Improving User Experience through Machine Learning: An Approach using XGBClassifier

BARRACHIN Carlyne

ABSTRACT – The main objective of this work is to improve the user experience in three distinct domains (E-commerce, Entertainment, Food) using a classification algorithm, XGBClassifier. Based on user data, this algorithm predicts preferences in order to personalize recommendations. Databases used include beauty product reviews, movie ratings and culinary recipe reviews. The study explores data pre-processing, hyperparameter optimization via Optuna, and evaluates model performance using suitable metrics, such as balanced accuracy and macro-average precision. The results show that the algorithm strongly favors majority classes, indicating a need for data rebalancing or methodological improvements to deal with minority classes. Solutions such as integrating analyses based on natural language processing (NLP) are proposed to capture more nuances in user preferences.

Keywords – **User Experience; XGBoost; XGBClassifier; Optuna: Performance Metrics**

1 INTRODUCTION

Nowadays, most people use websites and applications in their daily lives. Whether it's to buy products, to watch films or even to find recipe ideas, it's useful to use these kinds of tools. That's why, for each of these domains, we need a platform that's adapted to users' behavior, interactions and preferences. Ensuring that the user experience, which focuses on the overall experience a user feels when interacting with a service or system, is favorable to regular, pleasant and efficient use of the platform is important. In this context, understanding user behavior and designing adaptive systems are challenges. For example, as pointed out in the article [1], the integration of user data-driven methods and machine learning models can significantly improve the way systems respond to user needs by enabling real-time adaptation of interface elements.

In the experiments presented in this report, I used a machine learning algorithm to analyze and predict users' preferences, but it didn't dynamically modify interface elements in real time, as the model in article [1] does. Instead, the algorithm serves as a **decision-making tool** to inform potential improvements.

This work aims to apply a single algorithm to datasets from 3 different domains: **E-commerce**, **Entertainment** and **Food**. The chosen algorithm is **XGBClassifier**, a specific **XGBoost** tool for **classification**. For these domains, the aim is to improve the user experience by personalizing recommendations based on user's history. This includes movies viewed, products purchased, purchase histories and favorite recipes.

The paper is structured as follows: **Section 2** presents a review of the three selected domains. It highlights the application of this algorithm in these three areas and the reasons for its use. **Section 3** provides details of the datasets used, describing the features selected for each domain and justifying their relevance. In addition, it explains the pre-processing steps carried out to prepare the data for analysis, to ensure coherence and compatibility between experiments. **Section 4** presents the experiments performed, including the process of optimizing hyperparameters using Optuna to improve XGBClassifier's performance. **Section 5** presents the results obtained for each dataset are discussed to assess the algorithm's effectiveness in predicting user preferences. Finally, **Section 6** concludes the paper by providing numerous remarks.

2 A MACHINE-LEARNING METHOD IN 3 FIELDS

2.1 XGBoost

XGBoost or Extreme Gradient Boosting [2] is a machine learning library widely used for classification and regression tasks due to its

performance and flexibility. This machine learning algorithm, which uses decision trees as its basic model, is based on a technique known as boosting. The special feature here is that these trees are built sequentially. Each tree corrects the errors made by previous trees. In this way, several decision trees are combined until the predictions are better. This framework uses advanced techniques such as missing value handling, which improves performance and reduces the risk of overlearning. It provides for example **XGBRegressor** for regression problems but in my experiments, I chose to use **XGBClassifier**, which is used for classification problems. XGBoost is particularly well suited to large, complex datasets, using advanced regularization and parallel computing techniques to avoid overlearning and speed up training. Moreover, XGBoost can handle a wide variety of data types, including numerical and categorical variables, making it ideal for this project. The choice of classification instead of regression is motivated by the nature of the problem, where the aim is not to predict recommendations based on a continuous numerical value but rather to classify ratings into discrete categories, such as [0, 1, 2, 3, 4, 5]. This approach resulted in an easier-to-interpret result, corresponding to the rating scales typically used in recommendation systems or real-world rating frameworks.

2.2 E-commerce

For this domain, I've chosen to use the '*Sephora Products and Skincare Reviews*' dataset. It contains relevant information on online products from the Sephora online store. This dataset is composed of several .csv files. One file contains detailed information on over 8000 beauty products, while five other files contain around 1 million user reviews on over 2000 products in the **Skin-Care** category.

The aim is to analyze preferences and purchasing behavior to predict which products are likely to be of interest to each user, and thus improve the shopping experience. This study could be particularly useful for a Sephora employee, who could use the results to highlight relevant products on the home page, for example.

