Recommendation based on user experiences

Hai Vu

December 13, 2014

Abstract

Latent factor model is one common technique to build recommendation systems. Standard latent factor model however does not take into account the order in which each individual user makes the ratings. Modeling the shift in user behaviour over time will not only allow making better recommendations to users, but also discover their hidden categories, "level of experiences" or "progression stages". In this project, we apply the standard latent factor technique to a movie review data set and then apply the new technique to model user experience on this data set.

We find that the improvement in prediction depends on the time span of the dataset. But more importantly, we can use the model to draw interesting insight from the discovered "user experiences" - guiding user to the next level of experiences.

1. Introduction

Recommender systems follow 2 main strategies: content-based filtering and collaborative filtering. Collaborative is often the preffered approach as it requires no domain knowledge and no feature gathering effort. The 2 primary methods for collaborative filtering are latent factor models and neighborhood methods.

In user-user neighbourhood methods, similarity between users is measured by transforming them into the item space. Similar logic applies to item-item similarity. In latent factor methods, both user and items are transfomed into a latent featuee space. An item is recommended to a user if thu are similar, their vector representation in the latent feature spase is relatively high.

We select latent factor model because it allows us to identify the hidden feature of the users. These features are time indepedent. We first discuss standard recommeder system that discover time-depedent features in order to make suggestion to users. We then discuss new model (introduced in [3]) to deal with time-depedent feature that we named "user experience". We experiment with a movie data set, evaluate the improvment over standard recommender system, and discuss other benefits of discovering user experiences.

2. Modelling user's time-indepedent latent features

Rating is modelled as a sum of average score, item and users biases, and the similarity of user and items in the latent feature space.

- $\widehat{r}_{ui}=\mu + b_i + b_u + q^T{}_i p_u$

The Baseline algorithm

In principle, the factorization method maps users and items onto the feature space and the similarity on this space is measured by inner product. The higher the product of these 2 projected vectors the better the user rates the movie. Teh hidden feature space has k dimensions.

The challenge is to find the feature space, or in other word, to decompose the user-item matrix into the product of 2 matrices. The first matric links users to the hidden features and the second matric links products to the features. Given that the user-product is sparse we don't try to factorise the matrix (using Singular Value Decomposition approach). Instead we minimize the error on available ratings.

- min $\sum_{u,i} (r_{ui}-(\mu + b_i + b_u + q^T{}_i p_u)^2+\lambda(||q_i||^2 +||p_u||^2)$
- where $\lambda (||q_i||^2 +||p_u||^2)$ is regularization
- μ is the mean rating, b_i and $b_u$are bias for items and users respectively. Hidden feature space has k dimensions.

We will use Stochastic Gradient Descent to find the optimal solution. We chose it over Alternating Leaset Square due to is easy implementation and fast running time**.** Taking derivation to each parameters we arrive at the following update at each iteration:

- $\varepsilon_{iu} = 2\ (r_{iu} - (\mu + b\_item_i + b\_user_u + q_i\ p_u^T)$
- $q_i \leftarrow q_i + \eta\ (\varepsilon_{iu}\ p_u - 2\ \lambda\ q_i\ )$
- $p_u \leftarrow p_u + \eta\ (\epsilon_{iu}\ q_i - 2\ \lambda\ p_u)$
- $b\_item_i \leftarrow b\_item_i + \eta_{bias}(\varepsilon_{iu} - 2\lambda\ b\_item_i$
- $b\_user_u \leftarrow b\_user_u + \eta_{bias}\ (\varepsilon_{iu} - 2\ \lambda\ b\_user_u\ )$
- μ can be calculated from the dataset.

3. Modeling time-depedent user experiences

When a user rates a product, she stays at specific experience level, denoted by an integer e where e = 1,..k (k is the highest exprience level). We assume that

- users at the same exprience level will behave similarly (apart from their hidden features)
- experience level never decreases over time

Our intuitive expectation of user experience is that its level never decreases. We believe that monitonicity is natural assumption and will not narrow too much the scope of application. In addition, the monitonicity constraint will also help to avoid overfitting because only a limited set of experience sequence could be mapped to the sequence of ratings. Now the new model consists of 2 set of parameters :

- set 1: μ, b_item, b_user, q, p for each user experience level
- set 2: the assignment of each rating to an experience level. For each user, the assignment is subjected to the monotonicity constraint.

