Recommendation System

Kaige Yang
University College London
`Kaige.yang.11@ucl.ac.uk`

Abstract

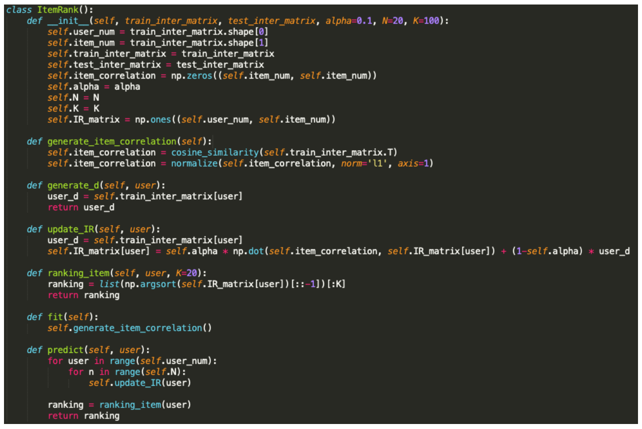
This report presents the detail of a set of widely used recommendation systems.

1 Introduction

In this report, the following algorithms are implemented and tested. An unified framework is created for all algorithms. The framework consists of a data interface to take into training data, a fit function to train underlying model and a predict function to rank items for each user. A set of metric functions are created to evaluate ranking results with respect to test data. Figure 1 shows an example of the framework.

Collaborative Filtering	ItemKNN	UserKNN	MF	BPR-MF	NeuCF
Feature-Based	FM	DeepFM	Wide&Deep	-	-
Random-Walk	ItemRank	RecWalk	P3alpha	RP3beta	-
Graph-Based	GCMC	NGCF	GNMF	-	-

Table 1: Algorithms



```python
class ItemRank():
    def __init__(self, train_inter_matrix, test_inter_matrix, alpha=0.1, N=20, K=100):
        self.user_num = train_inter_matrix.shape[0]
        self.item_num = train_inter_matrix.shape[1]
        self.train_inter_matrix = train_inter_matrix
        self.test_inter_matrix = test_inter_matrix
        self.item_correlation = np.zeros((self.item_num, self.item_num))
        self.alpha = alpha
        self.N = N
        self.K = K
        self.IR_matrix = np.ones((self.user_num, self.item_num))

    def generate_item_correlation(self):
        self.item_correlation = cosine_similarity(self.train_inter_matrix.T)
        self.item_correlation = normalize(self.item_correlation, norm='l1', axis=1)

    def generate_d(self, user):
        user_d = self.train_inter_matrix[user]
        return user_d

    def update_IR(self, user):
        user_d = self.train_inter_matrix[user]
        self.IR_matrix[user] = self.alpha * np.dot(self.item_correlation, self.IR_matrix[user]) + (1-self.alpha) * user_d

    def ranking_item(self, user, K=20):
        ranking = list(np.argsort(self.IR_matrix[user])[::-1])[:K]
        return ranking

    def fit(self):
        self.generate_item_correlation()

    def predict(self, user):
        for user in range(self.user_num):
            for n in range(self.N):
                self.update_IR(user)

        ranking = ranking_item(user)
        return ranking
```

Figure 1: Example

We test algorithms on three publicly available datasets: namely 1) The **MovieLens** dataset; 2) The **iPlayer** dataset; 3) The **Netflix** dataset. Each dataset is splitted into training and test set. A user graph and item graph are also constructed based on the training set, which are used to facilitate graph-based algorithms.

2 ItemRank

Problem
A recommender system deals with a set of users u_i, $i = 1, ..., N$ and a set of items p_j, $j = 1, ..., M$. The goal is to compute a score r_{ij} that measures the preference of user u_i to item p_j. So we need a scoring algorithm to rank items for every user and the suggest to a user the top-ranked items with respect to personalized scores.

Model
The idea underlying the ItemRank is that we can use the model expressed by a item correlation graph to forecast user preferences. For every user in the training set we know the ratings she assigned to a certain number of items. Thanks to the item correlation graph, we can spread user preference through the graph. The spread algorithm we apply has to possess two properties: propagation and attenuation.

