[추천시스템] Learning to Rank

Info.

- Learning to Rank 관련 내용/논문에 대한 요약
- 논문: A short Introduction to Learning to Rank (by Hang Li)
- 목차
  - Ranking Problem
  - Approach
  - Tutorial with LightFM
  - Discussion

작성중

1. Ranking Problem
   a. Introduction
      i. 전통적 영역은 IR(Information Retrieval) 영역이며,
      ii. 쿼리가 주어지면 시스템은 랭킹에 기반해 문서를 제공 → sort by *f(q,d)*

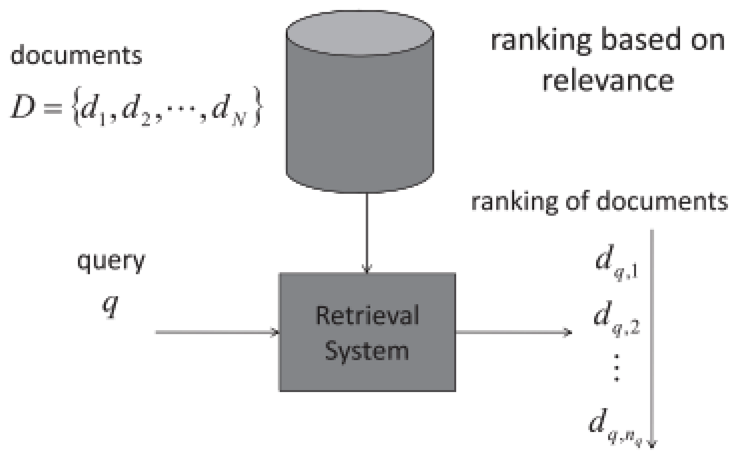


Fig. 1 Document retrieval.

      iii.
      iv. 예를 들어, *BM25* 모델의 경우, 조건부 확률를 이용하는 모델임
         1. P(r|q,d) where r takes 1 or 0
      v. 다른 모델로, *LMIR(Language Model for IR)*도 존재.
         1. P(q|d) → 확률 구할때 쿼리 및 문서에 포함된 단어의 출현빈도를 이용
      vi. 최근 (특히 웹검색)에는 기계학습 기법을 활발히 적용 (e.g PageRank)
         1. 예를 들어, 로그 데이터, Click Through Rate 등 데이터 활용 → *Learning To Rank*
   b. Training and Testing
      i. A supervised learning task (see Fig. 2)

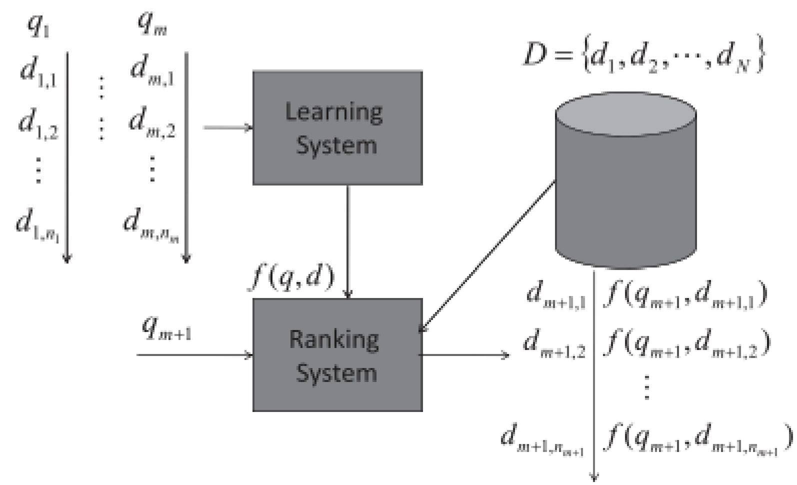


Fig. 2 Learning to rank for document retrieval.

ii.

