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

MSc Data Science

Project Report

eCommerce Recommender System For Increased Basket Value

Konstantin Orlovskiy, Student ID: 13157188

Planning

The work on the project started 21st June as planned. The timing and stages were well planned and the process was pipelined to count in the nature of working on machine learning project. On the course of working on the project having the clear plan helped to adjust and keep going when getting stuck with some minor issues. The plan was being reviewed on weekly basis.

Original 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												▨

Preprocessing stage took significantly longer time and was driven by multiple factors. Firstly, there was a need of making trade-off decision considering the computational capacity of used laptop in order to avoid too long run of each experiment and stick to the frequent iterations process with evaluation of results and revisiting the settings. Secondly, the used LightFM package has 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 to pass to the LightFM model. These package modules were designed to simplify the work but having unclear documentation required additional effort to make it work. The project report submission deadline was moved 2 weeks further by the university so the Gantt diagram was updated accordingly.

Actual Pace

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	Sep 14	Sep 21
Data preparation	🟩		🟩	🟩	🟩									
Hyperparameter tuning		🟩	🟩	🟩	🟩	🟩		🟩		🟩				
Model training		🟩	🟩	🟩	🟩	🟩		🟩		🟩				
Post-processing							🟩		🟩		🟩			
Model evaluation							🟩		🟩		🟩			
Report writing												🟩	🟩	
Report formatting														🟩
Submit report														🟩

Further background review

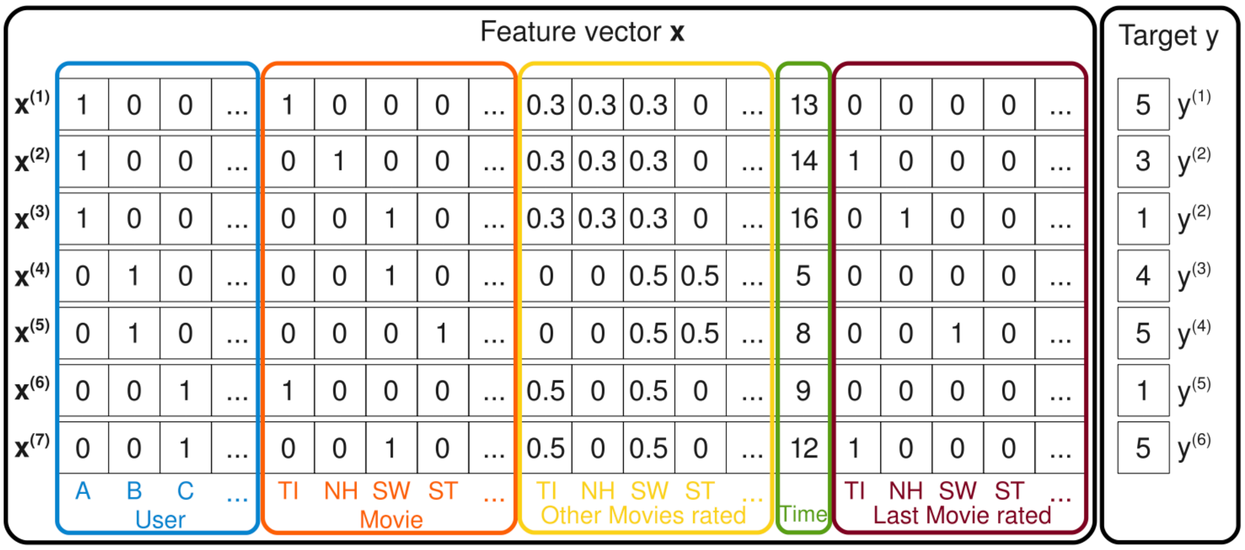
The LightFM package was chosen for recommender system development in this project. Reason for that is the aggregation by LightFM the benefits of two commonly used techniques for predictions based on implicit feedback: collaborative filtering and content-based filtering [https://arxiv.org/pdf/1507.08439.pdf]. As a result, the hybrid model performs not worse than pure collaborative filtering model. 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.

Collaborative filtering (CF) method is based on matrix factorisation (MF) of interactions described in the project proposal, when users and items get a latent representation which reveals hidden dependancies. This methodology requires the high density interactions data (low sparsity) to show acceptable performance. Cold start problem is the main downside since the technique will not have new item/user data to create their representation.

Content based (CB) perform better than CF on the sparse interactions data when CF is facing cold start problem so processing the user/item information is the more suitable way of generating relevant recommendations. The items metadata (item features) is normally uploaded when product is listed, this info is already available when new user joins.

The LightFM model is combining the CF and CB and balancing their input into the predictions based on the sparsity of the data [https://arxiv.org/pdf/1507.08439.pdf]. Three matrices factorisations are done to implement this combination: user/item, item/feature and user/feature. Interesting fact is that 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. This combination can be considered as an example of factorisation machines (FM), which were described in detail by Steffen Rendle in his paper [https://www.csie.ntu.edu.tw/~b97053/paper/Rendle2010FM.pdf].

FM is a powerful solution where the most of available interactions and metadata can be counted in for making the predictions on target. This is being achieved by creating a matrix with dedicated range of columns for specific type of information: users, items, timestamp, previous activity, etc. The resulting matrix represents the comprehensive view on the state of various variables that contribute to the recommendation.