2.3 Entertainment

I chose to use the “MovieLens” dataset, which contains information on films and user ratings. This one includes 32 000 204 ratings and 2 000 072 tags applied to 87 585 films, created by 200 948 users between January 1995 and October 2023. Main files include “ratings.csv” for movie ratings, “tags.csv” for tags applied by users, “movies.csv” for movie information, including genres, and “links.csv” including links to other data sources. The only file we don't need is “links.csv”.

The aim here is to analyze users' genre and content preferences to predict which films are likely to interest them in order to improve their engagement on the platform. This study could be useful for a streaming platform, which could use these results to offer personalized recommendations, increasing time spent on the platform and user satisfaction.

2.4 Food

For this experiment, I used the ‘Food.com’ dataset. This contains many recipes and recipe reviews. From this dataset consisting of over 231 636 recipes and over 1 116 492 recipe reviews covering 18 years of user interactions I retrieved two files: RAW_interactions.csv containing reviews and RAW_recipes.csv containing recipes.

The aim here is to be able to recommend the recipes most likely to please, so that users waste less time searching for recipes.

3 FEATURES & ENCODING

3.1 Sephora

3.1.1 Features

These data include many attributes, but those kept are presented in the tables 1 and 2.

First, I have chosen to retain the characteristics in Table 1, as they are relevant to the analysis of user preferences in terms of price, brand and product category. For example, keeping ‘price_usd’ is useful in order to take into account whether the user is more likely to buy expensive products or not. Then, keeping categories and brand_name provides information on preferred product types and brands.

There are many features that I have found unnecessary for the product data set. Indeed, the ‘primary_category’ column has been removed, as we only have reviews available for the 'Skincare' category. Then, characteristics such as **product variation** (e.g., size or type), **ingredients**, or

brand_id do not add value in this context because when a product is recommended, it's usually the base product, and then the user can choose options such as shade, size or other variations. Thus, for the first recommendation, I choose an initial **global recommendation**.

To get an overview of the products data, we can observe the distribution of Secondary and Tertiary Categories in **Figure 15** and **Figure 16**, as well as the most popular brands : **Figure 17**.

product_id	Unique product ID
product_name	Full name of the product
brand_name	Full name of the product brand
price_usd	Price of the product in US dollars
highlights	A list of tags or features that highlight the product's attributes
secondary_category	Second category in the breadcrumb section
tertiary_category	Third category in the breadcrumb section

Table 1- *Product Data Content*

For the review dataset, the features I have decided not to use are information related to **reviews** as it adds a level of detail that is too specific for this recommendation model. Furthermore, **demographic characteristics** are not essential. While this information can be useful for more targeted recommendations, it is not directly necessary for predicting a user's general interests based on their history.

author_id	Unique author ID
is_recommended	Indicates if the author recommends the product or not (1-true, 0-false)
rating	The rating given by the author for the product on a scale of 1 to 5
product_id	Unique product ID
brand_name	Full name of the product brand
product_name	Full name of the product

Table 2 *- Reviews Data Content*

To simplify visualization and analysis, I decided to combine the two previous DataFrames. This was achieved by using an inner merge on the common columns: 'product_id', 'product_name', and 'brand_name'. In this way, products with no reviews or reviews corresponding to no products are excluded.

3.1.2 Encoding

The XGBClassifier model requires input variables to be integers, floats, Booleans or categories. However, the DataFrame columns brand_name, highlights, secondary_category, and tertiary_category contain string. These columns therefore require encoding before they can be used.

I have tested to use **One-Hot Encoding,** where each value becomes a separate column made up of binary values, but it generated at least 300 columns, which was too many. The risk of overlearning due to too many columns was too high. So, I opted to use **pandas Categorical Types**. The **'category'** type is compatible with XGBoost, which natively manages categorical variables via its own encoding mechanisms.

Unlike the other columns, I chose to keep the first 6 'highlights' and place them in six columns: 'highlight1' to ''highlight6'. I did several tests and the results were better with this number of highlights. Furthermore, on average, a user assigns around 6 tags per products and the maximum number of highlights is 10.

Finally, I chose **Label Encoding**, where each column value is replaced by a unique integer, for the 'product_id' and 'author_id' columns. They could'nt be used directly in the model because they contain alphanumeric values. They must therefore be encoded to make them numeric.

3.2 MovieLens

3.2.1 Features

Tables 3, 4 and 5 show the main features retained. **Figure 14** shows the distribution of genres to give an overview of this important attribute. I decided to keep all the features except **'timestamp'** for data associated with ratings and tags. Indeed, although the date on which a film was watched can provide an indication for recommending films according to specific periods or seasonal trends, I preferred to remove this column. I prefer to focus on users'

preferences, history and interactions, which are more direct and reliable indicators of their tastes.

