COMP9417 Group Project

William Feng (z5309999)

Asha Raghav (z5363204)

Prayag Rawat (z5312819)

Humans Learning Deeply

Table of Contents

The GitHub repository can be found here: https://github.com/William-Feng/recsys-dataset.

Introduction

In the ever-evolving landscape of e-commerce, where online shopping has steadily grown from a novelty to a daily essential, the customer experience has become pivotal in increasing sales for retailers. The wealth of options for customers to explore may be overwhelming and such sensory overload often leads to abandoned shopping carts, leaving both the eager shopper and the hopeful retailer in a state of dissatisfaction. Online retailers consequently depend on recommender systems, which utilise consumer data to predict and present relevant products pertaining to customer interests during the shopping experience to increase the likelihood of an order.

As OTTO is the largest German online shop, in this report we employ machine learning techniques to build the OTTO – Multi-Objective Recommender System on Kaggle, which was held until January 2023 and had a competition prize pool of $30,000 [1]. Based on real-time behaviour, the proposed system selects and recommends relevant items to customers from OTTO's vast range of over 10 million products to improve the customer experience and ensure seamless navigation for shoppers.

The goal of this competition was to predict the next item a user would click, cart or order based on their existing history . Past session data was collected through "anonymised behaviour logs of the OTTO webshop and the app", consisting of the article id (aid) values, event type, and Unix timestamp of the event for each unique session id [2]. Please see our GitHub repository for a closer look into the starter files, the models we created and all the relevant documentation.

Implementation

Exploratory Data Analysis

In data science, Exploratory Data Analysis (EDA) is a method to analyse and investigate datasets to discover patterns, spot anomalies, test hypotheses and check assumptions before applying any machine learning models to the data.

Even from the provided dataset (see the the appendix for the sourced table of statistics), we can immediately see that there are significantly less orders (5,098,951) compared to clicks (194,720,954) and carts (16,896,191), and so our initial assumption was that 'orders' is the most important in portraying a user. This hypothesis was soon confirmed with our EDA and made sense since a user may click through products out of curiosity and add items to their wishlist or cart out of impulsiveness, but actually having an item as a component of a certain order means that an individual really did like it, having a higher significance in our recommendation system.

Specifically, after loading the dataset, we performed analysis to identify the best-sold items and user actions regarding the clicks, carts and orders, producing recommendations for each based on the

[1] Otto – multi-objective Recommender System (2023).
[2] Otto-De, Otto-DE/Recsys-dataset (2023).

best-selling article items. Then, we generated predictions based on the testing jsonl data and created a submission file in a specified format, where each of the session types have up to 20 labels for the article ids that it predicts.

We assumed that the users who have recently clicked on a certain item would be most likely to purchase it in the future (as they were perhaps thinking about it or simply got distracted by another product that caught their attention). As a result, we added the time elapsed as part of our feature engineering (sorted such that the most recently viewed items have more significance), and running the cross-validation on the results confirmed our previous assumption. We then continued testing different aspects, such as prioritising items that have previously been in their cart to exist in their final order to take precedence over the most popular items (which were appended to the predictions list to constitute the 20 article ids, as is explained in the model evaluation later).

During our EDA, we made several observations and gained valuable insights into the dataset, which influenced our feature engineering and prediction strategies. Some of the key findings are as follows:

1. **Recent Interactions Influence Future Behaviour:** We observed that users who had recently clicked on specific items were more likely to purchase those items in the future. This led us to prioritise recently viewed items in our predictions, acknowledging their higher likelihood of conversion.
2. **Preference for Popular Items:** Our analysis revealed that certain items consistently appeared as popular choices among users. We considered this popularity factor when generating recommendations, making sure to include these popular items in the final prediction list.
3. **Time-Dependent Behaviour:** Users exhibited time-dependent behaviour, where recent interactions carried more significance than older ones. As a result, we incorporated time elapsed as a key feature in our feature engineering, ensuring that more recent interactions had a higher impact on predictions.
4. **Cart-to-Order Influence:** According to the Baymard Institute, 69% of all e-commerce visitors abandon their shopping cart [3]. Therefore, we gave higher priority to orders placed by users, as they are a better indicator of user preference, and this assumption is later confirmed.
5. **Personalised Recommendations:** By understanding user preferences through their historical interactions, we were able to create personalised recommendations. This approach allowed us to cater to individual user behaviour, leading to more relevant and accurate predictions.
6. **Diverse Behaviour Across Types:** We noticed slight variations in user behaviour across different types of interactions (clicks, carts, orders). Understanding these distinctions helped us tailor our predictions based on the specific user action, frequency and duration.
7. **Effect of User Demographics:** Although not explicitly included in the dataset, we speculated that user demographics, location, and device type could play a significant role in their preferences. Collecting such data could further enhance the personalisation of recommendations.

When the refined set of predictions from the EDA was uploaded to the Kaggle submissions, we obtained a score of 0.38506 as shown in the appendix, and despite this being slightly lower than the

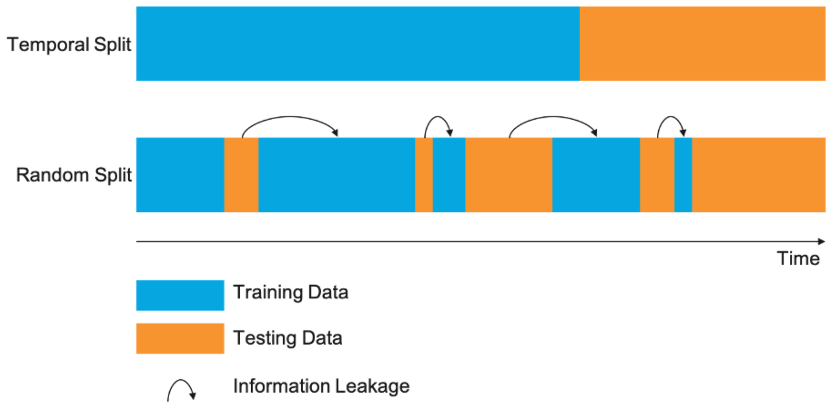
[3] Baymard Institute, Cart & Checkout Usability Research (2023).

results obtained by the machine models later, it still proved useful and broadened our understanding of the provided task. These findings also guided our decision-making process and influenced the creation of our ensemble models, including the choice of features and the ranking of predictions.

