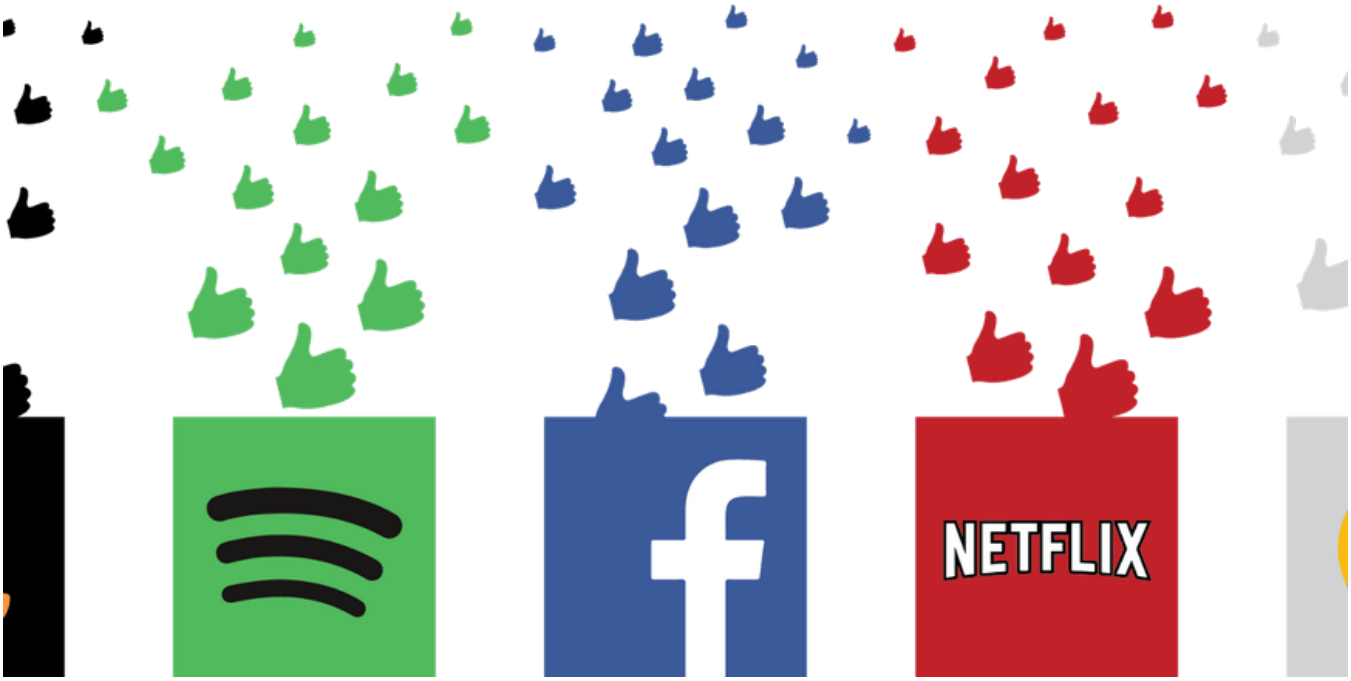


Recommender System

Mixed Recommender System- MF(Matrix Factorization) with item similarity based CF(Collaborative Filtering)

Sourish Dey May 20, 2020 · 10 min read ★

In industries like e-commerce, retail, news-group or music apps, recommendation system is one of the most important aspects in each of the 5 pillars of customer life cycle- reach,acquisition, Develop/nurture, retention and Retention. Today from e-commerce industry(email/On-site product

recommendations)to online advertisement (personalized suggestion with right contents, at right time matching user preferences), Recommender systems are one of the most essential components to influence user online journeys and to gain enhanced customer insight. Therefore it's not surprising that 35% of Amazon.com's revenue is generated by its recommendation engine.In this article we will discuss to improve Recommendation system quality of an e-commerce platform using hybrid approach-combining two building blocks.

Recommendation Engine algorithms

The basic principle of a recommendation engine is filters the data using past user behaviour and product/item characteristics or capturing the interaction between both, using suitable algorithm and recommends the most relevant items to users. There are many approaches and hence many algorithms according to the specific business need and strategy. With out going into much detail, following are a few options although not exhaustive in the order of complexity(from most primitive to most advanced).

a) Popularity Based: Non-personalized and recommend most popular products to every user. A good solution during cold-start stage(when there is sufficient data available)

b) Content Based: Classifier for the user's likes and dislikes based on content's feature.