- Propagation: An item is related to a item liked by a user will also be a good suggestion for this user.
- Attenuation: Good items transfer their positive influence through the graph, but this effect decrease its power of we move further away from good items.

Item Correlation Graph
We compute the correlation matrix $\tilde{C} \in \mathbb{R}^{M \times M}$. Each entry $\tilde{C}_{ij}$ represents the correlation between item i and j which is measured the by the number of common users. Denote $\mathcal{U}_{ij}$ as the user set where users liked both item i and j. Formally,

$$\tilde{C}_{ij} = |\mathcal{U}_{ij}| \tag{1}$$

where $|\cdot|$ denotes the cardinality of a set. Moreover, we define $\tilde{C}_{i,i} = 0$.

We normalized the matrix $\tilde{C}$ to obtain a stochastic matrix

$$C_{ij} = \frac{\tilde{C}_{ij}}{w_j} \tag{2}$$

where w_j is the sum of entries in j-th column of $\tilde{C}$.

Then, C is the correlation matrix, every entry contains the correlation index between item pairs. This correlation matrix can also be viewed as the adjacency matrix of correlation graph $\mathcal{G}$. Nodes in $\mathcal{G}$ represents items and there is an edge e_{ij} if $C_{ij} > 0$.

Rank Algorithm
Consider a grapg $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the PageRank algorithm computes an importance score $PR(n)$ for every node according to the graph topology. Formally,

$$PR(n) = \alpha \sum_{(q,n)\in\mathcal{E}} \frac{PR(q)}{w_q} + (1-\alpha)\frac{1}{|\mathcal{V}|} \tag{3}$$

where w_q is the out-degree of node q and α is a decay factor. The equivalent matrix form is

$$\mathbf{PR} = \alpha \cdot \mathbf{C} \cdot \mathbf{PR} + (1-\alpha)\frac{1}{|\mathcal{V}|} \cdot \mathbb{I}_{|\mathcal{V}|} \tag{4}$$

A generalized form is

$$\mathbf{PR} = \alpha \cdot \mathbf{M} \cdot \mathbf{PR} + (1-\alpha) \cdot \mathbf{d} \tag{5}$$

where $\mathbf{M}$ is a stochastic matrix, its non-negative entries has to sum up to 1 every column and vector has non-negative entries summing up to 1. The vector $\mathbf{d}$ can be tuned to bias the PageRank by boosting nodes corresponding to high value entries and matrix $\mathbf{M}$ controls the propagation and attenuation mode.

ItemRank: We apply the above equation to recommendation system

$$\mathbf{IR}_{u_i} = \alpha \cdot C \cdot \mathbf{IR}_{u_i} + (1-\alpha) \cdot \mathbf{d}_{u_i} \tag{6}$$

where $\mathbf{d}_{u_i}$ is built according to user preference.

The unnormalized $\tilde{\mathbf{d}}_{u_i}$ is defined as

$$\tilde{d}^j_{u_i} = 0 \; or \; r_{ij} \tag{7}$$

The normalized $\mathbf{d}_{u_i}$ is defined as

$$\mathbf{d}_{u_i} = \frac{\tilde{\mathbf{d}}_{u_i}}{|\tilde{\mathbf{d}}_{u_i}|} \tag{8}$$

The itemrank scores $\mathbf{IR}_{u_i}$ can be computed iteratively.

$$\mathbf{IR}_{u_i}(t+1) = \alpha \cdot C \cdot \mathbf{IR}_{u_i}(t) + (1-\alpha) \cdot \mathbf{d}_{u_i} \tag{9}$$

This dynamic system has to be run for every user, luckily it only needs on average about 20 iterations to converge. The ItemRank scores induce a sorting of items. The higher is the probability that a given user will prefer it to a lower score item.

3 RecWalk

RecWalk is a novel framework for top-n recommendations that aims to combine the potential of item-based models to discern meaningful relations between the items, with the inherent ability of random walks to diffuse these relations across the itemspace and exploit the rick network of interactions they shape.