iii. Training set consists of
  1. queries and documents
  2. And each query is associated with a number of documents, the *relevance* of documents is also given
  3. $\{q_1, q_2 \ldots, q_m\}$
  4. $\{d_{i,1}, d_{i,2} \ldots d_{i,ni}\}$
  5. $\{y_{i,1}, y_{i,2} \ldots, y_{i,ni}\}$

iv. Aim to train *a local ranking model f(q,d) = f(x) → which yields a single score.*

v. More generally, *global ranking model F(q, D) = F(x) → which yields a list of scores*

vi. 랭킹 모델의 역할은 집합에서 score 혹은 position 을 기준으로 list 를 만들어내는 역할
  1. "Ranking is nothing but to select a permutation $\pi_i \in \Pi$ for the given query q_i and the associated D_i using the score"

vii. Test dataset
  1. a new query q_{m+1} and associated D_m+1 → T = $\{(q_{m+1}, D_m+1)\}$
  2. Use the trained model to sort D_m+1 based on scores, and give the list of docs

viii. 기존 regression or classification의 테스트셋과 차이점
  1. Query and Docs forms a group → i.i.d while instance within groups are not

c. Data Labeling
  i. Relevance judgments are usually conducted at five levels, for example, perfect, excellent, good, fair, and bad
  ii. Log data

d. Evaluation
  i. NDCG(Normalized Discounted Cumulative Gain
  ii. MAP(Mean Average Precision) with two levels (1, 0)

e. Relation with Ordinal Classification
  i. "In ranking, one cares more about accurate ordering of objects,
  while in ordinal classification, one cares more about accurate ordered-categorization of objects"
  ii. for example,

  1. the ordinal one → correct assignment of the number of stars is critical
  2. but in raking → given a query, the objective is to correctly sort related documents
     a. The number of documents to be ranked can vary from query to query

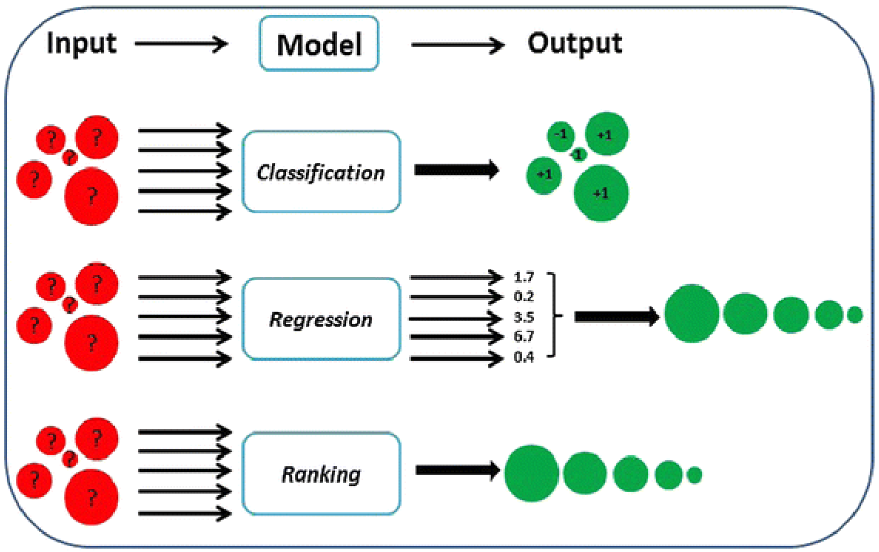


Figure 2
Different computational schemas in virtual screening.

3.
4. Source: https://jcheminf.biomedcentral.com/articles/10.1186/s13321-015-0052-z

2. Approach
   a. Reference
      i. https://medium.com/@nikhilbd/pointwise-vs-pairwise-vs-listwise-learning-to-rank-80a8fe8fadfd
      ii. https://www.slideshare.net/kerveros99/learning-to-rank-for-recommender-system-tutorial-acm-recsys-2013
   b. Overview

