PERSONALIZATION REPORT

A Recommender System for E-Commerce

Abstract

The holistic approach

In this project, we designed a dual-model solution for E-commerce companies, which aim to increase total sales revenue by providing high quality recommendations for their end users through multi-channel recommendation system.

Addison Li, Chuqi Yang, Haozheng Ni, Haoran Li

Contents

Part I: Intro:

Industrial background:

As recommendation systems are widely applied in E-Commerce, we selected the dataset from RetailRocket, a SAAS company which designs recommendation systems to help e-commerce companies to provide better recommendations for the end consumers. RetailRocket's customers include Vans, Nespresso, and Groupon, to name a few.

Business Objective:

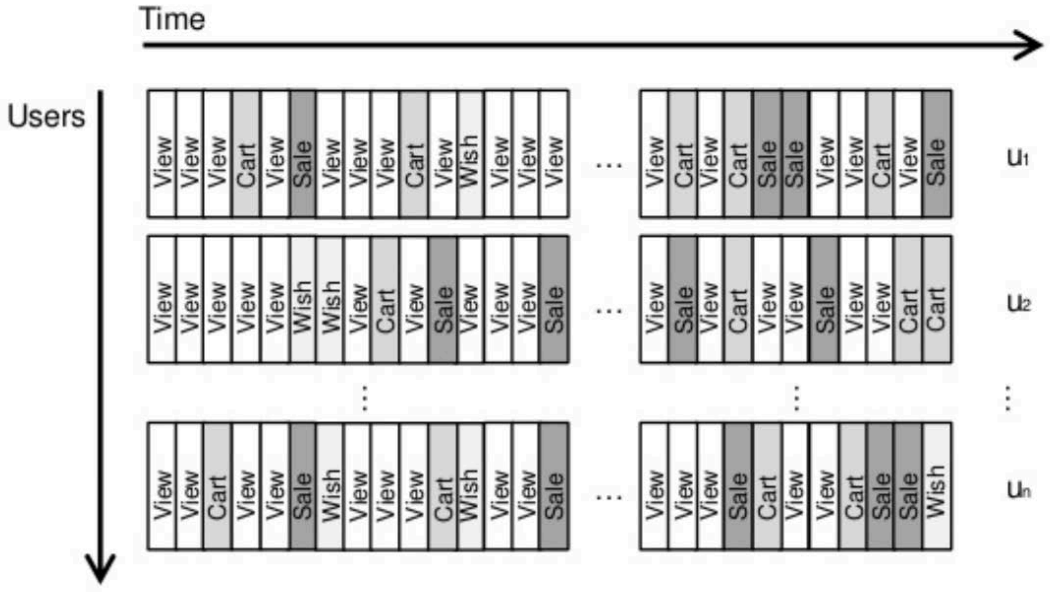
Positioning ourselves as the product managers and data scientists at RetailRockets, our goal is to help our customers, ie: E-Commerce companies to increase their sales revenue. We designed our product as a recommendation trio package, which is consisted of email, web, and mobile recommendations, with a core of double-model recommendation system.

In order to reach this goal, we separated the end consumers and strategized differently.

1, For new customers who we have little preexisting data of, we recommended them the most popular items from the data we collected from the existing shoppers.

2, For existing customers, we aim to increase their purchase by improving recommendation relevancy, enhancing shopping experience, and expanding customer retention with our recommendations. Our two models are: 1) Latent Factor Model; 2) Hybrid model: User-based KNN Collaborative Filtering and past user transaction behaviors. They work well and complement one another to make appropriate recommendations for these consumers.

Dataset overview:



The raw data includes 1,407,580 users, and 235,061 items. It contains: userID, itemID, categoryID, transactionType, ParentID, and Time Stamp. This data set does not come with explicit rating records. Therefore, we make predictions based on implied preferences reflected by users' behaviors.

We first eliminated Time Stamp and ParentID to move forward with our exploratory analysis. We didn't consider this factor because the quantity of the purchase, i.e., the time stamps per user is very sparse, simply not sufficient to provide reasonable conclusions. ParentIDs became insignificant after our items were aggregated under the main categoryIDs. Therefore, neglecting them won't influence our prediction. Due to the differences in implementation, more details on cleaning, sub-setting, and data manipulation will be explained in depth in the following technical section.

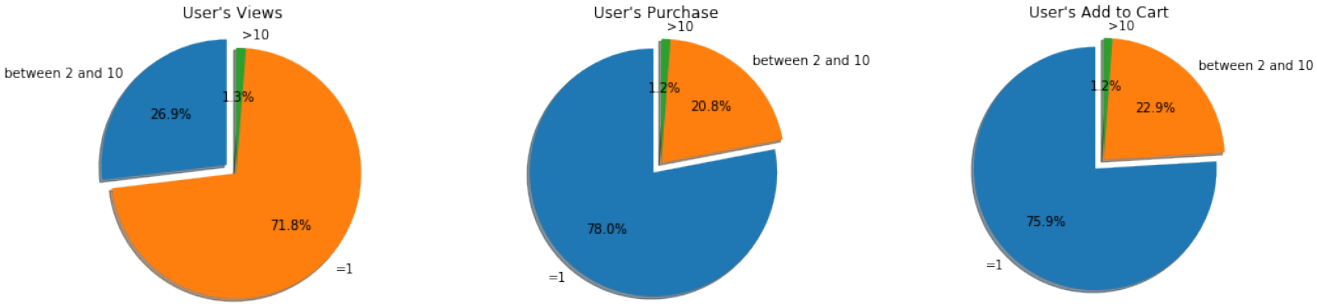
Part II: Technical Approach:

Preparation:

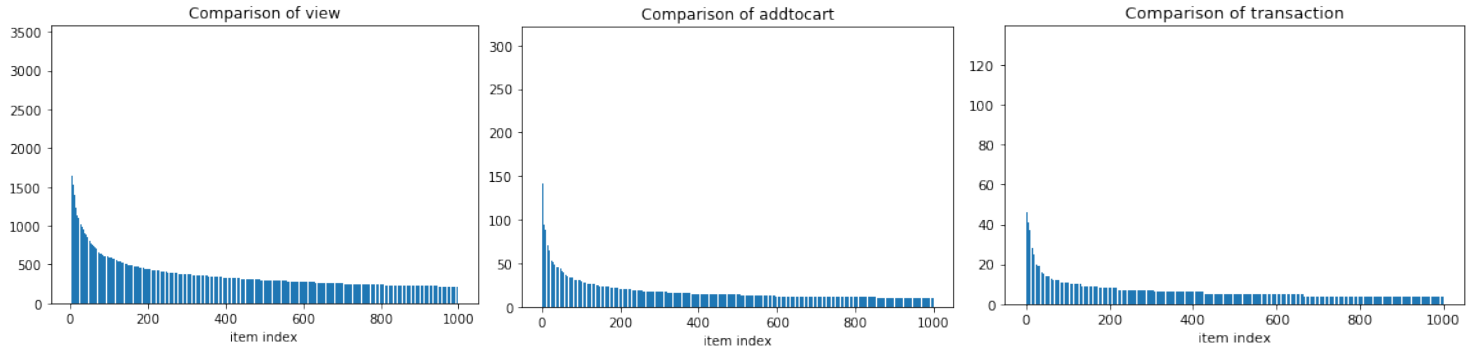
- Data cleaning and segregation – for new customers:

We first distinguished our users into two groups, new users, and existing users. Since we don't have any information about the new users' personal preference or shopping behaviors, we decide to recommend them the most popular items based on our existing customers' shopping history. Since there is no subsequent data about how these new users behave after our recommendations, we have yet any method to test the conversion rate of this group. Therefore, we focused on our existing users.

Most users in our dataset are not active, most of them only made one views, add to cart or purchase activity – graph to the right. This observation leads to our sampling method of users in next section.



Moreover, the items have long-tail effect, meaning that some items are more popular than the rest, which motivates us to sample the non-popular items as well.

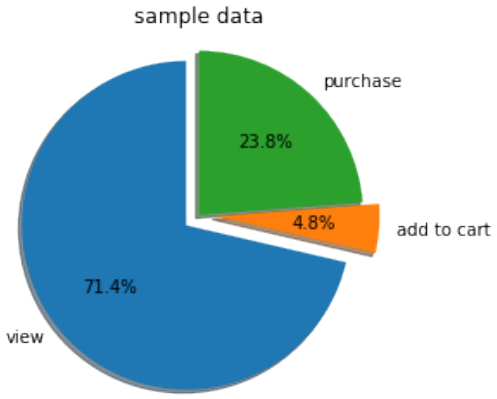


- Systematical sampling – for the existing users:

Due to the specific characteristics of our data, randomly selecting users or items won't provide enough quality data for us to process. To ensure enough data to conduct training and testing, we filtered our end consumers and only kept those who have at least three purchases and at least three views or add-to-cart activities, which yields to 1016 users and 235,061 unique items.

