COMP9727 Recommender Systems

Lina Yao

University of New South Wales

lina.yao@unsw.edu.au

Overview

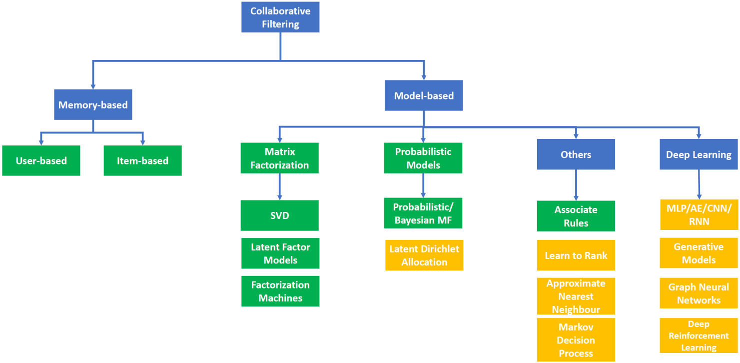
Learn to Rank
- Basics of Learn to Rank
- Memory-based pairwise learn to rank
- Model-based pairwise learn to rank
- Model-based Listwise Learning to Rank

Basics of Learn to Rank

- Search (Document Search, Entity Search, etc)
- Key Phrase Extraction
- Question Answering
- Document Summarization
- Opinion Mining
- Sentiment Analysis
- Machine Translation
- ... ...
- Recommender System(Collaborative Filtering)

Shi, Y et al. List-wise learning to rank with matrix factorization for collaborative filtering.

Overview



Collaborative Filtering
Memory-based
Model-based
User-based
Item-based
Matrix Factorization
Probabilistic Models
Others
Deep Learning
SVD
Latent Factor Models
Factorization Machines
Probabilistic/ Bayesian MF
Latent Dirichlet Allocation
Associate Rules
Learn to Rank
Approximate Nearest Neighbour
Markov Decision Process
MLP/AE/CNN/ RNN
Generative Models
Graph Neural Networks
Deep Reinforcement Learning

Why Learn to Rank

- No need to predict category labels, e.g., mapping to an unordered set of classes in ordinal classification
- No need to predict value of $f(x)$, e.g., rating score
- relative ranking order is more important, e.g., top-k recommendation

Problem Formulation

To create a ranking list of objects using the features of the objects.

- Given a set of input vectors and corresponding labels with specified order
- Learn a function that specifies a ranking over the input vectors, which minimizes the cost/loss.

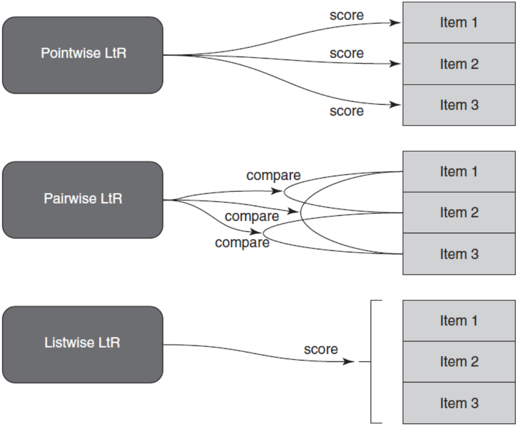
Ranking as a recommendation

- Most recommendations are presented in a sorted list
- Recommendation can be understood as a ranking problem
- Implicit feedback

Given a set of ordered pairs of instances as training data

- learn an item scoring function
- learn a classifier for classifying item pairs into two types of relations (correctly ordered vs. incorrectly ordered