Model Training

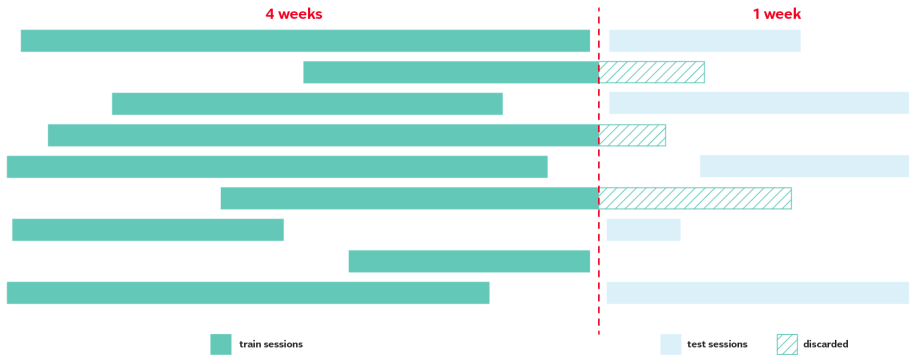
The provided train and test data was created on the basis of a time-based validation split. The training set contained four weeks of observations, whilst the test set consisted of user sessions from the subsequent week.

Information leakage tends to occur when information that should not be present in the training data prevents the model's learning ability which results in unsatisfactory prediction performance and may lead to unfavourable generalisations [4]. One particular type of leakage we wanted to avoid is implicit leakage - when the dataset is split randomly, leading to a portion of the training data to occur after the test data, as shown in Figure 1. If this information is available in the training set, then it may include future information which the model should not have.



[Figure 1. Information leakage]

This situation was mitigated, however, as the train sessions which overlapped with the test period were discarded in order to prevent influence on recommendations, as highlighted in Figure 2 below.



[Figure 2. Dataset split avoids information leakage]

[4] C3.ai, What is Information Leakage? (2023).

Model Evaluation

The evaluation metric for this Kaggle competition is 'Recall@20' (an evaluation metric for information retrieval) for each action type (clicks, carts, orders), with all three values weight averaged according to the equations below:

$$score \ = \ 0.10\, R_{clicks} \ + \ 0.3\, R_{carts} \ + \ 0.6\, R_{orders}$$

where

$$R_{type} = \frac{\sum_{i=1}^{N} \left|\{\text{predicted aids}\}_{i,type} \cap \{\text{ground truth aids}\}_{i,type}\right|}{\sum_{i=1}^{N} \min\left(20, \left|\{\text{ground truth aids}\}_{i,type}\right|\right)}$$

Thus, Recall@20 measures the ability of a model to recommend relevant items to users for each action type, with the overall assessment characterised by a weighted average of these recalls [5]. The ground truth value plays an important role in this metric as it provides the actual interactions that occurred during a session.

Note that it is trivial to achieve 100% recall - for example, by returning all article ids for each user. For this reason, we are restricted to returning at most 20 labels for each user. This metric ranges between 0 and 1, with a value of 0 indicating that the model did not identify any of the relevant labels, and thus performs poorly, whilst a value of 1 means the model identifies all labels perfectly.

As a reference, 2000 teams scored a recall above 0.5 in the Kaggle competition, with first place achieving a private leaderboard score of 0.60503. Note that after running a few experiments on the data, we found that the local cross validation (CV) does in fact track the Kaggle scores with a strong positive correlation, which thereby suggests that a higher CV score should indeed mean a better ranking on the leaderboard.

Model Selection

After some initial exploratory data analysis, we chose to use a variety of different models for the recommendation system, including the use of a two stage Re-Ranker model which utilised co-visitation matrices for the first stage, combined with a second custom re-ranking stage. This model was chosen due to the ability of co-visitation matrices to efficiently capture article id co-occurrence patterns in user interactions, whilst also significantly reducing the candidate pool.

A disadvantage of the Re-Ranker model however is that it requires re-calculation/updating of the co-visitation matrices each time new data is added, which can be computationally expensive and time-consuming. This may be a significant disadvantage for recommendation systems which require near real-time recommendations, for example for shopping websites such as OTTO where capturing current trends may be vital in generating sales. Therefore, we opted to use other models, including

[5] Powers, Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation (2007).

LightGBM Ranker and XGBoost which are both gradient boosting algorithms known for their effectiveness in learning complex patterns and performing well on large datasets [6]. These models were selected for their ability to handle non-linear relationships between the predictors and the target variable. Their gradient boosting nature allows them to make accurate predictions by combining multiple weak learners, providing robustness against overfitting, whilst also being extremely efficient and scalable. These models can both handle large datasets efficiently, and are renowned for their speed and scalability, making them a perfect choice for our system. Both these boosting frameworks provide unique advantages over the other, and so we decided to use both in order to measure their performance. For example, whilst LightGBM has faster training speed due to its ability to handle large datasets better, it does not provide a wide range of flexibility in its objective function and regularisation techniques like XGBoost.

Another model that we decided to examine was Word2Vec, a popular word embedding technique known for its ability to capture semantic relationships between words in a vector space. In the context of recommender systems, this model was utilised to embed products into a continuous vector space, enabling a meaningful representation of these products based on their context. Specifically, the embeddings obtained from Word2Vec capture the latent relationships and contextual relevance between different article ids based on how they appear together in user sessions, allowing an understanding of the underlying structure of user interactions - such as products which are frequently accessed together or those which share similar patterns of engagement. In essence, using this model allowed us to capture subtle user preferences and similarities between products, leading to improved recommendation accuracy.

Finally, we created a model which ensembled all of the models above in an attempt to improve predictive performance of our recommendation system, by leveraging the strength of each individual model. Specifically, the co-visitation matrices captured co-occurrence patterns between products, Word2vec leveraged article ids embeddings to find similar items and LightGBM/XGBoost employed gradient boosting. Ensembling can improve the robustness of the recommender system by reducing the impact of individual model weaknesses or biases and provide better generalisation. In turn, this enables more accurate and stable recommendations for a wider range of user preferences and helps to mitigate overfitting as it is less likely to memorise noise. By fine-tuning the weights for each model, we can build an effective recommendation system which can provide significant predictive gains.

Note that further details about each of these models will be provided in their respective sections later in this report.

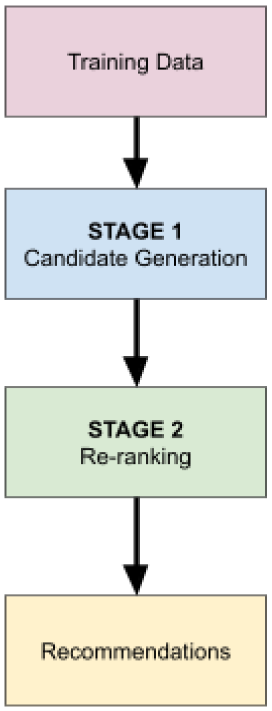
[6] Bishop, Pattern recognition and machine learning (2016).

