Recommender Systems Algos

Notes on the course Recommender Systems as taught by Paolo Cremonesi and Maurizio Dacrema in Politecnico di Milano during the academic year 2019/2020

Ranking Metrics

Average Precision at N

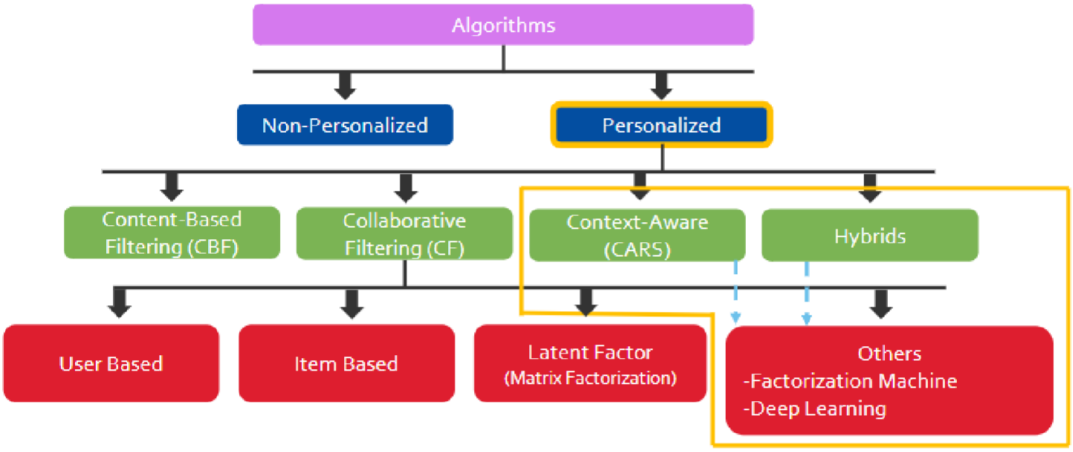
$$AP@N = \frac{1}{\min(m, N)} \sum_{k=1}^{N} P(k) rel(k)$$

- m is the number of relevant items in the entire system.
- N is the number of items recommended.
- $P(k)$ is the precision at cutoff k.
- $rel(k)$ is 1 if item k is relevant, 0 otherwise.

MAP at N

$$MAP@N = \frac{1}{|U|} \sum_{u=1}^{|U|} (AP@N)_u$$

Taxonomy



Non Personalized

1 - Top Popular

Implicit & Explicit

recommend items with the most number of interactions.
s stands for score, b is a binary variable telling us whether the interaction happened or not.

$$s_i = \sum_{u \in U} b_{ui}$$

sort $\bar{s}$ in descending order and take the first k items.
Most Popular items are those who have most ratings whatever is their value. It is good to remove items the user have already seen.

4 - Top Rated

$$\hat{r}_i = \frac{\sum_{u \in U} r_{ui}}{N_i + C}$$

N_i is the number of users that have given a score to item i.

C is the shrinkage.

Personalized

Global Effects

- compute the average rating of all items by all users

$$r_{avg} = \frac{\sum_{u,i} r_{ui}}{N^+}$$

  where N^+ is the number of interactions.

- subtract if from each rating in the URM

$$r'_{ui} = r_{ui} - r_{avg} \qquad \forall (u,i) \in \text{Interactions}$$

- compute for each item the item bias

$$b_i = \frac{\sum_u r'_{ui}}{N_i + C} \qquad \forall i \in I$$

  where N_i is the number of users that interacted with item i

- remove it from each rating in the URM

$$r''_{ui} = r'_{ui} - b_i \qquad \forall (u,i) \in \text{Interactions}$$

- compute for each user the user bias

$$b_u = \frac{\sum_i r''_{ui}}{N_u + C} \qquad \forall u \in U$$

  where N_u is the number of items rated by u.

- remove it from each rating the URM

$$r'''_{ui} = r''_{ui} - b_u \qquad \forall (u,i) \in \text{Interactions}$$

$\hat{r}_{ui} = r_{avg} + b_i + b_u$

Content Based

- Content Based Filtering:
  use the ICM
- Knowledge Based Filtering:
  when the user only buys occasionally so we can't create user profiles (e.g. a consumer electronics shop). We use additional, often manually provided, information about the users and the available items. content not only of items but even of users.

1 - IBCBF

Item Based Content Based Filtering.

