Authors: Rasul Dent Justine Diliberto Anna Nikiforovskaja Cindy Pereira

Intelligent systems and recommendations project

M2 NLP

Université de Lorraine, IDMC

1 Data exploration and analysis

This first part presents the primary steps of the development of a recommender system, that are data exploration and analyses.

1.1 Presenting dataset

The dataset studied is a part of the Diginetica corpus. It contains 6 dataframes, dealing with e-commerce session-based data. The aim is to implement a recommender system to provide an optimized product ranking. Users have been anonymized, sessions are bounded by periods of inactivity, and queries without clicks are not represented. Furthermore, "a single user might have multiple sessions, and one session can have multiple users (if user re-logins into another account)".

1.2 Primary data attributes

- item id: integer representing a distinct product
- category id: integer representing a set of distinct products
- product name tokens: hash-codes corresponding to words
- query id: serial label for query
- sessionId: serial label for unique session
- time-frame: milliseconds since beginning of session
- priceLog2: price after log transformation
- event dates: what day the click/view/purchase occurred

1.3 Data files and questions they can answer

- products: Which products and hashed keywords correspond with different price brackets?
- product categories: Which items belong to the same categories?
- train purchases: Which items were bought together in the same session?
- train items views: How far into a given session did a user view an item?
- train/test queries: All of the above -> which products should be recommended in a session?

Number of items	184,047
Number of users	204,789
Number of clicks	1,127,764
Number of views	1,235,380
Number of purchases	18,025
Number of categories	1,217

Table 1: Data counts

1.4 Statistics

We have calculated some statistics on the dataset to understand the possible issues while building a recommendation system.

The first thing we can notice just by seeing numbers of items, users and purchases is that our data is extremely sparse. This is a common problem while building a recommendation system, however we will have to choose the recommendation methods with that in mind.

Afterwards we have decided to look at the distribution of ratios between views and purchases. In fig. 1 we can see, that in general the ration is quite small, the most popular ratios are between 0 and 0.1 and there are approximately 10^5 such cases. However, sometimes it is still possible to have for one item the number of purchases to be higher than the number of views. We think it might be because people sometimes need specific items more than once. This can be true for nails, light bulbs or even car tires, which are never bought as one instance.

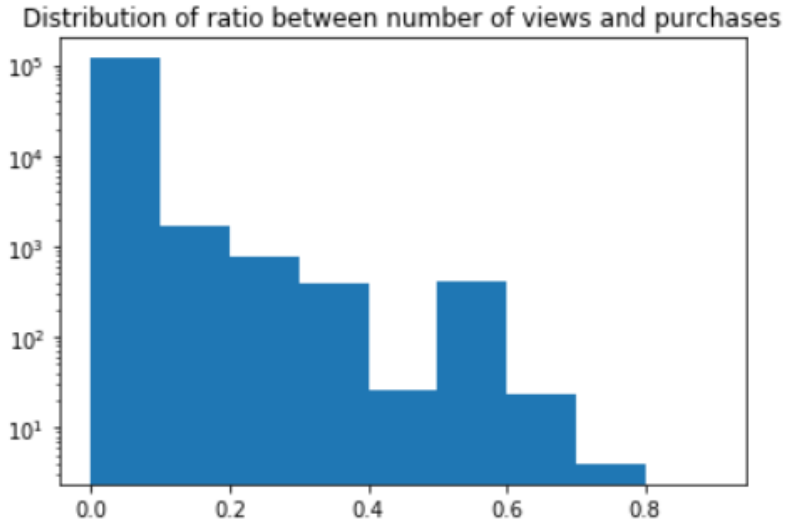

Figure 1: Distribution of ratios between number of views and purchases, on axis x we see the ratio, on axis y the number of items with this ratio.

We also study a distribution of ratio between the number of clicks and number of views, which is shown in fig. 2. An interesting part here is that there is now not a huge difference between different ratios in terms of how many items have the same ratio. And there are also some products which are viewed more than clicked. This might be a sign a user came to the product from outside of the site, as the clicks from outside of the site are not represented in our data. The ratio of products more viewed than clicked is of approximately 0.3069.

