Theory Underlying Retrievers and Rankers

Mohamed Benaicha
mohamed.benaicha@hotmail.com
www.mohamedbenaicha.com

Steps

- Retrieval: retrieve top K recommendations
  - Collaborative filtering: based on implicit assumptions or explicit user ratings
- Ranking: order top K recommendations
  - Pointwise, pairwise and listwise approaches

Retrieval

Training (using implicit assumptions)

Embeddings used in retrievers to represent users/ book/ other features are trainable weight matrices.

User 1	User 2	User 3	User 4
2	3	1	1.3
-1	-0.2	3	-2
1.4	2	2.2	-1.6

User embeddings = 3

Book embeddings = 3

Book 1	3	1.5	-0.5
Book 2	2	1	-1.3
Book 3	-1.2	2	0.5

0	1	0	1
0	0	1	1
1	1	0	0

Book/User Rating Matric
(1/0-> implicit;
ratings -> explicit)

0	1	0	1
0	0	1	1
1	1	0	0

Training (using user ratings)

User embeddings = 3

User 1	User 2	User 3	User 4
2	3	1	1.3
-1	-0.2	3	-2
1.4	2	2.2	-1.6

Book embeddings = 3

				User 1	User 2	User 3	User 4
Book 1	3	1.5	-0.5	0	3	0	2
Book 2	2	1	-1.3	0	0	5	4
Book 3	-1.2	2	0.5	4	4	0	0

Retrieval

User 1
2
-1
1.4

Book 1	3	1.5	-0.5	3.8
Book 2	2	1	-1.3	1.18
Book 3	-1.2	2	0.5	-3.7

Retrieve TopK = 2, i.e. top 2 books. Books 1 and 2

Training

Ranking

dim = 16	dim = 48	dim = 16	dim = 48	
Book 1 embeddings	Book 1 features embeddings	User 1 embeddings	User 1 features embeddings	Book 1 Rating
Book 2 embeddings	Book 2 features embeddings	User 1 embeddings	User 1 features embeddings	Book 2 Rating
Book 3 embeddings	Book 2 features embeddings	User 2 embeddings	User 2 features embeddings	Book 3 Rating
Book 4 embeddings	Book 4 features embeddings	User 2 embeddings	User 2 features embeddings	Book 4 Rating
Book 5 embeddings	Book 5 features embeddings	User 2 embeddings	User 2 features embeddings	Book 5 Rating
Book 6 embeddings	Book 6 features embeddings	User 3 embeddings	User 3 features embeddings	Book 6 Rating

Features

Target

DNN Layers

↓

Rating prediction

Loss: Pointwise/Pairwise/ListWise

Metric: NDCG

Inference

Ranking

From the retrieval model, we know User 1's top 2 movies are movies 1,2; so we pass them into the ranker to get rating predictions that are ranked, with **book 2 being ranked first, then book 1:**

Book 1 embeddings	Book 1 features embeddings	User 1 embeddings	User 1 features embeddings	2.8
Book 2 embeddings	Book 2 features embeddings	User 1 embeddings	User 1 features embeddings	3.25

Features

Prediction

DNN Layers

Ranking

Ranking techniques: pointwise, pairwise, listwise

Pointwise (MSE loss): uses a simple feature-to-rating mapping and reduces MSE between predicted and actual rating – loses context

(Book1, User1, other features) -> 4 -> learn feature weights of book 1 and user 1 to estimate accurately predict a 4
(Book2, User1, other features) -> 5 -> “”””” book2 “”””” a 5
(Book3, User2, other features) -> 3 …
(Book4, User2, other features) -> 2 …
(Book5, User2, other features) -> 4 …

Pairwise (hinge loss): uses a simple feature-to-rating mapping but pairs books per user (query) – captures some context

((Book1, Book2), User 1, , other features) -> P(Book1 rating > Book2 rating) = 0 -> learn feature weights of book 1, book2 and user 1 predict a proba of 0 for Book1, Book2 pairs given the user is User 1
((Book3, Book4), User2, other features) -> P(Book3 rating > Book4 rating | User 2) = 1 “””
((Book4, Book5), User2, , other features) -> P(Book4 rating > Book5 rating | User 2) = 0 “””

Ranking

Ranking techniques: pointwise, pairwise, listwise

Listwise ranking (List MLE): the authors of ListMLE claim ListMLE is a close representation of the actual loss function we wish to minimize by maximizing the sum of *m* log-likelihoods of getting a prediction $y^{(i)}$ given inputs $x^{(i)}$ where ***g*** is a list of book ratings (g1,g2,g3,g4, g5); the gradient descent algorithm for adjusting parameters (*w*, i.e. *Θ*) doesn't differ either

```
[sum(log(P(y[i]|x[i] ; g))) for i in range(0,m)]
```

$$\sum_{i=1}^{m} \log P(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}; \mathbf{g}).$$

→ Maximize through gradient descent

Algorithm 1 ListMLE Algorithm

Input: training data$\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}$
Parameter: learning rate η, tolerance rate ϵ
Initialize parameter ω
repeat
  for $i = 1$ **to** m **do**
    Input $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ to Neural Network and compute gradient $\triangle\omega$ with current ω
    Update $\omega = \omega - \eta \times \triangle\omega$
  end for
  calculate likelihood loss on the training set
until change of likelihood loss is below ϵ
Output: Neural Network model ω

Xia et al. (2008)

Listwise ranking data structure (for training; for inference, do not include ratings):

```
{
'Users':[User1,…,…], # shape = (n,) where n is batch size
'Books': [['Book1', 'Book2', 'Book3', 'Book4', 'Book5'],[..]…], # shape = (n,5)
'Ratings': [[2,5,3,2,5], [..],…], # shape = (n,5)
}
```

Ranking

The NDCG metric used in the ranking model is:

1. A sum of discounted relevance
2. Where each element ($r_1 \ldots r_k$) comprising the sum is:

$(2^{g(r)} – 1) / \log(r + 1)$

g = score of book in position r,

r = position in the list

$$DCG_k = \sum_{r=1}^{k} \frac{rel_r}{\log(r+1)}$$

3. The sum above is calculated for the ideal list and the current list being fed into the forward pass; to get normalized DCG, the latter is divided by the former

Normalized DCG = DCG / Ideal DCG

- Intuitively each term in DCG is a discounted relevance, i.e:

  [2^{score} - 1 / log(position in list+1)]

- If properly ranked, the term is greater since it will have a greater numerator and a small denominator, whereas the poorly ranked books should not contribute well to the DCG since the numerator would be small with a large denominator

$$NDCG_n = \frac{DCG_n}{IDCG_n},$$