Cosine Similarity

$$s_{ij} = \frac{\sum_a i_a j_a}{\sqrt{\sum_a i_a^2 \sum_a j_a^2} + C}$$

where x_{if} tells you if feature f is present in item i.

$$\tilde{r}_{ui} = \frac{\sum_{j \in KNN} r_{uj} s_{ij}}{\sum_{j \in KNN} s_{ij}}$$

2 - IBCBF with TF-IDF

it is useful to use TF-IDF as well:

TF

$$TF = \frac{N_{ai}}{N_i}$$

N_{ai} is the number of times attribute a appears in item i (usually 1).
N_i is the number of attributes of item i.

$$IDF = \log \frac{N_I}{N_a}$$

N_I is the number of items.

N_a is the number of items in which attribute a appears.

$$\text{TF-IDF}(a, i) = TF(a, i) \cdot IDF(a)$$

In the TF-IDF model, the document is represented not as a vector of boolean values for each keyword but a s a vector of the computed TF-IDF weights.

Collaborative Filtering

1 - UBCF

User Based Collaborative Filtering

Implicit

Cosine Similarity

$$s_{uv} = \frac{\sum_i r_{ui} r_{vi}}{\sqrt{\sum_i r_{ui}^2 \sum_i r_{vi}^2} + C}$$

$$\tilde{r}_{ui} = \frac{\sum_{v \in KNN} r_{vi} s_{uv}}{\sum_{v \in KNN} s_{uv}}$$

Explicit

Pearson Similarity

$$s_{uv} = \frac{\sum_i (r_{ui} - r_u)(r_{vi} - r_v)}{\sqrt{\sum_i (r_{ui} - r_u)^2 \sum_i (r_{vi} - r_v)^2} + C}$$

$$\tilde{r}_{ui} = \frac{\sum_{v \in KNN} (r_{vi} - r_v) s_{uv}}{\sum_{v \in KNN} s_{uv}} + r_u$$

2 - IBCF

Implicit

Cosine Similarity

$$s_{ij} = \frac{\sum_u r_{ui} r_{uj}}{\sqrt{\sum_u r_{ui}^2 \sum_u r_{uj}^2} + C}$$

$$\tilde{r}_{ui} = \frac{\sum_{j \in KNN} r_{uj} s_{ij}}{\sum_{j \in KNN} s_{ij}}$$

Explicit Ratings

Adjusted Cosine Similarity

$$s_{ij} = \frac{\sum_u (r_{ui} - r_u)(r_{uj} - r_u)}{\sqrt{\sum_u (r_{ui} - r_u)^2 \sum_u (r_{uj} - r_u)^2} + C}$$

$$\tilde{r}_{ui} = \frac{\sum_{j \in KNN} r_{uj} s_{ij}}{\sum_{j \in KNN} s_{ij}}$$

probably we can avoid to put the bias into the prediction formula because they rule out each other, being considered always r_u.

Association Rules

Support

$$\text{Support}(i, j) = \frac{\text{SupportCount}(i, j)}{N_U}$$

Confidence

$$\text{conf}(i, j) = \frac{\text{SupportCount}(i, j)}{\text{SupportCount}(i)}$$

Similarity as Confidence! - not symmetric

$$P(i|j) = \frac{\# < i, j >}{\# < j > + C} = s_{ij} = \text{conf}(j, i)$$

$$score_{i \in I} = \sum_{\text{r} \in \text{ rules recommending } i} \text{support}_r \cdot \text{conf}_r$$

it's a weighted sum of supports, the weights are the confidences.

Generally speaking:

- you compute the support for any pair of items (i, j) and if it is greater or equal to $minsup$, you consider it as a frequent itemset.
- for each frequent itemset (i, j), you consider all the association rules associated to such frequent itemset (in this case $i \to j$, $j \to i$) and for each association rule, if its confidence is greater than $minconf$ it is a valuable association rule.

Machine Learning Recommenders

SLIM, BPR, IALS

SLIM

Sparse Linear Methods.
It is a family of algorithms whose goal is to minimize Residual Sum of Squares.
we can see a recommendation task as a machine learning task in which we need to approximate the weights of a similarity matrix.
the formula that SLIM wants to optimize is

$$L = |R - RS|_2 + \alpha|S|_1 + \beta|S|_F$$

The one above is probably just an example of possible optimization. SLIM refers t the fact that I want to minimize the residual, and add some penalization for overfitting.

Elastic Net

$$L = |R - RS|_2 + \alpha|S|_1 + \beta|S|_2$$

Closed Form Solution

It is possible if only ridge is present as a regularization term.

$$S_j = (R^T R + \beta I)^{-1} R^T \bar{r}_j$$

Ra**t**to **r**a**b**b-m**Ino r**a**t**to **r**o**j**o.