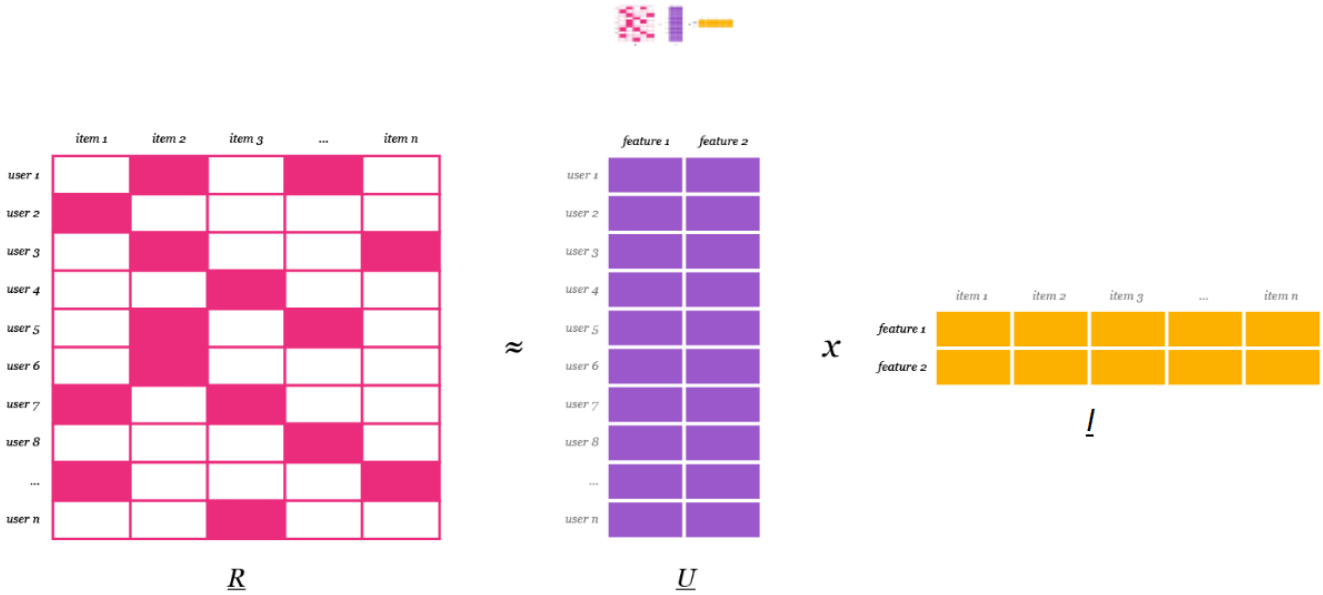
c) Collaborative filtering:

i)User-User collaborative filtering: Based on similarity score between user pairs using suitable distance metric(cosine similarity, Euclidean Distance, Pearson's Correlation etc) picks up the most similar users and recommends products which these similar users have liked or bought previously.

ii) Item- Item collaborative filtering: Here similarity scores between each pair of items are calculated. Using this score matrix based on preference of specific user/user group in the past, similar products are recommended.

d) Matrix Factorization(MF): MF is the 1st generation of of machine learning/ model based approach. The basic idea is to find the user and item embeddings(latent factor representation) by decomposing user-item matrix(past user-item interaction).Although the detailed discussion is beyond scope of this article matrix decomposition is a way of reducing a giant matrix into its constituent parts based on linear algebraic technique. It

is an approach that can simplify more complex matrix operations that can be performed on the decomposed matrix rather than on the original matrix itself and has similarity to the techniques like PCA, SVD etc.



Matrix Factorization

In recommendation system context the decomposition happens from massive user-item matrix(sometimes called rating matrix) to individual user and item matrix. Let us assume that we have to find k latent features/factors with rating matrix R(MxN), user matrix U and item matrix I then the decomposition happens according to following fashion:

R=U∑I , where:

· N is the total number of movies

· K is the total latent features/factors

· R is MxN user-movie rating matrix, where M and N is total number of users and items resp.

· U is Mxk user embedding matrix

· I is Nxk item embedding matrix

· Σ is kxk diagonal matrix which represents the essential weights of latent features

This decomposition task can be achieved using different ML algorithms such as ALS(alternating least squares), BPR(Bayesian Personalized Ranking), LMF(LogisticMatrixFactorization) etc.

e) Other embedding methods: ML based embedding techniques like node2vec, prod2vec etc

f) Deep learning and reinforcement learning(RL) technique: These techniques are new and still in the experimentation phase, such as deep neural network based recommendation,Autoencoder, Markov decision process (MDP) etc.