Hyperparameter Tuning

Learning Algorithm	**Hyperparameters**	**Effect**	**Values Tested**
Re-Ranker	Timing Window	The maximum difference between clicks/orders/carts is considered as a larger time window may capture more interactions but might also introduce noise and irrelevant associations between products. On the other hand, a smaller time window may lead to more specific and relevant co-visitations.	1 - 48 hours in increments of 4 hours
	Time Decay Weights	More recent events/interactions may be given more weight with gradual reduction to older weights. This assists with capturing changing user preferences and gives higher importance to recent behaviours.	‘none’ or ‘logarithmically spaced’
LightGBM	Objective	Specifies the loss function to be optimised during the training process and guides the model on how to train and make predictions.	lambdarank
	Metric	Evaluates the model performance during training and validation stages, which is crucial for measuring predictive power.	ndcg, map
	Boosting Type	Affects model performance and training speed. The choice of algorithm can improve the model’s generalisation capabilities and prevent overfitting.	gbdt, rf, dart
	Number of Estimators	Represents the number of boosting iterations. Increasing generally leads to better model performance as it allows the model to capture more complex relationships in the data, however, setting this parameter too high can lead to overfitting.	1, 5, 10, 15, 20, 25, 30
Word2Vec	Negative Sampling Distribution	A higher value proportionally samples aid values with a high frequency of occurrences. A negative value samples more low frequency values as opposed to high frequency values.	-0.5, 0.5, 1
	Number of Epochs	A larger number of epochs corresponds to more number of iterations through the training data, therefore although this may lead to decreased error, there may be a threshold upon which a larger number of epochs could lead to overfitting.	1, 5, 10
	Window-Size	As window size denotes the maximum distance between current and predicted aid values within a session, a smaller time window would lead to higher correlated values and more relevant aid values being predicted, similar to the Re-Ranker timing window.	1, 5, 10
XGBoost	Objective	The objective function guides the model on how to best split the data at each decision node, with the ultimate goal of minimising the loss between the predicted and actual output values.	ndcg, map, pairwise
Ensemble Methods	Weights	The weights determine the influence of each individual machine learning model (or in our case, the significance of each prediction generated by the respective models).	Gridsearch
	Decay	Exponential decay reduces the influence of predictions that are generated towards the end of a certain session (prioritising those that are generated first as the predictions are made in sorted order).	0.95 - 1.0

Re-Ranker

This is a two-stage model which involves first filtering out and reducing the vast number of products (article ids) by a factor of 100, down to a few thousand. This not only helps in increasing efficiency and scalability, but also helps to reduce noise (for example by limiting products that are not very popular or are irrelevant). The candidates generated at this stage are then passed into the second stage, where they are ordered according to a ranking system. The top (n = 20) ranking candidates for each user are then used as recommendations.



In the first stage, we used co-visitation matrices - a statistical method which employs conditional probabilities to capture relationships and interactions between objects. These matrices can be significantly useful in recommender systems as they can be applied to suggest items that are "visited" together by different users. One caveat of this approach however, is that for large datasets, it can be computationally expensive to generate co-visitation matrices. As a result, we had to resort to generating only three matrices:

Co-visitation Matrices	
Clicks	Records the number of times two article ids are clicked within 24 hours of each other, with items clicked within a shorter time frame given a higher weighting.
Carts to Orders	Records article ids present in a cart which were then later ordered.
Buy to Buy	Records whether two article ids were bought within 24 hours of each other.

$$P(A|B) \;=\; \frac{P(A \cap B)}{P(B)}$$

$$where\ A \;=\; user\ clicks/carts/orders\ a\ product\ A$$

$$B \;=\; user\ previously\ clicked/carted/ordered\ on\ product\ B$$

[The formula for Conditional Probability]

Each of the entries in the co-visitation matrix represent a conditional event. In particular, each entry represents a weighting of how likely a product is to be clicked, carted or ordered respectively. This combination of features leads to an informative and comprehensive representation of user behaviour patterns.

In the second stage, candidates generated from the first stage are prioritised and ordered according to various factors, with the goal to present the most relevant and personalised recommendations for each user. For this model, the re-ranking is done according to some hand-crafted rules which leverage both the broad coverage of the initial candidates and the user-specific preferences to deliver a curated list of items that are most likely to meet a user's needs.

Our Re-Ranker model uses a combination of the user's history, the co-visitation matrix weights, a weighting scheme based on the event type and their recurrences, as well as time decayed weights with more recent events given higher preference.

This combination of these custom rules allows it to perform considerably well, with a score of 0.6682. Amalgamating the user's history (which leads to more personal recommendations), the co-visitation matrices (which provide the most probable interactions), weighting scheme (emphasising important events such as how orders more accurately reflect a user's tastes compared to clicks) and time decay weights (symbolic of the user's current interests) all come together to provide an accurate recommendation system.

This model also provides a good balance between bias and variance through these custom rules. For example, the use of the co-visitation matrices captures meaningful patterns and relationships in the data, reducing the risk of overfitting. Weighting and regularisation schemes employed help control the impact of individual features, and prevent the model from becoming prone to noise in the data. Also, by incorporating the user's own history and session information, the model generalises well to new users and scenarios, making it less likely to memorise specific instances of the training data.

However, the model is still prone to some problems. For instance, it may not generalise well to new users who have limited or no existing history. Instead, the model simply recommends the most popular items, and although this may be a good starting basis, it is not curated to the user's nuances. Thus, it may be beneficial to collect more information about users, such as demographics, location or device type. These features may also help to provide more in-depth recommendations, something which the model lacks, though of course, we are not provided with such information in this dataset. The model is also hard to scale as when dealing with a large number of users and items since the co-visitation matrices require regeneration, and these computational costs may become significant.

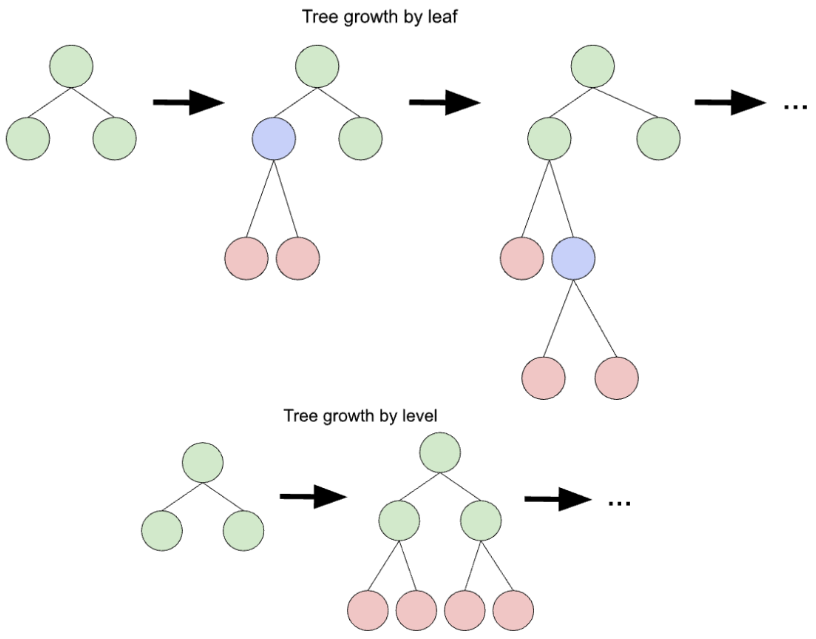
Hyperparameter Tuning

Due to the large size of the dataset, a grid search to fine-tune the hyperparameters was infeasible. After some preliminary analysis, it was found that shorter timeframe windows make up the bulk of the dataset, as shown in Figure 7 (see appendix). The 3D graph shows that across all sessions, user events (clicks/carts/orders) occur within a relatively short time period of two days as the data points were much more dense in the lower region of the Time Difference axis. Therefore, we reduced our search space to between 1 and 48 hours, and discovered that a time window of 24 hours produced the optimal recall score. This time window also allowed for better temporal co-occurrences between article ids and captured items that were closely interrelated, leading to better recommendation outcomes. It also served as a good trade-off to user interests as a larger time window may enhance the diversity of recommendations but could lead to slower adaptation to rapidly changing user interests. The reduction in the time window also helped reduce excess noise or irrelevant co-occurrences, in turn helping it to generalise better by filtering out irrelevant interactions.