	Category1,2,3	Category2,3,4	Category1,6	Category4,5	…	Category5,9
User1	VA	M	P	M	…	P
User2	M	VA	VA	P	…	VA
User3	VA	M	M	P	…	M
…	…	…	…	…	…	…
Usern	VA	P	M	P	…	M

With 235,061 items and a significant amount of missing data, our user-item matrix is very sparse. This is far from enough to make reasonable prediction. To address this issue, we grouped the items that belong to the same category after recording each user's behavior toward each item, transforming our user-item matrix into user-category matrix. Thus, our matrix ended up with 1016 users and 1962 categories. The proportions of view, add to cart and purchase are more balanced.



Through this transformation, the user will no longer get a recommended item from the exact subcategory that he or she had purchased from, but an item from the general category that he or she has shown activities in. As long as the users purchase anything we recommended, the overall sales revenue will increase and our business objective won't be compromised.

Modeling details:

As mentioned earlier, we built two models to serve our existing customers: Latent Factor Model and a hybrid model that combined User-based KNN Collaborative Filtering and past user transaction behavior.

Latent Factor Model

Part 1 Model

Model

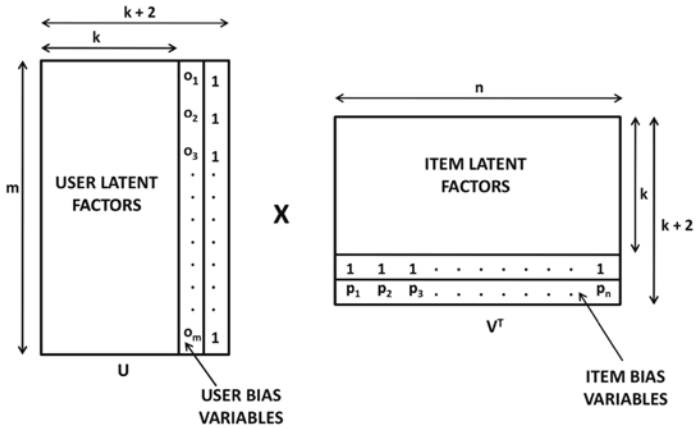
Latent factor model uses latent vectors that represent inter-item relationships in lower dimension, which is ideal for our data set since the items are naturally grouped under sub-categories, indicating that we can make use of their relevancy for prediction purpose.

Our experiment is conducted as follows:

Firstly, we incorporated user bias and item bias in our model. These two biases are two vectors of the existing users' shopping behaviors and biases toward different items. User bias represents the popularity of items based on information from the active existing shoppers. Item bias represents individual users' personal preference toward certain item.

The baseline model only contains those two biases. The mode is represented by the following formula and picture:

$$\widehat{r_{ij}} = o_i + p_j + \sum_{s=1}^{k} u_{is} v_{js}$$



Our target is to minimize the following by stochastic gradient descent method implemented by ourselves.

$$J = \sum_{(u_i, v_j, r_{ij}) \in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} v_{sj} \right) + \frac{\beta}{2} \left(\sum_{s=1}^{k} (||U||^2 + ||V||^2) \right)$$

Then, we divided our data into three parts for training (TR), cross validation (CV), and testing (TE) respectively, 80% for TR and CV, 20% for TE. We then assigned different weights to our categorical data M, V, A, P according to their hierarchical influence over purchasing decisions, transforming them into quantitative values such as 0M, 1V, 2A, 5P. In the training data, Stochastic Gradient Descent is used to tune different sets of parameters through rounds of cross validations.

To find out the best parameter set, we used Hit Rate and Precision as the main metric for this model. The Hit Rate is calculated as follows:

$$Hit\ Rate = \frac{the\ \#\ of\ recommended\ items\ \cap\ the\ \#\ of\ Purchases\ in\ TE}{the\ \#\ of\ recommended\ items\ \cap\ total\ records\ in\ TE}$$

For every user, we make a total of 50 recommendations. We then count how many of these items appeared in the Test set. The number of items serves appeared in both sets serves as our denominator, while the number of items appeared in both recommendations and ended being purchased serve as our numerator.

Hit Rate is indispensible because it enables us to measure how close we can “understand” a user’s preference, which increases the relevancy of our recommendations.

The other metric we look at is Precision, which represents the accuracy of our prediction. It is calculated as follows:

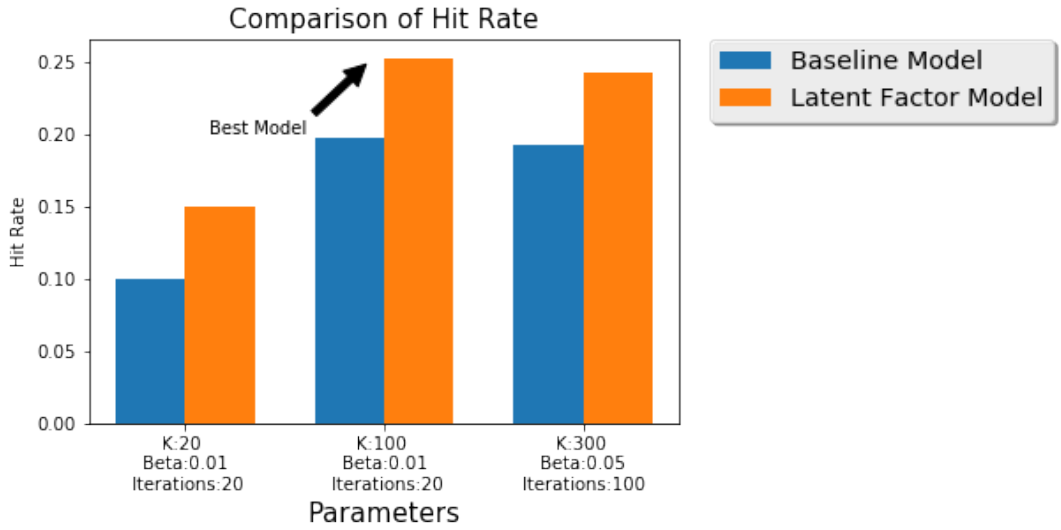
$$Precision = \frac{the\ \#\ of\ recommended\ items\ \cap\ the\ \#\ of\ Purchases\ in\ TE}{the\ \#\ of\ recommended\ items}$$

This is important because it indicates the percentage of how many items we recommended to the user ended being purchased, which could be directly translated into sales growth for our customer companies.

Finally, we picked the set of parameters, which generated the most accurate prediction results to examine the final Test set.

Result:

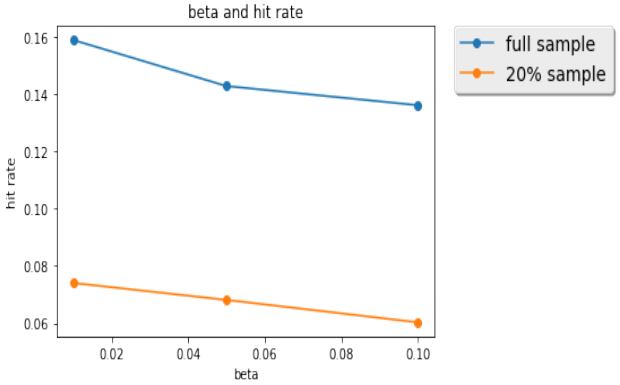
The best result we obtained from cross validation is 100 latent factors with regularization 0.01 and 20 iterations of Stochastic Gradient Descent. Compared with the baseline model and other parameter sets, this set yielded the prediction results of the highest accuracy.

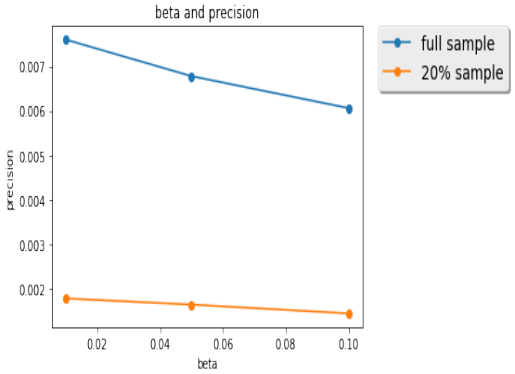


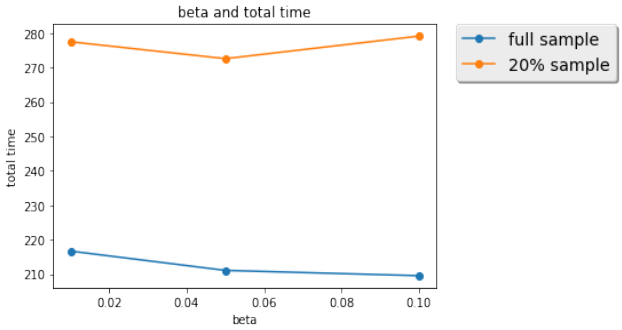
The following plots recorded how Hit Rate and Precision changed as each parameter changed, while other parameters remained fixed. The learning rate of Stochastic Gradient Descent is set to be 0.01 because any larger value can cause our data to overflow. To run through our entire data set, cross validation takes close to 10 hours. This time reduced drastically down for a smaller sample size with 20% data. In the graph above, K stands for number of latent factors, Beta stands for regularization term, and Iteration represents the iteration of Stochastic Gradient Descent optimization. In the graphs below, the run time comparison results for full and 20% sample size are displayed together.