RecWalk produces recommendations based on a random walk with node-dependent restarts designed to prolong the influence of the personalized initialization on the successive K-step landing probabilities of the walk-thereby eliminating the need of clipping the walks early.

Setting
Let $\mathcal{U}$ be the set of users and $\mathcal{I}$ a set of items. Let $R \in \mathbb{R}^{|U| \times |I|}$ be the user-item interaction matrix. Each user is represented by r^t_u which is the corresponding row of $\mathbf{R}$ and each item is represented r_i be the corresponding column of $\mathbf{R}$. We use a item model $W \in \mathbb{R}^{|\mathcal{I}| \times |\mathcal{I}|}$ the ij-th element of which gives a measure of proximity or similarity between item i and j.

Random Walk and Item Models
If W denote the item model and define

$$S = Diag(W\mathbf{1})^{-1}W \tag{10}$$

be the transition probability matrix of the walk. Personalized recommendations for each user u can be produced by utilizing the K-step landing probability distributions of a walk rooted on the items consumed by u:

$$\pi^T_u = \phi^T_u S^K, \;\; \phi^T_u = \frac{r^T_u}{||r^T_u||} \tag{11}$$

An alternative approach is personalized PageRank

$$\pi^T_u = \phi^T_u \sum_{k=0}^{\infty} (1-p)p^k S^k \tag{12}$$

where p is damping factor.

Issues: The landing probability will converge to a stationary distribution irrespectively of the initialization of the walk. This means for large enough K the landing probabilities will no longer be personalized in the sense that they will become independent of the user-specific staring vector ϕ^T_u.

RecWalk: We define $\mathcal{G} = (\{\mathcal{U}, \mathcal{I}\}, \mathcal{E})$ to be the user-item bipartite network. The adjacency matrix

$$A_{\mathcal{G}} = \begin{pmatrix} 0 & R \\ R^T & 0 \end{pmatrix} \tag{13}$$

Consider a random walker travel over the graph $\mathcal{G}$. Suppose the walker currently occupies a node $c \in \mathcal{U} \cup \mathcal{I}$. In order to determine her next transition she flips a biased coin that yields heads with probability α and tail with probability $(1-\alpha)$.

- If heads, then

  - if $c \in \mathcal{U}$, move to one of the items liked by user c uniformly at random.
  - if $c \in \mathcal{I}$, move to user that have rated the item c uniformly at random.

- If tails
  - if $c \in \mathcal{U}$, the walker stays.
  - if $c \in \mathcal{I}$, the walker moves to a related item following item model M (see detail later).

The Transition Probability Matrix
The transition matrix P that governs the behaviour of our random walker can be expressed as

$$P = \alpha H + (1-\alpha)M \tag{14}$$

Where H is the transition matrix of random walk on the user-item bipartite graph.

$$H = Diag(A_{\mathcal{G}}\mathbf{1})^{-1}A_{\mathcal{G}} \tag{15}$$

The Matrix M

$$M = \begin{pmatrix} \mathbf{I} & 0 \\ 0, & M_I \end{pmatrix} \tag{16}$$

where M_I is the item-item transition matrix.

$$M_I = \frac{1}{||W||_\infty}W + Diag(1 - \frac{1}{||W||_\infty}W\mathbf{1}) \tag{17}$$

The first term normalizes the elements and the second term enforces stochasticity by adding residuals to the diagonal, appropriately. This ensures that $W(ij) > W(i'j') \to M_I(ij) > M_I(i'j')$. This prevents items that are loosely related to the rest of the item-space to disproportionately influence the item-item transition and introduce noise to the model.

How to find W
The construction of matrix W itself can be approached in several ways. We propose to user SLIM method. Concretely, for any given item i we find the set of its C closest neighbors in terms of cosine similarity between r_i and r_j. We form a matrix $N_i \in \mathbb{R}^{I\times C}$ by selecting the corresponding columns of the matrix R. We then solve for each item the optimization problem

$$\min_{x\in\mathbb{R}^C} \frac{1}{2}||r_i - N_i x||_2^2 + \gamma_1||x||_1 + \frac{1}{2}\gamma_2||x||_2^2,\ s.t.,\ x \geq 0 \tag{18}$$

We fill the corresponding elements in the i-th columns of the matrix W.