Ranking as a recommendation



Methods

- Pointwise methods $f(user, item) \rightarrow \mathbb{R}$
  - minimizing the loss function between predicted value and ground-truth value
  - ranking score is based on regression or classification
- Pairwise methods $f(user, item_1, item_2) \rightarrow \mathbb{R}$
  - loss function is defined on pairwise preferences
  - the observed entries should be ranked higher than the unobserved ones
  - maximizes the margin between observed entry and unobserved entry
- Listwise methods $f(user, item_1, item_2, ..., item_n) \rightarrow \mathbb{R}$
  - indirect loss functions, e.g., similarity between ranking list and groundtruth as loss function; KL-divergence as loss function by defining a probability distribution
  - directly optimizing the ranking measures, like nDCG (Normalized Discounted Cumulative Gain), MRR (Mean Reciprocal Rank)

Learn to Rank for Recommendation

1. Basics of Learn to Rank
2. Memory-based pairwise learn to rank
3. Model-based pairwise learn to rank
4. Model-based listwise learn to rank

EigenRank

Higher accuracy in rating prediction $\not\Rightarrow$ better ranking effectiveness
Item i and j's groundtruth ratings are 3, and 4.
The predicted $\hat{i}$ and $\hat{j}$ produced by method1 is $\{2{,}5\}$; The predicted $\hat{i}$ and $\hat{j}$ produced by method2 is $\{4{,}3\}$;
Which method is better?

Liu, N.N. and Yang, Q., 2008, July. Eigenrank: a ranking-oriented approach to collaborative filtering.

EigenRank

- Design a similarity measure for evaluating the consistency between each pair of users' rankings on a set of items $\Rightarrow$ users sharing similar preference with target users
- Aggregating the partial and incomplete item rankings derived from the ratings of a set of similar users

EigenRank

Recall the similarity calculation (i.e., user-based) in neighbourhood-based CF. The Pearson Correlation Coefficient (PCC) measures the similarity between two users based their normalized ratings on the set of items they rated in common

$$s_{u,v} = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \overline{r}_u)(r_{v,i} - \overline{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - \overline{r}_u)^2 \sum_{i \in I_u \cap I_v} (r_{v,i} - \overline{r}_v)^2}}$$

Then, the unknown ratings can be predicted

$$\hat{r}_{u,i} = \overline{r}_u + \frac{\sum_{v \in N_u \wedge U_i} s_{u,v}(r_{v,i} - \overline{r}_v)}{\sum_{v \in N_u \wedge U_i} s_{u,v}}$$

EigenRank - Ranking-sensitive similarity

The similarity is calculated by the users' preferences over the items, reflecting their ranking of the items.
Kendall Rank Correlation Coefficient

$$s_{u,v} = 1 - \frac{4 \times \sum_{i,j \in I_u \cap I_v} I^{-}((r_{u,i} - r_{u,j})(r_{v,i} - r_{v,j}))}{|I_u \cap I_v| \cdot (|I_u \cap I_v| - 1)}$$

where $I^{-}(x)$ is an indicator function, it is equal to 1 if $x < 0$, and 0 otherwise.
The value is negatively correlated with the number of disconcordant pairs, where a pair of items i and j is disconcordant if i is ranked higher than j in one ranking but lower in the other.

EigenRank

The goal is produce a ranking of the items for an active user rather than predicting the rating scores. $\Psi : I \times I \rightarrow \mathcal{R}$, where $\Psi(i,j) > 0$ means that item i is more preferable to j and vice versa. $|\Psi|$ indicates the strength of preference.

$$\Psi(i,j) = \frac{\sum\limits_{v \in N_u^{i,j}} s_{u,v} \cdot (r_{v,i} - r_{v,j})}{\sum\limits_{v \in N_u^{i,j}} s_{u,v}}$$

- relaxed assumption on ranking function Ψ for transitivity, i.e., $\Psi(i,j) > 0 \wedge \Psi(j,k) > 0$ doesn't mean $\Psi(i,k) > 0$.
- the more often the users in neighbourhood N_u assign i a higher rating than j, the stronger the evidence for $\Psi(i,j) > 0$ and $\Psi(j,i) < 0$.