	Feature vector **x**																				Target y	
	User				Movie					Other Movies rated					Time	Last Movie rated						
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...		TI	NH	SW	ST	...		
$\mathbf{x}^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5	$y^{(1)}$
$\mathbf{x}^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3	$y^{(2)}$
$\mathbf{x}^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1	$y^{(2)}$
$\mathbf{x}^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4	$y^{(3)}$
$\mathbf{x}^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5	$y^{(4)}$
$\mathbf{x}^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1	$y^{(5)}$
$\mathbf{x}^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5	$y^{(6)}$

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 certain period of time when item or user features may have looked differently (stock update, price change, new specification added, title updated, image updated, etc.). Additionally, the timestamp of interaction is important for fashion industry, for instance, so this is a good enhancement to factor in the trends. However, the preparation of the data and determining the noisy parts of it to be removed was expected to be too extensive when being limited by the project deadline so the choice was made in favour of LightFM package.

Implementation

Preprocessing

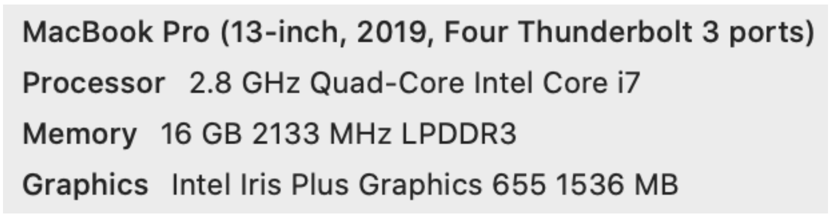
The Retailrocket data consists of three data sets: events, item properties and category tree. While first two are the important source of information, the category tree data is in some way replicated in item properties since the products have the category id information. For the further steps the category tree data will not be used but will be considered for a deeper exploration in future work.

Events data

The events dataset contains events types ‘view’, ‘addtocart’ and ‘transaction’. Since there’s a plan to use neural networks to process the data, these events types need to be converted from string values to numerical values: ‘view’ = 1, ‘addtocart’ = 2, ‘transaction’ = 3. Numbers still represent the categorical values but in format acceptable by neural networks. The transformed form of events cannot be considered as rating due to the nature of LightFM package and deep learning models overall. This was proved at the later stage of project when attempt of passing the weights matrix with 1/2/3 as well as 1/700/1000 instead of interactions matrix did not lead to any change of performance metrics. We’re able to pass weight of features to the model while the interactions themselves are either positive or negative so it is a binary problem. The task is to identify which event types are considered as positive and then pass to model. Originally the idea was to use event types as analogue to rating, however the events represent the implicit feedback so cannot be used as rating.

The user may have interacted with the item multiple times which is now stored in data frame. For current purpose of recommendation we're 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 will be kept for this item/user pair.

In order to test computational cost of working with the full range of interactions, the LightFM model was run on and the run time exceeded 6 hours. This result is not suitable for running experiments in the project considering the computational capacity of 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.

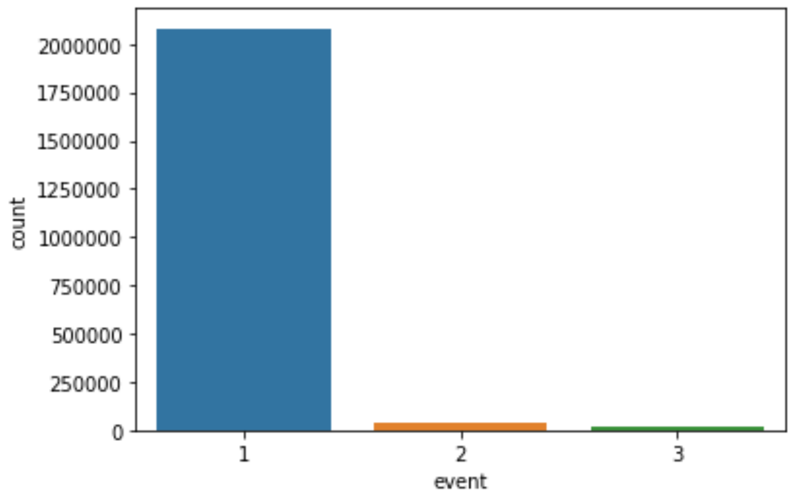

MacBook Pro (13-inch, 2019, Four Thunderbolt 3 ports)
Processor 2.8 GHz Quad-Core Intel Core i7
Memory 16 GB 2133 MHz LPDDR3
Graphics Intel Iris Plus Graphics 655 1536 MB

