Department of Computer Science and Information Systems

Birkbeck, University of London

2020

MSc Data Science

Project Proposal

eCommerce Recommender System

For Increased Basket Value

Konstantin Orlovskiy, Student ID: 13157188

Presentation of the problem

Introduction

Before eCommerce, the traditional brick and mortar stores were growing the sales by displaying the bestselling products in the most visible places because the bestsellers were more likely to be purchased in higher volumes but niche items were not. While this approach worked well for business to achieve increased sales numbers, there was no customer personalisation available due to limitations of physical space in store space to do personalised placements, and lack of customer data available.

eCommerce significantly extended the seller's capabilities to provide the personalised customer experience on their websites. At this point the recommendation systems started developing [2]. In order to make the process scalable the online retailers needed the computer to make recommendations based on the historical customer behaviour and provided preferences / information.

Between 2016 and 2018 years 90% of the data in the world was generated to date [9]. This is the source of vast amount of information for machine learning application and in particular providing personalisations, which is motivating development of recommender systems. The available computational power also grows exponentially which makes the process doable.

Recommender systems overview

The importance of the recommendation engines is growing in online retail nowadays: 35% of Amazon's revenue and 75% of Netflix's revenue is generated by recommender systems [8]. Personalisation is a key to customer value growth and retention rate decrease led by the improved shopping experience.

User may provide explicit and implicit feedback to the items. Explicit feedback is when the user rates the item giving a clear understanding on the level of interest, for instance, star reviews on Amazon. Implicit feedback is the set of user-item interactions such as view, add to wish list, add to cart, purchase, etc., which do not comprehensively describe how the item fits the user needs or interests. In practice, the recommender systems mostly deal with the implicit feedback which is collected by tracking user behaviour on website [4].

There are several approaches of using the user, item and interactions data for product recommendations. Three most commonly used methods are:

- Item-based collaborative method;
- Content based method;
- Hybrid method as a combination of the two above.

Item-Based collaborative filtering recommendation algorithms were described by Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl in their paper [11]. The idea behind this approach is for each user to identify another user or group of users having nearly similar interests so the recommendations can be made for the items which the current user has not had the interactions yet while those identified did.

The table below is the item-user interactions matrix. It visualises that User_1 and User_2 have similar interests, and User_2 is likely to be interested in Item_E. At the same time User_3 and User_4 show the same interest, and User_4 is likely to be interested in Item_D.

	Item_A	Item_B	Item_C	Item_D	Item_E
User_1	✓		✓		✓
User_2	✓		✓		?
User_3		✓	✓	✓	
User_4		✓	✓	?	

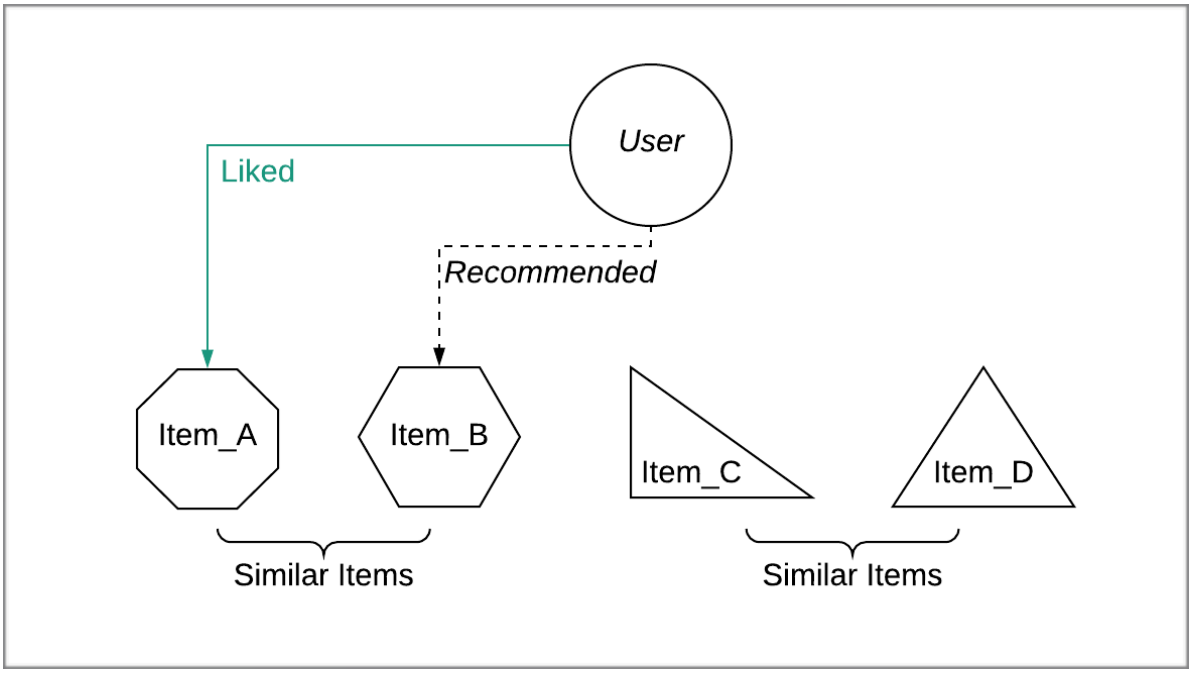
Machine learning algorithms like KNN and Matrix Factorisation are useful for working with the interactions matrix. KNN stands for k-nearest neighbour and is directly seeking similarity between users. Matrix factorisation is the way of latent representation of users and items when original interactions matrix is decomposed into two matrices, dot product of which is approximately equal to the original matrix and contains the predictions where the values were missing in original matrix. Example is shown below (values are used for demonstration purposes and were not calculated).

