Department of Computer Science and Information Systems
Birkbeck College, University of London
2020

MSc Data Science

Project Report

eCommerce Recommender System For Increased Basket Value

Konstantin Orlovskiy, student ID: 13157188

Peter Wood, supervisor

Contents

Abstract

Before eCommerce, the traditional brick and mortar stores were growing the sales by displaying the bestselling products in the most visible places. The bestsellers were more likely to be purchased in higher volumes in comparison with niche items. While this approach worked well for business to achieve increased sales numbers, there was no customer personalisation available due to limitations of physical space to do personalised placements, and lack of customer data available.

The eCommerce significantly extended the seller's capabilities to provide the personalised customer experience on their websites. Having loads of data and capable computational capacity of modern computers to deal with the volume of data, gave a great start to developing the recommendation systems and their implementation in the industry [3].

This project is aiming to deliver the machine learning-based recommender system, which will stimulate the buyers to increase basket value. The system is recognising the products that represent the highest interest for the user and then bundles and discounts them in order to increase the amount user spends. User is benefiting from discounted personalised recommendations, and this leads to incremental revenue growth of the website.

Supervisor: Peter Wood

Acknowledgements

I want to thank my wife Alena for supporting me on the way.

Big thanks to the Retailrocket team for providing the data set for public use.

Introduction

Between 2016 and 2018 years, people generated 90% of the data in the world to date [19]. It is the source of a vast amount of information for machine learning application and in particular providing personalisations, which is motivating the development of recommender systems. The available computational power also grows exponentially, which makes the process doable.

The importance of the recommendation engines is growing in online retail nowadays: 35% of Amazon's revenue and 75% of Netflix's revenue comes through recommender systems [18]. Personalisation is a key to customer value growth and retention rate decrease led by the improved shopping experience.

Any solution will require the data to generate recommendations. This data typically includes user interactions with website pages, user feedback, information about product properties, user profile information, etc. The user may provide explicit and implicit feedback to the items. Explicit feedback is when the user rates the item giving a clear understanding of 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, return etc., which do not comprehensively describe how the item fits the user needs or interests. In practice, the recommender systems mostly deal with implicit feedback, which is collected by tracking user behaviour on the website [7].

While high-quality recommendation usually 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 complementary product [6], up-sell more expensive product, etc. This project is aiming to develop the recommender system based on the implicit feedback and item properties which will lead to increase the customer basket value by recommending discounted bundles combined from the top of recommended products that fit best to the particular customer. It is expected that in production, the use of the developed system will lead to incremental growth of eCommerce website revenue.

The report will cover the overview of existing techniques for providing recommendations, the motivation behind the choice of the dataset as well as an approach to processing, and finally, use of machine learning algorithms to generate the recommendations meeting the project purpose.

Background Review

Partially the background overview was given in the proposal to this project [9]. Below is a more in-depth review.

Common techniques

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

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

Collaborative filtering

Collaborative filtering recommendation algorithms were described by Badrul Sarwar et al. [24]. 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 technique is the way of latent representation (embeddings) 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 the original matrix. An example is shown below (values are used for demonstration purposes and were not calculated).

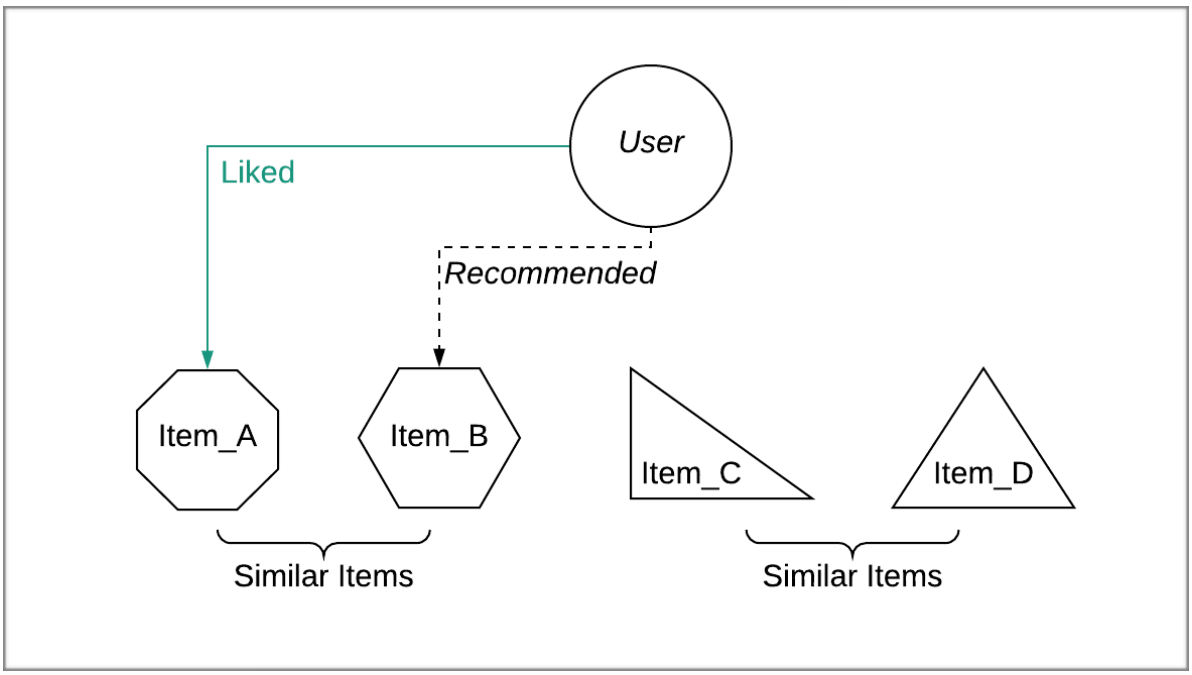
		Items																	
		A	B	C	D											A	B	C	D
Users	1		1	2	4		1	0.8	2.3						1	3	1	2	4
	2	3					2	3.1	3.2		A	B	C	D	2	3	4	2	4
	3	5			4	≈	3	4.7	4.5	●	1.6	2.6	4.2	4.1	= 3	5	3	3	4
	4	1		1			4	0.2	2.9		1.5	4.9	3.4	2.2	4	1	3	1	3
	5		3		3		5	4.1	3.7						5	4	3	3	3

Typically, there are many users and even more items, so the user-item interactions matrix usually is big. At the same time, lots of values are not available due to the number of items per user and vice versa. Thence, 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 [26] contains various functions for creating a sparse matrix. Sparsity is a challenge for collaborative filtering, and it may face 'cold start' problem when a new user that had no interactions cannot get the relevant recommendations to the lack of available data.

Content-based filtering

Another approach is content-based filtering when the products are being recommended based on the user and product metadata (description) [10]. The content-based method performs better than collaborative filtering on the high sparsity interactions data when collaborative filtering is facing cold start problem, so processing the user and item information (features) is the more suitable way of generating relevant recommendations. The idea is in finding similarities not between the interactions but between the items or users. The similarity is driven by the product/ user information meaning various properties. The item's metadata (item features) is usually uploaded when product is listed, and this info is already available when a new user joins. Therefore, the predictions may have more correlation with customer interest than pure collaborative filtering.

However, the range of recommended items is limited with the content of current user interaction history and preferences so that no products will be recommended outside of this scope/category. The image below visualises the process when User liked Item_A, which is similar to Item_B so that Item_B will be recommended to the User. At the same time, Item_C and Item_D will unlikely be recommended.



Neural Network algorithms are useful for pattern finding and revealing the item similarities by processing the properties data. Usually, 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 like a black box. In the case of eCommerce recommendations, the cost of making wrong suggestions is not high, so considering the low risks, the business may rely on the less interpretable prediction process leading to outstanding results.