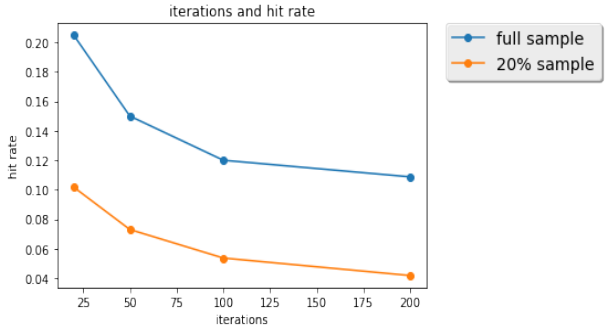
The baseline model only contains the user factor and item factors, without adding the interaction with them, i.e

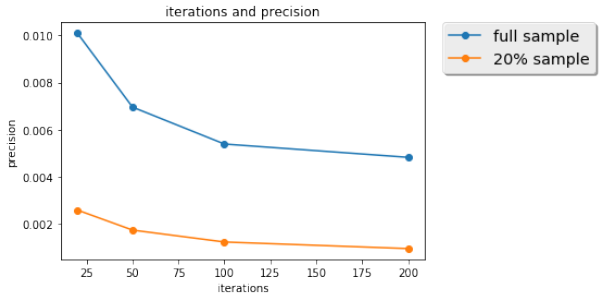
$$r_{ui} = \mu + b_i + b_u$$

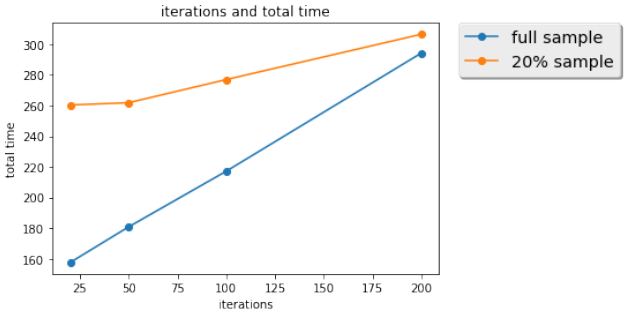

beta and hit rate
hit rate
beta
full sample
20% sample

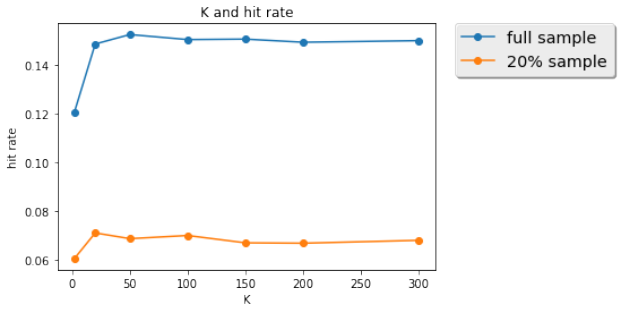

beta and precision
precision
beta
full sample
20% sample

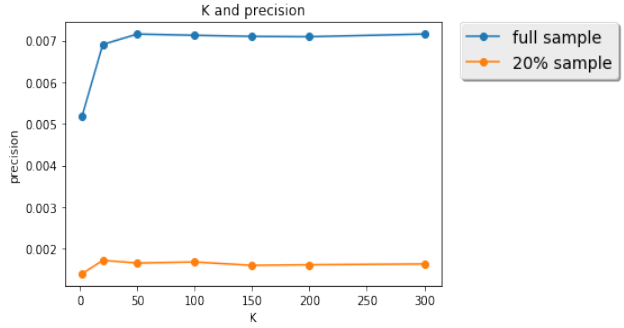

beta and total time
total time
beta
full sample
20% sample

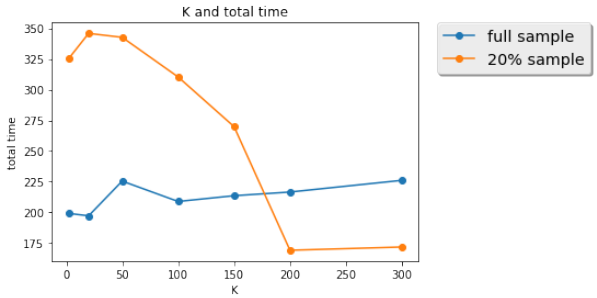

iterations and hit rate
hit rate
iterations
full sample
20% sample


iterations and precision
precision
iterations
full sample
20% sample


iterations and total time
total time
iterations
full sample
20% sample


K and hit rate
hit rate
K
full sample
20% sample


K and precision
precision
K
full sample
20% sample

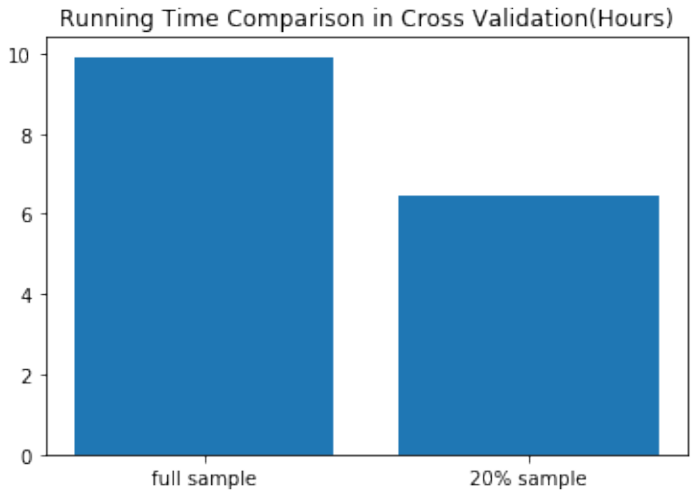


The changes in overall accuracy and evaluation metrics：

Based on the results, both full sample and small sample support that the prediction precision increases with the number of latent factor variables going up to some point, and a moderate K shall be selected to avoid over fitting. Havier regularization may not be helpful in increasing precision, while large number of iterations of Stochastic Gradient Descent will lead to over fitting.

Run-time VS data siz:

As we can see from the graph blow, a sample of 20% of full size costs about 6 hours to run, while the full size data only costs a little under 10 hours. This means that our model can scale relatively well when data size increased by 5 times.



Part 2 Model

We chose Time SVD as the new feature to be added to our latent factor model.

Model

The motivation behind Time SVD is that users' preference is changing over time. We need to have a recommendation strategy that reacts to this change. In addition, other external factors, such as newly added items and shifting in trends can also influence the popularity of items. Therefore, by adding Time SVD, we incorporated these observations, making one step further than conventional SVD to model item factors and user factors as functions of time. As we were designing our feature, we referenced the paper by Y. Koren[1], which proposed the method of adding Time SVD++. However, since our data naturally contains implicit feedback, we removed the implicit matrix decomposition from the model. To reflect the implied relationship, we used 1,2,5 to represent and computed "view", "add-to-cart", and "purchase" numerically.

In SVD, the baseline model that models user and item effect is defined as:

$$b_{ui} = \mu + b_u + b_i$$

where b_u, b_i are user and item effect respectively. Now we regard them as a function of time, i.e:

$$b_{ui}(t) = \mu + b_u(t) + b_i(t)$$

The author made another assumption that user effect can fluctuate in very short period of time, and thus $b_u(t)$ is modeled on the daily basis. While the effects of items may not change frequently, we divided the time periods into different bins and $b_i(t)$ remained unchanged in each bin. So we further define

$$b_i(t) = b_i + b_{i,Bin(t)}$$
$$b_{u(t)} = b_u + \alpha_u * dev_u(t)$$

where $dev_u(t)$ measures how much current time stamp differs than the median time

stamp, i.e

$$dev_u(t) = sign(t - t_u)|t - t_u|^{\beta}$$

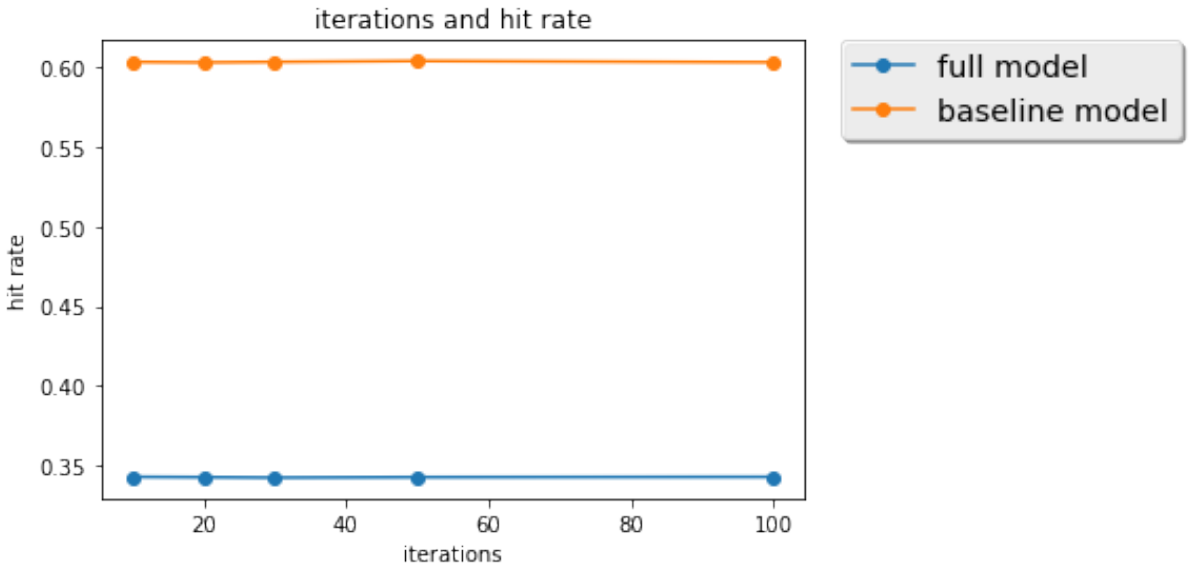
Now we are ready to define the full model, where user factors were added to the baseline model above.