Business problem and data:

The business problem is to develop a recommendation engine for a e-tailer dealing with thousands of products(sku) for its online users and the raw data is can be found here. After preprocessing the data and adding user action(here 'click') and removing all missing userIds we have following fields.

a) sessionId: unique index

b) userId

c) itemId

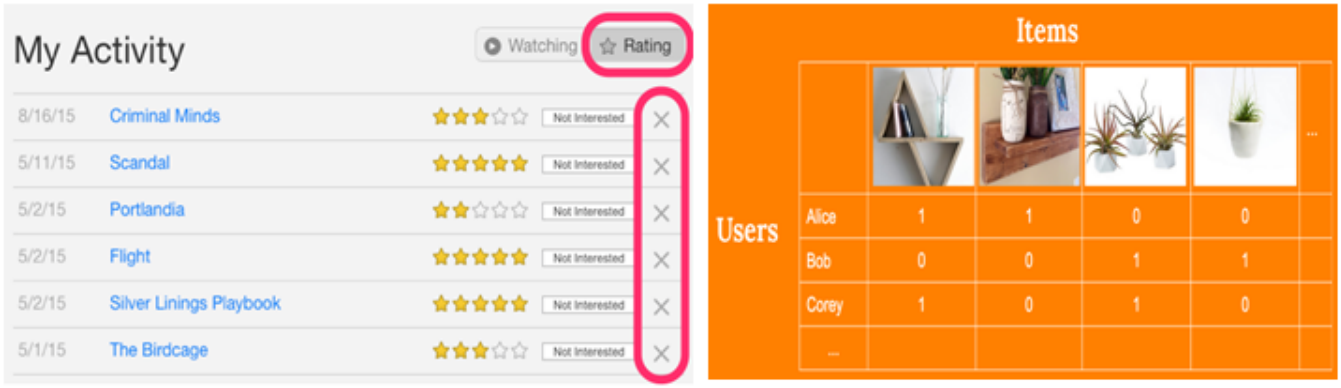
d) action

e) timeframe: unique for each sessionId

> Precisely based on past user action(here clicks/views), we need to recommend personalized 20 products from the universe of thousands of possible options, for each customer, which they are most likely to click during next online session. In most of the cases being important indicators of ultimate customer journey towards conversion/sales, online retailers do care about click, views, impressions etc to have better understanding of users.
>
> While the base solution is a popularity based approach(non personalized and static most popular item list), the goal is to improve the recommendation quality which affects top line of the business.

In this context it's essential to discuss Types of data in recommender systems.

a) Explicit feedback data: As the name suggests is an exact number given by a user to a product. Some of the examples of explicit feedback are ratings of movies by users on Netflix, ratings of products by users on Amazon. Explicit feedback takes into consideration the input from the user about how they liked or disliked a product. While explicit feedback data are quantifiable, Explicit feedback is hard to collect as they require additional input from the users — think When was the last time you rated a movie on Netflix? Also People normally rate a movie or an item on extreme feelings — either they really like the product or when they just hated it.



Explicit vs. Implicit Feedback

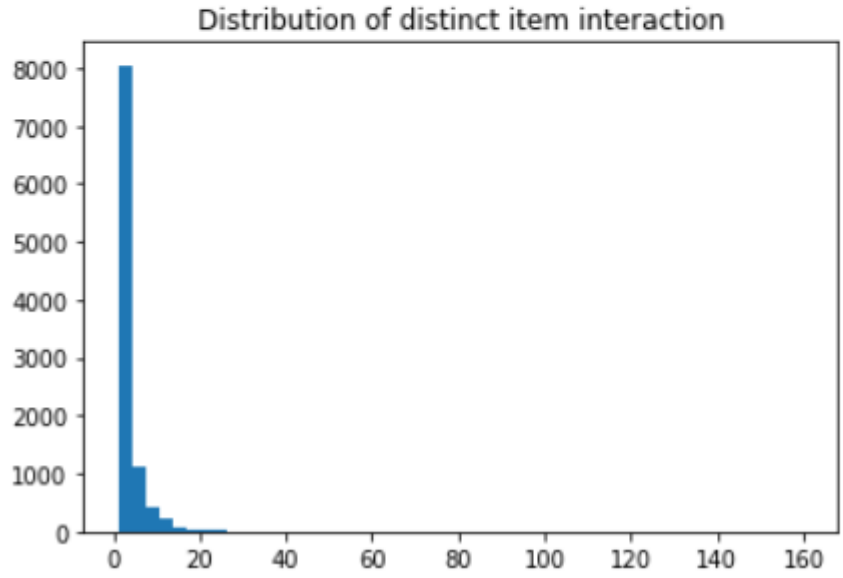
b) Implicit feedback data: Implicit feedback doesn't directly reflect the interest of the user but it acts as a proxy for a user's interest and most of the data available in the world of recommendation system is implicit feedback data. Examples of implicit feedback datasets include browsing history, clicks on links, count of the number of times a song is played, the percentage of a web page users have scrolled.