		Items																
		A	B	C	D											A	B	C	D
Users	1		1	2	4		1	0.8	2.3						1	3	1	2	4
	2	3					2	3.1	3.2		A	B	C	D	2	3	4	2	4
	3	5			4	≈	3	4.7	4.5	●	1.6	2.6	4.2	4.1	= 3	5	3	3	4
	4	1		1			4	0.2	2.9		1.5	4.9	3.4	2.2	4	1	3	1	3
	5		3		3		5	4.1	3.7						5	4	3	3	3

Typically, there are many users and even more items so the user-item interactions matrix are normally big. At the same time lots of values are not available due to the number of items per user and vice versa. This is why for storing this data the sparse matrices are commonly used since they benefit from storing only the data itself but not blanks which saves the disk space as well as lowers the computational cost of processing the data. Python library SciPy [12] contains various functions for creating sparse matrix.

Collaborative filtering has the highest accuracy since the predictions are based on cosine similarities between activities of different users. When there is no interactions data for user or item, for example new user or new item, the recommendations cannot be made and this is called cold start problem. Bottleneck of collaborative filtering is scalability when need to process the interactions data of millions of users and items as huge matrices.

Another approach is content based filtering when the products are being recommended based on the user and product metadata (description) [7]. This approach is more scalable since based on finding relationships between products which are more or less static. It also solves the cold-start problem when the user or item has no historical activities data yet but the provided preferences data suffices to make the recommendations. However, the range of recommended items is limited with content of current user interaction history and preferences so no products will be recommended outside of this scope. Image below visualises the process, when User liked Item_A which is similar to Item_B, so Item_B will be recommended to the User. At the same time Item_C and Item_D will never be recommended.



Neural Network algorithms are useful for the purpose of pattern finding and revealing the item similarities by processing the properties data. Normally this data is very complex and neural networks show high performance in finding hidden patterns. The downside of using neural networks is the interpretability and the results are harder to communicate to the business as the prediction process is not visualisable and for key stakeholders may look as a blackbox. In case of eCommerce recommendations, the cost of making wrong suggestions is not high so considering the low risks business normally relies on the less interpretable prediction process leading to outstanding results.

The level of personalisation can be improved by using hybrid type of filtering which combines the strengths of both content based and collaborative filtering [1].

While high quality recommendation normally leads to the purchase, there are more advanced commonly used techniques of stimulating customer buying multiple items (bundle): frequently bought together (actively used by Amazon), buy 3 for 2, cross-sell complimentary product [3], up-sell more expensive product, etc.

The project is aiming to develop the recommendation system using hybrid approach and recommend the discounted bundle of products the user is likely to be interested in which may lead to an increase of the basket value. The size of the discount is a subject to setting by the eCommerce website team, For the purposes of this project 10% will be offered. This approach supposed to catch several types of product combinations the customer can be potentially interested in.

Plan for developing the solution

Data collection

After the research of available datasets in the Internet, for the purpose of the project the RetailRocket dataset was chosen. The data was collected from the real-world eCommerce website in a period of 4.5 months by company Retail Rocket which is a provider of personalisation technologies services aiming to personalise the customer shopping experience. The dataset was published on Kaggle 23rd March 2017, current version 4 [10].