For this experiment, two merges were necessary to create the final dataset. First, I merged the ratings and movies DataFrames using an inner merge. This excluded films without ratings and ratings without corresponding films. Next, I performed another internal merge between this new dataset and the tag one. The final dataset thus contains the following features: movieId, title, genres, userId, rating and tag.

movieId	Unique movie ID
title	Title of the movie
genres	All associated genres. This is a pipe-separated list.

***Table 3** - Movies Data Content*

userId	Unique viewer ID
movieId	Unique movie ID
rating	Rating given on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars)

***Table 4** - Ratings Data Content*

userId	Unique viewer ID
movieId	Unique movie ID
tag	User-generated. They usually consist of a single word or a short phrase.

***Table 5** - Tags Data Content*

3.2.2 Encoding

Like the Sephora experiment, I used **LabelEncoder** to encode 'movieId' and 'userId'. Secondly, the genre column contained 14 unique categories, making **One-Hot Encoding** an appropriate choice. This method created a binary column for each genre, representing whether a film belonged to that genre. Finally, the encoding of the tag column presented a significant challenge due to its high cardinality. A user can assign an unlimited number of unique tags to films, resulting in a large amount of data. Initially, I tried One-Hot Encoding, even after limiting the dataset to the first tag for each movie-user pair. However, this approach generated 280 000 columns. To remedy this problem, I decided to use **Pandas' Categorical data type**. This approach assigns a categorical code to each unique tag, significantly reducing memory usage and ensuring that the feature remains manageable for the model. So, I chose to keep the first 5 tags and put them in five columns, 'tag1' to 'tag5'. I did several tests and the results were better with this number of tags. Furthermore, the choice of five tags is supported by **Figure 1** , which shows the distribution of the number of tags assigned by users to films. This distribution shows that a large majority of user-film pairs (around 76%) involve 5 tags or less and, on average, a user assigns around 5 tags per movie. Beyond 5 tags, the number of occurrences falls sharply, making it less advantageous to keep more than 5 tags in the encoding process.

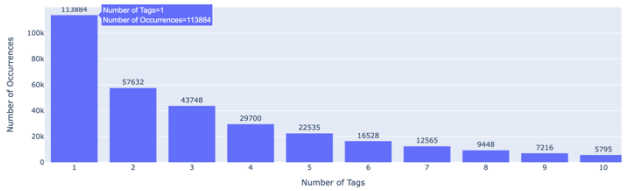

***Figure 1** - Distribution of Tag Counts per User per Movies*

3.3 Food.com

3.3.1 Features

From these two files I created 2 dataframes composed of the attributes found in the 6 and 7 tables. For the reviews database, I decided to remove the date and review columns. The reasons are the same as for the previous domain datasets.

user_id	Unique user ID: the person who wrote the review
rating	The rating given by the author for the product on a scale of 0 to 5
recipe_id	Unique recipe ID

***Table 8** - Reviews Data Content*

Then, the recipes dataframe contained numerous columns. The ones I haven't kept are the following: contributor_id, submitted, steps and description. I decided to keep the n_step column instead of step because n_step can tell whether users prefer shorter or longer recipes, whereas the recipe steps don't provide significant

information on whether a recipe should be recommended. Similarly, the “description” column, which contains a personal note from the person who published the recipe, was excluded. Indeed, its unstructured and subjective nature offers limited predictive power in the context of this classification task. I've decided to keep the minutes column, as it may reflect users' time preferences. Indeed, some prefer quick recipes, while others prefer longer ones. Then, in the nutrition column, users may prefer recipes according to their caloric or nutritional content. Finally, the Ingredients column (List of ingredients) is also important, as it may or may not correspond to users' dietary preferences.

name	Recipe name
recipe_id	Unique recipe identifier
minutes	Preparation time (in minutes)
tags	Qualifying tags recipe
nutrition	Nutritional values
n_step	Number of steps to complete the recipe
ingredients	Ingredients needed to make the recipe

***Table** 7 - Recipes Data Content*

3.3.2 Encoding

I inner merged the two data frames by ‘recipe_id’ and encoded the categorical columns. I used **LabelEncode** for the recipe ID and the user ID. Regarding ingredients, there was an average of 10 ingredients per recipe, 230 474 unique ingredients and the maximum number of ingredients was 43. There are too many unique ingredients to use One-Hot coding, so, as with previous datasets, I convert the column into a **category using pandas**. I only get the first 10 ingredients, which I split into 10 different columns and convert into categories. For labels, the average number is 255, which is far too many. I decided to keep only 10 and convert them into categories, as I did for the ingredients. For example, we can find the distributions of the first 15 ingredients before and after columns creation in **Figure 18** and **Figure 19**.