Hybrid filtering

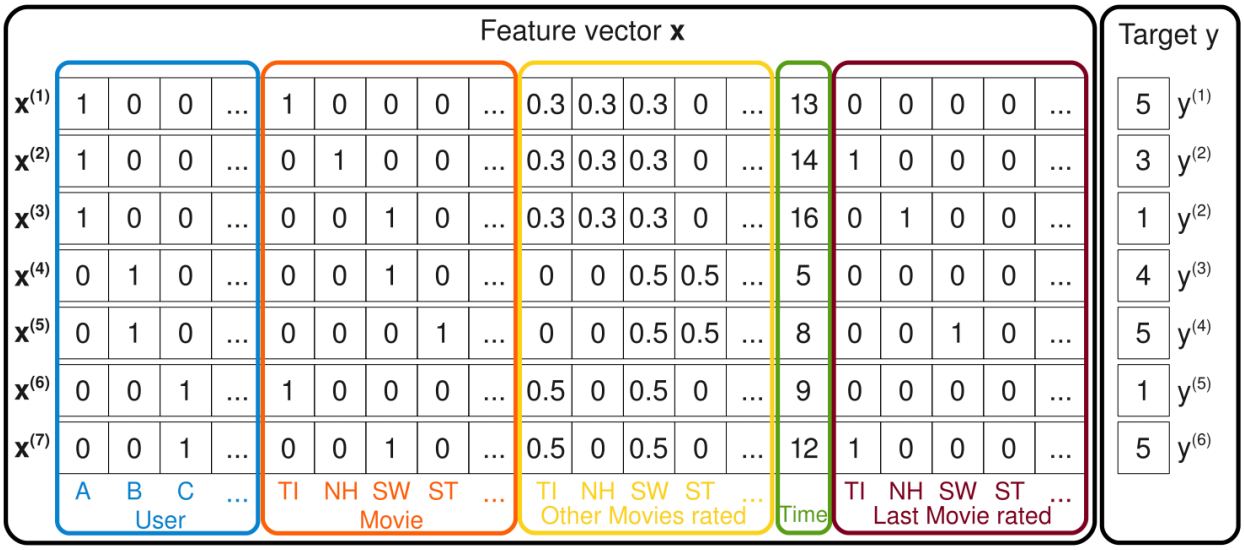
The level of personalisation can be improved by using a hybrid type of filtering which combines the strengths of both content-based and collaborative filtering [2]. Hence, the performance of hybrid filtering based models is not worse than separately collaborative or content-based.

The python package LightFM was chosen for recommender system development in this project. Reason for that is the aggregation by LightFM the benefits of two commonly used techniques for predictions meaning that it is hybrid filtering [12]. This section is dedicated to in-depth analysis of how the predictions are made by LightFM model and comparing the process to the other techniques.

The LightFM model is balancing both filtering types input into the predictions based on the sparsity of the data. Three matrices factorisation is done to implement this combination: user/item, item/feature and user/feature. When the user had interaction with only one item, and no user info is available, the model is using this item latent representation as user embedding. The combination of content-based and collaborative filtering can be considered as a specific example of factorisation machines (FM), which were described in detail by Steffen Rendle in his paper [27].

Factorisation machines is a powerful solution where most of the available interactions and metadata can be counted in for making the predictions on target. There is a matrix created with a dedicated range of columns for a specific type of information, including both implicit and explicit feedback: users, items, timestamp, previous activity, reviews, etc. The resulting matrix

represents the comprehensive view of the state of various variables that contribute to the recommendation.

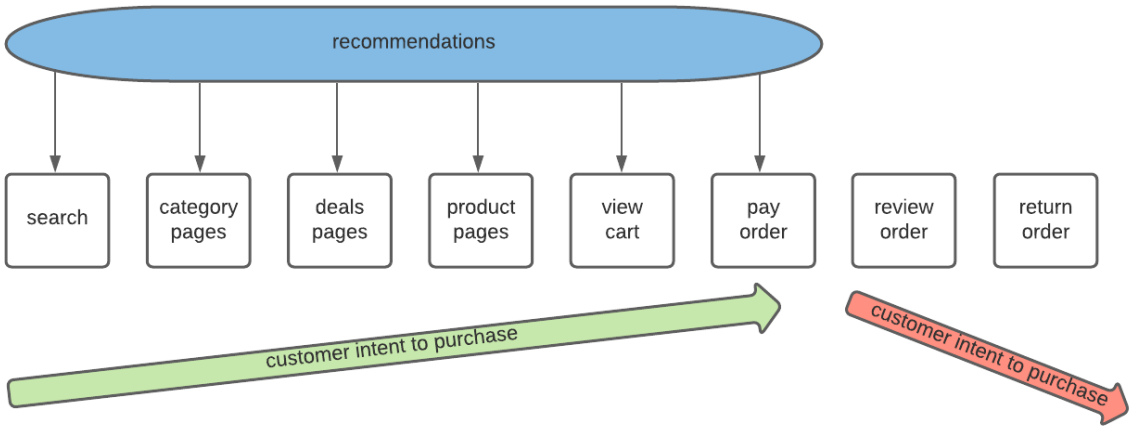


The most significant advantage in application to the current project is that the timestamps could be considered which is allowing to reveal hidden patterns in the specific time when item features may have looked differently: stock update, price change, new specification added, title updated, image updated, etc. Similarly, users may have had different intent: the latest activity trend, level of returns, basket value, etc. Even the fact that user started giving lower rank reviews to items in a bad mood can be counted in, so the latent representation is not impacted by the biased ranking and the relevant items are given the most accurate score. Additionally, the timestamp of interaction is vital for the fashion industry, for instance, so this is an excellent enhancement to factor in the trends.

However, the preparation of the data and determining the noisy parts of it was expected to be too extensive. In order to develop at least minimum viable recommender in time, the choice was made in favour of LightFM package having the range of implemented solutions to automate the process of data preparation and transformation.

Recommendations placement

User journey on the eCommerce website includes: search, browsing category pages, browsing deals pages, browsing product pages, viewing cart, paying order (checkout), reviewing the order, returning the order, etc. These stages are related to different customer intent and may include recommendations displayed on these pages. The closer to checkout stage the visitor is, the higher is the intent to pay. The diagram below illustrates the relationship between customer intent to purchase and the activity.



The proposed solution is not the stage-specific while can be easily adjusted to meet a specific interest, for instance, by limiting the pool of items by tag ‘sale’ for the recommendations made on the deals pages. The customer is already looking for discounted products and is much interested in having a good deal more than usual when browsing the products for the purpose. However, discounting the bundle of ‘on sale’ products may hit the P&L (profit and loss report) and therefore, the additional step needs to be implemented for excluding the items having issues with profitability. An interesting fact that deals page is the second most visited page on Amazon after homepage, which can be the case for the website the data is used from for this project so that the recommendation may convert into the sale very well.

Similarly, the pool of recommended products may be limited by the specific category which the user is browsing, providing the most relevant recommendations based on current intent. The trade-off will be in the potentially lower rank recommendations since the current category limits the full list. There is a room for experimenting and learning the consumer behaviour to find the balance between the focus on current intent and actual fit of the products to the particular buyer. The nature of the website products portfolio is essential to take into account when running the experiments.

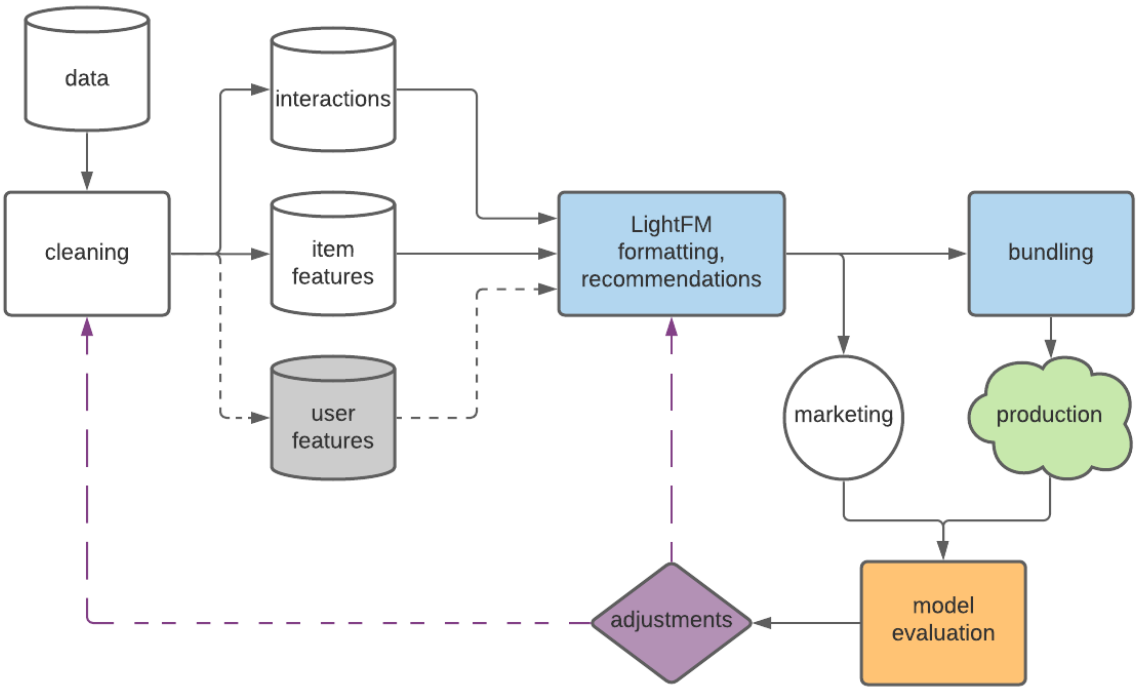
When viewing the cart and paying the order, the visitor may be offered the bundles that have the combination of the product in cart with another product which is not in cart. It will be the case of up-selling or cross-selling the items which are widely used in eCommerce and proved to work [23]. Implementation will require developing additional functional of filtering the bundles containing target product.