In addition to this, the distribution of the number of items per category has been computed and can be viewed in fig. 3. We can clearly notice a regular decrease in the quantity of categories with bigger number of items. This means the categories are not evenly distributed. In other words, some categories are most common than others, and there are many categories that have very few items.

Finally, we studied the price distribution. In fig.4 we can see that the items cost either less than 20¤, or between 45¤ and 100¤. This can be explained by the fact that people often buy cheap products (less than 20¤), or they will choose quality, which will significantly increase the price. A few products will have a price in between, but they are exceptions.

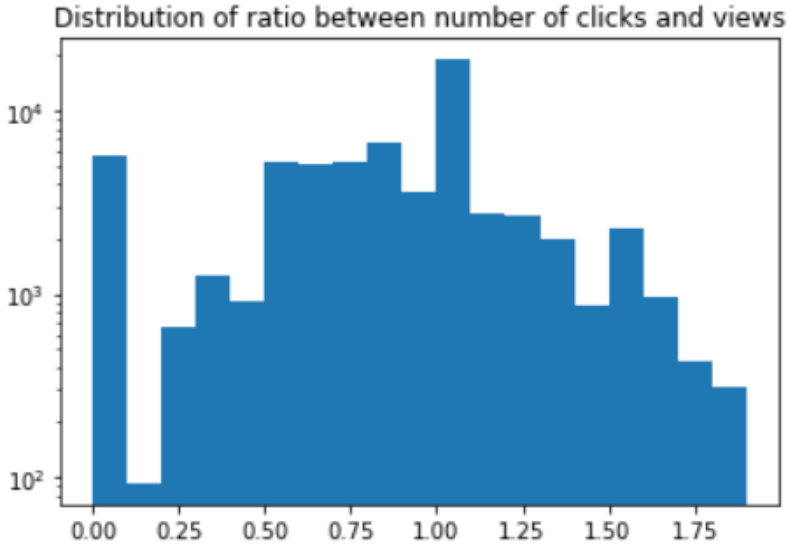

Figure 2: Distribution of ratios between number of clicks and views, on axis x we see the ratio, on axis y the number of items with this ratio.

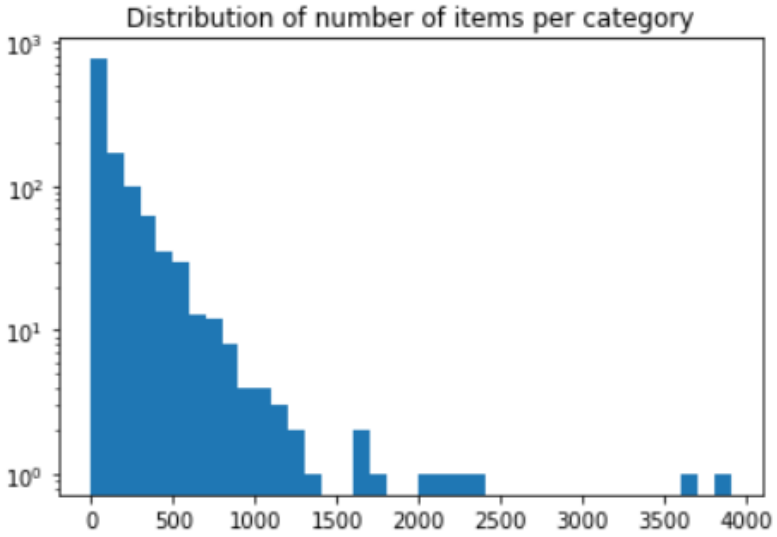

Figure 3: Distribution of number of items per category. On axis x we can see the possible numbers of items in categories, on axis y the amount of categories with this number of items.

1.5 Conclusions

We have a variety of data relating users, products, time, and price. Although we are able to note multiple correlations, several questions remain regarding the distribution of prices and how to account for purchases involving large quantities. As usual, we will have to reduce the dimensionality and redundancy of the dataset.