To train the model we need to find the above set of parameters to minimize error expression as in previous section. As objective function is not convex, we will find the local optimum by alternatively optimize one parameter set while keeping the other set fixed and repeate until the error becomes small enough.

The algorithm

- **Step 0 - Initialization**: for each user, assign experience level evenly so that each user at each level makes rating appoximately equal number of times. (while statisfying monitonicity constraint)
- **Step 1** - Filter ratings according to exprience level and **Train model for each individual experience level** (training smaller experience levels first). Collect all ratings belonging a specific level and apply the standard recommendation method to this dataset. In this step we keep parameter set 2 fixed while optimizing parameter set 1.
- **Step 2 - Reassign ratings** to experience levels**.** For each user, reassign rating times to exprience levels to minimise the error. In this step we keep parameter set 1 fixed while optimizing parameter set 2.
- Repeat steps 1,2 until convergence.

To avoid overfitting we introduce a **smoothing component** into our error function so that parameters of experience level k will not significantly different from those of experience level k-1 . This change impacts the derivation of error and the update rule as follows:

- $q_i \leftarrow q_i + \eta\ (\varepsilon_{iu} p_u - 2\lambda(q_i - q\_other_i))$
- $p_u \leftarrow p_u + \eta(\epsilon_{iu} q_i - 2\lambda(p_u - p\_other_i))$
- $b\_item_i \leftarrow b\_item_i + \eta_{bias}(\varepsilon_{iu} - 2\lambda(b\_item_i - b\_item\_other_i))$
- $b\_item_u \leftarrow b\_item_u + \eta_{bias}(\varepsilon_{iu} - 2\lambda(b\_item_u - b\_item\_other_u))$

While fitting model for exprience k, we use q_other, p_other, b_item_other, b_user_other parameters of the model k-1.

For Step 2, we use dynamic programming to reassign each rating to new experience level.

4. Implementation details

Data set

Availabel at: http://grouplens.org/datasets/movielens/

Format of data files: each row includes user id , movie id, rating, timestamp.

Dataset 100K: number of users = 943, number of movies = 1676, number of ratings = 100,000. Timespan = 214 days. Rating = 1-5. Number of rating / user: at least 20

Dataset 1M: number of users = 6040, number of movies = 3900. number of ratings = 1,000,000. Timespan = 1038 days. Rating = 1-5. Number of rating / user: at least 20

Filtering relevant users

We select only users whose have been at least 10 days (Latest rating and earliest ratign are done at least 10 days apart) in the system and make at least 20 ratings. Users who spend less time in the system have not potentialy reveal her behaviour and therefore will not be considered.

Learning rate and convergence

As we run standard latent factor model repeteadly at step 1 of the Algorithm above, we want to use the same learning rate η and $\eta_{bias}$for each iteration. We obtain these values by running a baseline algorithm on the data set without considering the user exprience at all. For this dataset, we get η_{bias}= 0.02, η = 0.03. Note, we use different learning rate η for p, q and for bias parameters because the bias parameters are simpler to learn.

Initialisation

At step 1 of the algorithm, The initial value of q and p are randomly generated so that <q,p> is in comparable range of the maximal rating value.

Cross validation, testing

For validation, we split the data set so that we can make 10-fold cross validation. Regarding testing, for each tuple (user, movie, timestamp) in the test dataset, we find the user experience level by looking up the level associated with the closest timestamps in the traingning set. Giventhe level , the parameters (μ, b_item, b_user, q, p) are then obtained and Root Mean Square RMSE is calculated.

Avoid overfitting

Overfitting is avoided by

- using regularisation to make the model simple : parameter of the experiences level k should be similar to that of level k-1. We arrive at $\lambda = 2$.
- forcing simple asssignment : monotonicity constraint on experience level for each user

Number of hidden features

Each user has k number of hidden features that are time indepedent and e number of experience levels. The maximal value of k is 20 and e is 5. We don't see RMSE improvement with large k or e value.

Dynamic programming

For Step 2 , we use dynamic programming to reassign each rating to new experience level. Dynamic programming is feasible because the best assignment for ratings r_1, r_2, ...r_n contains the best assginment for ratings r_1, r_2, ...r_{n-1} as its subset. (Here r_1, r_2, ...r_n are ordered according to the time of ratings)

5. Result, discussion

How much improvement does the new model bring ?