As discussed above in our case the data is clearly an Implicit feedback data of user action as a proxy of user preference.

Solution Approach:

Before coming to concrete solution, let's explore our data a bit.

```
   sessionId  userId  itemId action  timeframe eventdate  userID
0         48       2   24764  click       8863  4/9/2016       2
1         48       2   24764  click     496381  4/9/2016       2
2         48       2   24764  click     265216  4/9/2016       2
3         48       2   24764  click     519975  4/9/2016       2
4         48       2   24764  click     456437  4/9/2016       2
unique userId: 9999
unique itemId: 16177
avg. distinct item count per user: 3.216821682168217
avg. distinct user count per item: 1.9883167459974038
```



So in most of the cases a user interacted with only few items(<5) and vice versa.

To develop a better recommendation engine over base popularity based system, we explored following options:

a) Pure CF based: We tried item-item similarity/item-item CF method over User-User collaborative filtering user taste changes over time and so user-user similarity matrix. Item-item similarities remains more static over time. However readers can try user-user CF as well.

b) Pure MF based: ALS algorithm based MF method.

c) Hybrid recommendation: A mix of MF and item-item CF in sequential order. After MF-ALS the intermediate result is further filtered using item-item cosine similarity. It's worth exploring other distance based approach like Euclidean, Pearson's Correlation etc.

Train-validation design:

To measure the performance production like A/B test environment is simulated. We splitted the data in to train-validation based on time stamp of each user.Took all timestamps except the last timestamp for training the algorithm and predict top 20 for each user with products.Then we validate the prediction vs. actual user action for each user in respective last timestamp.

Evaluation Metric:

When we mention quality of recommendation we essentially take care about following:

c) Mean Reciprocal Rank(MRR) at k: As product recommendation algorithm generates lists according to user preference rank of the items matters and is a measure to evaluate systems that return a ordered list. : In our case MRR at 20.

d) Normalized Discounted Cumulative Gain(NDCG): Apart from MAP/relevance measure NDCG further tunes the recommended lists evaluation. As a context of recommendation system/information retrieval NDCG is a important metric as it is able to use the fact that some items are “more” relevant than others. Highly relevant items should come before medium relevant items, which should come before non-relevant items.

Detailed overview of these metrics can be found here.

e) AUC: As a ranked list solution we measure AUC between two lists — predicted and actual with cut-off at 20 items for each user and then take the average across all users.

Methodology:

The detailed solution framework with code and data can be found in this github repository.

A) Data preparation: We sequentially ordered actions(click) of each user to find the last timestamp. After splitting the data into train-validation based on timestamp we included only those userId, who are present both in training and validation datasets to get a fair evaluation.

B) Data Transformation: We need to transform the sequential transaction like data to suitable user-item matrix with count of each item interaction for each user.It’s same like term frequency matrix in Bag of Words(BoW) method. Also during data exploration we notice there is high sparsity, which is quite usual in the world of recommendation

system. In real business scenario it might be impossible to store and process such humongous matrix. So we need to convert the big user-item matrix to sparse format.

C) Algorithm Selection: On the top of popularity based solution we explored three options:

Pure cosine similarity based item-item CF: Based on users' interaction with product universe we calculated similarity(distance between pairs of item vectors) scores for items pars and finally cosine similarity matrix. Then matrix dot product between similarity matrix(12620 X 12620) and user-item matrix(5352 X 12620)generates item scores(a list 12620 scores for each user)across all users and picking up top 20 product for each user solves the problem.