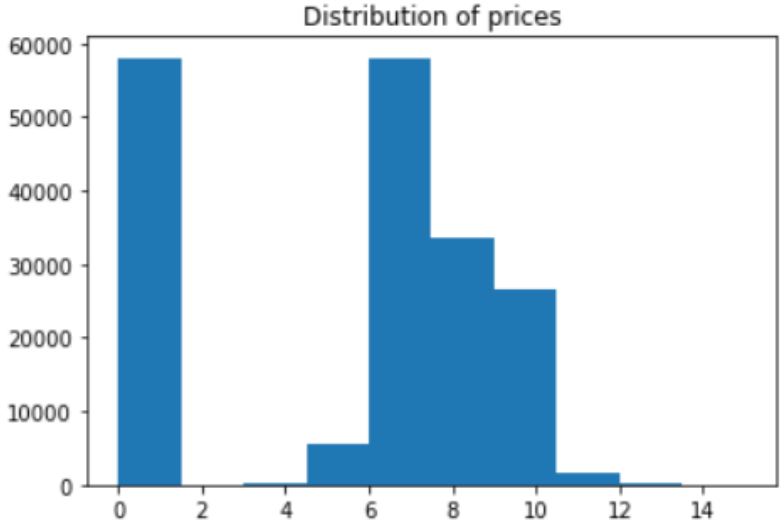

Figure 4: Distribution of the prices. On axis x we can see the log2 price, and on axis y the number of items involved.

2 Proposed Recommender Systems

2.1 Research and ideas

When building a recommendation system for a site which sells different types of products, the main goal is to generate more purchases thanks to this system. Therefore we have decided to concentrate both our methods on predicting the list of products the user most possibly will buy.

We have decided to look at the recommendation problem from two different points of view. We either want our system to be a little bit invasive and predict what the user might want to buy in addition to what they already have in a shopping cart. Or, we implement also another more classical way of recommendation. This would propose to each user possible products to buy based on the user's activity, in a separate page.

In the following sections we describe our ideas in more details.

2.2 Recommender system 1: time-weighted feedback model

In line with our stated goals, the first proposed model will be largely user-based. Since we have both absolute (dates) and relative (session lengths, session IDs) temporal information at our disposal, a time-weighted feedback model seems most appropriate. In particular, we are considering the algorithm described by Ding and Li [1], which is based on K-means clustering. However, given that our feedback is primarily implicit, we will have to define a way to infer ratings from information such as clicks and views. We also intend to integrate cost and category into our notion of item similarity. In line with the requirements of this project, we will use MRR as the evaluation metric for this model.

2.3 Recommender system 2: graph-based

Another system that we propose is implemented using a graph-based approach. Indeed, it is possible to link several items that were bought together in one order. A useful reference for this task is the review by Wang et al. [2].

Our recommender could suggest products when a user has already added some items to their order. We have found out that a lot of products are grouped into orders, as people tend to buy several needed products which are somehow connected to one another. The distribution of number of products in each order can be seen in fig. 5.

Based on that, we could build a bipartite graph with one set of vertices representing orders and the other representing products (items), as we have said earlier that items from the same orders are linked.

Afterwards we propose to apply the algorithm introduced by Zhang et al. [3], but using orders instead of users. However, their algorithm uses ratings, which we could introduce as following:

- If the item is actually in the order, we give it a maximum rating (for example, 5)