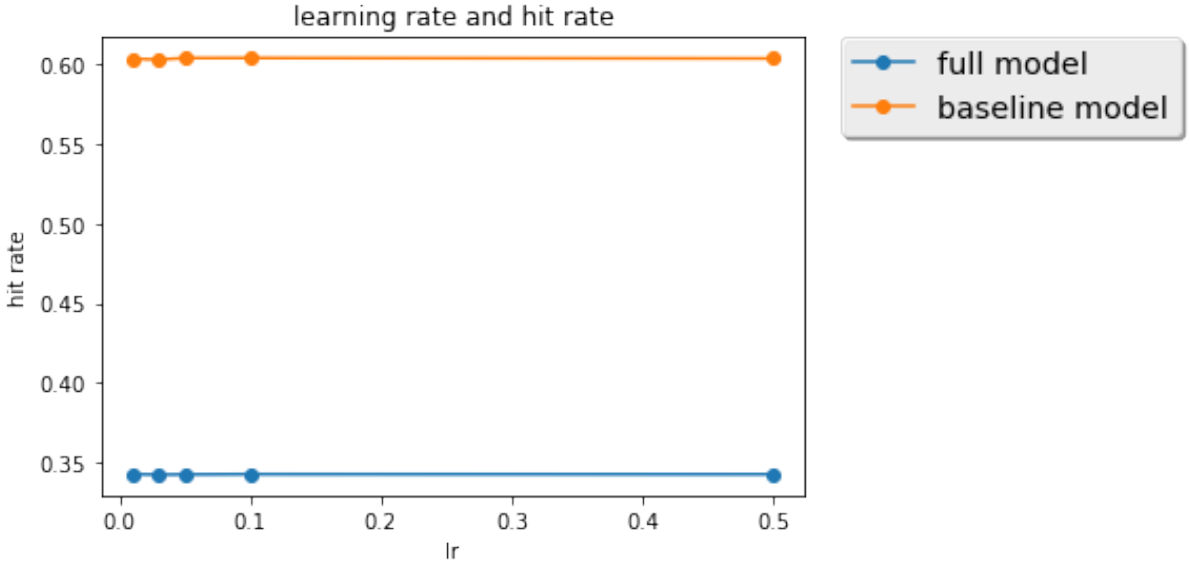
$$r_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^T p_u(t)$$

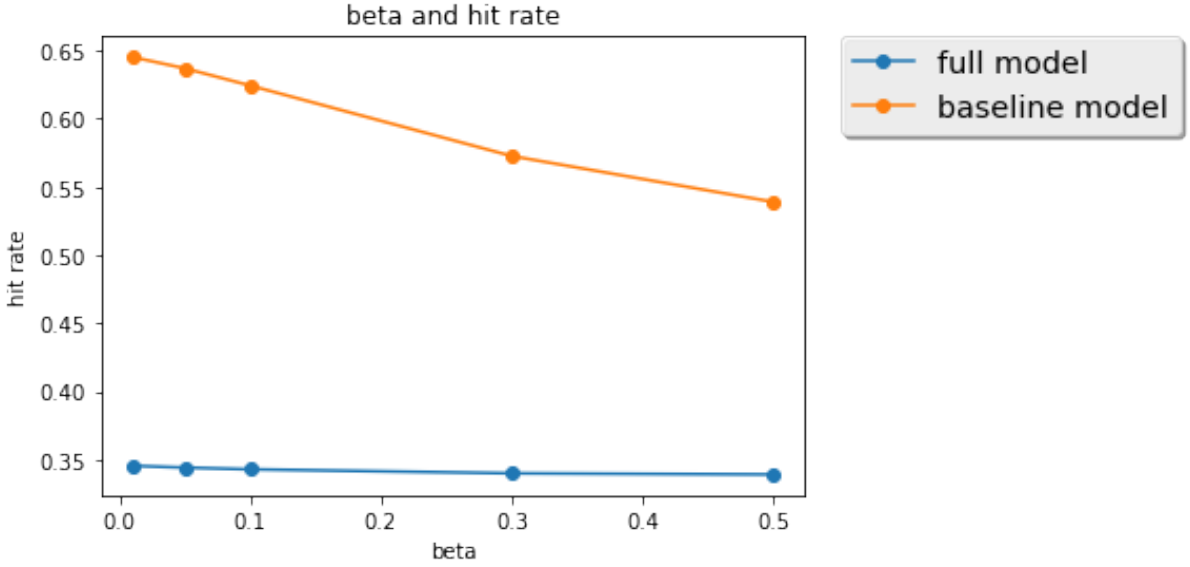
Since there are no available Time SVD++ library in Python, we wrote our own and used stochastic gradient descent method to estimate parameter. We tried cross validation on different numbers of latent factors, numbers of bins($nBins$), power in deviation(β), learning rate, and iterations in sgd. The plot against hit rate is shown below, and the hit rate is defined as proportions of items among 20 recommendations that were eventually purchased.

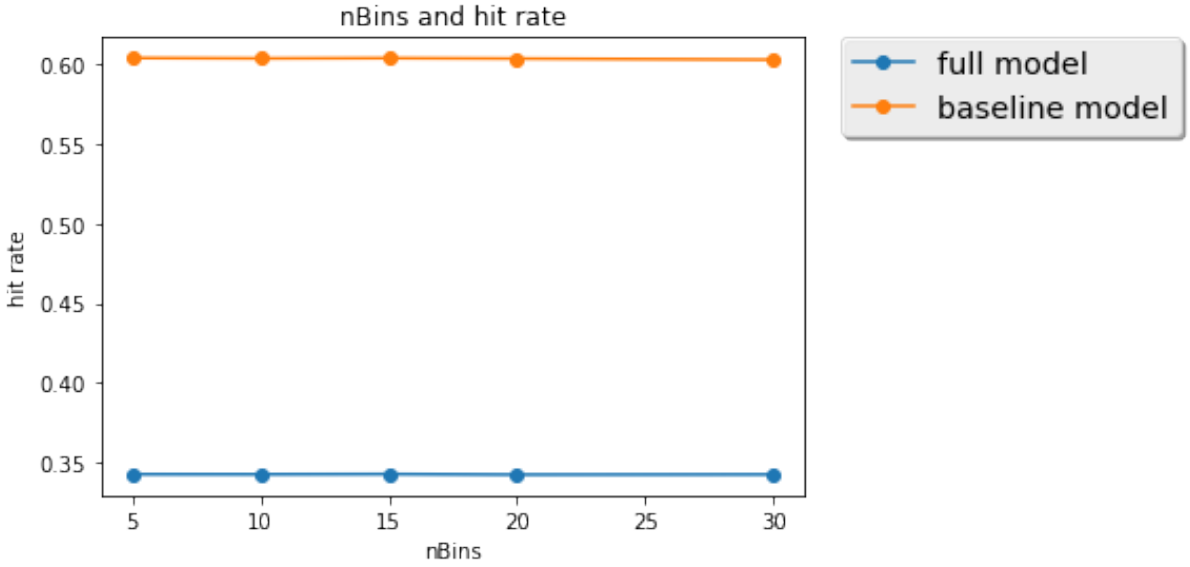
The baseline model only contains the user factor and item factors, without adding the interaction with them, i.e

