RecSys Challenge: A two-step approach

Alexis Forest (01948163), Harry Walker (01867526), William Flynn (01997714), Coraline Duval (01930178)

ABSTRACT

In this report, we detail our approach to the RecSys challenge. We discuss some of the innovations of our work in terms of feature engineering, as well as the two-step approach employed for prediction. Overall, we settle for an optimized XGBoost model to predict session buy events and a kNN collaborative-filtering approach to predict what items will be purchased (conditional on an buy event). Finally, we discuss model performance and what we would have done differently in the absence of the computational limitations faced.

1. CHALLENGE DESCRIPTION

In this assignment we were provided with a sequence of click events performed by users on an ecommerce website. We were tasked with two problems: identifying whether a session will end in a purchase and, if so, what item(s) will be purchased.

1.1. Dataset

The dataset was split between **a click dataset** containing click history information for each session such as their date, the item on which they clicked etc., and **a buy dataset** which summarized all the sessions which resulted in a purchase with their corresponding id, the purchased item, the quantity bought and price. Thus, *a priori*, there were few features which could actually be used for prediction. Additionally, the dataset was relatively unbalanced, with only 5.5% of sessions ending with a buy event and 3.5% of clicked items being purchased.

1.2. Evaluation Metrics

The evaluation measure for this challenge takes into account the ability to predict both aspects – buy event and items purchased. It was defined as follows for each session in the test set:

$$\text{Score} = \sum \begin{cases} if\ \boldsymbol{s} \in Sb \ \rightarrow \frac{|Sb|}{|S|} + \frac{A_S \cap B_S}{A_S \cup B_S} \\ else \ \rightarrow -\frac{|Sb|}{|S|} \end{cases}$$

Where ***s**:* session in the test set, **Sb**: sessions in test set which end with buy, **S**: all sessions in the test set, $\mathbf{A_s}$: predicted bought items in session s, $\mathbf{B_s}$: actual bought items in session ***s***. Thus, there was a clear emphasis on evaluating the two stages of this process.

2. METHOD OVERVIEW

2.1. Approach

We segmented our analysis into two stages. First, we predicted whether a session would contain a buy event based on its click information, that is, we created a binary classifier.

Second, we selected only the sessions that resulted in a buy, and calculated the probability of purchase for each item in a given session *j*. Further, we applied an optimised threshold to ultimately predict the purchased items. This latter classifier was implemented via a collaborative filtering approach based on the nearest neighbours.

2.2. Feature creation

A major emphasis was placed on feature engineering given the lack of diverse or granular information provided *a priori.* We built on insights from the winning papers, and created multiple new attributes to enrich our dataset and ultimately, our analysis. For example, Romov and Sokolov (2015) noticed that sessions with a higher number of items clicked on had a higher probability of purchase. Volkovs (2015) highlighted that users who performed targeted browsing (clicking on items within the same category) were also more likely to buy. To illustrate, we noticed that there was a correlation between users browsing a popular item (top quartile of sales) and a session ending in buy.

Furthermore, as there were more than 50k different items and more than 300 item categories, we

quickly had a lot of categorical variables. We chose to aggregate some items and categories into the most important one to reduce the dimension of the dataset and avoid hundreds of dummy variables.

Session-specific features (classifier 1)

Regarding the first classifier, and conscious of computational runtime limitations, we settled on 9 key features as listed below. It is worth mentioning that the features used for the first classifier are strictly "global" features about each session at an aggregate level. The session features are as follows, with all time measurements for clicks measured in seconds:

1. **total_time**: total amount of time that a user spent on each session (sum of time spent on each item in the session)
2. **median_time_clic**: median amount of time spent between clicks during each session
3. **nb_clic**: number of clicks in the session
4. **nb_2clic**: number of items with 2 clicks
5. **nb_3clic**: number of items with >= 3 clicks
6. **unique_item_count**: number of unique items in the session
7. **max_time_on_item**: the maximum amount spent on an item
8. **month**: corresponding month in which the session was held
9. **top_item_count**: number of items in the session which fall into the top 25% of the most bought items.

Session and item-specific features (classifier 2)

For the second classifier, as we want to predict the probability of buying each item of a session, a focus on the item characteristic must be undertaken. Thus, we expanded on the session-specific features above and also included item-specific features as well.