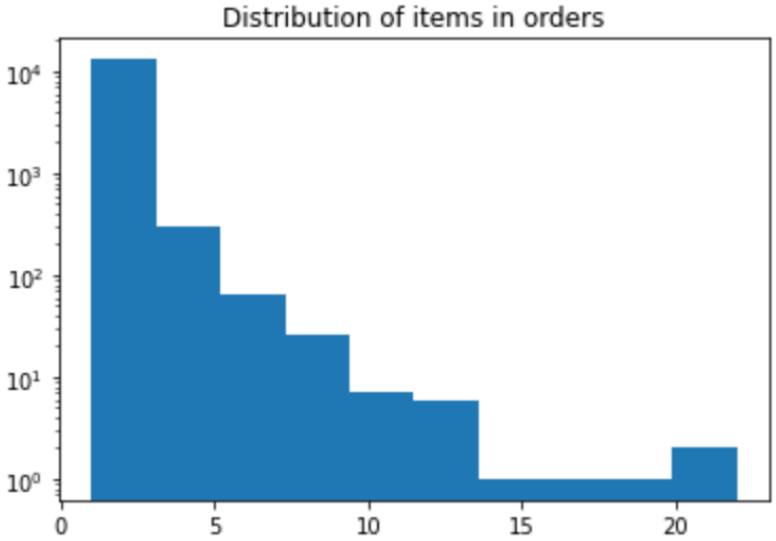

Figure 5: Distribution of the items over orders. Axis x : the number of items in the order. Axis y : the number of orders with this number of items.

- If the item was viewed in the same sessions as the sessions in the order but wasn't bought, we give it a medium rating (for example, 3)
- If the item appeared in queries during the same sessions, but was not viewed, we give it a minimum rating (for example, 1)

The code is published and thus usable, along with their evaluation as a preprocessing step. And we can also use MRR by ranking items in test orders, but the ideas are quite different, thus we won't be able to compare the algorithms by metric. However, we will be able to compare the computational time of the algorithms and notice how they behave.

3 Recommender Systems Implementation

This last part introduces two recommender systems that have been implemented.

3.1 Recommender system 1: time-weighted feedback model

The first model aims at producing recommendation based on previous user activity and time. This model will only affect connected users, as information on specific users is needed to compute a list of recommended items.

3.1.1 Variables

The variables used for the model are listed below, as well as their meaning. The data represented by these variables was gathered from the corpus, in preprocessing steps.

- C: user-item matrix
  - Rating user-item (did they buy / view / click)
- T: user-item matrix
  - Time when they bought / viewed / clicked the item
- n: number of closest items to a specific item
- K: number of clusters
- l: number of items to recommend

The matrices evoked above have been split between train and test sets such that in every set, all users and all items appear.

3.1.2 Functions

The different functions that enabled the creation of the model are presented below.

- ComputeItemSimilarity(C,n): compute n closest items for each item
  - output: M = {itemId: [n closest itemIds]}
- Predicted opinion of user i on item j
  - $F(T_0) = \frac{1}{2}f(0)$
  - $\lambda = \frac{1}{T_0}$
  - $f(t) = e^{-\lambda t}$
  - $O_{ij} = \frac{\sum_{c=1}^{k} O_{ic} * sim(I_j, I_c) * f(t_{ic})}{\sum_{c=1}^{k} sim(I_j, I_c) * f(t_{ic})}$
- LearningParameters(C,M,T,n,K,l): search for T_0 for each user such that:
  - $min[\frac{\sum_{i=1}^{N} |p_i - q_i|}{N}]$, where:
  - p_i = prediction (result of O_{ij})
  - q_i = actual rating
  - output: P = {user: T_0}
- PredictRecommendations(M,T,P,l): apply O_{ij} for the right T_0
  - output: l best results for each user

3.1.3 Data transformation

As detailed above, it was necessary to first transform the raw data into one matrix of users and item ratings, and another matrix of users and times for the ratings. To generate these matrices, we relied on 4 files : "train-item-views", "train-purchases", "train-queries", and "train-clicks". Since there were no explicit ratings, we decided to treat purchases, views, clicks, and nothing as ratings of 3,2,1, and 0, respectively.

For views and purchases, we were able to directly convert the dates provided into elapsed time, while for clicks we had to first match them to users and dates by using the "queryId" and "train-queries". After filtering out the data not associated with users, we originally created a sparse matrix of tens of thousands of users and many tens of thousands of items. As the initial matrices were 10.5Gb each, it became apparent that we would have to significantly reduce the dimensions of the matrices.