This data is represented by three files which reflect different behaviours and relationships between visitors and items. Each file contains unprocessed data, however, for the privacy reasons the values have been hashed.

Events dataset (file “events.csv”): contains the historical information about visitor interaction with items:

- timestamp (standard Unix format in milliseconds)
- visitorid (unique visitor identifier)
- event (values: view, addtocart, transaction)
- itemid (unique item identifier)
- transactionid (unique transaction identifier, empty for non-transactions events)

	timestamp	visitorid	event	itemid	transactionid
0	1433221332117	257597	view	355908	NaN
1	1433224214164	992329	view	248676	NaN
17	1433223236124	287857	addtocart	5206	NaN
19	1433221078505	158090	addtocart	10572	NaN
130	1433222276276	599528	transaction	356475	4000.0
304	1433193500981	121688	transaction	15335	11117.0

Properties dataset represents the item properties change log since they may change over the time: price, stock level, availability, etc. It consists of two files ("item_properties_part1/2.csv"). For the privacy reasons the property names and values have been hashed excepting "categoryid" and "available".

	timestamp	itemid	property	value
0	1435460400000	460429	categoryid	1338
1	1441508400000	206783	888	1116713 960601 n277.200
2	1439089200000	395014	400	n552.000 639502 n720.000 424566
3	1431226800000	59481	790	n15360.000
4	1431831600000	156781	917	828513
5	1436065200000	285026	available	0
6	1434250800000	89534	213	1121373
7	1431831600000	264312	6	319724
8	1433646000000	229370	202	1330310
9	1434250800000	98113	451	1141052 n48.000

Categories dataset (file "category_tree.csv"), shows the category tree in child-parent form. Field "categoryid" can be mapped with item properties data.

	categoryid	parentid
0	1016	213.0
1	809	169.0
2	570	9.0
3	1691	885.0
4	536	1691.0

Preprocessing

Events types “view”, “addtocart”, “transaction” are the implicit customer feedback and reflect the level of customer’s interest to the item. They can be considered as rating and will be transformed from categorical to numerical format with initial weights: view=1, add to cart=2, purchase=3. The weights are subject to tuning together with hyperparameters in order to achieve the best prediction results. This transformation also makes the data suitable for deep learning algorithms.

Initially the plan is to train the model on the whole data. If the results not suffice the cleaning from noise will be performed. Iteratively experimenting with data and methodology is a key to achieving the best results.

Firstly, the items meeting all three requirements below can be potentially removed since they have lack of item information available and it will be impossible to test purchase predictions on this part of dataset.

- no “transaction” events;
- no “addtocart” events;
- less than 2 ”view” events (alternatively try 3, 5 or 10 as threshold).

Secondly, the users that have not added any item to cart or haven’t made any purchase can be also potentially excluded as they are not likely to convert to buyers. Similarly, it won’t be possible to test purchase predictions on these users. This approach helps to keep focused on the most valuable users and provide the most relevant recommendations to them.

Traditional way of splitting the data into train and test by dividing the dataset into 20% and 80% parts will lead to training the model on separate user population than the validation set. Therefore the

timestamps will be used for train and test split which is useful as the user might interact with the item multiple times. Alternatively, some of user-item interactions can be “masked” randomly for training and the “masked” data then used for test.

Model training

Recommendations will be made in hybrid manner (combination collaborative and content based filtering) using user interactions with the items: view, add to cart, purchase. Python library LightFM [5] fits the needs of the project since it implements the hybrid recommendation filtering and includes matrix factorisation models.

There will be two phases of recommendation:

- item_1-to-user: for better customer experience in order to avoid recommending the same product all the time, one random of the top 5-10 recommended items will be chosen every time the recommendation is provided.
- item_2-to-item_1 which includes three potential approaches:
    - item_2 is another top-ranked item from step 1 (randomly selected from top of list);
    - item_2 with highest rating for item_1 in cosine distance matrix;
    - item_2 is frequently bought together with item_1.

As a result, the designed recommender system will return the two items offered with a predefined discount represented by a vector [item_1, item_2, discount] which in production supposed to be used by UI of the website.

Model evaluation

The project objective is to provide recommendations of discounted product bundle the customer can be potentially interested in, which does not necessarily mean that in test dataset these items are expected to be purchased together. The goal is to have the promyt high customer interest to the item shown by either “addtocart” or “transaction” event.

