Machine Learning Ranking: Pairwise Approach

Ranking SVM for Learning to Rank in Biomedical Information Retrieval

Alex Ferroni, Luca Ilardi, Gabriele Rinaldi

Biomedical Informatics

Objectives

- Understand the Machine Learned Ranking (MLR) process in the biomedical domain.
- Implement a simplified pipeline inspired by Joachims (2002).
- Use Ranking SVM to order biomedical documents by relevance.

Problem Setup

- **Dataset:** LOINC biomedical codes (component, system, property, long_common_name).
- **Component:** Type of measurement (e.g., Glucose, Bilirubin)
- **System:** Biological sample (e.g., Blood, Plasma)
- **Example queries:** 'glucose in blood', 'bilirubin in plasma', 'white blood cells count'.
- **Goal:** Rank documents by query relevance.

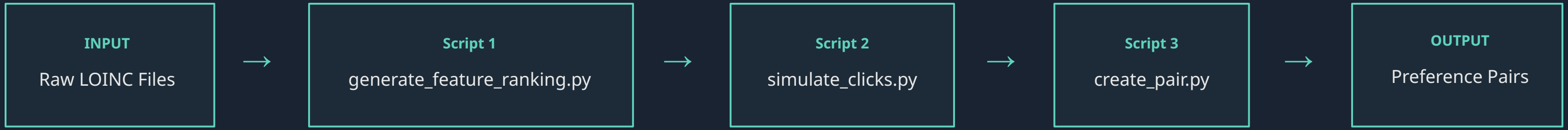
Ranking SVM Approach

- Pairwise Learning to Rank: model learns preferences between document pairs.
- If doc A > doc B → w·xA > w·xB.

> **If doc A > doc B → w·xA > w·xB**

- Transforms ranking into binary classification problem.
- Simple, interpretable, and widely used.

Our Data Pipeline: From Raw LOINC to Training Pairs



Script 1: Feature Engineering

- Normalize codes
- Calculate similarity features
- Create baseline_ranking

Script 2: Simulated Clicks

- Apply probabilistic model
- P(click) = C / (rank + 1)
- Generate clicked column

Script 3: Preference Pairs

- Apply Joachims' rule
- Infer preferences
- Output: CSV pairs

Step 1: Feature Engineering and Baseline Rank

Script: generate_feature_ranking.py

Code Normalization

- Reduces semantic gap between queries and LOINC fields.
- Example: bld → "blood", plas → "plasma", ser → "serum"

Similarity Features (TF-IDF Cosine Similarity)

- Calculates cosine similarity between query and each LOINC field.
- Features: component_similarity, system_similarity, property_similarity, long_name_similarity

Term Count Features

- Counts number of query terms present in each LOINC field.
- Features: component_query_terms_count, system_query_terms_count, property_query_terms_count

Baseline Rank

- Computes baseline_similarity as average of other similarities.
- Documents ordered by this score to create rank_position.

Step 1: Practical Example of Features

Query:"glucose in blood"

Document A (Relevant)

component:"Glucose"

system:"Bld" → "blood"

slong common name::"Glucose [Mass/volume] in Serum,Plasma or blood "

Feature	Value
Long_common_name_similarity	0.95
system_similarity	0.91
baseline_similarity	0.93

Rank: 1

Document B (Non-Relevant)

component:"Bilirubin"

system:"Ser" → "serum"

long common name:"Bilirubin total [Mass/Volume] in Synovial fluid "

Feature	Value
Long_common_name_similarity	0.10
system_similarity	0.05
baseline_similarity	0.07

Rank: 45

Step 2: Generating Training Signals (Simulated Clicks)

The Problem

- No access to real user click data in the biomedical domain.
- Impossible to train directly on authentic user behavior.

The Solution: Probabilistic Model

- Simulate clicks using a probabilistic model based on ranking position.
- Higher-ranked documents have higher click probability.
- Probability decreases as rank position increases down the list.

$$P(\text{click}) = C / (\text{rank} + 1)$$

Step 3: Creating Preference Pairs

Script: create_pair.py

Joachims' Rule: "If a user clicks document **i** but skips a higher-ranked document **j**, then document **i** is preferred over **j**" (i > j)

Example: Simulated Clicks

Rank	Document	Clicked
1	Doc A	**TRUE**
2	Doc B	FALSE
3	Doc C	FALSE
4	Doc D	**TRUE**

Interpretation: User skipped Doc B (Rank 2) and Doc C (Rank 3) to click Doc D (Rank 4). This means Doc D is preferred over Doc B and Doc C.

Inferred Preference Pairs

Pair 1

pref_id: Doc D

not_pref_id: Doc B

label: 1 (Doc D > Doc B)

Pair 2

pref_id: Doc D

not_pref_id: Doc C

label: 1 (Doc D > Doc C)

Final Output

File: **conceptual_preference_pairs.csv**
Contains all preference pairs inferred from all simulated clicks.

Step 4: Transforming Pairs for SVM

The Problem

- SVM cannot understand qualitative preferences like "Doc D > Doc B".
- Requires numerical input to learn. Must transform preferences into numerical vectors.

The Solution

- Create a difference vector for each preference pair.
- This vector becomes SVM input, classified as positive or negative.