First experiment

Setting	Value
All users	All

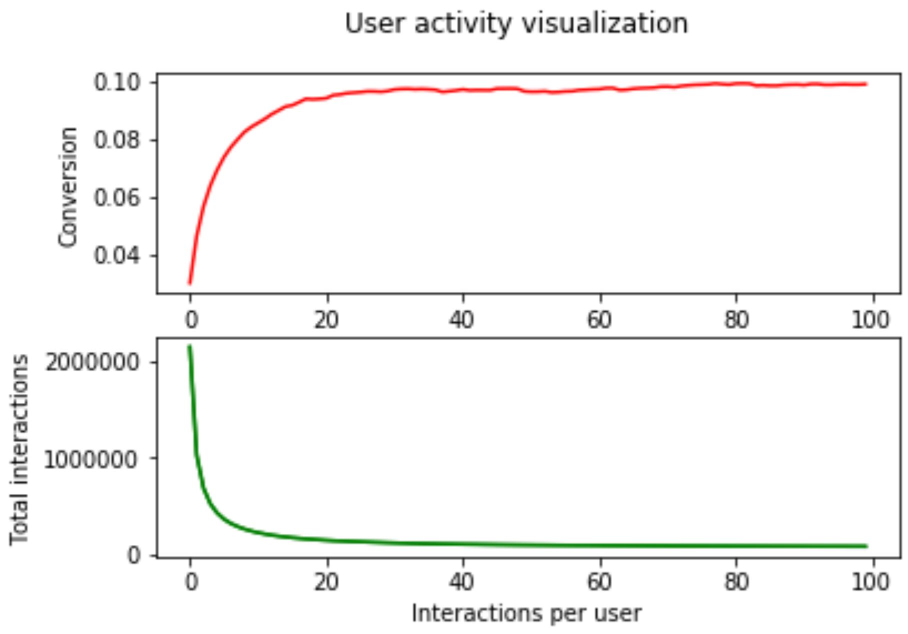
All items	All
Train/test split ratio	80 to 20
Same users/items in train/test	No effort
weight view / addtocart / transaction	1 / 2 / 3
no_components	10
learning_schedule (default ‘adagard’)	adagard
loss	warp
learning_rate (default ‘0.05’)	0.05
item_alpha (default ‘0.00’)	0.00
max_sampled (default ’10’)	10
num_epochs	10
item_features	None
Model trained (time, min)	0.57
auc_score train	0.95790213
auc_score train (time, min)	128.14
auc_score test	0.7760904
auc_score test (time, min)	36.25
Parameter “k”	10
precision_at_k train	0.007139213
precision_at_k train (time, min)	128.39
precision_at_k test	0.0009220791
precision_at_k test (time, min)	36.27
Total time to run, min	**329.62**

Further exploratory analysis of the dataset revealed the ratio between events 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%.



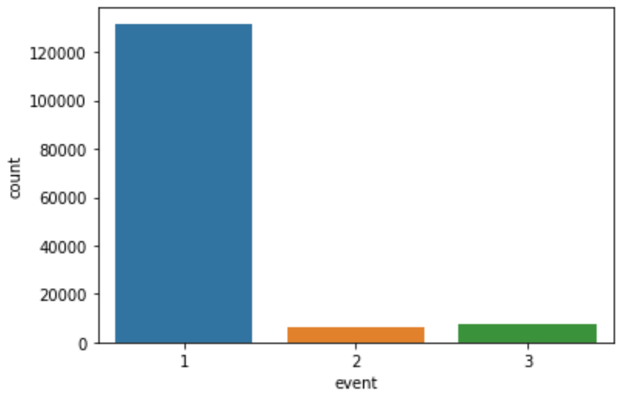
Add to cart event type technically is not transaction, however the fact that user interaction with item led to this step means the high interest / importance of this item to the user. So the assumption is that **add to cart event can be considered as positive interaction along with the transaction event type.** Ratio of addtocart 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 level 3.33%. This is close to ecommerce industry average [source]. The deeper look at the consumers generating various event types in the activity log reflects there're two major user types: '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 group of 'low activity' users is mostly browsing and not making many purchases. The ratio of positive interactions is low so it is harder to predict what they really like. These users create a 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. This will lead at least to improvement of customer experience and users are more likely to find the needed item. For the purposes of this project 'low activity' group will be left aside as the goal is basket value, not user experience (UX). This will allow to get rid of noise and decrease the size of the data frame which will save computational time which is crucial in production to let the recommender system provide near to real time recommendations.

The group of 'high activity' users has higher ratio of positive interactions (add to cart, transaction) meaning that the conversion is higher. For this group of users the view interactions may be not considered since the user has not proceeded to the purchase so this means there's low interest to the item. This approach will lead to focusing on what is really important to user and avoid making recommendations on the noise. For the purposes of 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 represent 0.07% of original. This 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. Cleaning process also improved the sparsity of interactions data to a level of 99.85% which is challenging for the collaboration filtering requiring sparsity less than 99.5% for better performance [link to source]. However, the fact of item features availability partially solves this problem. The further cleaning of data frame is not required. The used approach is more data driven than the one used in proposal when the activity thresholds was supposed to be selected empirically.

Having the time log of user interactions was originally planned to be used for splitting into train and test sets and leaving the latest interactions for test. However, this approach may impact the generalisation of the model considering the fact that user behaviour depends on the seasonality, various eCommerce events (Black Friday, Cyber Monday, Christmas sale, etc.). So the split was made randomly with ratio 80% for train and 20% for test. As an additional step, the test set was cleaned from users that are not present in train set. Rationale for this decision lies in the user cold start problem: there’s no user feature data available so the model will not be able to rank the products for user which did not have any interactions so the test results for this user in test set will create noise and lower the model performance. In the real life use these users will be recommended the most popular items (bestsellers). 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 certain products.