Early Stopping

- Speak about early stopping.
- divide dataset in training, validation, and test set.
- Use MSE on training set, but do early stopping on MAP.

BPR

Bayesian Pair-wise Ranking.

BPR is able to maximize AUC.

AUC plots fallout (x) with respect to recall (y) .

$$\text{Fallout} = \frac{\text{num of non relevant recommended items}}{\text{num of non relevant items}}$$

$$\text{Recall} = \frac{\text{num of relevant items suggested}}{\text{num of relevant items}}$$

BPR works both with implicit and explicit ratings.

Problem: popularity bias

Explicit

- positive rating: from 4 to 5
- negative rating: from 0 to 2
- Consider the user bias! no formula provided but you can figure it out.

Implicit

- positive rating: 1

- negative rating: 0

pick random triplets of user, positive, and negative ratings and optimize through stochastic gradient descent

$$\tilde{R} = RS$$

$$\tilde{x}_{uij} = \tilde{r}^{+}_{ui} - \tilde{r}^{-}_{uj}$$

$$\max \sum_{uij} \log \frac{1}{1 + e^{-x_{uij}}} - ||S||_2$$

Slim BPR

simply, S is computed through Slim. same as above.

$$\tilde{r}_{ui} = \bar{r}_u \bar{S}_i$$

$$\tilde{r}_{uj} = \bar{r}_u \bar{S}_j$$

$$\tilde{x}_{uij} = \tilde{r}_u - \tilde{r}_{uj}$$

IALS

Implicit Alternating Least Squares.
It is for implicit ratings.
We have to set up a preference and a confidence value for each user-item pair.

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

$$c_{ui} = 1 + \alpha r_{ui}$$

where α is a hyperparameter (usually set to a value between 15 and 40).

the loss function is:

$$\min_{X,Y} \sum_{u,i} c_{ui}(p_{ui} - X_u^T Y_i)^2 + \lambda \left(||X||_2 + ||Y||_2 \right)$$

Matrix Factorization Techniques

FunkSVD

Matrix Factorization with MSE using ALS and iterating.

we learn $\tilde{R}$ as a product between two matrices A and B.

$A : U \times F$

$B : I \times F$

F is a hyperparameter and represents the number of latent factors.

$$L = ||R - A \cdot B||_2 + \alpha||A||_? + \beta||B||_?$$

Check this out as well, could consider Frobenius instead if you want.

$$MSE = \sum_{(u,i) \in N^+} (r_{ui} - \tilde{r}_{ui})^2$$

matrix factorization advantage: there are way less parameters.

use ALS to update weights.

- $\text{num of latent factors } = 1 \rightarrow \text{Top Popular}$
- $\text{num of latent factors} = \infty \rightarrow \text{Overfitting}$

SVD ++

Version of FunkSVD that works better for explicit ratings.

It considers global effects.

$$\tilde{r}_{ui} = \mu + b_u + b_i + A_u \cdot B_i$$

same loss function of FunkSVD, simply put $\tilde{r}_{ui}$ as I've described it.

Adapted SVD ++

problem with SVD++: it is memory based.

let's make it model based.

$$\tilde{r}_{ui} = \mu + b_u + b_i + \Big(R_u \cdot B \Big) \cdot B_i^T$$

$$\tilde{r}_{ui} = \mu + b_u + b_i + \sum_f \Big(\sum_j r_{uj} b_{jf} \Big) b_{if}$$

the product of this matrices with its transpose $(B \cdot B^T)$ returns as a matrix with the same dimensions of a similarity matrix.
There is only one difference between the original SLIM and this kind of SLIM. In the first, matrix S was a generic matrix, so you were learning all possible values. Here, matrix S has been decomposed into the product of two identical rectangular matrices.
The two algorithms are the same if I constrain SLIM to learn a product between two identical rectangular matrices instead of simply S.

Asymmetric SVD

another way to solve the cold start problem, other than Adapted SVD++.

$$\tilde{r}_{ui} = \mu + b_u + b_i + \sum_f \left(\sum_j r_{uj} Q_{fj} \right) Y_{fi}$$

it is called asymmetric because $Q^T Y$ is not symmetric anymore.

Asymmetric SVD is model-based and it has a good quality.
It is considered one of the best recommendations techniques at least for dataset with explicit ratings.