Additionally, the marketing team may generate recommendations for email campaigns when each user receives a personalised offer based on the expected interest. It is a powerful tool for the team when usually the email campaigns include the same range of items for all the users, or as the best case for the user group determined manually by the marketing team. Automation of process with a custom offer for each consumer is an immense contribution to the marketing activities outcomes.

Design

Diagram describing the architecture of the proposed solution is below. The user features functional is implemented in the solution and can be used if the user data is available (will require preprocessing). It is a sensitive topic across industry, and not all the consumers are providing personal information as well as permit to use it.

Intermediate results of recommendation before bundling is ready to be passed to the marketing team. The results of user interaction with the recommendations are evaluated at the final stage of the cycle, and as actionable - the required adjustments can be made for the cleaning (activity thresholds, positive interaction criteria, etc.) and recommendation phases. The iterative approach is beneficial for keeping the solution improving and therefore leading to both basket value growth and increased user experience.



Dataset

After the research of available datasets on the Internet, for 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 [21].

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

Events dataset (file 'events.csv'): contains the historical information about visitor interaction with items. Events types 'view', 'addtocart', 'transaction' are the implicit customer feedback and in some way, reflect the level of customer's interest in the item. The 'view' means the user visited the item page. The 'addtocart' is when the item was added to the cart by the user. The 'transaction' event means the user purchased the item.

- 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 changelog since they may change over 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. While events and item properties data are the vital source of information, the category tree data is in some way replicated in item properties since the products have the category id information. For further steps, the category tree data was not used but may be considered for a more in-depth exploration in future work.

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

Preprocessing

Events data

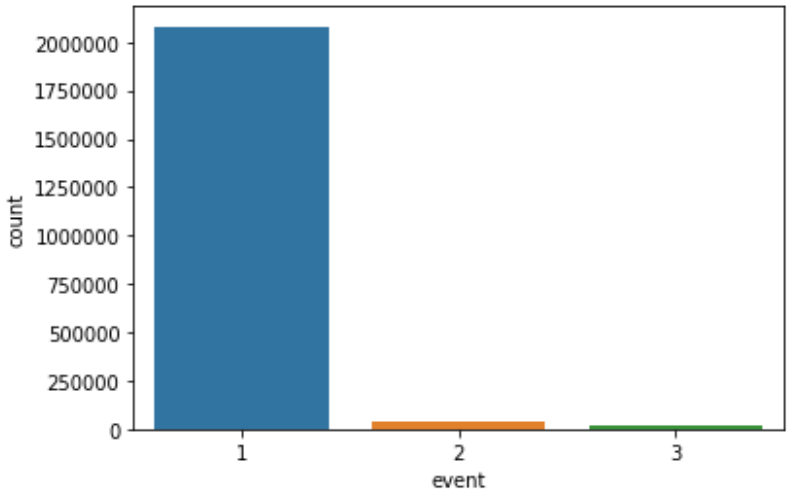
The events dataset contains event types ‘view’, ‘addtocart’ and ‘transaction’. Since there was a plan to use neural networks to process the data, these events types were converted from string values to numerical values: ‘view’ = 1, ‘addtocart’ = 2, ‘transaction’ = 3. Numbers still represent the categorical values but in a format acceptable by neural networks. The transformed form of events is still implicit feedback and cannot be considered as a rating.

The user may have interacted with the item multiple times which is now stored as activity log in the data frame. For the current purpose of recommendation, we are interested in the highest level of user interest to the item, which means purchase. The categorisation by numbers 1/2/3 simplifies the cleaning process and those interactions having higher number are left meaning that if the user has viewed (1), then added to cart (2) and then purchased (3) the item, the purchase interaction (3) will be kept for this item/user pair. It is helpful in need of excluding certain level of interaction from the further processing, for instance, non-positive interactions, which was the case for this project.

In order to test the computational cost of working with the full range of interactions, the LightFM model was run on all available activity data, and the run time exceeded 330 minutes. This result is not suitable for running experiments in the project considering the computational capacity of the used laptop and the time limitations of the project report upload. The decision was made to do the extensive analysis of the data aiming to clean the noise and less important information. As a first step, removing the users having only one interaction led to a significant decrease of run time and took ~50 minutes, which was still not suitable. The rationale of removing one interaction users lies in the inability to do a train/test split on only one event, and considering the number of such users, it would add the significant level of noise.

MacBook Pro (13-inch, 2019, Four Thunderbolt 3 ports)
Processor 2.8 GHz Quad-Core Intel Core i7
Memory 16 GB 2133 MHz LPDDR3
Graphics Intel Iris Plus Graphics 655 1536 MB

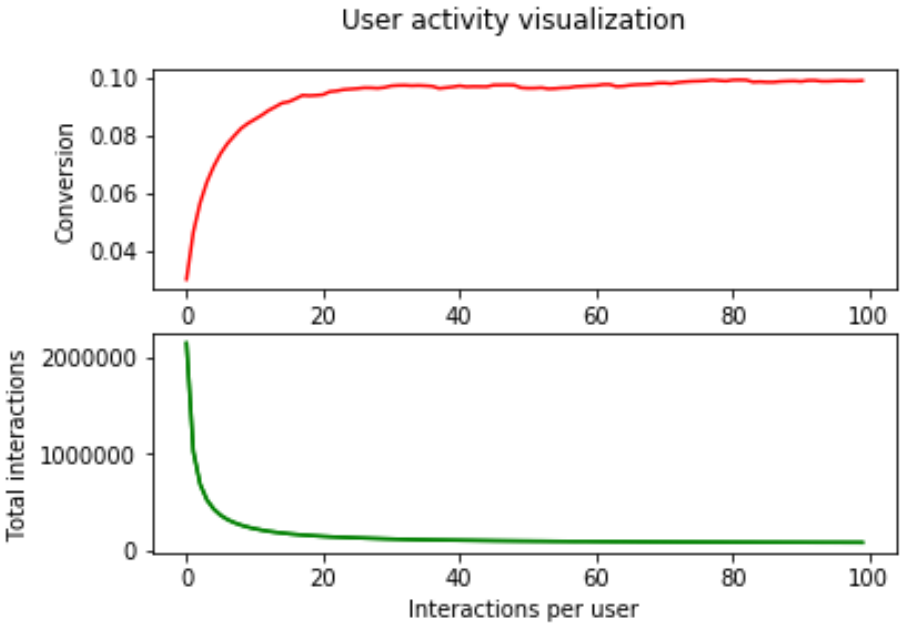
Further exploratory analysis of the dataset revealed the ratio between event types is tens of times uneven and the number of views is incomparable to the number of add to cart and transaction events: view is 96.67%, add to cart is 2.52%, transaction is 0.81%.



Add to cart event type technically is not a transaction; however, the fact that user interaction with item led to this step means the high interest/importance of this item to the user.

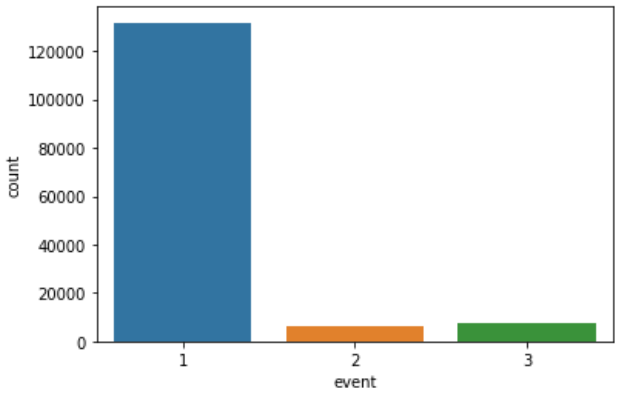
So the assumption is that **add to cart event can be considered as positive interaction along with the transaction event type**. The ratio of add to cart and purchase to total is relatively similar, which is additionally supports this assumption.