Session-specific features

1. **categories**: columns "0" to "S" indicates how many items in the session belonged to each category. Only categories 0 to 12 and 'S' are chosen as they are the most popular.
2. **item_list**: a list of all the items in the session

Item-specific features

1. **category**: category of the item
2. **ratio_time**: amount of time spent on item compared to total session time
3. **last_item**: id of last item that appears in the session

3. 1st CLASSIFIER – MARKOV AND XGBOOST

In order to train machine learning models, we split the data into three sets: training (60%), validation (20%) and testing (20%). We included a validation set to enable hyperparameter tuning with early-stopping rounds and avoid the computational costs associated with cross-validation. This split was done on the dataset sorted by ascending timestamp (i.e., we used sessions early on in the year to predict the outcome of future sessions).

We approached the first binary classification problem (1 = buy, 0 = otherwise) with two very different models. Firstly, we tried to model customer's journeys using the probabilistic Markov chain model (see Jupyter Notebook Appendix). However, the model performed poorly to predict a purchase. Markov chain assumes a memoryless property that does not hold within our problem framework. Indeed, by not taking into account prior clicks (for example, if a user clicked several times on the same item), we lose valuable information. Moreover, in order to have a model small enough to train, we could only consider an item's category, and not the items themselves, missing out on granularity which would likely lead to higher accuracy otherwise.

Our final model was an optimised XGBoost Classifier algorithm with parameters (*n_estimators*: 174, *learning_rate*: 0.05, *scale_pos_weight*: 4). The *scale_pos_weight*

parameter allows us to account for the unbalanced nature of the dataset, weighing the minority class according to the ratio of majority class to minority class observations and taking its square root. Finally, we combined a high initial *n_estimators* (2000) with a low *learning_rate* and early-stopping on validation to obtain a more accurate model while simultaneously preventing overfitting.

4. 2nd CLASSIFIER – K-NN

In our k-NN model, we used the same training set as for the XGBoost, but only considering sessions that led to a purchase, and for the testing set, we used the sessions we predicted a purchase would be made on with the XGBoost model testing set. We wanted to use the validation set to find the optimal K but did not manage to in the imparted time.

We first attempted to use our own custom metric for the kNN regressor that would make use of all of our features. This custom distance would be composed of:

- For numerical features as a 1-norm distance;
- For categorical or Boolean features as 1 if they were equal, 0 otherwise;
- For the list of items clicked as the Jaccard distance between the two sets of items to get the similarity between two sets of clicks.

We also added weights to each distance's feature to be able to give more or less importance to a feature over another. By default, we first tried it with equal weights.

Unfortunately, we quickly realised that scikit's k-NN regressor only accepted numerical features and thus had to drop the **category**, **last_item** and **item_list** features. Then, we tried running the custom distance with the k-NN regressor, but after a run-time of 20 hours we opted to use the default Euclidean distance with K=10. We used a threshold of 0.5 on the probability of the prediction from the k-NN regressor to decide if an item was bought or not.

5. RESULTS AND DISCUSSION

Overall, for the first classifier we obtained an accuracy of approximately 93%, sacrificing some precision to obtain a slightly higher recall through optimal weighting tuning. Note, that this performance is not particularly good given the unbalanced nature of the dataset. In fact, an accuracy of approximately 94.5% could be achieved simply by predicting no purchase for all sessions. However, we acknowledged the need to weight the positive classes more, especially since these would be used later in the second classifier.

Combining the results from the second classifier we obtained a score, as defined in section 1.2., of approximately 167. In comparison to some of the winning papers this is rather underwhelming. However, it is worth comparing this to the baseline achieved by the models without hyperparameter tuning (-12252). Nevertheless, there are multiple avenues we would have explored in further detail had we not run into computational running time issues due to the magnitude of the dataset. In particular we would highlight:

- Combining the sample weighting in the XGBoost with threshold optimisation and cross-validation. Currently, the threshold is at default (0.5), and more accurate predictions could be achieved by tweaking this parameter. Moreover, cross-validation, while computationally intensive, would allow for a more precise learning stage.
- Implementation of our custom distance metric. The function defined in section 4. is custom-built to work accurately on this specific dataset. Unfortunately, the runtime was far from feasible despite feature engineering, but we would expect a greatly improved performance relative to the default distance in k-NN.

In conclusion, we have discussed our approach to the RecSys challenge. We exploreed some of the innovations of our work in terms of feature engineering, our two-step approach employed for machine learning, and finally the advantages and shortcomings of our models.

REFERENCE LIST

Romov, P. and Sokolov, E., 2015. Recsys challenge 2015: ensemble learning with categorical features [Online]. *Proceedings of the 2015 International ACM Recommender Systems Challenge* (pp. 1-4). Accessed 5 May 2021. Available from: https://dl.acm.org/

Volkovs, M., 2015. Two-stage approach to item recommendation from user sessions [Online]. *Proceedings of the 2015 International ACM Recommender Systems Challenge* (pp. 1-4). Accessed 5 May 2021. Available from: https://dl.acm.org/