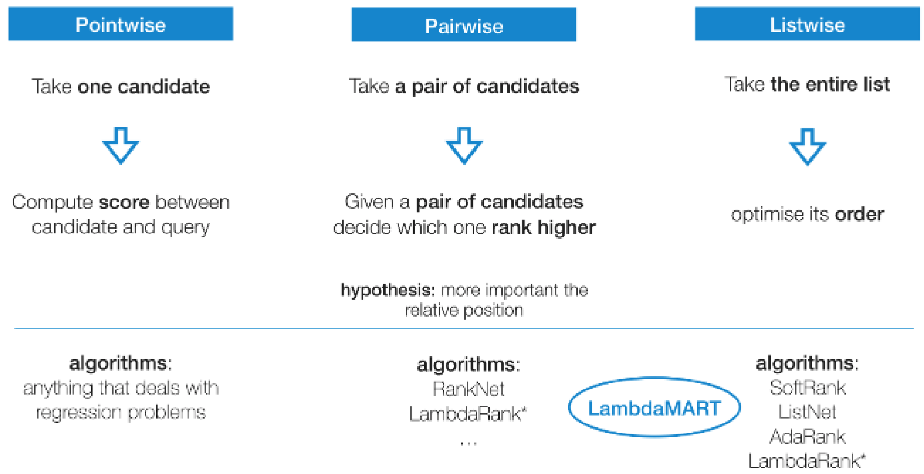


Fig2. Summary of the three main approaches of Learning to Rank.

i.
ii. Source: https://jobandtalent.engineering/learning-to-retrieve-and-rank-intuitive-overview-part-iii-1292f4259315

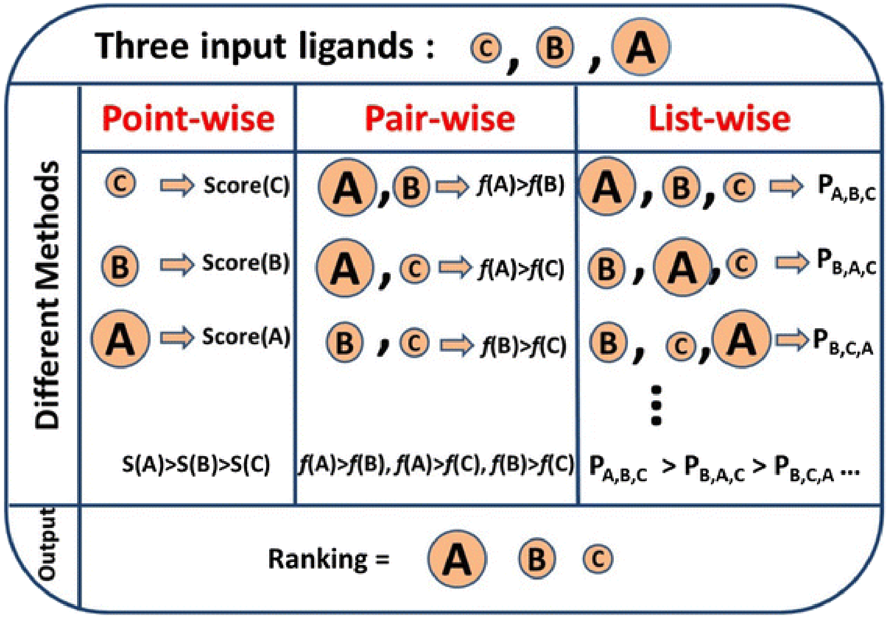


Figure 9

Three different approaches of *LOR*.

iii.

iv. Source: https://jcheminf.biomedcentral.com/articles/10.1186/s13321-015-0052-z

c. Pointwise

   i. Look at a single document at a time
   ii. Take a single document and train a classifier / regressor on it to predict how relevant it is for the current query
   iii. All the standard regression and classification algorithms can be directly used for pointwise learning to rank
      1. PRank
   iv. Loss function

$$L^r(f;\mathbf{x},\mathcal{L}) = \sum_{i=1}^{n} \left(f(x_i) - l(i)\right)^2.$$

      1.