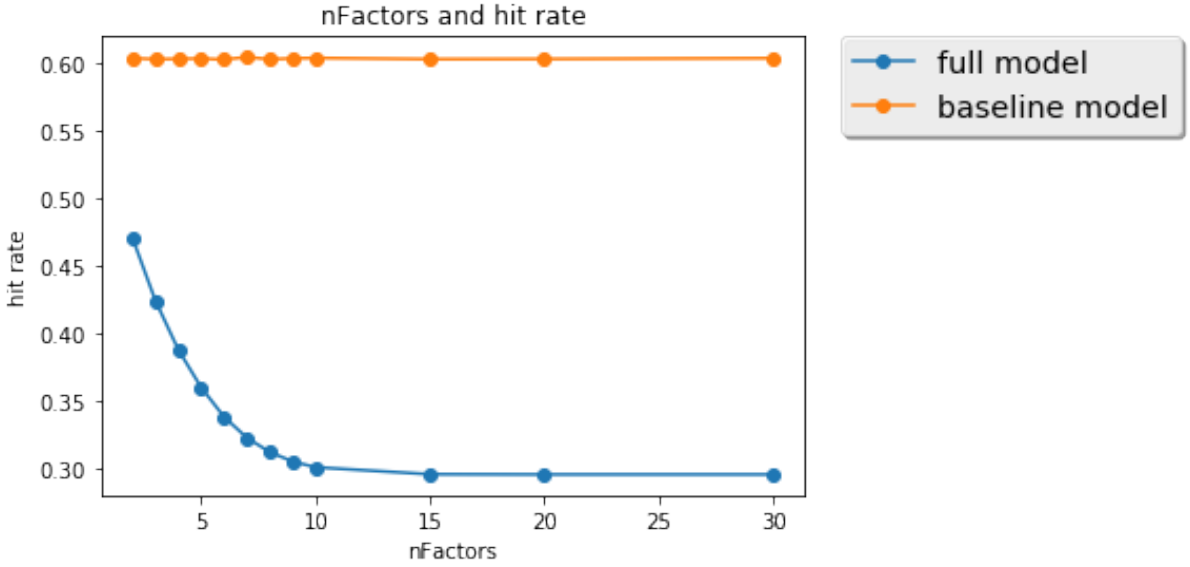
$$r_{ui}(t) = \mu + b_i(t) + b_u(t)$$




learning rate and hit rate
hit rate
lr
0.60
0.55
0.50
0.45
0.40
0.35
0.0
0.1
0.2
0.3
0.4
0.5
full model
baseline model


beta and hit rate
hit rate
beta
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.0
0.1
0.2
0.3
0.4
0.5
full model
baseline model



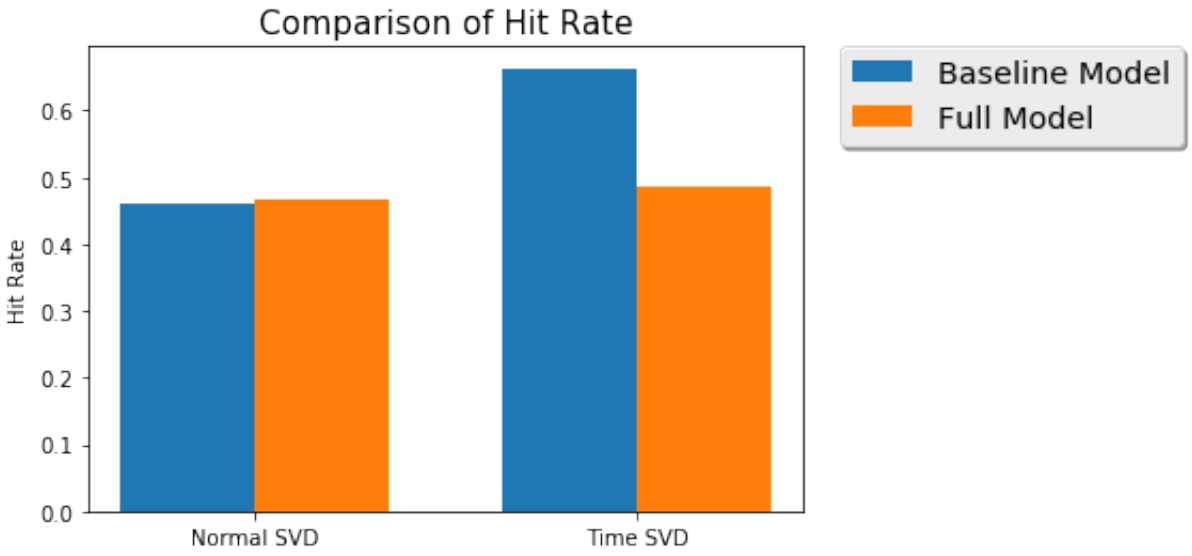


Notice that in splitting the training and testing sets, we should make sure that no records in the test set appear earlier than the train set, as we are trying to learn the behavior through time. It is clear that the general baseline model behaves better than the comprehensive model in the cross validation set. The number of latent factors is a critical parameter for the new comprehensive model. It is also interesting that the number of bins does not affect too much in both models, which is quite against the intuition as items shall yield changing popularities through time and different splits of time line should capture different patterns.

We made more investigation later.

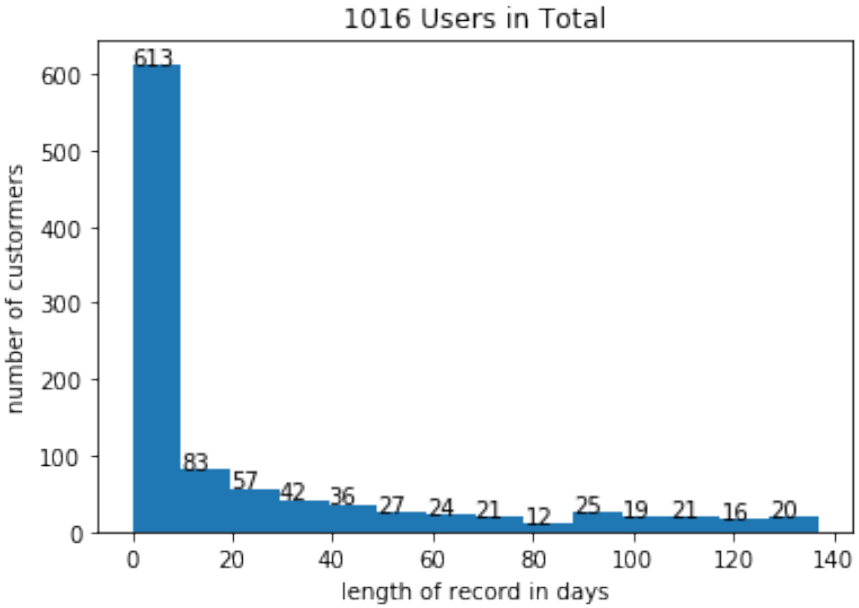
Result:

We compared the time SVD and conventional SVD predictions on the test set, with the best parameters obtained from validation set.



From the plot, it is obvious that the comprehensive model of Time SVD performs similarly as conventional SVD, and the baseline model is better than the rest. This supports the conclusion in the paper that time SVD does has its advantage when we have ample information on time stamps.

To better understand why the comprehensive model does not outperform the baseline model, we analyzed the length of records in days for each customer, the plot of which is shown below



Also, a big number of our customers only have activities within last ten days. This number is as high as 50% or even higher, which indicates that the time data on customers who have at least two purchases and two other activities are in fact very sparse. What's worse, up to 344 activities from these users were made in one day, which left very small room for modeling the behaviors through time. The paper used 200 bins to show best result.

The requirement on long period of time, and moderate size of data per customer to train the factor is one major limitation of time SVD. This observation explains why the comprehensive model does not outperform baseline model.

Many further improvements can be made if we have more data for each customer in a longer period of time. For example, we can amplify the effect by raising to exponential

$$b_{u(t)} = b_u + \frac{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|}}$$

Moreover, we can incorporate the change of behaviors through time cycle, and maybe regularization the decomposition can also be helpful.

User Behavior & Neighborhood Hybrid Model

Part 1 Model

Model

KNN Collaborative Filtering is a classic application for user-based item recommendation by leveraging top-K most similar users. However, limited by lacking the explicit rating data, we strengthened this method by adding users' transaction behavior history. Thus, for each user, we not only look for the top-K most similar users, but also took into consideration of their purchase and browsing history to make more tailored predictions to each user.

The process goes as follows:

We first constructed user-category matrix by transforming item preference to category preference. We combined "view" and "add to cart" as one behavior VA because for this model, they both indicate that the user would have a bigger purchasing potential. Since we don't have extra data to further distinguish their influence, we combined these two factors. For every user, we first grouped all the data points we have about the items in the same categories, and summed up the number of each of their behaviors, such as U_1C_1 = (20VA, 7P, 2M).

	Category1	Category2	Category3	Category4	Category5	Category6	Category7	…
User1	1VA 1P	1VA 1M	1VA 1M	2M	1M 1P	1P	1P	
User2	1M 1VA	1M 1VA	1M 1VA	1VA 1P	1P 1VA	1VA	1VA	
User3	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
Usern	…	…	…	…	…	…	…	…

Assign weights:

Since P (Purchasing) is a stronger buying indicator than VA (View and Add to cart), we assigned the weight of 1 point for P, 0.5 for VA, and 0 for M (Missing). We then sum up all preference values for this user to arrive at a preference total per category:

	Category1	Category2	Category3	Category4	Category5	Category6	Category7	…
User1	0.5+1	0.5	0.5	0	1	1	1	…

User2	0.5	0.5	0.5	1	1+0.5	0.5	0.5	…
User3	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
Usern	…	…	…	…	…	…	…	…

Finally, we acquired percentages of interest:

We used the preference value of a single category divided by the total preference to get a percentage, which stands for this user's interest about the items in this category among all the potential categories of this user.