The ratio of positive interactions to total means the conversion, it is at the level of 3.33%. It is comparable to the eCommerce industry average according to latest studies [4]. The more in-depth look in the activity log at the consumers generating various event types reveals that there are two primary user types by activity: ‘low activity’ and ‘high activity’ users. From the plot below, it is clear that the highly active the user, the higher the conversion rate is.



The 'low activity' users are mostly browsing and making few purchases. The ratio of positive interactions is low, so it is harder to predict what they like. These users create noise for the pool of more active users having higher conversion. All types of interactions (view / add to cart/transaction) can be counted as positive for 'low activity' users. It will lead at least to the improvement of customer experience, and users are more likely to find the item they want. For this project 'low activity' group will be left aside as the goal is basket value, not user experience (UX). It will allow us to get rid of noise and decrease the size of the data frame saving the computational time, which is crucial in production. The recommender system will then be able to provide near to real-time recommendations and have the ability to retrain the model on the latest data and count in the most relevant activities.

The group of 'high activity' users has a higher ratio of positive interactions (add to cart, transaction), meaning that the conversion is also higher. For this group of users, **the view interactions cannot be considered as positive** since the user has not proceeded to the purchase, so this means there is low interest to the item. This approach will lead to focusing on what is attractive to the user and avoid making recommendations on the noise. For this project 'high activity' group will be used. Split point of 20 interactions between low and high activity users is chosen as the threshold at which the conversion rate growth slows down significantly.




```
Original number of users: 1407580
Cleaned dataset number of users:  1015
Cleaned portion: 0.07 %

Original number of items: 235061
Cleaned dataset number of items:  9490
Cleaned portion: 4.04 %

Original number of interactions: 2756101
Cleaned dataset number of interactions:  13675
Cleaned portion: 0.5 %

Original sparsity:  99.999167 %
Cleaned sparsity:  99.85803 %
```

The described steps of cleaning the data lead to the following results. The number of users represents 0.07% of the original. It is the user group worth focusing on to achieve the goal of the project - increase basket value. These users are much more likely to convert to a purchase. The cleaning process also improved the sparsity of interactions data to a level of 99.85%, which is challenging for the collaboration filtering performing better at the lower level sparsity. However, the fact of item features availability partially solves this problem, and the hybrid model should perform at least as good as the pure collaborative filtering model.

While the 'high activity' users have a higher level of conversion, the 'view' interactions were excluded from their interactions data as they are not considered as positive interactions since the user has not proceeded so this means there is low interest to the particular item. Similarly, when working with the explicit feedback, researchers are considering as positive only those ratings being over a selected threshold like over 4.0 stars, for instance. These decisions are made to laser focus on what matters for the users, which are the most important for the eCommerce business.

Train/test split cannot be done based on the time log of user interactions: use earlier interactions for the train, and leaving the latest interactions for the test. This approach may seem to be correct as it mimics the real-life having test happening later than training. However, this will impact the generalisation of the model because user behaviour depends on the seasonality, various eCommerce events (Black Friday, Cyber Monday, Christmas sale, etc.). So the split was done randomly with a ratio of 80% for train and 20% for the test.

As an additional step, the test set was cleaned from users that are not present in the train set. The rationale for this decision lies in the user cold-start problem: there is no user feature data available so the model will not be able to rank the products for the user who did not have any interactions, so the test results for this user in test set will create noise. In the real-life, these users will be recommended the most popular items (bestsellers). If the user fills in a sufficient amount of information in a personal account before browsing this will lead to adjusting the prediction based on the similarity of user profile (features) with other users who have the historical log of interactions. When the interaction data is collected, the user may be given more accurate recommendations based on the behaviour and therefore, the potential interest to the particular products.

There's no need for such action on items since the Retailrocket data frame contains the item features data and the content-based filtering part of the algorithm will solve the cold start problem for items which were not present in the train set. It is essential to know what to measure in order to make the right decisions. Otherwise, the noise in results may get the eCommerce team confused or lead to wrong assumptions, and the recommender system will not meet the expectations.

Item properties data

The information about item properties is stored in the form of a historical log of changes with total size ~890MB. The LightFM model, unlike the factorisation machines technique, is

unable to process this format, so there is a need to trim the properties data. Assuming that the eCommerce team was continually improving the catalogue and adding more accurate item info, the use of the latest values is reasonable. Removing the redundant data decreased the data size to 353.6 MB.

Additionally, the ‘available’ property was removed. It will not make sense to consider any value as fixed (in stock or not in stock) for training purposes since the stock position of the item may differ at the train and test stages which will lead to making incorrect predictions. In production, this property can be used in real-time to filter out unavailable items from prediction. There was no interactions data analysis done on user behaviour when interacting with items that are out of stock.

As the next step, the items which are not present in the cleaned data frame were also removed. It won’t be possible to evaluate the predictions on them in the isolated environment, where there are no interactions both in train and test sets. Additionally, this step significantly decreases the number of the data frame size to 7.8 MB. Cleaned item properties data frame represents 0.87% of the original. It is crucial considering the calculations for this project are made on a laptop with average computational specifications.

The item features information should be passed to the lightFM model in a format of Compressed Sparse Row (CSR) matrix. This matrix should have a mapping of all items related to the train/test process and all features/values pairs available in a cleaned properties data frame. Then the matrix is to be filled in with the properties values. LightFM model would be then capable of creating a latent features vector for each item (embedding). Two tables below visualise the difference in the commonly used and required format. The LightFM documentation did not have the clear explanation of the required format [13], and it took significantly more time than expected to figure out which format works by running simple examples on a small size data frame.

Common format of user features

	Colour	Material
Brush	brown	
iPhone	black	metal

LightFM required format

	‘Colour : brown’	‘Colour : black’	‘Material : metal’
Brush	1		
iPhone		1	1

Below follows the code showing how the transformation piece was resolved. The algorithm iteratively runs over the data frame, and for each row concatenates the property name and value in the format, and then adds them to a temporary array dedicated to the current user. When all the records for the current item are over, the tuple consisting of item id and its property/value combinations list are added to the output array. It is done assuming the data Fram was sorted by item id in advance, which is the case.

```
item_features_values = []
current_item = df_properties.itemid[0]
current_item_features = []
for index, row in df_properties.iterrows():
    if row['itemid'] == current_item:
        current_item_features.append(str(row['property']) + ':' + str(row['value']))
    else:
        item_features_values.append((current_item, current_item_features))
        current_item = row['itemid']
        current_item_features = [str(row['property']) + ':' + str(row['value'])]
item_features_values.append((current_item, current_item_features))
```

Data preparation

LightFM package has built-in Dataset class [14] which simplifies the process of preparing the matrices in the required format of coordinate format (COO) and compressed sparse row (CSR). This class using '.fit( )' method, creates the internal mapping of four dimensions: users, user features, items, item features. The used data frame does not have user features available so that the dataset class will have three dimensions. Since the train and test set interactions tables need to be separate, the variable 'dataset' is used as a mockup to build interactions separately for train and test data.

Dataset class method '.build_interactions( )' returns two COO matrices. One matrix contains interactions (binary), another has weights of these interactions, which were set for view, add to cart and transaction before. The idea behind the LightFM implementation is to consider in the implicit feedback as a binary problem, with either positive or negative interactions and then rank the items for a particular user by the likelihood of having positive interaction (add to cart or purchase). The weights matrix will not be needed for this project. Similarly, the required format of CSR matrix for item features was built using '.build_item_features( )' method.

```
dataset = Dataset()
dataset.fit(
    users = (x for x in df_events['visitorid']),
    items = (x for x in df_events['itemid']),
    user_features=None,
    item_features=item_features_mapping)

dataset_train = dataset
dataset_test = dataset

(interactions_train, weights_train) = dataset_train.build_interactions(df_events_train_interactions)
(interactions_test, weights_test) = dataset_test.build_interactions(df_events_test_interactions)

item_features = dataset.build_item_features(item_features_values)
```

LightFM Model Training

LightFM model includes both collaborative and content-based filtering so benefits from this combination. Providing features data improves the results on cold start problem when user or item did not have any interactions before. In this particular case, the only item cold-start problem will be resolved since there is no user data available. In order to check the impact of using a hybrid model, the training and test steps were run first without item features and then including them.