The second hyperparameter for this model involved comparing the introduction of regularisation (by decreasing the influence of past interactions over time) versus weighting all events equally. This weighting system played an important role in determining how the importance of past interactions decreases over time. By emphasising recent events, the algorithm was able to stay responsive to current user behaviours, whilst also promoting exploration of new items, which was particularly important in a dynamic environment such as OTTO where shopping preferences are bound to evolve rapidly. We found that not applying these weights resulted in a significantly reduced ‘recall@20’ score of 0.287, reinforcing the importance of adapting to a user’s current psyche.

LightGBM Ranker

Light Gradient Boosting Machine (LightGBM) is a gradient boosting framework which uses tree based learning by ensembling weak learners iteratively. It differs from other frameworks as its trees grow vertically, rather than horizontally, as depicted in Figure 3. The tree grows by choosing the leaf with the largest loss, which is more efficient than reducing loss by level. This strategy allows the model to be more efficient as it grows deeper trees with fewer leaf nodes, and hence reduces the computational cost significantly - which is ideal for our case since our dataset is very large.



[Figure 3. Leaf growth vs. level growth]

LambdaRank was used as the objective function for this model. It subverts the difficulty associated with the direct optimisation of quality measures (such as mean reciprocal rank, winner takes all and pairwise correct [7]) in information retrieval methods by using an implicit cost function. Specifically, the algorithm is designed to optimise the pairwise ranking quality, which aligns with the reranking task at hand.

Coupled with the LambdaRank objective function, the Normalised Discounted Cumulative Gain (NDCG) evaluation metric was used. This metric measures the quality of the ranked list and takes into account their relevance, helping to improve the recommendation quality of the labels.

[7] C. Burges, R. Ragno, Q. Le. Learning to Rank with Nonsmooth Cost Functions (2006).

The boosting type used for this model was DART (Dropouts meet Multiple Additive Regression Trees) which was used due its ability to improve the model's generalisation and prevent overfitting, which is crucial for improving the ranking algorithm. The choice of boosting can affect the model's performance and training speed, and DART provided a balanced approach between score and speed. Other boosting types include Gradient Boosting Decision Trees (gBDT) and random forest. DART overcomes the deficiencies with these algorithms, which include reduced predictive power and over-specialisation, leading to over-fitting [8].

Word2Vec

The Word2Vec model is from the Gensim library, used in Natural Language Processing as a simple neural network model [9]. It creates word embeddings, or vectorised representations of text in order to group together vectors of similar words and predict nearby words for each given word in a sentence. In the context of the OTTO Recommendation system, the "words" are AID (article id) values after a particular event type with the sentence being the event values corresponding to a particular session ID. Word2Vec operates by recognising the characteristics between certain AID values, thereby increasing the validity in predictions of future event occurrences for a particular user.

The AID values embeddings for Approximate Nearest Neighbour Search were created by utilising the Annoy library, used in lieu of the inbuilt Gensim library to improve computation time. The index_to_key property gives a list, with every entry corresponding to a string word/AID value [10]. The embeddings created from these word vectors were made and fed to the Word2Vec model to establish similarities in values. The test session types and AIDs were converted to lists and Approximate Nearest Neighbour Search was employed to generate labels from the Word2vec model [11]. This operated in conjunction with weightings of 0.1, 0.3 and 0.6 corresponding to the clicks, carts and orders in order to sort the candidate generated AID values. The labels were then reformatted into the submission format as required.

Hyperparameter Tuning

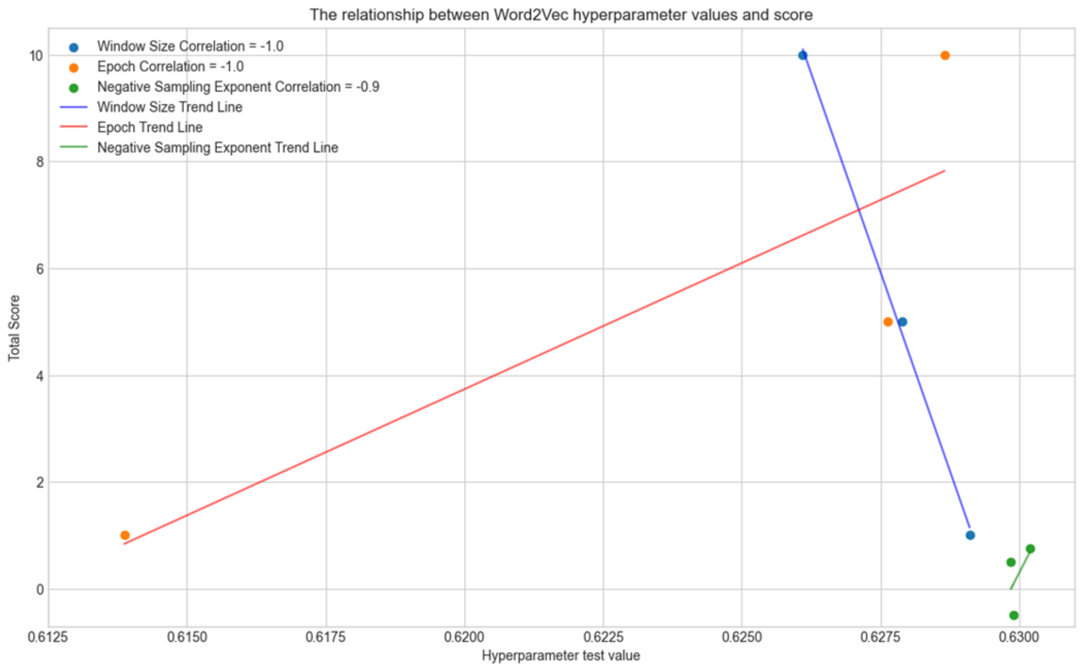
Initially, the only parameters used were vector_size, min_count and workers. The vector_size parameter represents the dimensionality of word vectors, and with a large dataset only requiring the prediction of 20 aid values per session, it seemed fitting to have roughly 30-32 as the vector size. The min_count parameter was set to 1, since words with occurrence less than this value will be disregarded and we did not want to ignore data in our analysis. The number of workers corresponds to the number of threads simultaneously training and since most commonly computers have 4 cores and we wanted our code to be accessible, a value higher than this would not be viable.