The next step is to subset our data for training, cross validation, and testing purposes.

1) Preliminary sub-setting into 80% - 20%:

   After the user-category matrix is formed, we transformed this matrix into a dictionary containing four columns, with each column dedicated to: "user – category – preference percentage – random variable ". We introduced a uniformly distributed random variable over the interval between [0, 1] to randomly subset our data into 20% and 80% respectively for testing and training purpose.

2) Reinsurance:

   By randomizing the users, we might end up having a training set and testing set containing different users, which would cause trouble for us to compare predictions. To rule out this possibility, we added secondary filtering criteria to ensure that the user data we used for testing must have appeared in the pre-existing training data. In this way, we are guaranteed to have sufficient data to train, cross validate, and test our results. Our final training set contains 1,603,819 records, and 389,573 records for testing set.

Calculation:

We started our similarity calculation with simple Pearson Correlation Coefficient:

$$Pearson(u,v) = \frac{\sum_{k \in I_u \cap I_v}(r_{uk} - E(u)) * (r_{vk} - E(v))}{(\sum_{k \in I_u \cap I_v}(r_{uk} - E(u))^2)^{1/2} * (\sum_{k \in I_u \cap I_v}(r_{vk} - E(v))^2)^{1/2}}$$

However, the PCC result ended up being 1 as a result that part of our users only have one category in common with other users. This is misleading because "1" can also indicate that two users has exact same shopping preference. To fix this undesirable coincident, we introduced weighted PCC and weighted Jaccard.

$$Sim(u,v) = 0.5 * \frac{\sum_{k \in I_u \cap I_v} \left(r_{uk} - E(u)\right) * \left(r_{vk} - E(v)\right)}{\left(\sum_{k \in I_u \cap I_v} \left(r_{uk} - E(u)\right)^2\right)^{\frac{1}{2}} * \left(\sum_{k \in I_u \cap I_v} \left(r_{vk} - E(v)\right)^2\right)^{\frac{1}{2}}}$$

$$+0.5 * \frac{|S_i \cap S_j|}{|S_i \cup S_j|}$$

After calculating all the similarities between users, we started to predict our recommendations using:

$$\widehat{r_{ij}} = E(u) + \frac{\sum_{u \in P_u(j)} Sim(u,v) * \left(r_{vj} - E(v)\right)}{\sum_{v \in P_u(j)} |Sim(u,v)|}$$

The result was disappointing because the formula above incorporated the average of users' ratings, which is not applicable for our model since we used an interest percentage to reflect the users' preference, rather than a rating system. By getting rid of the rating average U, we were able to acquire a more reasonable calculation.

$$\widehat{r_{ij}} = \frac{\sum_{u \in P_u(j)} Sim(u,v) * \left(r_{vj}\right)}{\sum_{v \in P_u(j)} |Sim(u,v)|}$$

Last but not the least, we incorporate two metrics to measure this model: MSE, and hit rate.

$$MSE\ for\ an\ individual\ user\ u$$

$$= \frac{1}{|rated\ categories|} \sum_{rated\ categories} (\widehat{r_{ui}} - r_{ui})^2$$

$$Total\ MSE = \frac{\sum_{All\ users} MSE\ for\ an\ individual\ user\ u}{n}$$

$$Accuracy\ Rate\ for\ an\ individual\ user\ u =$$

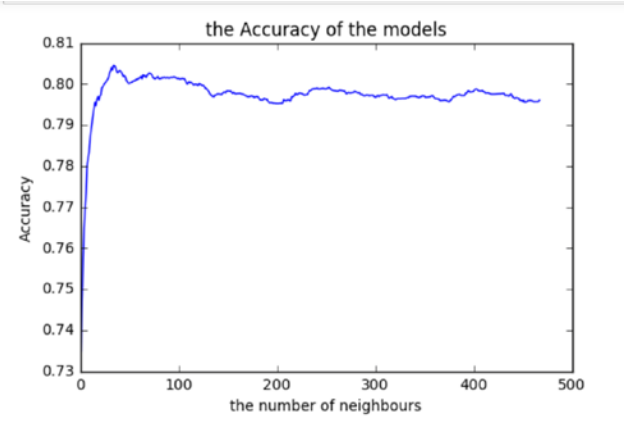
$$\frac{Number\ of\ the\ overlapping\ top\ \min(20, number\ of\ interested\ categories\ )\ recommended\ categories\ and\ top\ \min(20, number\ of\ interested\ categories\ )\ most\ interested\ categories}{\min(20, number\ of\ interested\ categories\ )}$$

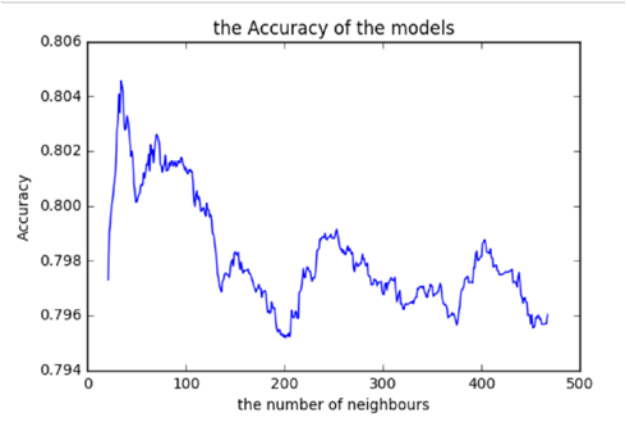
$$Total\ Accuracy\ Rate = \frac{\sum_{All\ users}\ Accuracy\ Rate\ for\ an\ individual\ user\ u}{n}$$

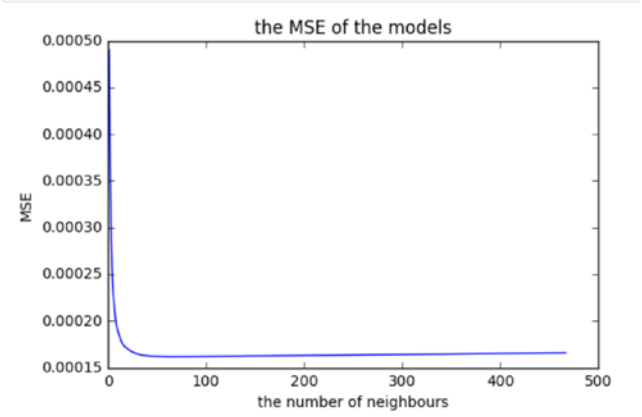
Results:

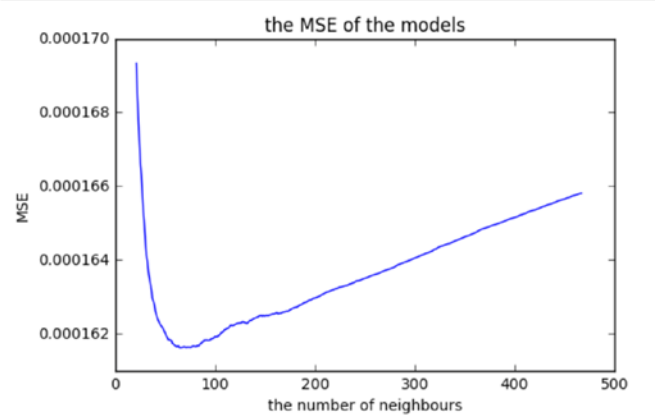
We eventually tuned parameters and found two sets of K values, 65 nearest neighbor would generate the smallest MSE and 34 nearest neighbor would maximize hit rate.

Following plots shows the results in cross validation, the left columns are plots for all values of k and the right column remove first several values because they perform really bad.

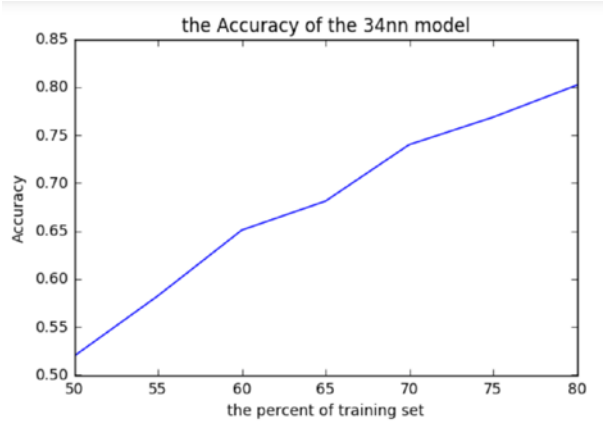


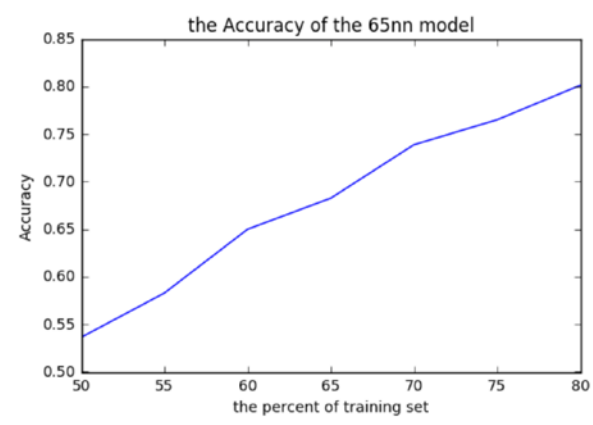


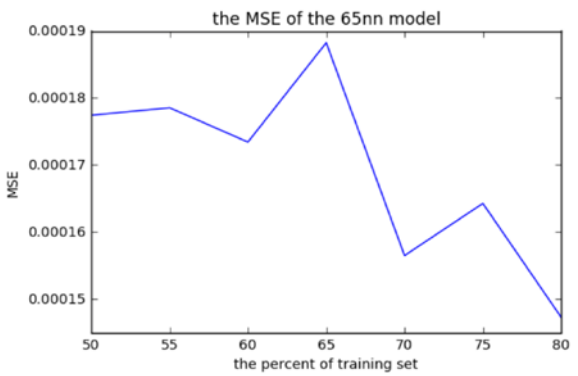




Based on the best two sets of parameters we have found, we tested the results by splitting training and testing data in different proportion, from 50% - 80%. The Accuracy below indicates the Hit Rate. The corresponding MSE and Hit Rate are plotted as follows:







In brief, the larger the number is of the training set, the better the Hit Rate, which is consistent with our intuition such that the more data we have about our neighbors, the more accurate the similarities would be. However, MSE's plots showed strange patterns as the number of neighbors in the training set increases to 34 and 65. It seems that moderate size of data does not strictly outperform smaller data, and the reason of this observation is not clear yet.

Part 2 Model

In the first part of our project, we designed a hybrid model that generates recommendation by weighing an individual user's past purchasing behaviors and behaviors of this user's top similar users. The weighted result determines which items to be recommended to this user eventually. To fix potential training and testing sets dependency and to achieve higher prediction accuracy, we modified several steps of this weighted model.

Model

1. Remove potential dependency between training and testing sets:

In the part I, we first calculated a "percentage of interest", a percentage score which represents an individual user's interest about the items in a certain category among all the categories. We then divided the testing and training sets into subsets. However, this could lead to dependency between our training and testing sets and cause a fundamental issue. As the items were merged into categories and the "interest percentage" scores calculated before testing and training subsets are made, our testing set could become subjected to inaccurate influence and then produce results with undesirable bias. Therefore, to remove this bias, we divided our training and testing datasets first and then calculated the user's "percentage of interest".

2. Further tuning parameters in search for better performance:

In part I, the weight parameters were initially set as 0.5/0.5 to reduce the calculations workload. 0.5/0.5 is also convenient as we believed that a user's past purchase behaviors and his/her similar users' behaviors play an equal weight in determining our recommendations. However, this may not be the case. In search for better performance, we trained 9 sets of parameters and eventually adjusted our weights to 0.9 for modified PPC and 0.1 for Jaccard, respectively, which produced the smallest MSE. After applying different PPC values and calculating all the scores for the training set, we combined them with the scores in testing set, which gave us a complete list of interest scores for a certain user regarding all the categories. We then picked the top 20 and recommended items from these categories.

3. In project II, we further segmented our entire user space to three categories and applied different recommending strategies:
    a) Type A: Users who have at least three purchase transactions and three views or add-to-cart activities, but has a purchase time scope within a year. For this group, we recommended items using the User Behavior & Neighborhood Hybrid Model described above.
    b) Type B: Users who have at least three purchase transactions and three views or add to cart activities, but has a purchase time scope over a year. For this group, we recommended items using the Time SVD model
    c) Type C: Users who have no purchasing activities but some browsing activities. For these users, we pushed the top popular items from the general popular item list.
    d) Type D: The rest of the users. For this group, we pushed the top popular items that are from the same categories as the items that this user browsed or previously purchased.

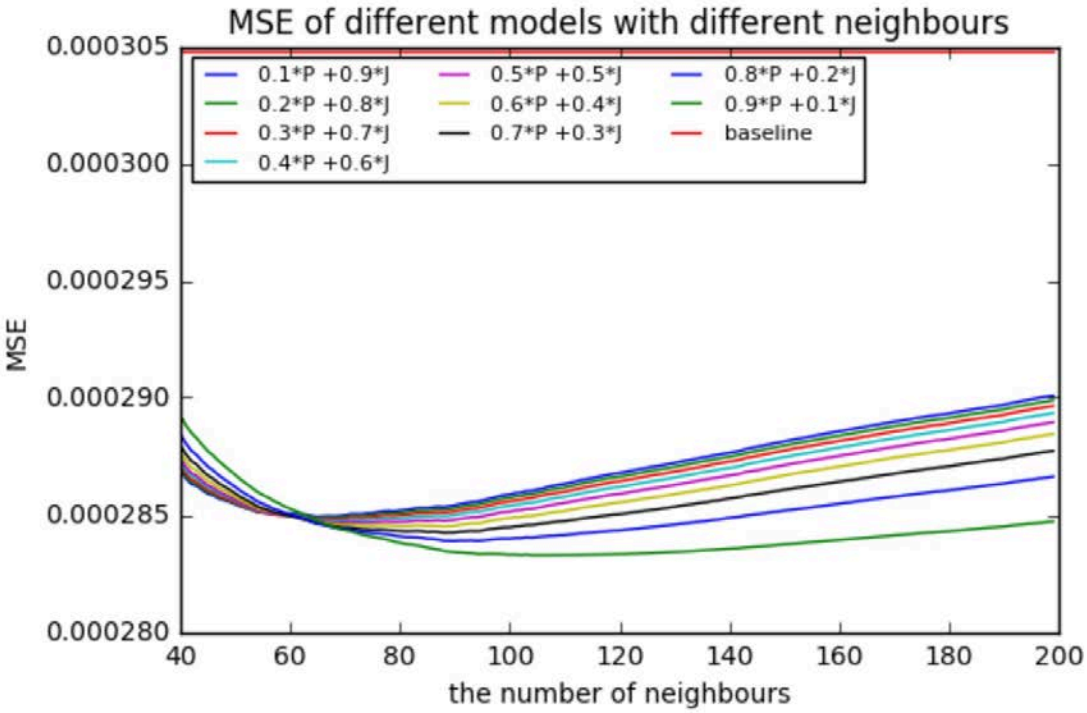
Results:

(notice: in part I, our model did not outperform the baseline model)

The baseline model is defined set forth below:

From the training set, for each category, we took the average of the computed user interest score. We assigned these average scores to the corresponding categories for all users in the testing set. For each user, we recommended the highest 20 user interest score corresponding categories. Essentially, we recommended the top 20 most popular items across all users.

- MSE:



We define our MSE as the following:

$$MSE\ for\ an\ individual\ user\ u = \frac{1}{|rated\ categories|} \sum_{rated\ categories} (\widehat{r_{ui}} - r_{ui})^2$$

$$Total\ MSE = \frac{\sum_{All\ users} MSE\ for\ an\ individual\ user\ u}{n}$$

(note: For the baseline, if for a category in the testing set, there is no interest score in the training set, we set the testing set score equals to 0)

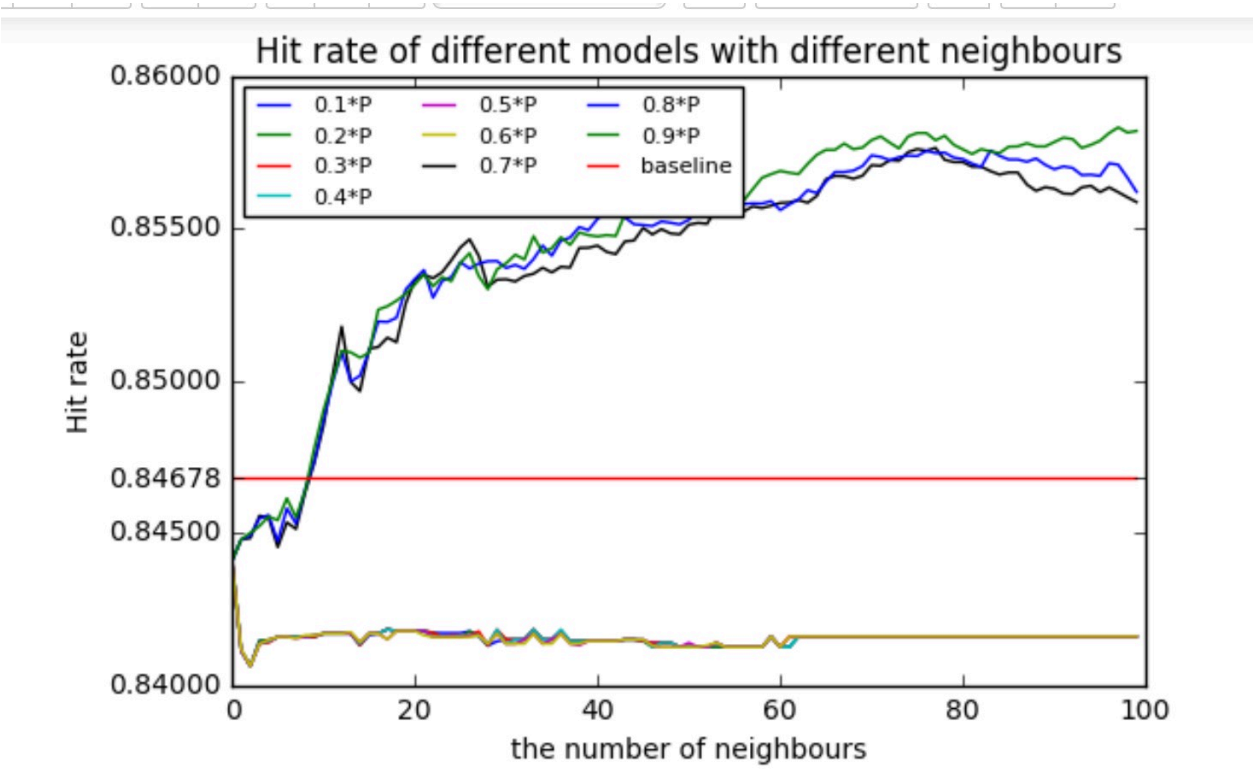
This MSE graph captured how accurate we were able to predict different users' interest for certain category and give a score. From the above graph, regardless of the weight, our model is clearly better than the baseline MSE, which equals to 0.000305. Specifically, we can see that when we use 0.9*Modified Pearson+ 0.1* Jaccard, with 109 nearest neighbors, we are able to get the smallest MSE = 0.000283.