Item Properties Data

The information about item properties is stored in the form of historical log of changes. The model is unable to process this format so there's a need to trim the properties data. The latest properties data is considered to be the best to describe items.

Assumption: the ecommerce team was constantly improving the catalogue and adding more accurate item info. So the use of the latest values is reasonable. Additionally, the 'available' property was removed. It won't make sense to consider any value as fixed (in stock or not in stock) for training purposes. In production this property can be used in real time to filter out unavailable items from prediction.

As the next step the items which are not present in the cleaned data frame were also removed since it won’t be possible to evaluate the predictions on them in the isolated environment where there’re no interactions. Additionally this step significantly decreases the number of the data frame size from 353.6 MB to 7.8 MB. Cleaned item properties data frame represents 1.26% of original. This is important 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 csr matrix. This matrix must have mapping of all items related to the train/test process and all features/value pairs available in cleaned properties data frame. Then the matrix is to be filled in with the properties values. LightFM model would be then capable to create a latent features vector for each item. Two tables below visualise the format. The LightFM documentation did not have the clear explanation of the required format [https://making.lyst.com/lightfm/docs/lightfm.data.html#lightfm.data.Dataset.build_item_features] and it took more time than expected to figure out which format works. The code showing how the transformation piece was resolved follows the tables.

Common format

	Colour	Material
Brush	brown	
iPhone	black	metal

LightFM required format

	'Colour : brown'	'Colour : black'	'Material : metal'
Brush	1		
iPhone		1	1

```
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 which simplifies the process of preparation the matrices in required format (COO, CSR matrix). 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 the dataset class will have 3 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.

Dataset class method '.build_interactions( )' returns two COO matrices: one with interactions (binary), another with weights of these interactions which were set for view, add to cart and transaction before. The idea behind the LightFM implementation is to count in the implicit feedback as binary problem, with either positive or negative interactions (addtocart and purchase) and then rank the items for a particular user by the likelihood of having positive interaction. The weights matrix will not be needed for the purpose of 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 order to check the impact of using hybrid model, the training and test were run first without item features, and then including them.

Originally the two step approach to recommendations was planned.

The model has a range of hyperparameters which are not a subject for training and are selected by the data scientist. [https://making.lyst.com/lightfm/docs/lightfm.html#lightfm.LightFM] There are various techniques of improving the model performance by hyperparameters tuning. The default values are shown below and not all of them were used for a start.

Firstly, the logistic loss suits when there are both positive and negative interactions available. The Retailrocket data set contains implicit feedback so no ratings provided. The fact of return of the item could be considered as negative interaction but this data is not present in the events data set. BPR (Bayesian Personalised Ranking) and Weighted Approximate-Rank Pairwise (WARP) loss work for the positive only 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 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. [https://making.lyst.com/lightfm/docs/lightfm.html#lightfm.LightFM]

Secondly, considering the size of data passed to the model and computational capacity, the number of components (neurones) was increased to 100. Normally higher number of components leads to improvement of performance with computational cost growth as tradeoff.