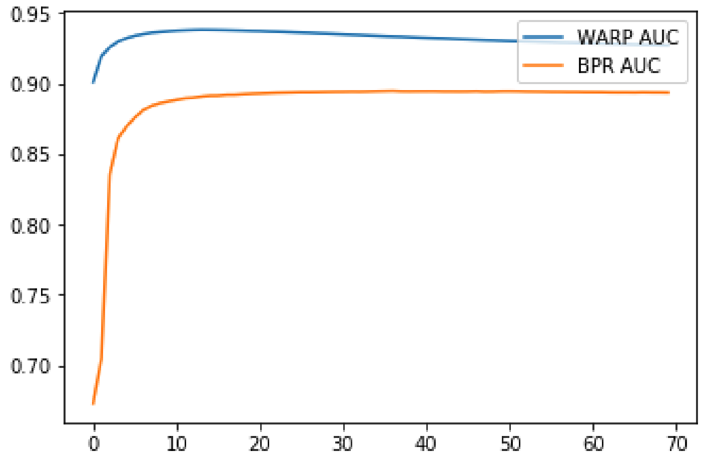
d. Pairwise

   i. Look at a pair of documents at a time
   ii. Come up with the optimal ordering for that pair and the goal for the ranker is to minimize the number of inversions in ranking
   iii. A better approach as it is closer to the nature of ranking
      1. RankNet
      2. LambdaRank
      3. LambdaMART
   iv. Loss function

1. $$L^p(f;\mathbf{x},\mathcal{L}) = \sum_{s=1}^{n-1} \sum_{i=1, l(i)<l(s)}^{n} \phi\big(f(x_s) - f(x_i)\big),$$
2. wherecan be logistic loss(log(1+e^{-z})) or exponential loss(e^{-z})

e. Listwise
    i. Look at the entire list of documents and try to come up with the optimal ordering for it
    ii. Complex compared to Pairwise, Pointwise approaches
        1. AdaRank
        2. ListNet
    iii. Loss function
        1. $$L^l(f;\mathbf{x},y) = \sum_{s=1}^{n-1} \Big(- f(x_{y(s)}) + \ln\big(\sum_{i=s}^{n} \exp(f(x_{y(i)}))\big)\Big),$$

3. Tutorial with LightFM
    a. Reference
        i. http://lyst.github.io/lightfm/docs/examples/warp_loss.html
        ii. https://towardsdatascience.com/how-to-build-a-movie-recommender-system-in-python-using-lightfm-8fa49d7cbe3b
    b. Dataset
        i. https://grouplens.org/datasets/movielens/100k/
    c. AUC comparison
        i. BPR(Bayesian Personalized Ranking pairwise loss)
            1. It maximizes the prediction difference between a positive example and a randomly chosen negative example. It is useful when only positive interactions are present.
            2. AUC: 0.88
        ii. WARP(Weighted Approximate-Rank Pairwise loss)
            1. Maximizes the rank of positive examples by repeatedly sampling negative examples until rank violating one is found
            2. AUC: 0.93
        iii. By epochs



        iv.

d. A simple recommender

i.

```
# fit
train, test = movielens['train'], movielens['test']

alpha = 1e-05
epochs = 70
num_components = 32

warp_model = LightFM(no_components=num_components,
                    loss='warp',
                    learning_schedule='adagrad',
                    max_sampled=100,
                    user_alpha=alpha,
                    item_alpha=alpha)

# build a function
def simple_recommendation(model, data, user_ids):

    n_users, n_items = data['train'].shape

    for user_id in user_ids:
        known_positives = data['item_labels'][data['train'].tocsr()[user_id].indices]
        scores = model.predict(user_id, np.arange(n_items))
        top_items = data['item_labels'][np.argsort(-scores)]

        print("User %s" % user_id)
        print("Known positives:")

        for x in known_positives[:10]:
            print("  %s" % x)

        print("Recommended:")

        for x in top_items[:10]:
            print("  %s" % x)
```

4. Discussion

a. 기존 맛집랭킹, 큐레이션 랭킹 모델 등의 고도화/개선
b. 오픈리스트 랭킹 모델 적용 (현재 랜덤 노출)