EigenRank

Given a preference function Ψ, which assigns a score to every pair of items $i, j \in I$, the aim is to choose a ranking of items in I that agrees with the pairwise preferences defined by Ψ as much as possible. $V^{\Psi}(\rho) = \sum_{i,j:\rho(i)>\rho(j)} \Psi(i,j)$ where $V^{\Psi}(\rho)$ is a value function measuring how consistent is the ranking ρ w.r.t. the preference function Ψ.
The goal is to produce a ranking ρ^{*} that maximizes this value function.

- Greedy order algorithm
- Random Walk

EigenRank - Greedy Order Algorithm

Always makes the choice that looks best at the moment and adds it to the current subsolution.
Greedy algorithms don't always yield optimal solutions but, when they do, they're usually the simplest and most efficient algorithms available.

EigenRank - Greedy Order Algorithm

Search through the possible rankings in an attempt to find the optimal ranking ρ^*

- produce the ranking from the highest position to the lowest position by always picking the item i that currently has the maximum potential and assign it a rank equal to the number of remaining items I.
- delete i from I and udpate the potential values of remaining items by removing the effects of t

NP-complete problem based on reduction from Cyclic ordering problem.
An approximation of optimal ranking (detailed algorithm refers to the paper).

Bayesian Personalized Ranking

- recommending the top k mostly relevant items only
- a ranked list - the top k is orderly arranged according to relevance
- reasons of missing values are complex
  - explicit feedback is not available in many cases, e.g., users may not rate an item that they don't like
  - explicit feedback may not capture the nature of data, e.g., a clickstream dataset may only reveal how frequent a user visit an item, but that may not be equivalent to say the user like this item

Steffen Rendle et al., "BPR: Bayesian Personalized Ranking from Implicit Feedback"

Implicit Feedback

Implicit feedback are more prevalent and readily available as the users haven't explicitly express their preference yet.

- Biased dataset. Implicit feedback is represented as positive data, e.g., only having records like "purchased" or "clicked", without negative data. It is challenging to ML algorithms
- Complex. A mixture of real negative data (e.g., are not interested) and unknown/missing values (e.g., haven't purchased yet)

Implicit Feedback

Let U be the set of all users and I the set of all items. The implicit feedback $S \subseteq U \times I$ is available.
The task of the recommender system is to provide the active user with a personalized total ranking $\geq_u \subset I^2$ of all items, where $\geq_u$ has to meet the properties of a total order:

- Totality. $\forall_{i,j} \in I : i \neq j \Rightarrow i >_u j \vee j >_u i$
- Antisymmetry. $\forall_{i,j} \in I : i >_u j \wedge j >_u i \Rightarrow i = j$
- Transitivity. $\forall_{i,j,k} \in I : i >_u j \wedge j >_u k \Rightarrow i >_u k$

For convenience, $I_u^+ := \{i \in I : (u,i) \in \mathcal{S}\}$ and $U_i^+ := \{u \in U : (u,i) \in \mathcal{S}\}$

Implicit Feedback

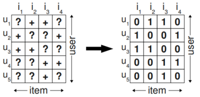


Figure 1: On the left side, the observed data S is shown. Learning directly from S is not feasible as only positive feedback is observed. Usually negative data is generated by filling the matrix with 0 values.

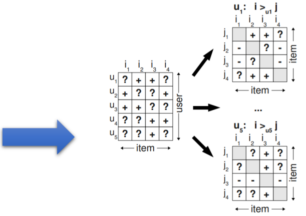


Figure 2: On the left side, the observed data S is shown. Our approach creates user specific pairwise preferences $i >_u j$ between a pair of items. On the right side, plus (+) indicates that a user prefers item i over item j; minus (–) indicates that he prefers j over i.

- training data consists of both positive and negative pairs and missing values. The missing values between two non-observed items are exactly the item pairs that have to be ranked in the future. That means from a pairwise point of view the training data and the test data is disjoint.
- The training data is created for the actual objective of ranking.