[8] Rashmi, Gilad-Bachrach, DART: Dropouts meet Multiple Additive Regression Trees (2015).
[9] Tensorflow contributors, TensorFlow Text (2023).
[10] Yard, Class: Annoy::annoyindex (2022).
[11] Spotify contributors, Spotify/Annoy: Approximate nearest neighbors in C++/Python optimized for memory usage and loading/saving to disk (2023).

The article ‘Word2Vec applied to Recommendation: Hyperparameters Matter’ stated that optimising hyperparameters “significantly improves performance on a recommendation task, and can increase it by an order of magnitude”, and from this we decided to test further values. Our motivations directed towards tuning the following parameters: negative sampling distribution (ns_exponent), number of epochs (epochs), and window-size (window). [12] Since the Word2Vec model had a lengthy run time of one and a half hours, our focus was on hyperparameter tuning of just these 3 parameters in order to enhance the capabilities of the Word2Vec model.

As reflected in Figure 4 below, as the number of epochs increases, the calculated score from the Word2Vec model increases. Likewise, this resultant score is optimised as the window size decreases. The negative sampling distribution aims to maximise the similarity of word incidences in the context, and hence, AID values that occur with a higher frequency have a higher probability of being chosen as a negative sample.

The hyperparameter of ns_exponent seemed to have a better score when the parameter was set to a higher positive value, yet the negative value interestingly had a better score than the corresponding positive value. This correlated with findings in the aforementioned article which stated “The optimal hyperparameter is −0.5, such that the optimal negative sampling distribution is one more likely to sample unpopular items as negative examples.” The results reflect setting the negative sampling parameter to be negative sampled words with a lower frequency, as opposed to those with a higher one. These findings were used to fine-tune our predictions and used to generate the best scoring predictions. For closer analysis, refer to Table 4 in the appendix with the Word2Vec scores.



[Figure 4. Word2Vec Hyperparameter Tuning Graph]

[12] Caselles-Dupré et al. Word2Vec applied to Recommendation: Hyperparameters Matter (2018).

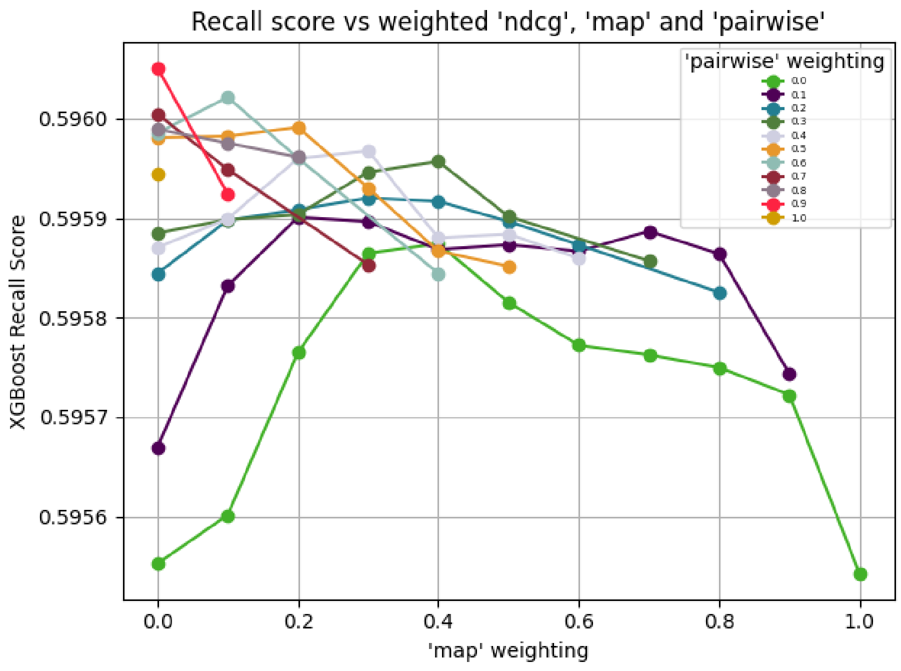
XGBoost Ranker

The eXtreme Gradient Boosting (XGBoost) is an optimised, scalable and distributed machine learning library developed with the primary goal of pushing the limit of computational resources for boosted tree algorithms [13]. It is popular for regression and classification problems, and works by creating a prediction model by additively generating weaker models (typically decision trees) formalised as a gradient descent algorithm over an objective function, and ensembling them together through "boosting". Not only is XGBoost performant, but beneath the surface it also has L1 (Lasso Regression) and L2 (Ridge Regression) regularisation to prevent the model from overfitting, as well as tree turning and built-in cross validation.

The XGBoost ranking models were trained on the GPU [14] using different objectives and the feature importance (a quantified measure of how much each feature contributes to the predictive power) were plotted for the trained model (see Figure 8 for an example). Once the models were trained, a weighted blend was used to make predictions on the test data and write the results in the specified format, such that the cross-validation score can be determined and examined thereafter.

Hyperparameter Tuning

As specified earlier, XGBoost relies heavily on an objective function and here, we tuned the following: **NDCG** (Normalised Discounted Cumulative Gain measures the quality of a ranking model based on the positive of relevant items), **MAP** (Mean Average Precision averages the precision of the system at the point of each of the relevant items in a list) and **Pairwise** (considers two items at a time during training and optimises the model such that the order of predictions respects those pairs).



[Figure 5. XGBoost Hyperparameter Tuning Graph]

[13] Xgboost contributors. Learning to Rank — xgboost 2.0.0-dev documentation (2023).
[14] Nvidia. What is XGBoost? (2023).

We then perform a manual gridsearch over all the weights of these three objective functions with a step size of 0.1 and the results are tabulated within the appendix. A plot of the recall scores against the weights is also shown in Figure 5. Note that the x-axis has the weighting of the ‘map’ objective, the legend shows the ‘pairwise’ weighting and thus, the remaining ‘ndcg’ weighting can be inferred since the sum of these weights is 1.0. For each of the colours, we see that the recall scores follow an approximately normal distribution, where there is a clear peak for most of these plots. This confirms our hypothesis that there is a relationship between the weights and the scores.

In summary, the best combination was 0.9 : 0.0 : 0.1 for ‘ndcg’, ‘map’ and ‘pairwise’ respectively (this is the pink data point in the top left corner of the graph). This makes sense as the purpose of the ‘ndcg’ ranking aligns best with the task at hand at generating the best predictions like in the LightGBM ranker, and ‘map’ is not so practical here since we are not asked to find an entire list of predictions with uniform weighting and binary relevance necessarily. Subsequently, this optimal set of weight combinations generated from XGBoost was used later in the ensemble methods.