The model has a range of hyperparameters which are not a subject for training and are selected by the data scientist [15]. There are various techniques for improving model performance by hyperparameters tuning. The default values are shown below, and not all of them were used for a start.

```
model_default = LightFM()
model_default.get_params()
```

```
{'loss': 'logistic',
 'learning_schedule': 'adagrad',
 'no_components': 10,
 'learning_rate': 0.05,
 'k': 5,
 'n': 10,
 'rho': 0.95,
 'epsilon': 1e-06,
 'max_sampled': 10,
 'item_alpha': 0.0,
 'user_alpha': 0.0,
 'random_state': RandomState(MT19937) at 0x7FEB436F7380}
```

Firstly, the logistic loss suits when there are both positive and negative interactions available, for instance, 5-star rating to 1 start rating). The Retailrocket data set contains implicit feedback, so no ratings provided. The fact of the return of the item could be considered as negative interaction, but this data is not present in the events data set. In this project, it is assumed that only add to cart and purchase are positive interactions. BPR (Bayesian Personalised Ranking) and Weighted Approximate-Rank Pairwise (WARP) loss work for the positive interactions. BPR focuses on the optimisation of Area Under the Curve (AUC), which means any positive example will be ranked higher than any randomly chosen negative (non-positive) example. As a starting point, WARP loss is chosen since it improves precision at ‘k’ metric by maximising the rank of positive examples, it shows how many positives are in the top ‘k’ ranked items [15].

Secondly, considering the size of data passed to the model and computational capacity, the number of components (neurones) was increased to 100. Typically a higher number of components leads to improvement of performance with computational cost growth as a tradeoff. The model then is more capable of seeing the hidden and more complex patterns in the data.

As test metrics, AUC score and precision at ‘k’ are used. Both of them are measuring the ranking quality in a slightly different way described above, which means for each particular user, the items can be sorted in the order of their fit/interest to the user. For both metrics, the highest value is 1.0 meaning the perfect ranking [16].

Set parameter ‘k’ value to check precision at ‘k’ top-rated items. Initially select k=3 as it will allow having 4 possible combinations (bundles) to recommend to the user and achieve the goal of increasing the basket value. If these three items are A, B, C - then the potential bundles are:

1. A + B + C
2. A + B
3. A + C
4. B + C

A user will not be offered set of 2 for each item (like A + A). It is a more straightforward task and can be implemented across the whole catalogue on all pages if there is a business need for that. It will lead to better user experience and avoid recommending too much which can cause the fear of missing the opportunity which is related to FOMO (fear of missing out) problem [1] when the users are afraid of missing something important predominantly in social networks.

Model training

In order to compare how the item features influence the performance of the model, two separate models were trained and tested. The 'epochs' parameter determines how many learning cycles were being done over train data. The more cycles, the better is training, but the downside of a too high number of epochs is that the model may start remembering the train data rather than learning the patterns which lead to a decrease of model generalisation. As a starting point, 100 epochs is chosen. Recap:

- Collaborative filtering model - no item features used;
- Hybrid filtering model - including item features.

```
model_collab = LightFM(no_components=100,
                       loss='warp',
                       random_state=2020)

NUM_EPOCHS = 100  # Initial number of epochs.

model_collab.fit(train,
                 item_features=None,
                 epochs=NUM_EPOCHS,
                 num_threads=4)
```

```
model_hybrid = LightFM(no_components=100,
                       loss='warp',
                       random_state=2020)



model_hybrid.fit(train,
                 item_features=item_features,
                 epochs=NUM_EPOCHS,
                 num_threads=4)
```

Two selected metrics (AUC and precision) were scored on both models for train and test. It is helpful to see and compare the results in order to revisit the parameters or get back to the preprocessing stage if the results do not suffice. On the course of the project, this iterative approach was commonly used.

Model metrics comparison

	Collaborative filtering (no item features)	**Hybrid filtering (incl. item features)**
Train AUC	0.9999983	0.99953717
Test AUC	0.65434647	0.84518975
Train precision@3	0.7899966	0.5577252
Test precision@3	0.012444445	0.030222224

AUC score - this metric is often over 0.95 on train set as the model learns well on the train data. Having high train AUC is not determining how well the model is generalised as it may overfit the train data and show low performance on the test set. Test AUC on the collaborative filtering is 0.65, which is significantly lower than the hybrid model. It proves that the collaborative model is overfitting the train data. In contrast, the hybrid model is more generalised and shows acceptable performance on the test set, wherein 84% of the cases any positive interaction is ranked higher than randomly chosen negative interaction.

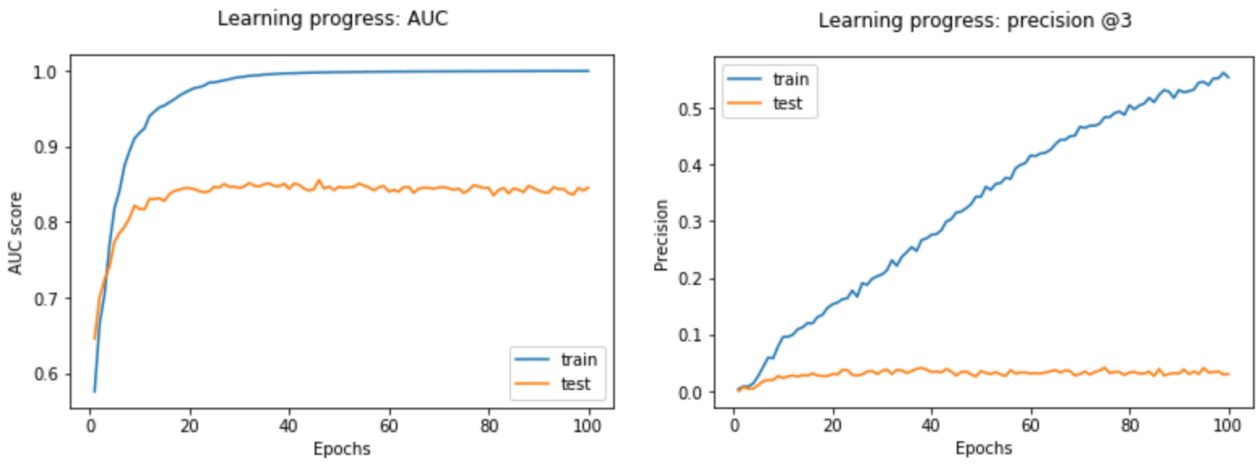
Precision at k=3 (precision@3) - there are no benchmarks on the precision at k for recommender systems as it strongly depends on the nature of the data. The below results show that collaborative model scores 0.78 on the train set so works better than hybrid model having 0.55 score, however, when it comes to the test, the hybrid model appears to be more generalised and performs twice better with score 0.03 which means that there are 3% of positive interactions in top 3 items across all users. The precision at k is always increasing with epochs and determining the ceiling number of epochs is essential to avoid overfitting the data with no improvement on the AUC side. Visualising the learning process is helpful to understand the model behaviour and select the most suitable threshold.

Learning process visualisation

In order to visualise the learning process, two functions were developed to automate the process and plot the results. Each iteratively runs the training and testing of the model and stores the result after each step and then visualises the dependency between evaluation metrics and epochs. Two functions have relatively similar code skeleton and differ by the measured metric.

The analysis of resulting plots revealed that the AUC improvement slows down significantly after 30 epochs. The train precision score is continuously improving with each epoch while test precision reaches the maximum between 20 and 40 epochs. In order to save the computational costs and optimise the process, the number of epochs is set to 30.

The behaviour of precision at k parameter depending on the epochs increase was expected: the more epochs pass, the more model starts learning train set (memorising the data) meaning that model's generalisation decreases and there is no further improvement in the test results. The selected threshold of 30 epochs corresponds to the plateau of the test precision curve, so no further consideration of precision at k results is needed, and the selected epochs threshold is sufficient.



Hyperparameter Tuning

The LightFM model has a range of fo hyperparameters that can be tuned in order to improve the performance. When trained, the model is not adjusting them, so they need to be selected manually by the data scientist. There is no right or wrong setup as it depends on the data used, so there are several techniques to define the combination that leads to model generalisation and performance improvement.

One of the techniques to automate the process of selecting the best performing combination of hyperparameters is random search. The idea is for each optimised hyperparameter to define the array of considerable values and then randomly sample various combinations of hyperparameters values and select the best performing one. This technique has proven to show good results with less computational cost in comparison to the grid search technique when the hyperparameters samples choice is extensively run across the full range of values in the defined arrays. The array of values for the hyperparameters was generated by custom function 'samle_hyperparameters( )' which can be later called by other functions running the hyperparameter tuning itself.

```
def sample_hyperparameters():

    # Choose the hyperparameters to tune and the range of values.

    while True:
        yield {
            'no_components': np.random.randint(10, 100),
            'learning_schedule': np.random.choice(['adagrad', 'adadelta']),
            'loss': np.random.choice(['bpr', 'warp', 'warp-kos']),
            'k': np.random.randint(1, 10),
            'n': np.random.randint(1, 20),
            'learning_rate': np.random.exponential(0.005),
            'item_alpha': np.random.exponential(1e-10),
            'max_sampled': np.random.randint(5, 30),
            'num_epochs': np.random.randint(10, 100)
        }
```

The author of the LightFM package Maciej Kula has also developed the algorithm for hyperparameter tuning. The original algorithm was posted on GitHub 23 Apr 2018 [11]. For this project the original code was considered in the development of the code that meets the project needs, meaning that the tuning was not only focused on AUC metric improvement, but also with a focus on precision at k performance. There are two attempts. AUC focused tuning when the best model will be chosen based on the best AUC result. Precision focused optimisation when the measure of success to choose the best model is precision at 'k'. This optimisation process if aiming to improve the ranking in the top of the list and does not take into account how well the rest of the recommended products list is sorted so this may affect the AUC metric.