BPR

- BPR-OPT, is derived by a Bayesian analysis of the problem using the likelihood function for $p(i >_u j|\Theta)$ and prior probability for the model parameter $p(\Theta)$
- LEARNBPR for learning

Recap

Bayesian statistics, used in the PMF related methods in week 2.

PMF - Modelling

$$\max p(U,V|R,\sigma^2,\sigma_U^2,\sigma_V^2) = \frac{p(R|U,V,\sigma^2)p(U|\sigma_U^2)p(V|\sigma_V^2)}{p(U,V,\sigma^2,\sigma_U^2,\sigma_V^2)}$$

$$\Downarrow$$

$$\max p(U,V|R,\sigma^2,\sigma_U^2,\sigma_V^2) \propto p(R|U,V,\sigma^2)p(U|\sigma_U^2)p(V|\sigma_V^2)$$

$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$ In BPR, our aim is to learn the model parameters that can produce a perfect ordering for users. The probability is written as $p(\Theta| >_u) \propto p(>_u |\Theta)p(\Theta)$

Prior

$p(\Theta) \sim N(0, \Sigma_\Theta)$ follows normal distribution with zero mean and variance-covariance matrix Σ_Θ.
$\Sigma_\Theta = \lambda_\Theta I$

Likelihood function $p(>_u |\Theta)$

There are two assumptions

- all users are independent
- the ordering of each pair of items (i, j) for a specific user is independent pair of items (i, j)

$\prod_{u \in U} p(>_u |\Theta) = \prod_{(u,i,j) \in U \times I \times I} p(i >_u j|\Theta)^{\delta((u,i,j) \in D_S)} \cdot (1 - p(i >_u j|\Theta))^{\delta(u,j,i) \notin D_S}$
where $\delta(b) = 1$ if b is true, 0 otherwise.
Recall our assumptions on totality and ANTIsymmetry

- *Totality.* if i is not j, then either $i >_u j$ or $j >_u i$
- *Antisymmetry.* if $i >_u j$ and $j >_u i$, then i is j

$\prod_{u \in U} p(>_u |\Theta) = \prod_{(u,i,j) \in D_S} p(i >_u j|\Theta)$

Likelihood function $p(>_u|\Theta)$

- Not sound yet to get a personalized total order, recalling the third assumption on transitivity. An individual probability that a user really prefers item i to item j as $p(i >_u j|\Theta) = \sigma(\hat{x}_{uij}(\Theta))$ where σ is the logistic $\sigma(x) = \frac{1}{1+exp(-x)}$
- Convert to differentiable sigmoid function for optimization
- $\hat{x}_{uij}(\Phi)$ is an arbitrary real-valued function of model parameter vector Φ capturing the relationship between user u and items i, j, e.g., $\hat{x}_{uij}(\Theta) = \hat{x}_{ui}(\Theta) - \hat{x}_{uj}(\Theta)$

$$\prod_{u \in U} p(>_u|\Theta) = \prod_{(u,i,j) \in D_S} p(i >_u j|\Theta) \Rightarrow$$

$$\prod_{u \in U} p(>_u|\Theta) = \prod_{(u,i,j) \in D_S} \sigma(\hat{x}_{ui} - \hat{x}_{uj})$$

Objective function: BPR-OPT

$$
\begin{aligned}
BPR-OPT &= p(\Theta| >_u) \propto p(>_u |\Theta)p(\Theta) \\
&:= \ln(p(\Theta| >_u)) \\
&= \ln(p(>_u |\Theta)p(\Theta)) \\
&= \ln \prod_{(u,i,j)\in D_S} \sigma(\hat{x}_{uij})p(\Theta) \\
&= \sum_{(u,i,j)\in D_S} \ln \sigma(\hat{x}_{uij}) + \ln p(\Theta) \\
&= \sum_{(u,i,j)\in D_S} \ln \sigma(\hat{x}_{uij}) + \lambda_\Theta ||\Theta||^2
\end{aligned}
$$