Ensemble Methods

Ensemble methods are a machine learning technique that combine multiple models to create a more robust and accurate prediction. They work on the principle that a group of weak learners can come together to form a stronger learner, thus increasing the accuracy and improving the model’s generalisability [15]. There were two approaches that we considered for the ensembling of our models and predictions data. Firstly, we attempted using a voting ensemble, where each prediction is given a vote and the final predictions are chosen based on the sum of votes [16]. However, this constantly resulted in a very low cross-validation score of 0.15 based on the predictions generated from our models due to the fact that some models were consistently overfitting to the training data, while others were underperforming on certain subsets of the data. As a result, the voting ensemble was not able to capture the strengths of individual models effectively, leading to suboptimal performance.

Consequently, we decided to implement a slightly modified version, namely the weighted average ensemble methodology [17]. Carefully chosen weights were assigned to each model based on their cross-validation scores. Models with higher cross-validation scores were given higher weights, indicating that they were more reliable and should have a larger contribution to the final ensemble prediction. This approach provided more fine-grained control over the influence of each model on the final result, enabling us to capitalise upon the advantages of individual models. However, it also introduced more sensitivity to the specific weights used, meaning that the performance of the ensemble could potentially be more affected by the choice of weights. Despite this sensitivity, the weighted average ensemble yielded better results compared to the simple voting ensemble, leveraging the strong performance levels of each model more effectively and collating all of their ideal predictions to more closely match the testing ground truth values.

[15] Mwiti, A Comprehensive Guide to Ensemble Learning: What Exactly Do You Need to Know (2023).
[16] Osmulski, How-to ensemble predictions (2023).
[17] Kumar, Ensemble Methods in Machine Learning: Examples, Data Analytics (2023).

After analysing the table of results generated by the individual models (shown in the appendix) and understanding the inner mechanics of how they worked, we used the predictions generated by five models in the ensemble methods phase: Word2Vec, Re-ranker Model, Exploratory Data Analysis (EDA), XGBoost, LightGBM. We adopted a form of Leave-One-Out Cross-Validation (LOCCV) wherein for each fold, we left out one model and then tuned the weights of the remaining four models. This was necessary and in order to reduce the risk of overfitting, we had to limit the weights of each model to be between 0.2 and 0.8. This process was then repeated for all model combinations in the dataset, and we found that removing LightGBM resulted in the optimal recall score, and thus our final model only consisted of the aforementioned four models.

We utilised a gridsearch approach in order to determine the weights and prevent the final predictions having suboptimal performance [18]. Appropriate step sizes and ranges from the itertools library were taken to avoid computing too many combinations and account for the bias-variance tradeoff. Since at most 20 predictions were generated for each model, we also decided to incorporate a decay factor such that for instance, the 3rd article id had higher significance than the 19th. This decay was computed to be 0.97 (meaning that we would multiply the weight by 0.97^3 against 0.97^{19}). Fourteen hours later, with 127 generated files totalling 104GB in size, a script was subsequently run in order to determine the best cross-validation score. These results are summarised in the appendix.

Final Results

Ultimately, the best normalised weights consisted of the following:

Model	Normalised Weights
Word2Vec Ranker	0.400
2 Stage Re-Ranker	0.175
Exploratory Data Analysis	0.150
XGBoost Ranker	0.275

The above set of weights generated a cross-validation score of 0.92643, which again uses the recall@20 metric defined in the problem statement. When the corresponding ‘submission.csv’ was uploaded to Kaggle as per the competition rules, the final private score was 0.89638. Note that the private score is what is used to determine the leaderboard rankings, since it is based on a hidden dataset to prevent users hard coding the correct values or overfitting the training dataset.

Our results exemplify the importance of not only model selection, but also ensemble methods to reduce overfitting, improve robustness and boost the overall prediction power, as the score evidently increased multifold compared to each model running individually.

[18] Great Learning, Hyperparameter Tuning with GridSearchCV (2023).

Improvements

In our efforts to improve the recommender system, we noticed several key opportunities for enhancement. In particular, it may have been advantageous to leverage the strength of the co-visitation matrices which provide valuable insights into user behaviour patterns, and combine them with the LightGBM or XGBoost machine learning models to enhance our learning capabilities. Specifically, replacing the custom re-ranking stage with one of these models may provide a more robust and accurate recommender system, and this area should be further explored..

Whilst we only used three co-visitation matrices due to computational constraints, we recognise the valuable insights they provided and their formidable predictive performance (its Kaggle private score of 0.57245 was the highest amongst the individual models without the ensembling phase). Therefore, it may be beneficial to incorporate more of these matrices into the candidate generation stage. For example, we could explore variations in user behaviour during different times of the day, such as morning versus night, as these patterns may influence the recommendations outcomes.

If possible, it may also be worthwhile gathering additional user data such as demographic information, location and device type, which could provide deeper insights into user preferences and behaviour, as currently there are no features relating to user information apart from the time at which the event was triggered. Incorporating such personalised data into the model could lead to more tailored and accurate recommendations, better catering to the needs of individual users.

Furthermore, although we applied grid search to tune the hyperparameters of some models such as XGBoost, the step sizes chosen were relatively large due to limited computing resources. It may be favourable to reduce this step size or use other methods such as Bayesian optimisation which is known for its ability to discover fairly good hyperparameter combinations in relatively few iterations. In a similar vein, the LightGBM Ranker model provides a large number of hyperparameters to tune, and exploring these options may have produced a better recall score [19].

The training data for each model was also a limited period of only four weeks, which can introduce biases and limitations. For example, user interests may change seasonally or certain products may only be available in certain parts of the year, and these interactions are not captured. Another issue that may arise due to the limited data collection period is that incomplete user profiles may be formed. For example, users might not engage with a wide range of content during the four-week period, hindering the model's ability to understand user preferences accurately, leading to surface level recommendations or in the worst case, contributing to disengagement from the OTTO site.

By taking the above measures, the recommender system can become more robust and reliable, leading to an enhancement in its ability to provide accurate and personalised recommendations to users, which leads to happier customers and better engagement for OTTO, thereby maximising company profits.

[19] LightGBM contributors, LightGBM Parameters (2023).

Conclusion

The development and implementation of the OTTO Recommender System offers an elegant solution that harmonises the objectives of both retailers and customers. By recommending pertinent products, it enhances the customer's shopping experience while concurrently fuelling the sales engines of retailers, such as OTTO.

The success of our endeavour was cemented through the masterful orchestration of five machine learning models. Our accomplishment was not just a triumph of programming acumen, but also a testament to the remarkable synergy of our three-member team, surpassing our initial aspirations. Our group performance ultimately manifested in our tangible results, and had we participated in the competition, we would have had a chance at scoring very well.

Notably, the use of parquet files inspired from competition notebooks saved us time in loading data, enabling us to shift our focus on hyperparameter tuning, and leveraging their potential in maximising our models' capabilities and boosting our competition score.