Before running the optimisation algorithms, the train data was split into train-train and train-validate. The algorithms select sample hyperparameters, train the model on train-train data, and score the metrics on train-validate data to find the best performing combination. Then the best model is tested on a test set. This approach allows keeping the train data unseen for the model during the tuning process to make sure the model does not learn the test data and shows fair results. Another method is to run a cross-validation hyperparameter tuning over the whole train set when the whole train data is split in different train-train, and train-validate pairs and the performance is measured. It leads to a high level of model generalisation. The methodology is implemented in scikit-learn package [25]; however, it was not used in this project as the function 'RandomizedSearchCV( )' needs the 'estimator' parameter (model) to be passed as input, but the LightFM estimator is not included in the scikit-learn package. Therefore, the custom solution was developed to run the hyperparameter tuning.

Model Evaluation

In production, the users will be given the recommendations and will possibly interact with them so the model can be evaluated on the recommendations' conversion rate and basket value. This project is done in an isolated environment, where the test set of users is not getting recommendations. The best possible way to evaluate the model, in this case, is to see the AUC, precision and recall metrics over the test data.

During the experiments, the focus was made on two metrics: AUC and precision at k. AUC is essential for understanding overall how well the array of items was ranked for the particular user. Precision shows how many actual positives are in top k ranked items. The recall is demonstrating the portion of true positives to total positives in top k items. Both precision and recall are the measures of relevancy, so one of them (namely precision) was chosen for the further evaluation of models.

Item features were successfully used to improve the model performance based on the metrics' scores. The best-achieved test AUC is ~0.84, which means that in 84% of the cases randomly selected positive interaction is ranked higher than any randomly selected negative interaction. This metric describes how well the products are ranked for each particular user. The best-achieved test precision is ~0.03 at k=3, which means in 3% of the cases, the positive interaction is in the top 3 recommended items. Both achieved AUC and precision are comparable to the available web results [22].

Performance comparison

	Original Hybrid Model	**AUC focused tuning**	**Precision focused tuning**
Train AUC	0.99953717	0.9620935	0.95148015
Test AUC	0.84518975	0.8172267	0.83780867
Train precision@3	0.5577252	0.079822	0.041453
Test precision@3	0.030222224	0.009778	0.009778

Hyperparameter tuning has not led to the improvement of both metrics. The extensive search over the array of hyperparameters was done with a focus on separately AUC and precision. The performance of the model after tuning has not improved in comparison to the original manually selected model. Researchers face this situation quite often, and it means the default hyperparameters fit the data better. The authors of the machine learning packages usually select default values for hyperparameters based on the empirically proved best performance in the industry which has been again proved with current data. Moreover, with new releases of the packages, the default values may be changed based on the latest findings. The original model with a mix of manually selected and default hyperparameters is proved to be the best and is selected to proceed further.

The model evaluation process should have the iterative nature based on the test results received continuously in production. It is an important process of any machine learning solution leading to an improvement of the existing model or making the decision to switch to another solution or algorithm. Collecting and analysing the customer feedback - explicit (reviews) and implicit (clickstream data) - requires the pipelined solution, which is described in the 'Production and Feature Work' of this report. Implementation will depend on the eCommerce website data architecture while the top-line view shown on the diagram will be similar.

Recommendations

The recommendation process implemented in the proposed solution consists of two phases. The first phase is identifying the list of top recommended items for the user. The second phase is the process of bundling of top N products (refers to parameter k, which was used to evaluate precision) and returning the recommendation of 4 bundles. Recommendations algorithm was developed with consideration of the approach used in LightFM package examples [17].

The two phases are implemented in two separate functions which give the flexibility for the website marketing team in using the full ranked list of items for each user for various activities like personalised email campaigns for a broader range of products or recommending new versions of the items or anything else. Additionally, this approach gives some flexibility when changes in the functionality are done separately per phase, so the code compilation process is simplified. Phase two as input takes the phase one output which is the only case of potential errors caused by formatting.

Ranking the products

First phase algorithm is generating the list of top N products sorted from top to less attractive for a particular user. In production, the parameter N can be empirically selected because some items from this list can be out of stock or unavailable for sale for any other reason. The functionality allows running recommendation for the list of users in one go. It may be needed when the recommendation process is not run live but overnight or several times a day, and the results are stored in between. Additionally, it may be helpful for the marketing team in email campaigns preparations, when the specific range of users are targeted so the task can be done in one go. The function returns the dictionary which has a similar structure as JSON (JavaScript Object Notation) format broadly used in Big Data systems.

```python
def recommend_items(model_trained, dataset, interactions, userids, N):

    '''
    model_trained - the trained model to be used for recommendations.
    dataset - LightFM Dataset class used to create mapping.
    interactions - coo matrix of known positive interactions.
    users - list of users for recommendations.
    N - top N recommendations to return.
    '''

    model = model_trained
    # Extract user and item mapping from created Dataset class.
    # Mapping is stored as tuple of arrays (user id map, user feature map, item id map, item feature map).
    # Need to extract arrays with  index 0 and 2.
    mapping_users = dataset.mapping()[0]
    mapping_items = dataset.mapping()[2]

    output = {}

    for user in userids:

        # Convert user to internal index.
        user_internal = mapping_users[user]

        # Extract internal indecies of items whith which user positively interacted in train set.
        positive_items_internal = train.tocsr()[user_internal].indices

        # Convert internal item indecies to itemids.
        positive_items = [key for key, val in mapping_items.items() if val in positive_items_internal]

        # Make prediction.
        scores = model.predict(user_internal, np.arange(len(mapping_items)))

        # Sort the recommended items by descending order of their importance.
        # Array has the internal indecies of the items.
        recommended_items_internal = np.argsort(-scores)

        # Convert internal item indecies to item ids and return top 10 recommendations.
        recommended_items = []
        for id_internal in recommended_items_internal[:N]:
            for key, value in mapping_items.items():
                if id_internal == value:
                    recommended_items.append(key)

        # Add user and recommended items to output.
        output[user] = recommended_items

    # Returns the recommendation in format{user_1:[item_1, item_2, item_3], user_2:[item_4, item_5, item_6], ...}
    return(output)
```

The recommendations may include items with which the user had positive interactions. Since the item features names and values are hashed, there is no proof whether the items can be considered for the repetitive purchase or not. If the item is refillable (paper towels, toothpaste, dishwasher machine detergent, cleaning chemicals), it can be the case they may be recommended one more time, while the sofa, iPhone or carpet are unlikely to be recommended the second time. If the information is provided, the pool of potentially recommended items may be reviewed for better UX. It will require developing additional functional. As an example, the function was run on a range of 3 user ids over a train set.

```python
# Make recommendation to the set of users based on the train interactions.
selected_users = [566009, 170470, 64931]

start_time = time.time()

plain_recommendation = recommend_items(model_trained=model_best,
                                       dataset=dataset,
                                       interactions=train,
                                       userids=selected_users,
                                       N=3)

print('Recommendation made in: ', round((time.time()-start_time), 6), " seconds")

plain_recommendation
```

```
Recommendation made in:  0.10763  seconds

{566009: [248676, 247842, 234603],
 170470: [134525, 352230, 369447],
 64931: [240755, 415610, 119433]}
```

The recommendation was made in ~0.1 second, which is sufficiently fast for providing live recommendations to the users while they are shopping. The value of N can be selected higher to be able to show the visitor new range of bundles every time which will potentially improve the experience: generate a broader range of bundles to recommend and then show recommendations in chunks determined by the user interface capabilities.