To make the problem more tractable, we resolved to focus on only the most active users and their most popular items. In concrete terms, this meant first calculating the total ratings of each user, and then picking the 1000 users with the highest total ratings. Then, for the items, we calculated interaction scores (sums of all ratings) given by this subset of users, and picked the 3000 items with the highest level of interaction. After identifying the most popular users and items, we created new matrices of size 1000 x 3000 which were then used as the bases of the recommendation system. Finally, we stored the relationships between user/item IDs and positions in the matrices as dictionaries to assist in identifying similar user and item profiles at the moment of inference.

3.1.4 Implementation

We started to implement the functions explained earlier to compute the n closest articles for a specific item.

To do so, we iterate through all the items in our matrix C (see section 3.1.1) and compute the similarity between the observed item and all the other ones. We tested multiple functions to compute this similarity: Cosine similarity, Pearson correlation coefficient, and conditional probability-based similarity.

With these results, we have created a dictionary in which keys are item IDs and the corresponding values are the n closest items (i.e. the n items with the highest similarity). This dictionary will be referenced as M for the next steps.

Using ComputeItemSimilarity, we are able to compute the predicted opinion of user i on item j. We tried two versions of this function:

- Without time: the real rating of user i on an item c is extracted, and multiplied by the similarity between item j and item c. The final predicted opinion of user i on item j is computed by adding the predicted opinions for all possible items c, and dividing it by the sum of the similarities.
- With time: the prediction taking time into account works the same way as the other version. However, we multiply the numerator and the divisor by a function f on the actual time when a user i acted on an item c. This function corresponds to the exponential of $-\lambda * t$. This lambda is the inverse of T0.

Then, we had to implement the function to learn the parameters. This function consists in finding, for each users, the best T0 for which the Mean Absolute Error (MAE) is minimum.

For each users, we test multiple T0 and compute the predicted opinion for each possible T0 and each article. We then compare these predictions will the real values and compute the MAE. The T0 with the lowest MAE is kept as the best T0.

Our final function concatenates these three parts to output a dictionary in which keys are users, and values are lists of highest predicted items for the user.

3.1.5 MRR metric

The Mean Reciprocal Rank is a metric to evaluate recommendation systems. It is based on the question "Where is the first relevant recommendation in a list of predictions?". To use such an evaluation metric, the system has to generate a list of recommended items for a specific user. As we output a dictionary with such lists for each users, this metric is appropriate for our model.

Mathematically, the MRR is the average of the reciprocal ranks of results for users. For example, given a list of recommended items, if the first relevant item is in position 2, its reciprocal rank will be 1/2. The closer to 1 is the MRR, the more performant is the system.

In our specific case, an item is considered relevant if the rating of the user on this item is of 3. Here, this maximum rating is attributed when the considered item is bought.

3.1.6 Results

We first tried to evaluate our model taking time into account. However, the computational time of the system was too high, which made it impossible for us to fully run it.

Thus, we evaluated our model ignoring the time parameter and with 2 predictions outputted for each user. The Mean Reciprocal Rank obtained was of 0.75. As we said before, the closer to 1 it is, the better the model is. We can thus state that our model still performs quite well.

We decided to evaluate the same model, but with a higher number of generated predictions per user. For the case of 4 predictions per user, the MRR is of 0.59. Interestingly, predicting more items does not make the model more accurate.

3.2 Recommender system 2: graph-based

This model is an adaptation of the model introduced by Zhang et al. [3] to the data we have, containing orders and items. The main idea of the system is to transform information about sessions preceding the given order into a weighted bipartite graph and then to apply Inductive Graph-based matrix completion model. By doing so, the model will be able to reconstruct order-items relationship. Therefore, the model will predict a list of suitable additional items for a given order.

3.2.1 Data Transformation

We build the graph data in the following way. For this model, we examine each orderID and assign all the bought items the highest rating, which is of 5 (using train-purchases.csv). Afterwards, we get all the session IDs from those purchases and find all the queries performed by the user during these sessions, using table train-queries.csv.

Then for each query:

- We first find all the items in the results of the query which were clicked by a user. (train-clicks.csv)
- We choose top ten items from each query (as it's obvious the user has low chances of looking further in the query)

- We then choose randomly 10 items which were clicked and assign them rating 3.
- Finally we select randomly 10 items among not clicked top-shown items and assign them rating 1.