```
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}
```

As test metrics are used AUC score and precision at 'k'. Both of them are measuring the ranking quality, 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. [https://making.lyst.com/lightfm/docs/lightfm.evaluation.html].

Set parameter 'k' value to check precision at 'k' top rated items. Initially select k=3 as it will allow to have 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

User won't be offered set of 2 for each item (like A + A) as this is simpler task and can be implemented across the whole catalogue on all pages if there is a business need for that. This will lead to better user experience and avoid recommending too much which can cause FOMO (fear of missing out) [source to the article] when the user gets stuck in front of too wide range of choices.

The project proposal also included the cosine matrix based item to item recommendation as well as frequently bought together items but this is already implemented into the lightFM model and does not need to be done separately. [source]

Model training

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 is being done over train data. The more cycles the better is training but downside of too high number of epochs is that the model may start remembering the train data rather than learning the patterns which leads to decrease of model generalisation. As a starting point 100 epochs is chosen.

- 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. This is helpful to see the results in order to revisit the parameters or get back to the preprocessing stage if the results do not suffice.

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

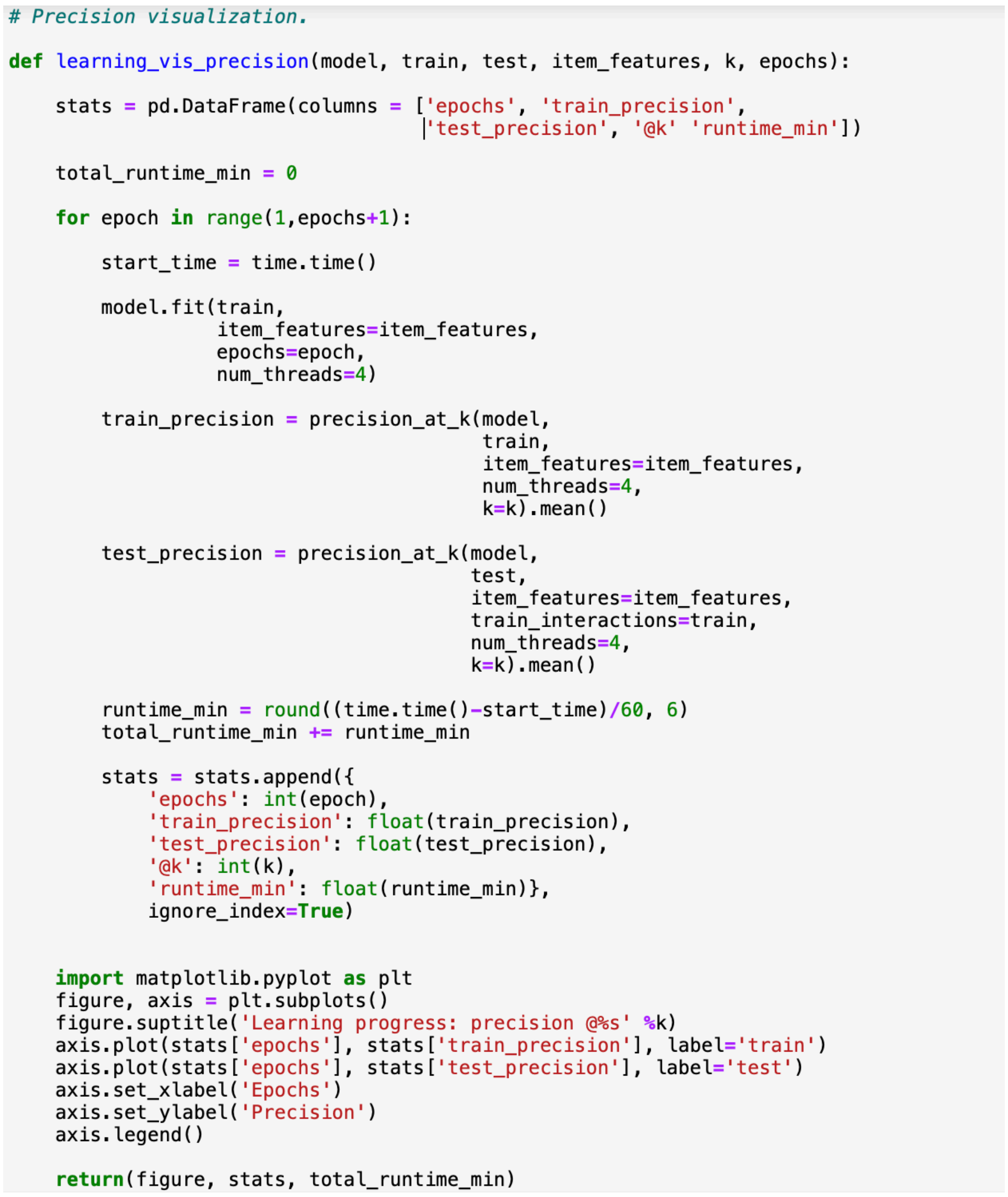
AUC score - this metric is normally over 0.95 on train set as the model learns well on the train data. Having high train AUC is not determining how well the model is generalised as it may overfit the train data and show low performance on the test set. Test AUC on the collaborative filtering is 0.65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

Precision at 3 - there are no benchmarks on the precision at k fore recommendation systems as it strongly depends on the nature of the data. The below results show that collaborative model shows 0.78 score on 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 important to avoid overfitting the data with no improvement on 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, the intermediate metrics of the model were iteratively stored in the data frame after each epoch. The below functions was developed to automate the process and plot the results.

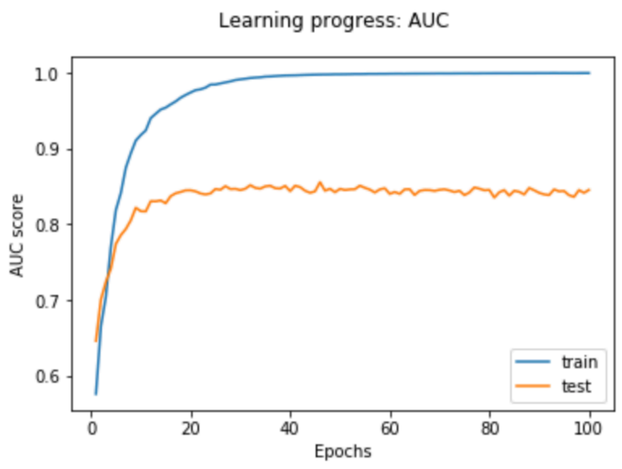
```python
# AUC visualization function.

def learning_vis_auc(model, train, test, item_features, epochs):

    stats = pd.DataFrame(columns = ['epochs', 'train_auc', 'test_auc', 'runtime_min'])

    total_runtime_min = 0

    for epoch in range(1,epochs+1):

        start_time = time.time()

        model.fit(train,
                  item_features=item_features,
                  epochs=epoch,
                  num_threads=4)

        train_auc = auc_score(model,
                              train,
                              item_features=item_features,
                              num_threads=4).mean()

        test_auc = auc_score(model,
                             test,
                             item_features=item_features,
                             train_interactions = train,
                             num_threads=4).mean()

        runtime_min = round((time.time()-start_time)/60, 6)
        total_runtime_min += runtime_min

        stats = stats.append({
            'epochs': int(epoch),
            'train_auc': float(train_auc),
            'test_auc': float(test_auc),
            'runtime_min': float(runtime_min)},
            ignore_index=True)