As discussed, there remain uncharted territories to explore. Given additional time and computational resources, we could have delved into different ensemble combinations, experimented with varying weights in smaller step sizes and further fine-tuned the hyperparameters associated with each of our models. Nevertheless, the experience gained from our first ever Kaggle competition proved worthwhile, and we have gained confidence at tackling such challenges and will undoubtedly participate in more competitions to continue pushing our boundaries and exploring the limitless realm of Machine Learning.

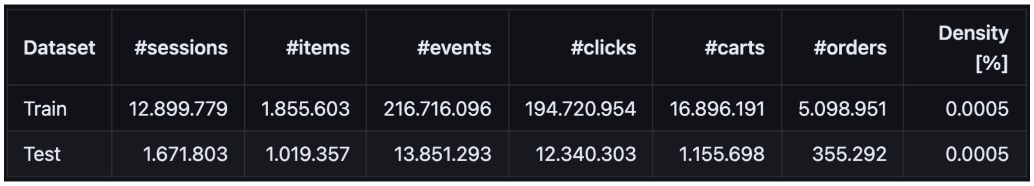
References

- Bishop, C.M. (2016) *Pattern recognition and machine learning*. New York, NY: Springer New York. (Accessed: 29 July 2023).
- Burges, Ragno, Le (2006) *Learning to Rank with Nonsmooth Cost Functions.* Available at: https://proceedings.neurips.cc/paper_files/paper/2006/file/af44c4c56f385c43f2529f9b1b018f6a-Paper.pdf (Accessed: 03 August 2023).
- Caselles-Dupré, H., Lesaint, F. and Royo-Letelier, J. (2018) *WORD2VEC applied to recommendation: Hyperparameters matter*, *arXiv.org*. Available at: https://arxiv.org/abs/1804.04212 (Accessed: 26 July 2023).
- Derrick Mwiti. (2023) *A Comprehensive Guide to Ensemble Learning: What Exactly Do You Need to Know.* Available at: https://neptune.ai/blog/ensemble-learning-guide (Accessed: 03 August 2023).
- Examining Feature Contributions. (2023) *Information Leakage.* Available at https://www.c3iot.ai/glossary/data-science/information-leakage/ (Accessed 03 August, 2023).
- Gensim: Topic modelling for humans (2022) *models.word2vec – Word2vec embeddings - gensim*. Available at: https://radimrehurek.com/gensim/models/word2vec.html (Accessed: 01 August 2023).
- Great Learning Team. (2023) *An Introduction to GridSearchCV | What is Grid Search | Great Learning.* Available at https://www.mygreatlearning.com/blog/gridsearchcv/ (Accessed: 03 August, 2023).
- Ke. et. al (2017) *LightGBM: A Highly Efficient Gradient Boosting Decision Tree.* Available at: https://papers.nips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf (Accessed: 28 July 2023).
- Kumar, A. (2023) *Ensemble Methods in Machine Learning: Examples*, *Data Analytics*. Available at: https://vitalflux.com/5-common-ensemble-methods-in-machine-learning/. (Accessed: 02 August 2023).
- Last, First. (2023) *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation.* Available from https://www.researchgate.net/publication/228529307_Evaluation_From_Precision_Recall_and_F-Factor_to_ROC_Informedness_Markedness_Correlation (Accessed: 03 August, 2023)
- LightGBM contributors (2023) *LightGBM/docs/Parameters.rst at master · microsoft/LightGBM*, *GitHub*. Available at: https://github.com/microsoft/LightGBM/blob/master/docs/Parameters.rst#core-parameters (Accessed: 31 July 2023).
- Nvidia (2023) *What is XGBoost?* Available at: https://www.nvidia.com/en-us/glossary/data-science/xgboost/ (Accessed: 02 August 2023).
- Osmulski, R. (2022). *How-to ensemble predictions* Available at: https://www.kaggle.com/code/radek1/2-methods-how-to-ensemble-predictions/ (Accessed: 25 July 2023).
- Otto – multi-objective Recommender System (2022) Kaggle. Available at: https://www.kaggle.com/competitions/otto-recommender-system/ (Accessed: 02 August 2023).
- Otto-De (2022) Otto-DE/Recsys-dataset: A real-world e-commerce dataset for session-based Recommender Systems Research., GitHub. Available at: https://github.com/otto-de/recsys-dataset (Accessed: 02 August 2023).
- Spotify contributors (2023) *Spotify/Annoy: Approximate nearest neighbors in c++/python optimized for memory usage and loading/saving to disk*, *GitHub*. Available at: https://github.com/spotify/annoy#full-python-api (Accessed: 01 August 2023).
- Tensorflow contributors (2023) *TensorFlow Text.* Available at: https://github.com/tensorflow/text/blob/master/README.md (Accessed: 31 July 2023).
- Ux Performance.Page Designs. (2023) *E-Commerce Checkout Usability: An Original Research Study – Baymard Institute.* Available from https://baymard.com/research/checkout-usability (Accessed: 03 August, 2023).
- Vatsal (2023) *Word2Vec explained*, *Medium*. Available at: https://towardsdatascience.com/word2vec-explained-49c52b4ccb71#:~:text=Word%20embeddings%20is%20a%20technique,which%20resembles%20a%20neural%20network (Accessed: 01 August 2023).
- Xgboost developers. (2023) *Learning to Rank — xgboost 2.0.0-dev documentation* Available at: https://xgboost.readthedocs.io/en/latest/tutorials/learning_to_rank.html (Accessed: 28 July 2022).
- Yard 0.9.27 (2022) *Class: Annoy::annoyindex*, *Class: Annoy::AnnoyIndex - Documentation by YARD 0.9.27*. Available at: https://yoshoku.github.io/annoy.rb/doc/Annoy/AnnoyIndex.html (Accessed: 01 August 2023).

Appendix

The GitHub repository can be found here: https://github.com/William-Feng/recsys-dataset.

Figure 6. Dataset Statistics (sourced from the README.md)



Dataset	#sessions	#items	#events	#clicks	#carts	#orders	Density [%]
Train	12.899.779	1.855.603	216.716.096	194.720.954	16.896.191	5.098.951	0.0005
Test	1.671.803	1.019.357	13.851.293	12.340.303	1.155.698	355.292	0.0005