Rank Algorithm
The recommendation score of user u for item i is defined to be the probability the random walker lands on node i after K steps, given that she started on node u.

$$\pi_u^T = e_u^T P^K \tag{19}$$

where $e_u \in \mathbb{R}^{U+I}$ is a vector that contains the element 1 on the position that corresponds to user u and zeros elsewhere.

An alternative approach is

$$\pi_u^T = \lim_{K\to\infty} e_u^T(\eta P + (1-\eta)\mathbf{1}e_u^T)^K \tag{20}$$

4 P3alpha [2]

$P^3\alpha$: A simple graph-based algorithm which implements a random walk between users and items. Items for user u are ranked based on the probability of a random walk with three steps staring from user u. An user-item graph bipartite is constructed. The transition probability matrix P is constructed as follows:
The probability p_{ui} to jump from user u to item i is computed from the implicit user-rating-matrix as $p_{ui} = (r_{ui}/N_u)^\alpha$, where r_{ui} is the rating of user u on item i, N_u is the number of ratings of user u and α is a damping factor. The probability p_{iu} to jump backward is computed as $p_{iu} = (r_{ui}/N_i)^\alpha$, where N_i is the number of ratings for item i. The three-step transition matrix is P^3.

5 RP3beta [5]

Our algorithm are based on walks over the graph $G = (V, E)$ constructed from the users' feedbacks on item(user-item bipartite graph). The vertices V of G represent the union of users and items. All edges in the graph are unweighted/undirected and no parallel edges exist. The adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$ where

$$A(ij) \begin{cases} 1 & if\ e_{ij} \in E \\ 0 & otherwise \end{cases} \tag{21}$$

Denote D as the degree matrix where $D(ii) = \sum_j A(ij)$.

The transition probability from vertex i to j is defined as

$$p_{ij} = \frac{a_{ij}}{d_{ii}} \tag{22}$$

For one step random walk, the transition matrix is given by

$$P = D^{-1}A \tag{23}$$

The s-step random walk transition matrix is

$$P^s = (D^{-1}A)^s \tag{24}$$

We observed that the ranking of items according to the transition matrix P^3 is strongly influenced by the popularity of the items. Hence, for most users, the well-known blockbuster items dominate the recommendation lists. To compensate from the influence of popularity and to leverage recommendation of items from the long tail, we introduce a simple reranking procedure dependent on item popularity. Suppose the original score of item i given by user u is p^3_{ui} (the transition probability after a random walk of length 3). We reweight the score with

$$\tilde{p}^3_{ui} = \frac{p^3_{ui}}{d^{\beta}_{ii}} \tag{25}$$

where $\beta \in R$ and $\beta > 0.0$.

6 ItemKNN [3]

In item-oriented approach a rating is estimated using known ratings made by the same user on similar items. Central to most item-oriented approach is a similarity measure between items where s_{ij} denotes the similarity between item i and j. Frequently, it is based on the Pearson correlation coefficient. Our goal is to predict r_{ui} the unobserved value by user u for item i. Using the similarity measure, we identity the k items rated by u, which are most similar to i. This set of item is denoted by $S^k(i, u)$. The predicted value of r_{ui} is taken as a weighted average of ratings for neighboring items.

`ItemKNN` is a traditional collaborative-filtering approach based on k-nearest-neighborhood (KNN) and item-item similarities. We use the cosine similarity s_{ij} between items i and j computed as

$$s_{ij} = \frac{r_i \cdot r_j}{||r_i||||r_j|| + h} \tag{26}$$

where vector $r_i, r_j \in \mathbb{R}^N$ represent the implicit ratings of users for item i and j. Parameter h (shrink term) is used to lower the similarity between items having only few interactions. The other parameter of the method if the neighborhood size k.