PureSVD

It is the only actual, covered, technique that uses Singular Value Decomposition.

$$R = U \cdot \Sigma \cdot V^T$$

$$[U] = |U| \times k$$

$$[\Sigma] = k \times k$$

$$[V] = |I| \times k$$

$$U^T U = I$$

$$V^T V = I$$

$$\Sigma \text{ is a diagonal matrix}$$

$$\tilde{R} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$$

we aim at finding the 3 matrices on the right member of the equation above by minimizing:

$$\min_{U,\Sigma,V} ||R - \tilde{R}||_2$$

the problem of this approach is that it tends to overfit too much.

How to create an approximate representation of R starting from pure SVD? just take into consideration not all the singular values in Σ (reduce U and V accordingly).

Relationship among FunkSVD and PureSVD

$$R = A \cdot B = \text{FunkSVD}$$

$$A = U\sqrt{S}$$

$$B = \sqrt{S}V^T$$

$$R = USV^T = \text{Pure SVD}$$

FunkSVD can be seen as pure SVD if we put such values as A and B (kind of, not exactly).

Pure SVD Trick - Folding in

Pure SVD can be seen as an item based technique that minimizes the mean squared error (computed on all the rating matrix)

$$R\tilde{V}\tilde{V}^T = \tilde{U}\tilde{S}\tilde{V}^T\tilde{V}\tilde{V}^T = \tilde{U}\tilde{S}\tilde{V}^T$$

$\tilde{V}^T\tilde{V}$ is equal to I, but $\tilde{V}\tilde{V}^T$ is not! this is because we are using approximated matrices.

$\tilde{V}\tilde{V}^T$ is a matrix of dimensions items$\times$ items $\rightarrow$ it is a similarity matrix.

$$RS = \tilde{U}\tilde{S}\tilde{V}^T$$

so if I have a new user (a new row in R) it is easy to compute!

Side Information Recommenders

S-SLIM

the basic loss function of S-SLIM is the following

$$\min_S \alpha||R - RS||_2 + (1-\alpha)||F - FS||$$

where $F = ICM$.

the variant with a penalization term is the following:

$$\min_S \left\{ \alpha||R - RS||_2 + \beta||F - FS||_2 + \gamma||S|| \right\}$$

the norm of the regularization term can probably be any kind of norm we want.

Variant of SSLIM

I think the following is just Collaborative Boosting not explained properly.

$$\min_{S_{CF}, S_{CB}} \left\{ \alpha||R - RS_{CF}||_2 + \beta||F - FS_{CB}||_2 + \gamma||S_{CF} - S_{CB}|| \right.$$

the norm of the regularization term can probably be any kind of norm we want.

Collaborative Boosted Content Based Filtering

it is an alternative version of SSLIM.

1. solve a SLIM problem
$$\min_{S^{CF}} ||R - RS^{CF}||_2 + \lambda||S^{CF}||$$
2. we impose the following:
$S^{CB}_{ij} \neq \sum_f a_{if}a_{jf}$ but $S^{CB}_{ij} = \sum_f a_{if}a_{jf}w_f$
3. we solve
$$\min_{\bar{w}} ||S^{CF} - S^{CB}||$$
4. we discard S^{CB} and we keep the weights vector $\bar{w}$.

we are assuming that S^{CF} is very good.

I think that the fifth step (not specified by the professor) is to solve S-SLIM taking into account an new weighted item content matrix (with the weights learnt).

Tensor Factorization

it is an algorithm belonging to the context-aware recommender systems family. It can be seen as matrix factorization in 3 dimensions.

Now our URM is 3 dimensional (it can have even more dimensions but we'll use this dimensionality since it is the wisest to use).

R can be factorized as $R = A \cdot B \cdot D^T$.

$[A] = U \times F$. It represents how much user u likes latent feature f.

$[B] = F \times I$. It represents how much latent feature f is important to describe item i.

$[D] = F \times C$. It represents how much feature f is relevant in context c.

$$\tilde{R} = \tilde{A} \cdot \tilde{B} \cdot \tilde{D}$$

$$\tilde{r}_{uic} = \sum_{f} a_{uf} b_{if} d_{cf}$$

we can learn the predicted values for the 3 matrices by minimizing $||R - \tilde{R}||_2$ or $||R - \tilde{R}||_F$.

Or we can use BPR to have the best accuracy as possible.

Add bias for user and items if the original R matrix contains explicit ratings.

Sparsity problem, solution: penalization term and reduce dimensionality as long as you can.

