
Maxime Heuillet, Benoit Debaque and Audrey Durand
Institute Intelligence and Data, Laval University, Canada
maxime.heuillet.1@laval.ca

1. Motivation

Existing CF-based frameworks [1, 2] adopt an off-line setting that requires the generation of a large benchmarking of pipeline performances used as the training matrix:

- the training matrix \( R \) is costly to generate and is immutable
- information \( C(t) \) from current dataset \( t \) and from recommendation \( k_t \) is wasted

Instead, we adopt a sequential setting where the information from each recommendation request is leveraged to improve the performance of the framework over time.

- an exploration policy collects the information \( C(t) \) about the current dataset \( t \)
- \( R(t) \) is updated after each request with information \( C(t) \) and recommendation \( k_t \)

2. Proposed method

Problem formulation: consider \( K \) pipelines available for recommendation. The goal is to maximize the performance of the recommended pipelines over a sequence of datasets. For a new dataset, we want to recommend a good pipeline by trying only \( c \) pipelines on this dataset (where \( c \) is small compared to \( K \)).

Our method: a Collaborative Filtering (CF) latent representation updated at each dataset (akin to a step) is leveraged in order to drive efficiently the exploration and the recommendation of pipelines over time:

1. Obtain a latent representation of \( R(t−1) \): extract a latent representation \( P, Q \) factorized from the knowledge \( R(t−1) \) with a matrix factorization [3].
2. LinUCB exploration policy: at episode \( t \), LinUCB [4] selects the set of top-\( c \) \( C(t) \) pipelines by using as context the latent representation \( P(t−1) \) factorized from the knowledge \( R(t−1) \).
3. Recommendation policy: recommend the pipeline \( k_t \) with the best observed performance from the set of selected pipelines \( C(t) \), i.e \( k_t = \arg \max_{k \in C(t)} r_{t,k} \).
4. Update the training matrix with the collected knowledge: create a new empty row in \( R(t−1) \), append \( C(t) \) and \( r_{t,k} \) obtained from recommendation \( k_t \) resulting in matrix \( R(t) \).

Practical consideration: since LinUCB requires a stationary context, we use a buffered replicate of \( P(t) \), denoted \( \hat{P}(t) \), updated after every \( s \) episodes instead of every \( s \) episode. We wait \( s \) steps before the first update (burn-in phase) during which pipelines in \( C(t) \) are selected uniformly at random.

4. Results

Figure 3 shows the cumulative regret averaged over 10 folds. Rows correspond to the exploration budgets \( c \) and columns correspond to exploration policies.

---

Figure 1: Information waste in off-line setting.

Figure 2: The sequential setting.

Figure 3: Cumulative regret averaged over a 10-folds cross validation with the resulting standard error. Parameter \( s = 10 \) steps between update. Lower is better. Based on UCB1 time-series exploration [4].

Take-home messages:

- the performance of CF-based frameworks is highly influenced by the exploration policy.
- recommending the best pipelines over \( C(t) \) (blue & green dotted lines) always achieves the best performance indicating that recommendations should not be based on inference (blue & green plain lines).
- the gap between the KNN-based (current standard in the literature [1, 5]) and the LinUCB-based strategies narrows as \( c \) is increased (although still low). This is impressive because the KNN-based approach uses a dense knowledge matrix of \( 140 \times 175 = 24.5k \) observations, while the LinUCB-based approach uses a sparse knowledge of at most \( c \times t \) observations for a decision at time \( t \).

---

5. Resources

Code: https://github.com/MaxHeuillet/sequentialAutoML

Acknowledgements: MITACS Accelerate Research grant (IT17584) and Thales Research and Technology (Canada).