Figure 7. Co-visitation Matrix

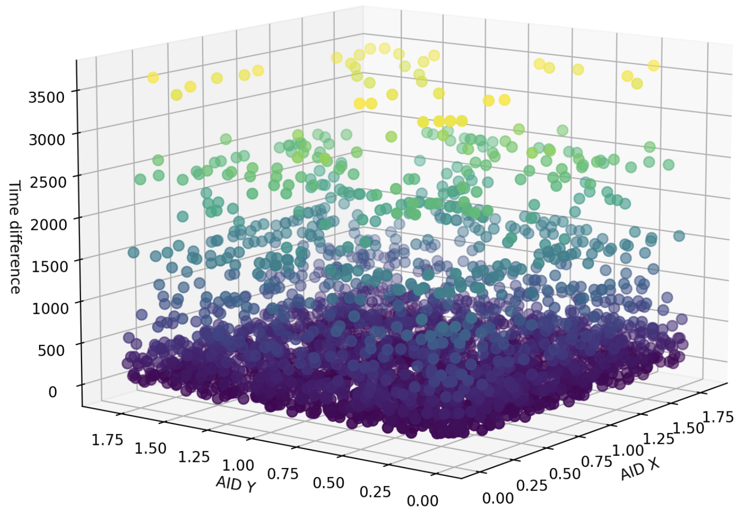


Figure 8. XGBoost Feature Graph

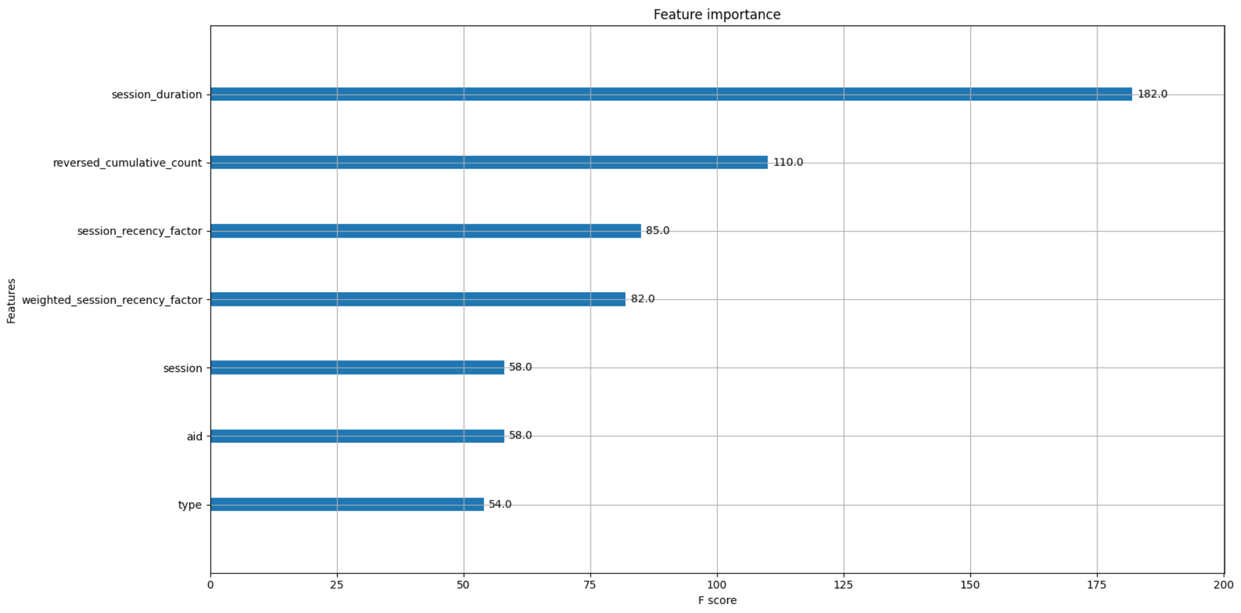


Table 1. Scores against generated ground truths/test labels

	Clicks	**Carts**	**Orders**	**Total**
Re-Ranker Model	0.6296	0.6092	0.704197	0.6682
EDA	0.48596	0.27828	0.43564	0.39347
LGBM	0.4717	0.5358	0.6494	0.5976
Word2Vec	0.5398	0.5696	0.6728	0.6286
XGBoost	0.47172	0.53490	0.64735	0.59605
Ensemble Methods	0.99116	0.86336	0.94718	0.92643

Table 2. Kaggle Score

	Private Score	**Public Score**
Re-Ranker Model	0.57245	0.5727
Exploratory Data Analysis	0.38706	0.38506
LGBM	0.48457	0.48317
Word2Vec	0.52179	0.52119
XGBoost	0.48355	0.48268
Ensemble Methods	0.89638	0.89509

Table 3. LightGBM Scores for num. estimators tuning

Number of Estimators	**Recall@20 Score**
1	{'clicks': 0.4718651071027275, 'carts': 0.5350577220613322, 'orders': 0.6489154907911113, 'total': 0.5970531218033391}
5	{'clicks': 0.4718078520529514, 'carts': 0.5362796048716626, 'orders': 0.6480484428097949, 'total': 0.5968937323526708}
10	{'clicks': 0.47173151198658325, 'carts': 0.5361447217042885, 'orders': 0.6488735368565315, 'total': 0.5973406898238638}
15	{'clicks': 0.47173866886780524, 'carts': 0.5363192763914786, 'orders': 0.6479505502957752, 'total': 0.5968399799816893}
20	{'clicks': 0.4717100413429172, 'carts': 0.5362319990478835, 'orders': 0.649502845875229, 'total': 0.5977423113737942}
25	{'clicks': 0.4717195838512132, 'carts': 0.5362716705676994, 'orders': 0.6495308151649489, 'total': 0.5977719486544004}

30	{'clicks': 0.4717004988346212, 'carts': 0.5361843932241044, 'orders': 0.6493070608471898, 'total': 0.5976096043590073}

Table 4: *Word2Vec Scores for hyperparameter tuning*

Epoch	**Window Size**	**Negative Exponent**	**Recall@20 Score**
6	10	1	{'clicks': 0.5346404979280829, 'carts': 0.5666521204427342, 'orders': 0.6710671682492623, 'total': 0.6260999868751859}
6	5	1	{'clicks': 0.5386984495809646, 'carts': 0.5692149006228429, 'orders': 0.6721020319688982, 'total': 0.6278955343262882}
6	1	1	{'clicks': 0.54254408042426, 'carts': 0.5699845281072717, 'orders': 0.6730949417539541, 'total': 0.62910673152698}
1	4	1	{'clicks': 0.5096057274134793, 'carts': 0.5535287816876264, 'orders': 0.6614177632959011, 'total': 0.6138698652251765}
5	4	1	{'clicks': 0.5382666510805698, 'carts': 0.5684690760503035, 'orders': 0.6721020319688982, 'total': 0.627628607104487}
10	4	1	{'clicks': 0.5397600536288967, 'carts': 0.5697623675963026, 'orders': 0.6729131413707748, 'total': 0.6286526004642453}
6	4	0.75	{'clicks': 0.5417782941335044, 'carts': 0.5708255643273694, 'orders': 0.67411582082873, 'total': 0.6298949912087992}
6	4	0.5	{'clicks': 0.5428375125543625, 'carts': 0.5708255643273694, 'orders': 0.6738640972212511, 'total': 0.6298498788863978}
6	4	-0.5	{'clicks': 0.5432382979027952, 'carts': 0.5715317173800928, 'orders': 0.6740039436698505, 'total': 0.6301857112062177}

The XGBoost Ranker Model and Ensemble Methods hyperparameter tuning have been omitted due to the lengthy nature of their output. They can be found at the process/XGBoost and process/ensemble directories of the GitHub repository respectively.