:
    - Components such as glucose, bilirubin, cholesterol, urea, and others.
    - Systems such as blood, serum, plasma, urine.
    - Query templates like “component in system” or “component concentration in system”.
3. Use the LOINC Core table to find matching documents for each query.
4. Limit to a maximum of 50 queries and up to 50 LOINC terms per query.
5. Merge the new data with the original dataset.

9. Results

The results from the training of the model and the application of it into an example of feature file is the following:
The model achieved a final accuracy of 0.44 on the test set.The top four documents, ranked by the learned model, are as follows:

LOINC Number	Long Common Name	Ranking Score
43223-7	Sodium/Creatinine [Ratio] in Urine	1.891516
41903-6	Blood pressure device Vendor software version	1.285821
41918-4	Blood coagulation device Vendor software version	1.285821
12587-2	Creatinine [Mass/time] in 6 hour Urine	0.419505

10. Discussion and Reflection

The model does not perform well because the clicked data do not represent absolute relevance information. Moreover, the SVM is also learning from other features, such as the TF-IDF score, which is not ideal for capturing semantic relationships. In every case, the underlying concept of the model remains valid: the idea of inferring relative preferences from user interactions and combining multiple features leads to an improved final ranking compared to the baseline model without click-based data.

10.1 What Was Learned

- How ranking problems can be reformulated as classification problems through pairwise comparisons.
- The importance of feature design and preprocessing for text-based ranking.

- How simulated user interactions (clicks) can serve as training signals when real labels are not available.
- The logic behind transforming document-level data into machine-learning-ready input.

10.2 Limitations

- The simulated click data is only an approximation of real user behavior.
- The features are simple and mostly text-based, not semantic or contextual (a W2V model can be surely better than a TF-IDF approach).
- The model assumes a linear relationship between features and ranking score.
- Results are not expected to generalize well without richer data.

11. References

1. Thorsten Joachims, *Optimizing Search Engines using Clickthrough Data*, KDD 2002.
2. Course slides on Machine Learned Ranking.
3. Scikit-learn documentation on Support Vector Machines.
4. LOINC database official documentation.