The rating of unobserved item is defined as

$$\hat{r}_{ui} = \frac{\sum_{j\in S^k(i,u)} s_{ij} r_{uj}}{\sum_{j\in S^k(i,u) s_{ij}}} \tag{27}$$

7 ItemKNN-CF

A neighborhood content-based filtering approach with item similarities computed by using item content features

$$s_{ij} = \frac{f_i \cdot f_j}{||f_i|| ||f_j|| + h} \tag{28}$$

where vectors $f_i, f_j \in \mathbb{R}^d$ describe the features of items i and j.

8 ItemKNN-Hybrid

A hybrid algorithm based on item-item similarities. The similarity is computed by first concatenating, for each item i, the vector of ratings and the vector of features $[r_i, wf_i]$ and by later computing the cosine similarity between the concatenated vectors.

9 UserKNN

A neighborhood-based method using collaborative user-user similarities.

10 MF

Latent factor model comprise an alternative approach to CF with the more holistic goal to uncover latent features that explain observed ratings. A typical model associates each user u with a use feature vector $x_u \in \mathbb{R}^d$ and each item i with an item feature $y_i \in \mathbb{R}^d$. The prediction is done by taking an inner product $r_{ui} = x_u^T y_i$.

Explicit Feedback
Many of the recent works, applied to explicit feedback datasets, suggested modeling directly only the observed ratings, while avoiding overfitting through an adequate regularized model, such as

$$\min_{x,y} \sum_{r_{ui}} (r_{ui} - x_u^T y_i)^2 + \lambda(||x_u||_2^2 + ||y_i||_2^2) \tag{29}$$

where λ is used for regularizing the model.

Implicit Feedback
Let us introduce a set of binary variable p_{ui} which indicates the preference of user u to item i.

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases} \tag{30}$$

Note that we will have different confidence levels among items that are indicated to be preferred by the user. We introduce a set of variable c_{ui}, which measure our confidence in observing p_{ui}

$$c_{ui} = 1 + \alpha r_{ui} \tag{31}$$

The objective function is

$$\min_{x,y} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u ||x_u||_2^2 + \sum_i ||y_i||_2^2) \tag{32}$$

We alternate between re-computing user features and item feature and each step is guaranteed to lower the value of the objective function.

11 BPR-MF

12 NeuCF

We present a generall framework named NCF, short for Neural network based collaborative filtering. NCF is generic and can express and generalize matrix factorization under its framework. To supercharge NCF modelling with non-linearity, we propose to leverage a Neural network to learn the user-item interaction function.

Setting
Let N denote the number of user and M be the number of items. we define the user-item interaction matrix $\mathbf{Y} \in \mathbb{R}^{N\times M}$ from users' implicit feedback where $y_{ui} = 1$ or 0.

Here the value of i for y_{ui} indicates that there is an interaction between user u and item i. However, it does not mean u actually like i. similarly a value of 0 does not necessarily mean u does not like i, it can be that the user is not aware of the item.

The recommendation time problem with implicit feedback is formulated as the problem of estimating the scores of unobserved entries in $\mathbf{Y}$, which are used for ranking the items. Model-based approaches assume that data can generated by an underlying model. Formally, they can be abstracted as learning

$$\hat{y}_{ui} = f(u, i|\Theta) \tag{33}$$

the predicted score of interaction y_{ui}, $\mathbf{\Theta}$ denotes model parameters, and f denotes the function that maps model parameters to the predicted score.

Loss function
Two types of objective functions are most commonly used : pointwise loss and pairwise loss.

- Pointwise loss usually follow a regression framework by minimizing the squared loss between $\hat{y}_{ui}$ and y_{ui}. To handle the absence of negative data, they have either treated all unobserved entires as negative feedback ot sampled negative instances from unobserved entries.
- Pairwise loss: The idea that observed entries shoudl be ranked higher than the unobserved ones. As such, instead of minimizing the loss between $\hat{y}_{ui}$ and y_{ui}, pairwise learning maximizes the margin between observed entry $\hat{y}_{ui}$ and unobserved entry $\hat{y}_{ui}$.