    import matplotlib.pyplot as plt
    figure, axis = plt.subplots()
    figure.suptitle('Learning progress: AUC')
    axis.plot(stats['epochs'], stats['train_auc'], label='train')
    axis.plot(stats['epochs'], stats['test_auc'], label='test')
    axis.set_xlabel('Epochs')
    axis.set_ylabel('AUC score')
    axis.legend()

    return(figure, stats, total_runtime_min)
```

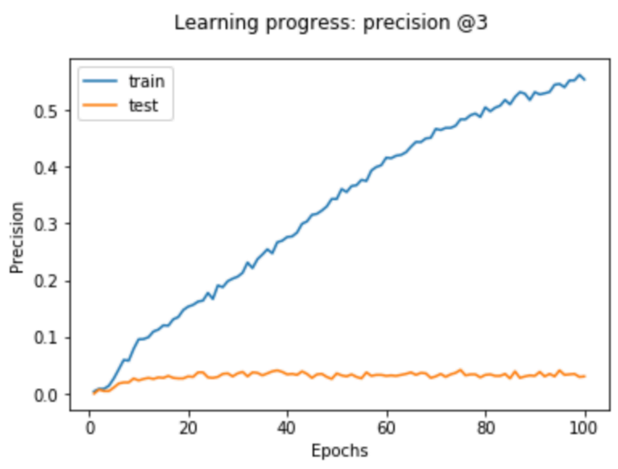


```python
# Precision visualization.

def learning_vis_precision(model, train, test, item_features, k, epochs):

    stats = pd.DataFrame(columns = ['epochs', 'train_precision',
                                    'test_precision', '@k' 'runtime_min'])

    total_runtime_min = 0

    for epoch in range(1,epochs+1):

        start_time = time.time()

        model.fit(train,
                  item_features=item_features,
                  epochs=epoch,
                  num_threads=4)

        train_precision = precision_at_k(model,
                                         train,
                                         item_features=item_features,
                                         num_threads=4,
                                         k=k).mean()

        test_precision = precision_at_k(model,
                                        test,
                                        item_features=item_features,
                                        train_interactions=train,
                                        num_threads=4,
                                        k=k).mean()

        runtime_min = round((time.time()-start_time)/60, 6)
        total_runtime_min += runtime_min

        stats = stats.append({
            'epochs': int(epoch),
            'train_precision': float(train_precision),
            'test_precision': float(test_precision),
            '@k': int(k),
            'runtime_min': float(runtime_min)},
            ignore_index=True)


    import matplotlib.pyplot as plt
    figure, axis = plt.subplots()
    figure.suptitle('Learning progress: precision @%s' %k)
    axis.plot(stats['epochs'], stats['train_precision'], label='train')
    axis.plot(stats['epochs'], stats['test_precision'], label='test')
    axis.set_xlabel('Epochs')
    axis.set_ylabel('Precision')
    axis.legend()

    return(figure, stats, total_runtime_min)
```

The analysis of resulting plots revealed that the AUC improvement slows down significantly after 30 epochs. The train precision score is constantly 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.

Behaviour of precision at k parameter depending on number of epochs was expected: the more epochs pass the more model starts learning train set (memorising the data) meaning that model's generalisation decreases and there's 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 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. There's no right or wrong setup as it depends on the data used so there're several techniques to define the combination that leads to model generalisation and performance improvement.

One of the techniques is random search [source to the paper]. The idea is for each optimised hyperparameter to define the array of the considerable values and then randomly sample various combinations of hyperparameter values and select the best performing one. This technique has proved to show good results with less computational cost in comparison to grid search technique when the hyperparameters samples are extensively run across the full range of values in the defined arrays. The array of values for the hyperparameters was generated by custom function:

```
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. Original algorythm was posted on GitHub 23 Apr 2018 [https://gist.github.com/maciejkula/29aaf2db2efee5775a7f14dc387f0c0f]. For the purposes of this project the original code was modified to meet the needs. 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 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 test set. This approach allows to keep 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.

Model Evaluation

In production the model can be evaluated on the recommendations conversion rate and basket value. The big difference is that in production the users will be given the recommendations and will possibly interact with them. Since this project is done in isolated environment, where the test set of users isn't actually getting recommendations, the best possible way to evaluate the model is to see the AUC, precision and recall metrics.

During the experiments the focus was made on two metrics: AUC and precision at k. AUC is important to understand how well overall the array of items was ranked for the particular user. Precision shows how many actual positives are in top k ranked items. 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 metrics 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. [source].

Hyperparameter tuning has not led to improvement of both metrics. The extensive search over the array of hyperparameters was done with focus on improvement of 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 normally select default values for hyperparameters based on the empirically proved best performance which has been again proved with current data. The original model with a mix of manually selected and default hyperparameters is proved to be the best and is selected to proceed further.

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

Recommendations

The recommendation process 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 3 products (refers to parameter k which was used to evaluate precision) and returning the recommendation of 4 bundles. Recommendations algorithm was developed based on the one the one used by the author of LightFM package, which is available here [https://making.lyst.com/lightfm/docs/quickstart.html]. The two phases are implemented in two separate functions which gives 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.