If we keep only users from dataset 100K, whose timespan is more than 10 days, we achieve RMSE of 0.887 versus the baseline RMSE 0.997 (the baseline model is the standard recommendation which does not consider user experience). We argue that users who just spend very little time in the movie system will not exhibit behaviour shift, instead they exhibit a **"noise behaviour"**. Once they spend more time in the system, their behaviour will be shaped and follow the monotonicity trend. Therefore removing these noise behaviours will help identify the key user experiences.

Dataset 100K: Baseline RMSE = 0.997. Keeping users who spend at least 10 days in the system. User experience based model: RMSE = 0.871. Improvement over baseline: 12%. Number of experience level :3. Regularisation$\lambda = 2$, number of time indepedent latent features k = 20, learning rate for bias parameter η_{bias} = 0.02,for other parameters η = 0.03.

Dataset	100K	1M
Baseline RMSE	0.997	1.002
User-exprience based model RMSE	0.871	0.847
Improvement over baseline	12%	15%

Does time depedent behaviour exist in the dataset?

This a fundamental question. Based on our experiements we must remove "noisy behaviour" (see above), and consider only users who have been engaged with the system for minum period of time. This duration is application dependent and could be measured in different ways. For a movie system in this project with timespan 200-1000 days (data set 100K and 1M) we observe that the threshold of 10 days is feasible.

Impact of model smoothing

When users jump from one experience level to the next, we don't expect too sudden change in their behaviour. Therefore it is reasonalbe to smooth the model parameters:

- Θ=(μ + b_item$_i$ + b_user$_u$ +q$_i$,p$_u$) to keep the sum $\Sigma_e||\Theta_e$-$\Theta_{e-1}||$ small .

This term is part of our error function describe in Section 3. However, to see how much we have gained with smoothing we also try to minimise

- $\Sigma_e||\Theta_e||$ instead of $\Sigma_e||\Theta_e$-$\Theta_{e-1}||$.

We observe that the smoothing gain in RMSE reduction is **2%** (from 0.8871 down to 0.871 on Dataset 1) .

Benefits of discovering user experiences

Instead of modelling user-experience and then using it to predict future user ratings, one can also just take the latest ratings of each indivudual users and run a standard recommendation on that filtered data set. Which one has better prediction power ? If all users have already reached the highest experience level, then our user-experience model provide less advantages. However, we expect that this is not the case in typical systems.

Another benefit is to identify a group of users who got stuck in certain experience level for longer than average time. Do they need special treatment ? Can they churn ? Or can we help them to progress so othat they stay with us ? This question leads to the next topic below.

Distinctive benefit over traditional recommendation system: guiding to the next level

While traditional system recommend the product the user most probably like at the current state of their experience, our user-exprience based system can do more: give them the product so that they progress one level and also like it.

Here is the outline:

- Step 1) For a user u, find his current exprience level, say e. Find the averate time a user spend in exprience level e.
- Step 2) If user user u has spent much longer than average at this experience level, then use parameter set $\Theta_{e+1}to$ make offer (instead of using Θ_e as normal recommednatio system does)

6. Conclusion

We have experimented with user-exprience based recommender system as suggested in [3]. We confirm that the improvement in term of RMSE over baseline recommender system is in the range of 12-15%.

Our work reveals that:

- For the model to work efficienlty and delivering improvement over baseline recommenders, the dataset needs to be cleaned upfront to remove users who spend too little time in the system. We introduce the conjecture about "noisy behaviour" that could reduce the performance improvement.
- We suggest to amek use of the learnt user experience to make offers to users so that they progress faster. The steps are outlined in section 5.
- We discuss the model smoothing that is necessary to reduce overfitting. We observed that model smothing will increase the performance about 2% in one of our dataset

As future work, the model could be extended to model different classes of progresstions (see [4]) .

References

[1] Y. Koren. Collaborative filtering with temporal dynamics. Commun. ACM , 2010

[2] Y. Koren and R. Bell. Advances in collaborative filtering. In Recommender Systems Handbook . Springer, 2011.

[3] From Amateurs to Connoisseurs: Modeling the Evolution of User Expertise through Online Reviews by J. McAuley, J. Leskovec. ACM International Conference on World Wide Web (WWW), 2013.

[4] Finding Progression Stages in Time-evolving Event Sequences by J. Yang, J. McAuley, J. Leskovec, P. LePendu, N. Shah. ACM International Conference on World Wide Web (WWW), 2014.