Objective function: BPR-OPT

$$\frac{\partial BPR-OPT}{\partial \Theta} = \sum_{(u,i,j)\in D_S} \frac{\partial \ln \sigma(\hat{x}_{uij})}{\partial \Theta} - \lambda_\Theta \frac{\partial ||\Theta||^2}{\partial \Theta}$$

$$\propto \sum_{(u,i,j)\in D_S} \frac{-exp(-\hat{x}_{uij})}{1+exp(-\hat{x}_{uij})} \cdot \frac{\partial \hat{x}_{uij}}{\partial \Theta} - \lambda_\Theta \Theta$$

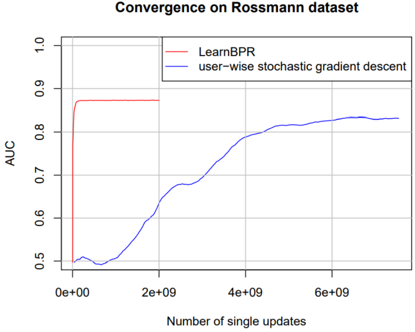
For each triple $(u,i,j) \in D_S$ an update will be performed as

$$\Theta \leftarrow \Theta + \alpha \left(\frac{exp(-\hat{x}_{uij})}{1+exp(-\hat{x}_{uij})} \cdot \frac{\partial \hat{x}_{uij}}{\partial \Theta} + \lambda_\Theta \Theta \right)$$

what is the problem?

BPR-OPT: Learning

Bootstrap sampling to sample triplets uniformly, in order to avoid picking up same user-item combinations in consecutive updates



procedure LEARNBPR(D_S, Θ)
  initialize Θ
  repeat
    draw (u, i, j) from D_S
    $\Theta \leftarrow \Theta + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\Theta \cdot \Theta \right)$
  until convergence
  return $\hat{\Theta}$

Learning with BPR: Matrix Factorization

Given triples $(u, i, j) \in D_S$, we first decompose the our estimator $\hat{x}_{uij} := \hat{x}_{ui} - \hat{x}_{uj}$.
To classify the difference of two predictions $\hat{x}_{ui} - \hat{x}_{uj}$. The prediction is produced $\hat{X} = WH^T$ where W and H are low-rank matrices with dimensionality k, which are seen as latent variables, modeling non-observing preference of an active user and the non-observed properties of item.
$\hat{x}_{ui} = \sum_{f=1}^{k} w_{uf} \cdot h_{if}$, same to $\hat{x}_{uj}$

Learning with BPR: Matrix Factorization

For the matrix factorization model the derivatives are

$$\frac{\partial \hat{x}_{uij}}{\partial \Theta} = \begin{cases} h_{ij} - h_{jf} & \text{if } \Theta = w_{uf} \\ w_{uf} & \text{if } \Theta = h_{if} \\ -w_{uf} & \text{if } \Theta = h_{jf} \\ 0 & \text{otherwise} \end{cases}$$

Regularizers are still used on users and item vectors. For the item feature vector, one for positive updates on h_{ij}, and the other is for negative update on h_{jf}.

Learning with BPR: Neighbourhood-based

Prediction for an active user u and item i depends on the similarity of i to all the items that the user has seen in the past, e.g., I_u^+.
Pick up k mostly relevant items, $\hat{x}_{ui} = \sum_{j \in I_u^+ \wedge j \neq i} c_{ij}$, where $c_{ij} \in C : I \times I$ is the symmetric item correlation matrix as model parameters Θ.

$$\frac{\partial \hat{x}_{uij}}{\partial \Theta} = \begin{cases} +1 & \text{if } \Theta \in \{c_{il}, c_{li}\} \wedge l \in I_u^+ \wedge l \neq i \\ -1 & \text{if } \Theta \in \{c_{jl}, c_{lj}\} \wedge l \in I_u^+ \wedge l \neq j \\ 0 & \text{otherwise} \end{cases}$$

One regularizer is for updates on c_{il}, and the other is for c_{jl}.