Pure MF: As a starting point we decomposed user-item matrix (5352 X 12620) to 32 latent dimension embedding matrix for both user(5352 X 32) and item space(12620 X 32) using ALS algorithm. Then the same approach as item-item CF generates user-item score matrix(5352 X 12620) and then pick up 20 items for each user.

The heart of the process is ALS which is an modified version of OLS(ordinary least square) loss minimization according to following equation:

$$\min_{y_*, y_*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \| x_u \|^2 + \sum_i \| y_i \|^2 \right)$$

$$P_{ui} = \begin{cases} 1 & R > 0 \\ 0 & R = 0 \end{cases} \text{ where } R = r_{ui}$$

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

where

· ***x and y:*** randomly initialized user and item matrix of latent factor dimension. These will get alternatively updated to minimize squared error and hence the algorithm is

called ALS.

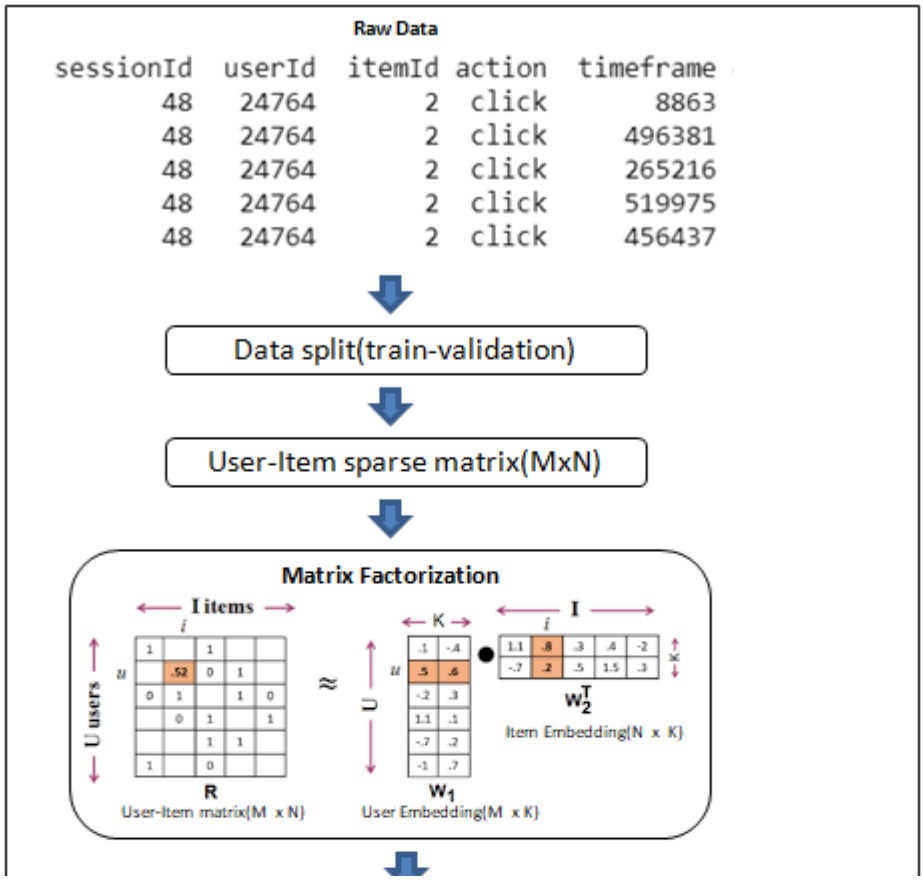
· ***rui:***rating, in our case count of interaction between user u and item i.