The random selection in the steps above is needed to decrease the amount of low ratings and therefore to balance data. In addition, this makes it easier for the algorithm to learn the underlying connection between low-rating items and high-rating items.

The resulting bipartite graph has 13506 orders, 31609 items and 191317 edges between them. Among those edges, 17761 of them have the highest score, the ratio is 0.09.

3.2.2 Training and Evaluation Parameters

The model was trained during 10 epochs. Checkpoints at epoch 5 and 10 are used to predict ratings in ensemble for the final testing. The model is evaluated using RMSE – Root-Mean-Square Error, which is defined as follows:

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(y_t - \hat{y}_t)^2}{T}},$$

where y_t – actual ratings, while $\hat{y}_t$ – predicted ratings.

3.2.3 Evaluation Results

The final RMSE value on the test dataset is 1.499327. This is a pretty good result, considering that the distance between ratings which are next to each other is of 2. A more detailed example of how the model works is displayed in fig. 6.

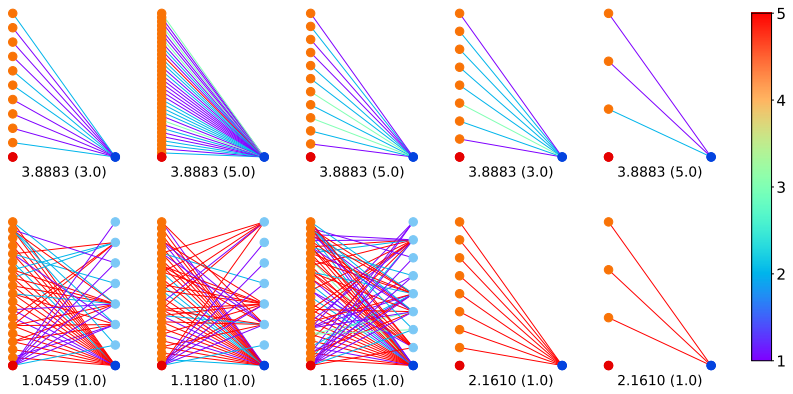

Figure 6: Top 5 and bottom 5 are subgraphs with the highest and lowest predicted ratings, respectively. In each subgraph, red nodes in the left are orders, blue nodes in the right are items; the bottom red and blue nodes are the target order and item; the predicted rating and true rating (in parenthesis) are shown under each subgraph. The edge ratings are visualized with the colormap on the right.

The results are pretty satisfying, and the model allows not to use users information while also processing thousands of items and orders. We believe this method is promising. In the original paper, the authors also add the possibility to take items features in consideration, which could be a nice thing to add.

4 Conclusion

Based on available data, we have managed to build two very different recommender systems. The first one is user-based and may be used to predict items that a specific registered user will like. The second one is order-based and can be used to recommend extra items to the shopping cart the user has created.

The first system turned out to be extremely heavy in terms of training and was unable to process large amounts of data. In contrast, the second one has managed to compute the predictions for a much larger amount of orders and items. Unfortunately, we applied the models in such different ways that we were unable to properly compare the quality of their predictions.

However, numerous modifications could still be done to improve both our models. For example, we unfortunately did not manage to run the first system while taking timestamps into consideration. It turned out that considering the time parameter increases the computational time a lot.

Finally, we could add some item features to the second recommender system and compare the results between the graph-based models with and without features.

References

[1] Y. Ding and X. Li, "Time weight collaborative filtering," in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, ser. CIKM '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 485–492. [Online]. Available: https://doi.org/10.1145/1099554.1099689

[2] S. Wang, L. Hu, Y. Wang, X. He, Q. Z. Sheng, M. A. Orgun, L. Cao, F. Ricci, and P. S. Yu, "Graph learning based recommender systems: A review," 2021.

[3] M. Zhang and Y. Chen, "Inductive matrix completion based on graph neural networks," in *International Conference on Learning Representations*, 2019.