Recommendation task is a ranking problem: for each user the items are being ranked and the user is expected to be interested only in the top of the list. Therefore, there’s no point in measuring the performance on the tail of this list since the user will never see the tail. This is a significant difference from traditional machine learning tasks where the performance of the model is evaluated based on the full set of results.

LightFM python library has the module with evaluation functions for measuring the performance of the model. It includes precision and recall metric for top “k” items in the top list. ROC AUC metric will be useful to evaluate the quality of ranking when a randomly chosen positive example (event “addtocart”, “transaction”) is expected to have a higher score than a randomly chosen negative example [6]. All these three metrics will be used to evaluate the results.

Production readiness

For better customer experience in production the predictions need to be made online based on current user behaviour so precomputing of recommendations won't fit the needs as it won't factor in the current user interactions. Iteratively updating the user-item relationship matrix and retraining the model is computationally extensive and is unlikely to be possible to do real-time.

Alternatively, the trained model can be updated online with new inputs using the method LightFM python library called "fit_partial()". This will allow to provide the recommendations in real time and save the computational power and memory which will have a positive impact on recommendations quality and user experience.

Timeline

The roadmap is shown on the calendar below. Iteratively repeating certain project stages is needed as the pipeline is built on assumptions with several alternatives which will be tested during experiment.

Calendar Plan

Week start date	Jun 22	Jun 29	Jul 6	Jul 13	Jul 20	Jul 27	Aug 3	Aug 10	Aug 17	Aug 24	Aug 31	Sep 7
Data preparation												
Hyperparameter tuning												
Model training												
Post-processing												
Model evaluation												
Report writing												
Report formatting												
Submit report												

Experiments are expected to be finished by August 10, 2020 to reserve enough time for writing and formatting the report, assuming the project writing speed is about 500 words per day and 10,000 words length is needed.

Calendar plan is a subject to review on weekly basis and can be adjusted to meet the project goals within the given timeframe.

References

[1] Balabanovic, M., Shoham, Y., 1997. Content-Based, Collaborative Recommendation. Communications of the ACM Vol.40, No. 3. Available online at https://dl.acm.org/doi/pdf/10.1145/245108.245124. Last accessed 19/01/2020.

[2] Basu, C., Hirsh, H., Cohen, W., 1998. Recommendation as Classification: Using Social and Content-Based Information in Recommendation. Aaai, available online at https://www.aaai.org/Papers/AAAI/1998/AAAI98-101.pdf. Last accessed 14/12/2019.

[3] Grozin, V., Levina, A., 2017. Similar Product Clustering for Long-Tail Cross-Sell Recommendations. CEUR Workshop Proceedings, available online at http://ceur-ws.org/Vol-1975/paper29.pdf, last accessed 26/01/2020.

[4] Hu, Y., Koren, Y., Volinsky, C., 2008. 2008 Eighth IEEE International Conference on Data Mining. Pisa, Italy. Available online at https://ieeexplore.ieee.org/abstract/document/4781121, last accessed 06/01/2020.

[5] Kula, M. 2016. LightFM documentation. Available online at https://making.lyst.com/lightfm/docs/home.html. Last accessed 24/03/2020.

[6] Kula, M. 2016. Model evaluation. LightFM documentation., available online at https://making.lyst.com/lightfm/docs/lightfm.evaluation.html. Last accessed 24/03/2020.

[7] Lops, P., de Gemmis, M., Semeraro, G., 2011. Content-based Recommender Systems: State of the Art and Trends. In: Ricci F., Rokach L., Shapira B., Kantor P. (eds) Recommender Systems Handbook. Springer, Boston, MA.

[8] MacKenzie, I., Meyer, C., Noble, S. How retailers can keep up with consumers. McKinsey&Company, available online at https://

www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers. Last accessed 16/12/2019.

[9] Marr, B., 2018. How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read. Forbes, available online at https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/. Last accessed 14/12/2019.

[10] Retailrocket, 2017. Retailrocket recommender system dataset (Version 4). Available online at https://www.kaggle.com/retailrocket/ecommerce-dataset, last accessed 02/03/2020.

[11] Sarwar, B., Karypis, G., Konstan, J., Riedl, J., 2001. Item-Based Collaborative Filtering Recommendation Algorithms. Hong Kong. Available online at https://dl.acm.org/doi/pdf/10.1145/371920.372071. Last accessed 11/01/2020.

[12] SciPy documentation, sparse matrices, available online at https://docs.scipy.org/doc/scipy/reference/sparse.html. Last accessed 19/01/2020.