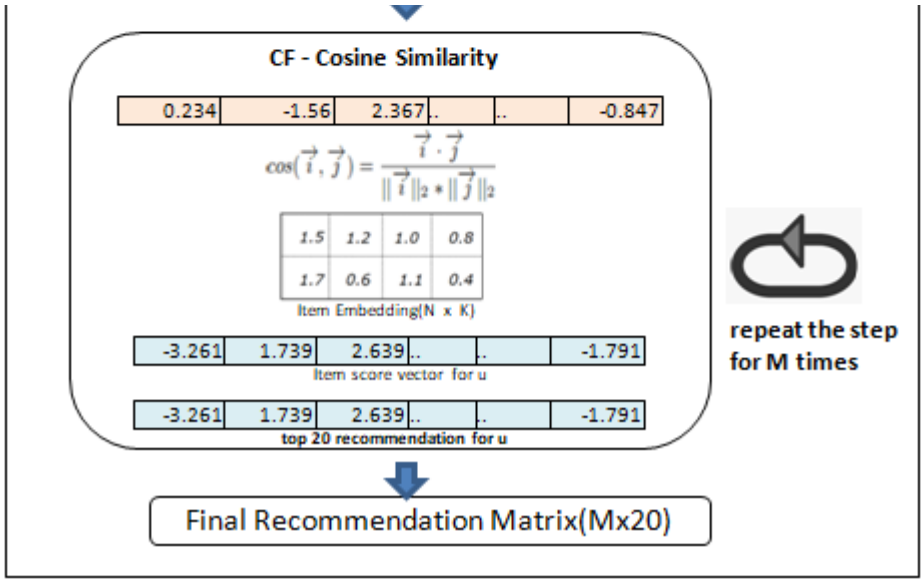
· ***α and Cui:*** *Cui or Confidence values — can be defined as the worth or the value we give to the interaction(user u and item i). With more count of User u clicking(a event) item i increase the interaction weight and hence algorithm's confidence on particular interaction and this confidence increases in linear scale of multiplier* ***α.***

· ***λ:*** *Regularization to reduce over fitting (we're using 0.1).*

· ***pui:*** *The binary preference for an item i for a user u. one if we know the preference and zero if we don't.*

c) **Mix of MF and item-item CF**: This hybrid approach provides even further lift to the overall performance. Intuitively it learns not only the user-item latent characteristics, but also incorporate item similarity component for each user to further refine the recommendation.





Detailed Methodology

After getting the decomposed user embedding(5352 X 32) and item embedding(12620 X 32) from ALS-MF algorithm, we apply cosine similarity technique between individual user embedding vector(1 X 32) and item embedding to get final score list(1 X 12620). After sorting the final score list picking up top k items creates the final recommended list(1 X 20) for each user and repetition across all users produce final recommendation matrix(5352 X 20). The amazing part of the solution is

While solution produces better quality recommendation than stand-alone MF-ALS, it comes with out significant additional computation cost, as the heavy computation part -ALS-MF training is executed only once to create the embedding. Here CF process-cosine similarity calculation doesn't involve heavy computation as it's done for each user level unlike usual cosine similarity(item-item CF) across entire user-item universe.

Performance Evaluation

In final hybrid solution we achieve 40.12% average match rate across users, almost 9% improvement over pure MF based recommendation. Although most of the intelligence lies in creating embedding during MF process,CF(further cosine similarity based filtering of product pairs for each user) adds significant extra lift to the recommendation engine. If we look at the AUC figure 67.5%(6.5% improvement from pure MF method),contrary to binary classification(with only 2 options), here with massive product space of 12620 options it's quite a decent and optimistic result for the business.

Methodology	Metrics			
	match rate	AUC	MAPk	MRRk
popularity	2.47%	50.13	0.011	0.0039
item-item CF	4.76%	52.49	0.018	0.0057
MF-ALS	36.79%	63.36	0.114	0.0178
MF-ALS with cosine similarity CF	40.12%	67.48	0.167	0.0232

Performance Comparison

Takeaways and Road Ahead:

For further improvement of the quality following few strategies are worth exploring:

a) Different MF/embedding generation algorithm: Embedding generation using Bayesian Personalized Ranking optimization(BPR) criterion involves pairs of items(the user-specific order of two items) to come up with more personalized rankings for each user and hence potentially might produce better recommendation than ALS. There are other algorithms such as LMF, network based embedding(e.g.node2vec) etc. are worth exploration.

b) Hyperparameter tuning: Apart from algorithm the other critical factor in embedding generation is number of latent factors k to compute and it's a hyper parameter. The optimum value of k can be found using suitable search. Also **regularization** factor to avoid over fitting to choose carefully. Another control liver is linear scale of multiplier α.

c) Other deep learning and RL based algorithm: Although in a nascent stage, still worth exploring.

Recommendation engine is a multi-faceted problem and there are many different approaches to solve the same problem according to the complexity of the business and organizational data science maturity and positioning. Not always designing a very complex recommendation system and maintenance overhead produces proportional ROI for the business.

Get this newsletter You'll need to sign in or create an account to receive this newsletter.

Machine Learning Matrix Factorization Recommendation System Data Science

Ecommerce Solution

About Help Legal

Get the Medium app