Factorization Machines

we turn the recommending problem into a regression problem.

columns corresponding to users.

columns corresponding to items.

columns corresponding to content.

$$\tilde{r}_k = \omega_0 + \bar{x}_k^T \bar{\omega} + \bar{x}_k^T W \bar{x}_k$$

$\bar{\omega}$ is a vector of weights of dimension $U + I$.

W is a matrix of weights of dimensionality $(U + I) \times (U + I)$. It models how much user u likes item j.

W is very big though, so read the following:

The way it is written in the paper it is the following, stick with it if you want to be sure:

$$\hat{y}(\bar{x}) := w_0 + \sum_{i}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < \bar{v}_i, \bar{v}_j > x_i x_j$$

$$n = \text{num cols}, \;\; w_0 \in \mathbb{R}, \;\; \bar{w} \in \mathbb{R}^n, \;\; V \in \mathbb{R}^{n \times k}$$

$< \cdot, \cdot >$ is the dot product of two vectors of size k:

$$< \bar{v}_i, \bar{v}_j >:= \sum_{f=1}^{k} v_{if} v_{jf}$$

a row $\bar{v}_i$ within V describes the i-th variable with k factors. $k \in \mathbb{N}_0^+$ is a hyperparameter that defines the dimensionality of the factorization.

- w_0 is the global bias
- w_i models the strength of the i-th variable
- $w_{i,j} =< \bar{v}_i, \bar{v}_j >$ models the interaction between the i-th and j-th variable. Instead of using an own model parameter $w_{i,j} \in \mathbb{R}$ for each interaction, the FM models the interaction by factorizing it.

FM can be applied to a variety of prediction tasks, among them there are:

- Regression (use MSE or similar loss)
- Binary classification
- Ranking

BPR is a good optimization metric to use in this case, but MSE is the one implemented in libFM.

source

Important: it is possible to prove that, if you use this formula and you strictly follow the rule of putting one only for the user and the item that did the rating, the formula is identical to collaborative filtering matrix factorization, but more complex to write.

Graph Based

Basic Algorithm

I'm gonna talk about it without considering features. the version with features is worth to be mentioned though (simply add nodes for the feature and you are on the horse (it is an italian dictum, it means you are good to go LeL)).

- we model our data as a graph.
- nodes represent users and items.
- an arc goes from a user node to an item node if such user consumed such item, and viceversa.
- random walk for n steps, where n is odd. usually $n = 3$.
- the graph can be formulated as a matrix A such that $|A| = (U + I) \times (U + i)$.

$$A_{i,j} = \begin{cases} 1 \text{ if } i \text{ consumed } j \text{ or if } i \text{ was consumed by } j \\ 0 \text{ otherwise} \end{cases}$$

- we create a probability matrix P of the same dimension of M that tells us what is the probability of going to node j from node i (i is strictly a user, j is strictly an item)

$$P_{ij} = \frac{A_{ij}}{\sum_k^{U+I} A_{ik}}$$

- the probability of getting to node j after $n = 3$ steps from node i is

$$P_{ij} = \sum_a \sum_b P_{ia} P_{ab} P_{bj}$$

P3Alpha

exactly as I've described before, it does not consider feature. the only thing to be said is that now P_{ij} becomes

$$P_{ij} = \left(\frac{A_{ij}}{\sum_k^{U+I} A_{ik}} \right)^{\alpha}$$

It considers 3 steps, as I've said earlier.

Page Rank

It is an example of Random Walk with Restart **probably**.

PageRank is an algorithm that measures the transitive influence or connectivity of nodes.
It can be computed by either iteratively distributing one node's rank (originally based on degree) over its neighbors or by randomly traversing the graph and counting the frequency of hitting each node during these walks. PageRank is used to rank websites in Google's search results.
It counts the number, and quality, of links to a page which determines an estimation of how important the page is. The underlying assumption is that pages of importance are more likely to receive a higher volume of links from other pages.

Google recalculates PageRank scores each time it crawls the Web and rebuilds its index. As Google increases the number of documents in its collection, the initial approximation of PageRank decreases for all documents.

It's defined as:

$$PR(A) = (1-d) + d\left(\frac{PR(T_1)}{C(T_1)} + \cdots + \frac{PR(T_n)}{C(T_n)}\right)$$

where

- we assume that a page A has pages T_1 to T_n which point to it (i.e., are citations).
- d is a damping factor which can be set between 0 and 1. It is usually set 0.85.
- $C(A)$ is defined as the number of links going out of page A.

The ***normalized*** sum of the the PageRanks is equal to one.

It is an iterative process.