Ranking the products

First phase algorithm generating the list of top N products sorted from top to less interesting. In production N can be imperially selected considering the fact that some items from this list can be out of stock or unavailable for sale for any other reason. The functionality allows to run recommendation for the list of users in one go. This may be needed when the email campaigns are prepared so the task can be done in one go. Function returns the dictionary which has the 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 user had positive interactions. Since the item features names and values are hashed, there's no proof whether the items can be considered for the repetitive purchase or not. If the item is refillable (paper towels, toothpaste) it can be the case they may be recommended one more time, while the same sofa or carpet are unlikely to be recommended second time. If the information is provided, the pool of potentially recommended items may be reviewed for better UX. As an example, the function was run on a range of 3 users over train set.

```
# 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]}
```

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.

Product bundles

The second phase of recommendation is creating the bundles of items recommended in phase one. Since the item prices are hashed in original data set and total number of bundles is 4 all bundles can be recommended to user with no negative impact on user experience. This 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 function which returns the similar dictionary containing the list of bundles respecting the bundle size set manually. Additionally, the bundler adds the fill list items to output. This functional allows to keep the view on all recommended items and it can be easily excluded if needed as comes as the very first item in the user recommendation list of bundles. As the next step the price per bundle needs to be calculated and discount applied. Since the prices are hashed in the item properties data, this step is skipped.

```
# 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 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% was aimed to be offered. However, since the item prices are hashed, the step of applying the discount has to be left aside this project

The number of recommended bundles isn't high (only 4), so the user may be recommended all 4 bundles on the same page. This will lead to improved user experience as all bundles are offered at once and the user is able to choose.

Production and Future Work

The developed solution is ready for implementation in production. Prediction process is staged and allows the eCommerce website team to benefit from the intermediate results as well as adjust the output according to the needs. The first stage functional - '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.

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 number of items per bundle. Having the bundle size setting functional allows to adjust 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 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 recommendation list. This 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 purchase.

Both stages output format is dictionary and corresponds the broadly used in modern data systems JSON format. This simplifies the implementation of the recommender system into existing data systems and removes complexity and extensive data transformation.

DIAGRAM OF 2 STAGE PREDICTION PROCESS.

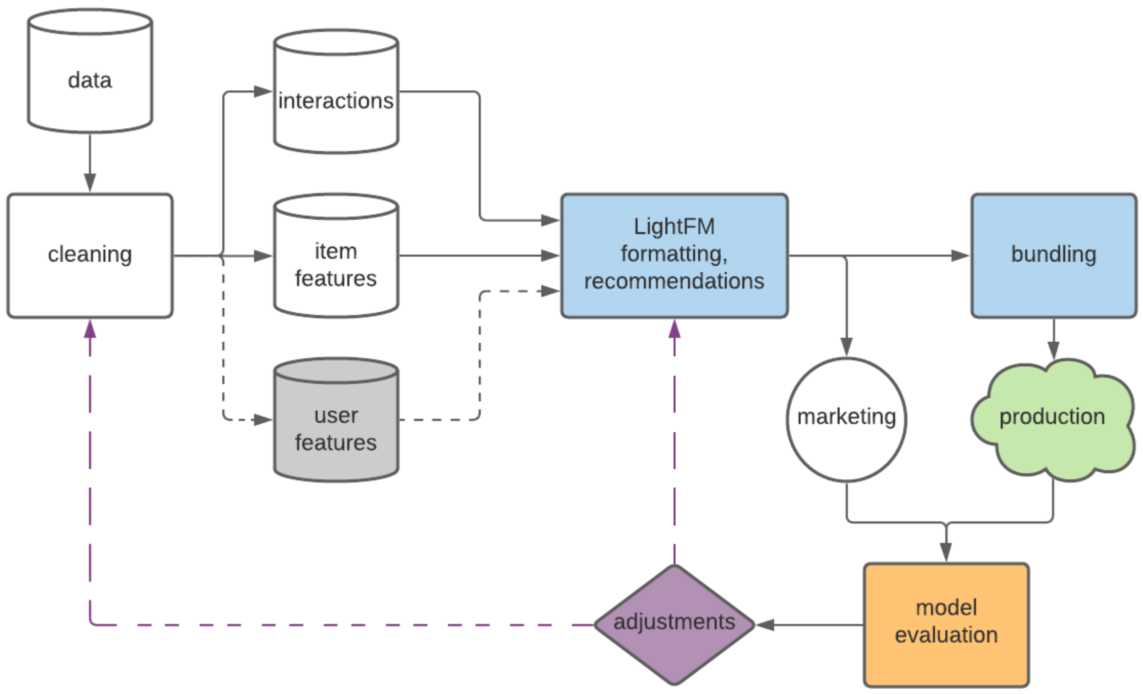
Before making recommendations for production, the selected model should be trained on the whole cleaned dataset (without train/test split) to count in all the meaningful interactions in the model. Also, all items from item features data should be added to the Dataset class variable. For academic purposes item features data was originally preprocessed to remove noise from the test results. In real life recommendations all of this data matters and should be used for predictions and the results can be evaluated based on the 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 compliment 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 eCommerce team and in can be potentially used for the predictions improvement by passing in to the model in CSR matrix format and therefore letting the model to created the latent representation of user features. 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 number of individual user interactions in comparison to the selected threshold (driven by conversion rate) may move user to 'high activity' group and let the user get relevant recommendations. 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 corresponding cell in CSR matrix needs to be cleaned which is not implemented in this project - therefore there's a need of reprocessing.