Product bundles

The second phase of recommendation is creating the bundles of items recommended in phase one. Since the item prices are hashed in the original data set, and the total number of bundles is 4, all bundles can be recommended to the visitor with no negative impact on user experience. It is in line with the project goal of increasing the basket value by providing the discount in case if the user is buying a bundle. The below algorithm is run over the output of 'recommend_items( )' function and for each user generates the list of bundles.

```
def bundler(plain_recommendation, bundle_size):
    '''
    plain_recommendation - is dictionary same as output of function recommend_items().
    '''

    recommend_products_bundles = {}


    for user, item in plain_recommendation.items():
        combos = [tuple(plain_recommendation[user])]
        for bundle in combinations(plain_recommendation[user], bundle_size):
            combos.append(bundle)
        recommend_products_bundles[user] = combos

    return(recommend_products_bundles)
    # Fromat is {user_1: [(bundle_1), (bundle_2), ...],
    #            user_2: [(bundle_1), ...], ...}
```

As an example, the above output was passed to the function which returns the dictionary containing the list of bundles respecting the bundle size set manually. Additionally, the bundler adds the full list items to output. This functionality allows to keep the view on all recommended items at the final stage of recommendations, and it can be easily excluded if needed as comes as the very first item in the user recommendation list of bundles.

```
# Make bundling of recommendation.
bundler(plain_recommendation,bundle_size=2)
```

```
{566009: [(248676, 247842, 234603),
  (248676, 247842),
  (248676, 234603),
  (247842, 234603)],
 170470: [(134525, 352230, 369447),
  (134525, 352230),
  (134525, 369447),
  (352230, 369447)],
 64931: [(240755, 415610, 119433),
  (240755, 415610),
  (240755, 119433),
  (415610, 119433)]}
```

The project was aiming to develop the recommendation system using the 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 a setting by the eCommerce website team. As the next step, the price per bundle needs to be calculated and discount to be applied. Since the prices are hashed in the item properties data, this step is skipped. Having the hashed information in provided data set applied the range of limitations on this project. As a learning, the choice of a data frame for the academic project

should be made carefully, and the limitations acknowledged as early as possible. However, coping with the limitations develops the expertise of working in the environment of privacy, for instance in compliance with the General Data Protection Regulation (GDPR) which is considered to be the strictest privacy and security law in the world [5].

The number of recommended bundles is not high (only 4), so the user may be recommended all four bundles on the same page. It will lead to improved user experience (UX) as all bundles are offered at once, and the user can choose. Potentially, the bundles can be sorted by the total value and positioned in the sequence from high to low, so the top value bundles have more visibility. In the case of chasing a broader number of bundles to be recommended, the order of prioritisation becomes more critical as it may impact the balance between user interest and willingness to have a good deal.

Production and Future Work

The developed solution is ready for implementation in production. The prediction process is staged and allows the eCommerce website team to benefit from the intermediate results. Also, the output can be adjusted according to the needs. The first stage functionality - 'recommend_items( )' function - includes the possibility of making predictions for the array of users at once and select the number of top-ranked items to be returned. The output does not count in the current availability of the items (in stock or not) and therefore, the additional step may be needed to exclude out of stock items from prediction. Hence, the implemented parameter N is providing additional flexibility considering the possible limitations.

The second stage function 'bundler( )' is designed to fit the first stage output format, which makes the process of recommendations seamless and flexible at the same time. The function generates all possible combinations from the recommended top N items based on the setting of the number of items per bundle. Having the bundle size setting functionality allows adjusting the proposed solution in production depending on the performance and needs of the experiment or A/B test. The access to the product prices could lead to the further extension of the functionality by using a broader list of items and bundle sizes and sorting them by the total bundle value, so the user is offered the highest value bundle at the top of the recommendation list. It may not necessarily mean the top of the list is the most relevant but offering the discount on the bundle is expected to be the stimulus for a user to purchase.

The output format of both stages is a dictionary and corresponds to the JSON format, which is broadly used in modern data systems. It simplifies the implementation of the recommender system into existing data systems and removes complexity and extensive data transformation.

The level of user experience can be potentially impacted by the repetitive offering of the same recommendations over time. If there is the case and eCommerce website team wants to sacrifice the recommendation quality to UX, the parameter N of 'recommend_items( )' can be increased, at the same time the settings need to be changed in 'bundler( )' function - at least bundle size, or code modification to add functional of generating bundles of various sizes. These changes require extensive analysis of results.

Before making recommendations for production, the selected model should be trained on the whole cleaned dataset (without train/test split) to be able to count in all the meaningful interactions in the model. Also, all items from the item properties file should be added to the Dataset class variable, and all item features matrix is to be built. For academic purposes, item features data was preprocessed initially to lower the size by removing redundant data as well as remove noise from the test results (availability property). In real life recommendations, all of this data matters and should be used for predictions, and the results can be evaluated based on user behaviour.

Item properties information is hashed in the used data frame. The eCommerce team will be able to work on the non-hashed information to do the feature engineering by exploring in detail the distribution of various features, applying scaling techniques, excluding redundant features or those creating noise. Additionally, the category tree file can complement this analysis. It was proved in this report that passing the item features to the model increase the AUC performance by 19% from 0.65 to 0.84, so the enhancement of item features data will lead to the increase of model efficiency. The Retailrocket data does not include user features information which was probably motivated by the privacy issues so the info may have been removed from the public data set. However, the user data can be available internally within the eCommerce team. In a format of CSR matrix, this data can be passed to the model and will potentially lead to the improvement of the predictions. The model will create the latent representation of user features and count them in when ranking the items for each particular user. This enhancement improves the content-based filtering capabilities of the solution and may lead to improvement of the prediction over sparse data.

The preprocessing stage needs to be iteratively rerun, for instance, once a day, to be responsive to the visitor behaviour and providing the most relevant recommendations considering the latest interactions. Increase in a number of individual user interactions in comparison to the

selected threshold (driven by conversion rate in the current project) may move a user to ‘high activity’ group and let the user getting recommendations of higher quality. Item features data needs to be reprocessed as well since the CSR matrix headers have the form of ‘property_name : property_value’ so when the item property is updated, the old column becomes not relevant anymore, so the corresponding cell in CSR matrix needs to be cleaned which is not implemented in this project - therefore there is a need of reprocessing.

Pitfalls

The developed technical solution can now be implemented into the website user interface, and, additionally, can be considered for marketing initiatives. In production, it is essential to measure the success of recommendations. Hung-Hsuan Chen et al. described four common pitfalls of evaluating recommender systems [8]. The team of researchers flagged the issues and ran the experiments to demonstrate them.

This project was done in an isolated environment without providing actual recommendations to users, so the measure of recommendations accuracy was based on two metrics: AUC and precision at 3. In production, when recommending the items to visitors, the first issue raises because the recommended products start getting higher visibility than others so are easily reachable by a visitor, which makes their choice biased. So the model trained on the post-recommendations data can be biased on the visibility of recommended items.

The second issue lies in the fact that the recommender system is impacting the distribution of test data. When the model is additionally trained on these recommendation interactions, it makes the model is biased as well. The test data should be carefully selected before evaluating the results of recommendations to avoid favouring the interactions with recommendations over the rest of the clickstream data for other items.

The third issue is driven by the fact that comparing the models by the click-through rate (CTR) on provided recommendations is a pitfall since the click on the recommended product is an intermediate result the ultimate goal of increasing the company revenues. CTR correlates with the user experience helping visitors in reaching the relevant products faster with less effort. The project goal is set to ‘increased basket value’, and CTR was not supposed to be the key measure.

The fourth issue described in the paper is related to the incremental revenue growth led by the recommender system. While the visitor may find the recommended product attractive and convert into a sale, the same product may have been found by a visitor without recommendation, merely using the website search engine. There is a concept of ‘impulsive purchasing’ when a specific stimulus determines consumer behaviour. It is described in detail by Regina Virvilaitė et al. in their paper [20]. Based on this concept, the websites having certain products portfolio (gifts, kitchen accessories) have a high probability of benefitting from recommender system implemented in the user interface: the user will buy the products they would not search for at all.

All four described issues need to be considered when the developed solution goes in production. The additional point to consider is the conversion of discounted products bundle. When offered 10% off (for instance), the user may be more likely to purchase this bundle because this is a good deal, while the user may not consider buying without this offer. So, counting in the fourth issue described above, the eCommerce website team may decide to limit the pool of products in recommendations by those moving slowly and stimulate the purchase. Theoretically, this approach may be able to bring incremental revenue growth. Most of the brands are successfully running clearance campaigns having the same intent, and the recommender system can add efficiency to this process and drive incremental sales.