Our model result is better because it works this way: Since our data is very sparse, in fact

that many of our users do not have many interested categories, therefore their interest rates for all other categories should equal to 0. However, the baseline model pushes the most popular items to the users based on all users' interest, which in most cases does not meet an individual's preference.

The MSE results are very valuable because it proves the diversity of our users. It helped us understand the interests of our users are wildly spread, therefore, a recommender system other than popular items and past purchased items is necessarily desirable.

- Hit Rate:



We define our Hit rate as the following:

$$Hit\ rate\ for\ an\ individual\ user\ u = \frac{Number\ of\ the\ overlapping\ top\ \min(20, number\ of\ interested\ categories)\ recommended\ categories\ and\ top\ \min(20, number\ of\ interested\ categories)\ most\ interested\ categories}{\min(20, number\ of\ interested\ categories)}$$

$$Total\ Accuracy\ Rate = \frac{\sum_{All\ users} Accuracy\ Rate\ for\ an\ individual\ user\ u}{n}$$

The result of our Hit Rate graph is very interesting. It is quite obvious that there is a clear diversion as the modified PPC weight changes.
When PPC ranges from 0.1 to 0.6, our hit rate stabilized around 0.841 after the number of neighbors reached 1. This means that with PPC in this range, no matter how many more similar users we introduce, it won't make any further contribution to improve our Hit Rate. This makes perfect sense because when our user data is sparse, introducing more similar users won't help increasing the interest score on a certain category between two users.

When PCC takes on a higher weight – such as 0.7, 0.8, 0.9, our hit rate monotonically increases and converges when the number of similar users reaches 80. This means a couple of things. First, we initially introduced Jaccard to fix the potential issues that the modified PPC may overlook. When our PPC takes on a higher weight, Jaccard was indeed able to fix the problem that when PPC is 1, it misrepresents how similar two users' interests are.

We can also see that PPC is a good metric to measure user similarities. From the graph above, hit rate increases as long as we introduced one similar user despite the minor fluctuation. This is consistent with what we found in our exploratory data analysis that when most of our users have limited numbers of purchases, they typically don't have strong past shopping preferences. Therefore, positive influence occurred when we introduced the purchase interest of their similar users.

One detail that we need to address is that there are minor fluctuations in our results. These small inconsistencies reflect that some of our users do have strong purchasing preference. Therefore, recommending based on their similar users is not the best strategy for this type of users. Therefore, we segregated this group of users (type B users) and applied a different model (Time SVD) for them.

PART III : Real World Application

As Product Managers and Data Scientists at RetailRockets, our primary goal is to design recommendation systems that could help our customers – E-Commerce firms to increase sales revenue. Our models can be integrated with a wide range of applications including but not limited to websites, email campaigns, online communities/blogs.

In the final stage, we integrated our two models as one comprehensive business solution.
Instead of merging them as one, we decided to apply different models to users at different maturity level. There are three main reasons for this decision:

1. Our two models are set up fundamentally differently from the designing stage. From the very beginning, the user-item model didn't incorporate the concept of time at all. It focused on the user's shopping behaviors and the behaviors of the similar users, while the latent factor model was built with a further interest to leverage data over time. In this way, the user-item model is naturally more useful to produce short-term and immediate recommendation, while the latent factor model with time stamp is ideal for long term and future targeting recommendation.

2. Further, as users move from one stage to another, for example, from mere browsers to new first-time buyer, from inactive buyers to regular shoppers, from regular shoppers to loyal customers, our recommendation strategy should capture this change and evolve as the users' maturity change. As our customers grow older, their interest and needs would change too. Therefore, applying different recommendations at different stages would be a more appropriate adaptation.

3. Finally, to increase the overall sales for E-commerce businesses cannot solely rely on the website recommendations. Developing consumer interests also plays an indispensable role in increasing customer activeness, stickiness and loyalty. Through carefully curated content recommendations in the format of subscription service, emails, and blogs, companies can build trust and likability amongst consumers. Hence, by applying latent factor model with time stamps to these types of recommendations, we are able to cultivate and develop

customer interests, uncovering more sales opportunities. Once a customer goes to the website, the user-item model on the website would give the customer the immediate and most relevant recommendation. We are able to maximize the opportunity to unleash our customers' purchasing potential.

Below is specifically how we decided to implement:

User type	**Purchase**	**View/add-to-cart**	**Time Stamp**	**Model**	**Recommendation format**
A	3 and more	3 and more	Within a year	User Behavior & Neighborhood Hybrid Model	Website /mobile app
B	3 and more	3 and more	over a year	Latent Factor with time SVD	Email subscription, blog, online community
C	1-3	some	n/a	Popular items within the same categories as the items they showed interests	Website, Email subscription, blog, online community
D	0	No activities	n/a	Top popular items across all categories	Website, Email subscription, blog, online community

type A) Users who have at least three purchase transactions and three views or add-to-cart activities, but has a purchase time scope within a year

type B) Users who have at least three purchase transactions and three views or add to cart activities, but has a purchase time scope over a year

type C) Users who do not have any purchase transactions

type D) The rest of the users

For type A users, our model outperformed baseline model and were able to recommend items of high-related interests.

For type B users, the test results proved our hypothesis that time SVD would perform better than SVD, which indicated that users' purchasing behaviors and interests in different categories would indeed change through the time.

For type C and type D users, since we don't have any access to the future data to verify the recommendation quality, we are yet to provide prediction accuracy.

Even though there is yet any ways to accurately test the recommendation quality for Type B, C and D users due to the fact that we only have data within a year, we believe a good recommendation system should be a comprehensive solution that can cater to different needs of users at different maturity.

By applying the proper model for the right consumer groups through the right channel, we can create a holistic and convenient one-stop shopping experience for the end consumers. As a result, this will maximize the sales revenues for our customers, making our product more valuable.

From a cost saving perspective, our customers don't have to invest heavily in engineering talents, development time, and potential opportunity cost. The more customer success cases we can build to establish our reputation, attract more potential customers, and expand our market share.

By providing a valuable core product, we can save our marketing budget by winning customer referrals. An expansion in market share and customer quantity will also contribute to our company evaluation, attract more funding, and stimulate our growth.

PART IV: Future work

Limitations:

While we are confident about our model and prediction results, we did encounter some serious limitation along the process.

Initially, we don't have any information about our items, or categories. It makes it hard for us to consider the intrinsic differences between categories. For example, cars sales and grocery sales can yield to totally different recommendation strategies. If this information is provided by RetaiRocket, we could have better tailored our models for specific industries.

In addition, all of our data points about users' feedback are implicit, rather than direct ratings. When transforming implicative information into real-values, we mostly based on our experience or intuition, which is not the systematic, nor scientific without mathematical support from real data.

Lastly, sparse data set remains one of the most common challenges. It requires most careful consideration when deciding on a strategy to fill up the missing data. Especially for E-Commerce, the more data we have for an individual user, the more we can understand this user's interest therefore predict his next shopping behavior.

Future work:

One interesting hypothesis is that consumers may have different shopping styles: some always stay with their usual choices, while others are willing to explore new things. Knowing this difference, we could develop different recommendation strategies to for the diverged groups.

Last but last the least, our recommendation system design could also line up with our customer companies' business priority. We could incorporate inventory and price information to maximize profit for our customer companies.

Summary

From the results of both of our models, we feel confident to put our package in a real company as a product. Overall, compared with baseline model, our product has increased prediction accuracy with an acceptable runtime and scalability. As we accumulate more data and knowledge about our end consumers, together with updated categorical information, different industry backgrounds, further tuning with our parameters, we would be able to modify our model and enhance our performance further, and create more value for our customers.