The developed technical solution can now be implemented into the website UI or considered for email campaigns, etc. In production it is important to measure the success of recommendations. Hung-Hsuan Chen et al. described four common pitfalls of evaluating recommender systems [https://www.kdd.org/exploration_files/19-1-Article3.pdf]. The team of researchers flagged the issues and run the experiments to demonstrate them.

Diagram describing the architecture of the solution is below. The user features element of the diagram is implemented in the solution and can be used when the user data becomes available. The results of user interaction with recommendations are evaluated at the final stage of the cycle and as the actionable the required adjustments can be made for the cleaning (activity thresholds, positive interaction criteria, etc) and recommendation phases. Iterative approach is beneficial for keeping the solution improving and therefore leading to both basket value growth and increased user experience.



This project was done in isolated environment without providing actual recommendations to users so the measure of recommendations accuracy was based on two metrics: AUC and precision at 3. When recommending the items to visitors in production, the first issue raises when the recommended products start getting higher visibility so are easily reachable by visitor which makes their choice biased.

Second issue lies in the fact that the distribution of test data is being impacted by the recommender system. When the model is additionally trained over time on these recommendation interactions, it makes the model being biased as well. The test data should be carefully selected before evaluating the results of recommendations.

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 is definitely correlated with the user experience helping visitors in reaching the relevant products faster with less effort. In the proposal to this project the goal was clearly set to 'increased basket value' and CTR was not supposed to be measured. Moreover, it would not be possible in isolated environment.

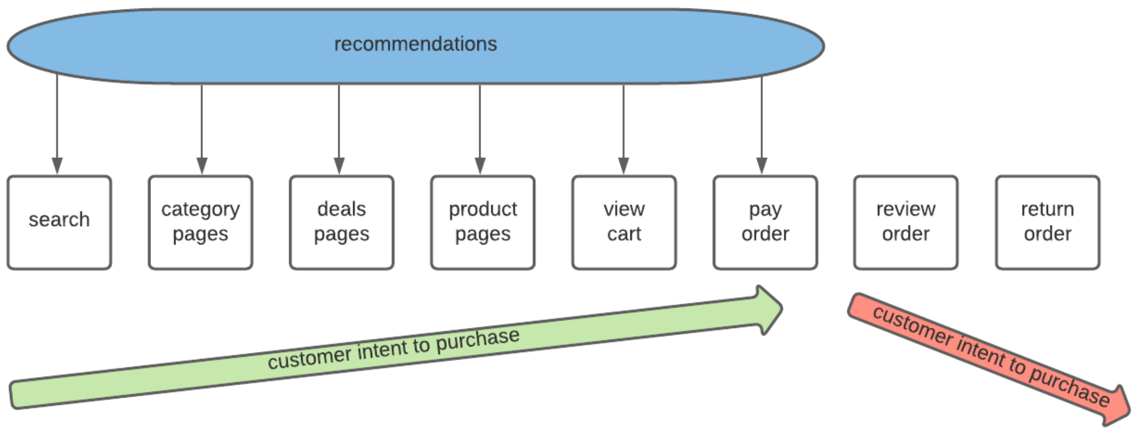
Fourth issue described in the paper is related to the incremental revenue growth led by recommender system. While the visitor may find the recommended product attractive and convert into sale, the same product may have been found by visitor without recommendation, simply using the website search engine. There is a concept of 'impulsive purchasing' when consumer behaviour is determined by a certain stimulus. It is described in detail by Regina Virvilaitė et al. in their paper [https://etalpykla.lituanistikadb.lt/object/LT-LDB-0001:J.04~2011~1367177966372/J.04~2011~1367177966372.pdf]. Based on this concept, the websites having certain products portfolio (gifts, kitchen accessories, etc.) have high probability of benefitting from recommender system implemented in the user interface.

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 user may not consider buying without this offer. So, counting in the fourth issue described above, the eCommerce website team may decide to limiting the pool of products in recommendations but those moving slowly and stimulate the purchase. Theoretically, this approach may be able to bring incremental revenue growth.

A/B testing would be helpful in evaluating the recommender system added value when comparing the revenue generated by two focus groups of similar users when first group is not given recommendations while another is offered the bundles of products with discount. This process is medium/long term and is related to the buyer 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. Hence, the scale of incremental revenue growth needs to be constantly re-evaluated.

There's a potential room for improvement of recommendations if the users/items interactions log will be limited by a certain period of time. 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 important to 'refresh' the latent representation of users and items by limiting the historical depth of data used. The same approach and pipeline fit this purpose however the decision on the interactions threshold may be reconsidered as well as hyperparameters used, so eventually it comes to tuning of the existing solution.

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. Proposed solution is not the stage specific while can be easily adjusted to meet a specific interest, for instance limiting the pool of items by tag 'sale' for the recommendations made on the deals pages.


recommendations
search
category pages
deals pages
product pages
view cart
pay order
review order
return order
customer intent to purchase
customer intent to purchase