A/B testing would help evaluate the recommender system added value when comparing the revenue generated by two focus groups of similar users: when the first group is not given recommendations while another is offered the bundles of products with discount. This process is a medium-to-long-term and is related to the buyer lifetime value. The fact of purchase after receiving recommendation may delay the next purchase, especially related to fast-moving consumer goods (FMCG) and refillable products specifically when buying a mega pack of toilet paper means postponing the next purchase of a similar item. Hence, the scale of incremental

revenue growth needs to be continuously re-evaluated over the customer lifetime value estimate. Among the essential machine learning applications, recommender systems, estimation of lifetime customer value and repricing algorithms are standing out as the sales-focused in comparison to others, which are mostly leading to safety and user experience improvement like chatbots, demand/supply, fraud detection.

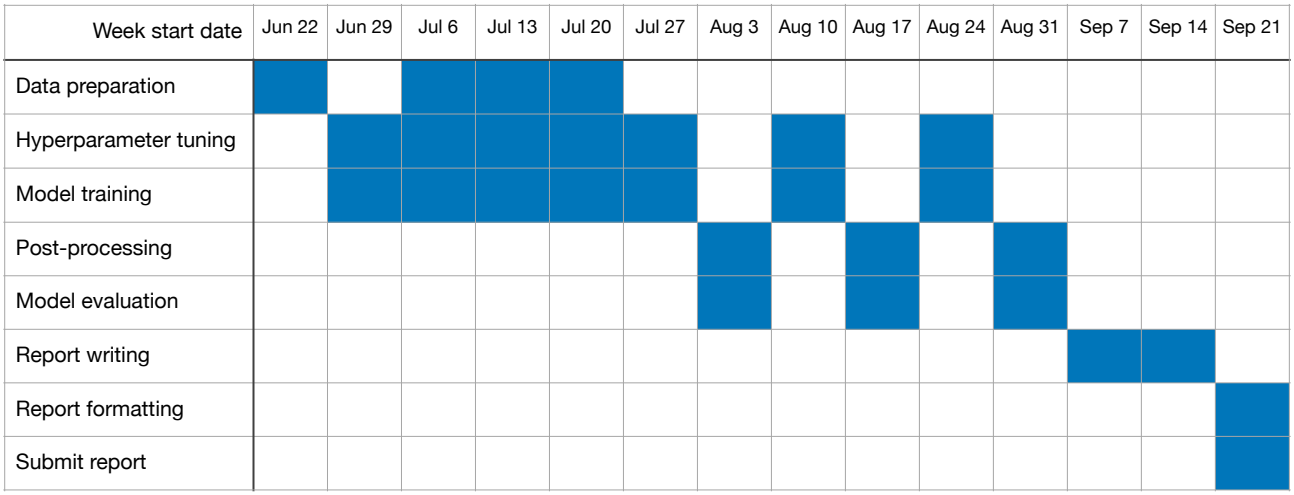
There is a potential room for improvement of recommendations if the users/items interactions log will be limited by a specific period. The project solution was developed on the 4.5 months interactions history, which was fully used. For the fast pace moving industries like fashion, it may be necessary to 'refresh' the latent representation of users and items by limiting the historical depth of data used. The approach and pipeline used in the project fit this purpose, however the decision on the interactions threshold may be reconsidered, as well as the choice of hyperparameters used - so eventually it comes to tuning of the existing solution. The factorisation machines technique allows counting in the trend without removing the older data while requires more extensive data preparation. Depending on the eCommerce website team capacity and goals, this technique may be considered.

Conclusion

The work on the project started on 21st June as planned. The timing and stages were well planned, and the process was pipelined to consider the nature of working on machine learning projects and included the multiple iterations aiming to improve the model performance. Having a clear plan helped to adjust and keep going when getting stuck with some minor issues. The plan was being reviewed every week.

The preprocessing stage took a significantly longer time and was driven by multiple factors. Firstly, there was a need of making trade-off decision considering the computational capacity of the used laptop in order to avoid too long run of each experiment and stick to the frequent iterations process with an evaluation of results and revisiting the settings. Secondly, the used LightFM package had unclear documentation on the format of the inputs that need to be passed to the package modules in order to build the matrices of required shape and form. These package modules were designed to simplify the work but having unclear documentation required additional effort to make it work. The iterations of data preparation, hyperparameter tuning and model training took less than one week each time, and after the code skeleton was created, the pace was fast enough to figure out the best option to proceed with to the further stages. The project report submission deadline was moved two weeks further by the university, so the Gantt diagram was updated accordingly.

Actual Pace



The developed recommender system can be considered as a core which then can be adjusted to fit the specific need. Recommendations are given in a form that is ready to use by cross-functional teams without a technical background. Collecting the feedback on the provided recommendations is a crucial part of improvement, and the proposed diagram of the solution is helpful to pipeline the whole process. Access to the non-hashed user and item data will potentially increase the quality of recommendations and lead to basket value growth.

References

1. Andrew K. Przybylski, Kou Murayama, Cody R. DeHaan, Valerie Gladwell, Motivational, emotional, and behavioral correlates of fear of missing out. Computers in Human Behavior, Volume 29, Issue 4, July 2013, Pages 1841-1848. Available online at https://www.sciencedirect.com/science/article/abs/pii/S0747563213000800, last accessed 02/09/2020

2. 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

3. 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

4. Digital Marketing Institute, What is a good conversion rate for e-commerce?. Available online at https://digitalmarketinginstitute.com/blog/what-is-a-good-conversion-rate-for-ecommerce, last accessed 06/07/2020

5. European Union (EU), The General Data Protection Regulation (GDPR), 2018. Available online at https://gdpr.eu/what-is-gdpr/, last accessed 10/08/2020

6. 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

7. 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

8. Hung-Hsuan Chen, Chu-An Chung, Hsin-Chien Huang, Wen Tsui, Common pitfalls in training and evaluating recommender systems, 2017. Available online at https://www.kdd.org/exploration_files/19-1-Article3.pdf, last accessed 28/08/2020

9. Konstantin Orlovskiy, 2020. MSc Project proposal, eCommerce Recommender System For Increased Basket Value. Department of Computer Science and Information Systems, Birkbeck College, University of London

10. 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

11. Maciej Kula, LightFM hyperparameter optimization. Available online at https://gist.github.com/maciejkula/29aaf2db2efee5775a7f14dc387f0c0f, last accessed 02/09/2020

12. Maciej Kula, Lyst, 2015. Metadata Embeddings for User and Item Cold-start Recommendations. Available online at https://arxiv.org/pdf/1507.08439.pdf, last accessed 16/12/2019

13. Maciej Kula, Lyst. LightFM Documentation, Dataset construction, build item features. Available online at https://making.lyst.com/lightfm/docs/lightfm.data.html#lightfm.data.Dataset.build_item_features, last accessed 02/09/2020

14. Maciej Kula, Lyst. LightFM Documentation, Dataset construction, dataset class. Available online at https://making.lyst.com/lightfm/docs/lightfm.data.html#lightfm.data.Dataset, last accessed 02/09/2020

15. Maciej Kula, Lyst. LightFM Documentation, LightFM model. Available online at https://making.lyst.com/lightfm/docs/lightfm.html#lightfm.LightFM, last accessed 02/09/2020

16. Maciej Kula, Lyst. LightFM Documentation, model evaluation. Available online at https://making.lyst.com/lightfm/docs/lightfm.evaluation.html, last accessed 02/09/2020

17. Maciej Kula, Lyst. LightFM Documentation, Quickstart. Available online at https://making.lyst.com/lightfm/docs/quickstart.html, last accessed 02/09/2020

18. 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

19. 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

20. Regina Virvilaitė, Violeta Saladienė, Jūratė Žvinklytė, The impact of external and internal stimuli on impulsive purchasing, Economics and management: 2011.16, 2011. Available online at https://etalpykla.lituanistikadb.lt/object/LT-LDB-0001:J.04~2011~1367177966372/J.04~2011~1367177966372.pdf, last accessed 04/09/2020

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

22. Retailrocket, Retailrocket recommender system dataset. Available online at https://www.kaggle.com/retailrocket/ecommerce-dataset/notebooks, last accessed 10/09/2020

23. Ron Kohavi, Rajesh Parekh, 2003. Ten Supplementary Analyses to Improve E-commerce Web Sites. Available online at https://www.researchgate.net/publication/2923317, last accessed 20/02/2020

24. 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

25. SciPy documentation, Randomized search of hyperparameters. Available online at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html, last accessed 02/09/2020

26. SciPy documentation, sparse matrices. Available online at https://docs.scipy.org/doc/scipy/reference/sparse.html, last accessed 19/01/2020

27. Steffen Rendle, 2010, IEEE International Conference on Data Mining (ICDM). Available online at https://ieeexplore.ieee.org/document/5694074, last accessed 17/02/2020