Matrix Factorization
Let q_u and p_i denote the latent vector for user u and item i, respectively. MF estimates an interaction y_{ui} as the inner product

$$\hat{y}_{ui} = f(u, i|p_u, q_i) = p_u^T q_i \tag{34}$$

Neural CF

$$\hat{y}_{ui} = f(P^T v_u, Q^T v_i|P, Q, \Theta) \tag{35}$$

where v_u and v_i are one-hot encode of user and item. P and Q are embedding parameters. Θ is the parameter of neural network.

The likelihood function is

$$p(\mathcal{Y}, \mathcal{Y}^-|P, Q, \Theta) = \Pi_{(u,i)\in\mathcal{Y}}\hat{y}_{ui}\Pi_{(u,i)\in\mathcal{Y}^-}(1-\hat{y}_{ui}) \tag{36}$$

The negative logarithm of likelihood

$$\begin{aligned} L = &-\sum_{(u,i)\in\mathcal{Y}} \log \hat{y}_{ui} - \sum_{(u,i)\in\mathcal{Y}^-} \log(1-\hat{y}_{ui}) \\ &- \sum_{(u,i)\in\mathcal{Y}\cup\mathcal{Y}^-} y_{ui}\log\hat{y}_{ui} + (1-y_{ui})\log(1-\hat{y}_{ui}) \end{aligned} \tag{37}$$

MLP

$$z_1 = \phi_1(p_u, q_i) = [p_u, q_i] \tag{38}$$

$$z_2 = \phi_2(W_2^T z_1 + b_2) \tag{39}$$

$$\hat{y}_{ui} = \sigma(\phi_L(z_{L-1}) \tag{40}$$

13 FM

14 DeepFM

15 Wide&Deep

16 GCMC

We consider matrix completion for recommender systems from the point of view of link prediction on graphs. Interaction data such as movie ratings can be represented by a bipartite user-item graph with labeled edges denoting observed ratings. Building on recent progress in deep learning on graph-structured data, we propose a graph auto-encoder framework based on differentiable message passing on the bipartite interaction graph.

17 NGCF

18 Experiment

18.1 Datasets

We test algorithms on three publicly available datasets: namely 1) The **MovieLens** dataset; 2) The **iPlayer** dataset; 3) The **Netflix** dataset.

iPlayer: The iPlayer dataset is obatined from [] which contains the implicit feedback of 9876 users for 23801 iPlayer programs.

# user	9876
# program	23801
start date	2017-01-01
end date	2017-04-30
# interaction	490852
sparsity	0.194%

Table 2: iPlayer Dataset

MovieLens: The MovieLens dataset [4] which contains the ratings of 6,040 users for 3,706 movies and it has been used extensively for the evaluation of top-n recommendation methods

# user	6040
# program	3706
start date	2017-01-01
end date	2017-04-30
# interaction	1,000,209
sparsity	4.47%

Table 3: MovieLens Dataset

Netflix: The Netflix is obtained from [1] which contains ratings of 480189 users on 17770 movies.

# user	80,476
# program	16,821
start date	2017-01-01
end date	2017-04-30
# interaction	1,977,844
sparsity	0.15%

Table 4: Netflix Dataset

Data Preprocessing: Before feed into algorithms, each dataset is converted into an interaction matrix $\mathbf{M} \in \mathbb{R}^{N \times M}$, user meta-data matrix $\mathbf{X}_U \in \mathbb{R}^{N \times d}$ and item meta-data matrix $\mathbf{X}_T \in \mathbb{R}^{M \times \tilde{d}}$.

Train and test dataset: The original datasets are splitted into a training set and test set following either ratio split (e.g., 8:2) or leave-one-out.

18.2 Results

References

[1] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. Citeseer, 2007.

[2] Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. Random walks in recommender systems: exact computation and simulations. In *Proceedings of the 23rd international conference on world wide web*, pages 811–816, 2014.

[3] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

[4] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[5] Bibek Paudel, Fabian Christoffel, Chris Newell, and Abraham Bernstein. Updatable, accurate, diverse, and scalable recommendations for interactive applications. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 7(1):1–34, 2016.