Numerical Example: Pair (Doc D > Doc B)

Training Example 1 (Positive)

Feature Vector Doc D: [sim=0.81, count=3]

Feature Vector Doc B: [sim=0.88, count=4]

Calculation	Value
V_D[0] - V_B[0]	0.81 - 0.88 = -0.07
V_D[1] - V_B[1]	3 - 4 = -1

Difference Vector: [-0.07, -1]

Label: +1

Training Example 2 (Negative)

Feature Vector Doc B: [sim=0.88, count=4]

Feature Vector Doc D: [sim=0.81, count=3]

Calculation	Value
V_B[0] - V_D[0]	0.88 - 0.81 = 0.07
V_B[1] - V_D[1]	4 - 3 = 1

Difference Vector: [0.07, 1]

Label: -1

Model Implementation

- **Script: train.py** – preprocess and normalize features.
- Split dataset into train/test sets.
- Train SVM classifier to predict relevance order.

SVM Training Process: Data Preparation

Step 1: Data Loading and Filtering

- Load numerical features from preference pairs dataset.
- Filter and discard unneeded metadata and textual columns.
- Retain only relevant numerical features for model training.

Step 2: Difference Vector Generation

- Generate normalized difference vectors from preferred and non-preferred document pairs.
- $Z_{ij} = Z_i - Z_j$ for preferred pairs (label +1)
- $Z_{ji} = Z_j - Z_i$ for non-preferred pairs (label -1)

Step 3: Train/Test Split

- Divide dataset into training and testing sets for model evaluation.
- Ensures model performance assessment on unseen data.
- Prevents overfitting and validates generalization capability.

Normalization and Data Leakage Prevention

Z-Score Normalization Formula

$$z = (x - \mu) / \sigma$$

z = new feature scaled

x = old feature value

μ = mean from all samples of the feature

σ = standard deviation from all samples

Why Normalization Matters

- SVM is particularly sensitive to feature scales.
- Features with larger ranges can dominate the learning process.
- Normalization ensures all features contribute equally to model training.

Preventing Data Leakage

- **Critical Rule:** Apply .fit() method **only on training data**.
- Compute μ and σ exclusively from training set statistics.
- Apply same transformation to test data using training statistics

SVM Loss Function and Hyperparameter Optimization

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

- w = weight vector
- ξ_i = penalty due to margin violation of i-th example
- C = user-defined regularization hyperparameter

Small C (e.g., 0.1 - 1)

Allows model to be more tolerant of margin violations. Favors simpler decision boundaries with larger margins.

Large C (e.g., 10 - 100)

Makes model strict regarding margin violations. Increases risk of overfitting by fitting training data more closely.

Hyperparameter Tuning Strategy

- **GridSearch** approach to determine optimal C value for the task.
- Kernel set to **linear** to find hyperplane separating features linearly.
- **KFold cross-validation (K=5)** employed to maximize data generalization.
- Best C value selected based on cross-validation performance.

Final Ranking Application

Script: ranker.py

- The trained SVM model learns preferences from simulated clicks and document feature vectors.
- For each query, the model scores documents based on learned preference patterns.
- Documents are re-ranked according to their SVM scores, replacing the initial baseline ranking.
- The **final ranking** integrates preferences learned from clicks, producing a ranking that reflects both feature similarity and user interaction patterns.
- This demonstrates how machine-learned ranking can improve upon simple feature-based baselines by leveraging implicit user feedback signals.

Dataset Expansion: Scaling to Realistic Scenarios

Components to Combine

- Glucose, Bilirubin, Cholesterol, Urea, and other biomedical measurements.
- Diverse set of laboratory test types to ensure comprehensive coverage.

Biological Systems

- Blood, Serum, Plasma, Urine, and other biological samples.
- Multiple system types to reflect realistic query variations.

Query Template Generation

- Templates like **"component in system"** or **"component concentration in system"**.
- Systematic combination of components and systems creates diverse query set.

Scaling Constraints

- Maximum of **50 queries** with up to **50 LOINC terms per query**.
- Use LOINC Core table to find matching documents for each query.
- Merge expanded data with original dataset for robust model training.

Results and Model Performance

Top Four Documents Ranked by the Learned Model

Rank	LOINC Number	Long Common Name	Ranking Score
1	43223-7	Sodium/Creatinine [Ratio] in Urine	1.891516
2	41903-6	Blood pressure device Vendor software version	1.285821
3	41918-4	Blood coagulation device Vendor software version	1.285821
4	12587-2	Creatinine [Mass/time] in 6 hour Urine	0.419505

Limitations and Future Improvements

Current Limitations

- Simulated click data is only an approximation of real user behavior.
- Features are simple and mostly text-based, lacking semantic or contextual depth.
- Model assumes linear relationship between features and ranking score.
- Results are not expected to generalize well without richer data.

Future Improvements

- Integrate **Word2Vec** or other semantic embeddings instead of TF-IDF.
- Incorporate contextual features capturing relationships between biomedical terms.
- Explore non-linear models (kernel SVM, neural networks) for complex patterns.
- Collect real user interaction data to replace simulated clicks.
- Expand dataset with more diverse biomedical domains and query types.

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.