Learn to Rank for Recommendation

1. Basics of Learn to Rank
2. Memory-based Pairwise Rank
3. Model-based pairwise Rank-oriented method
4. Model-based Listwise Rank

ListRank-MF

- Pointwise ranking can't be directly interpreted into a measure of ranking quality
- Pairwise ranking is computationally intensive, and hard to run at scale

Top One Probability

The top one probability for an item rated R_{ij} in user i's ranking list l_i with K items can be expressed as

$$p_{l_i}(R_{ij}) = \frac{\phi(R_{ij})}{\sum_{k=1}^{K} \phi(R_{ik})},$$

where $\phi(x)$ can be any monotonically increasing and strictly positive function, like exponential function.
It indicates the probability of an item being ranked in the top position for a given ranking list.

Cross Entropy

- Cross Entropy is a measure of the difference between two probability distributions from a given random variable or set of events
- Entropy is the number of bits required to transmit a randomly selected event from a probability distribution

In discrete space, $H(p, q) = -\sum_x p(x) \log q(x)$, where $p(x)$ and $q(x)$ are two probabilistic distributions

ListRank-MF

The ListRank-MF is formulated by using the cross-entropy of top one probability of the items, in the training example lists and the ranking lists from the ranking model (MF) as the loss function.

$$
\begin{aligned}
L(U,V) =& \sum_{i=1}^{M} \left\{ -\sum_{j=1}^{N} P_{l_i} R_{ij} \log P_{l_i}(g(U_i^T V_j)) \right\} + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2) \\
=& \sum_{i=1}^{M} \left\{ -\sum_{j=1}^{N} I_{ij} \frac{\exp(R_{ij})}{\sum_{k=1}^{N} I_{ik} \exp(R_{ik})} \log \frac{\exp(g(U_i^T V_j))}{\sum_{k=1}^{N} I_{ik} \exp(g(U_i^T V_j))} \right\} \\
& + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2)
\end{aligned}
$$

$g(x)$ is a logistic function to bound the range of $U_i^T v_j$, i.e., $g(x) = \frac{1}{1+exp(-x)}$

ListRank-MF

The training example lists consist of the training set items in the profiles of each user. The output of recommendation model is a recommendation list for each user i and is generated by ranking items in collection in descending order according to the value $U_i^T V$.

Items already contained among the training examples in the user i's profile are removed.

$$\frac{\partial L}{\partial U_i} = \sum_{j=1}^{N} I_{ij} \left(\frac{\exp(g(U_i^T V_j))}{\sum_{k=1}^{N} I_{ik} \exp(g(U_i^T V_k))} - \frac{\exp(R_{ij})}{\sum_{k=1}^{N} I_{ik} \exp(R_{ik})} \right) g'(U_i^T V_j) V_j + \lambda U_i$$

$$\frac{\partial L}{\partial V_j} = \sum_{i=1}^{M} I_{ij} \left(\frac{\exp(g(U_i^T V_j))}{\sum_{k=1}^{N} I_{ik} \exp(g(U_i^T V_k))} - \frac{\exp(R_{ij})}{\sum_{k=1}^{N} I_{ik} \exp(R_{ik})} \right) g'(U_i^T V_j) U_i + \lambda V_j$$

ListRank-MF

The complexity of ListRank-MF consists of the following components

- computation of loss function: $\mathcal{O}(2dS + d(M + N))$
- computation of gradient w.r.t. U: $\mathcal{O}(2dS + dM)$
- computation of gradient w.r.t. V: $\mathcal{O}(dS + pdS + dN)$

where d is the dimensionality of U and V, S is the number of observed ratings in a given user-item matrix, M is the number of users, N is the number of items, p the average number of items rated per user, and usually small. $S >> M, N$. Therefore, the total complexity of each iteration is $\mathcal{O}(dS + pdS)$.