4 EXPERIMENTS

4.1Hyperparameters optimization using Optuna

Hyperparameter optimization is a fundamental step in improving the performance of machine learning models. In this study, I used **Optuna**, a next-generation framework specifically designed for hyperparameter optimization, as presented in the article [3] by its creators.

Let's start with a brief explanation of this framework. Optuna, which combines flexibility, efficiency and ease of use, uses efficient optimization techniques. These include **Bayesian searches** to find the combination of hyperparameters that produces the best results. These searches focus on the **most promising areas** of the hyperparameter space. Instead of exploring all possible combinations of hyperparameters, as Grid Search does, or generating them randomly, as Random Search does, Optuna explores them **dynamically**. The method consists in concentrating on promising regions of the search space, making it a faster and often more accurate tool. Moreover, Optuna stands out for its integrated dynamic pruning functionality. This allows tests considered unpromising to be quickly stopped during the training process, thus avoiding wasting time and resources on poorly performing configurations. Its use in my project made sense, given the sheer volume of data, its easy integration with the XGBoost framework and the many advantages mentioned above. This one is also a complex algorithm due to its large number of hyperparameters, such as learning rate (learning_rate), number of estimators (n_estimators) and maximum tree depth (max_depth). The XGBClassifier algorithm does not require extensive tuning, as demonstrated by the research reported in the paper [4]. It points out that a huge investment in terms of time is not necessary, as no considerable gains can be expected from adjusting the hyperparameters for this model. However, instead of manually selecting hyperparameters, I opted for Optuna, which enables intelligent, automated optimization. The article [4] presents a large-scale study involving 26 machine learning algorithms, 250 datasets (for binary and multinomial regression and classification

problems), 6 score measures, and 28,857,600 algorithm executions. Among these algorithms we can find XGBClassifier. I therefore used the information mentioned to configure Optuna and explore several hyperparameters, notably by defining a logarithmic range for n_estimators between [10, 1000] and running 30 trials (n_trials = 30).

4.2 Performance Metrics

To help me define the metrics to use for a classification I used the resources in reference [5]. I therefore chose to evaluate my model by analyzing the results of the following metrics:

Confusion Matrix in Multi-class Classification

In this project, we are dealing with a multi-class classification problem rather than binary classification. Consequently, a multi-class confusion matrix is more appropriate. This matrix extends the concept of the binary confusion matrix to handle predictions across multiple classes, such as the grades 1 to 5. It provides a clear visualization of how the classification algorithm performs across all classes. More precisely, the confusion matrix is represented as an **N×N matrix**, where **N** is the number of classes. Each **row i** corresponds to the true class, while each **column j** corresponds to the predicted class. The element **M[i,j]** represents the number of instances where the true class is i but the predicted class is j. **Diagonal elements (M[i, i])** indicate the number of correctly classified instances for class i and **off-diagonal elements (M[i, j], where i ≠ j)** show misclassifications. This matrix is an essential tool for identifying systematic errors, such as consistent misclassification between certain classes, and helps evaluate the classifier's overall performance.

Balanced Accuracy

$$Balanced\ Accuracy = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i + FN_i}$$

where C is the number of classes, and TPi and FNi are the true positives and false negatives for each class i.

First, True Positives (TP) corresponds to the number of instances correctly classified as belonging to class i and false Positive (FP) corresponds to the number of instances incorrectly classified as belonging to class i, but which belong to another class.

Simple accuracy corresponds to the proportion of correct predictions out of all predictions. **Multi-Class accuracy** is the proportion of correctly classified instances across all classes in a multi-class setting. And **Balanced Accuracy** corresponds to the average recall for all classes, treating each class equally regardless of frequency. This last measure prevents the dominance of majority classes from overshadowing the performance of minority classes, offering a more reliable assessment in such scenarios.

Macro-Averaged Precision

$$\text{Precision} = \frac{1}{N}\sum_{i=1}^{N}\frac{TP_i}{TP_i + FP_i}$$

Precision measures the accuracy of positive predictions made by a model. The basic measure is calculated by dividing the number of true positive predictions by the total number of positive predictions, which includes both true and false positives. But for multiple classes, the accuracy is calculated for each class individually, then averaged.

Recall

Corresponds to the proportion of true positive instances correctly identified by the model.

4.3 Prediction Model

4.3.1 Sephora

The characteristics selected for training are as follows: **author_id**, **product_id**, **brand_name**, **price_usd**, **secondary_category**, **tertiary_category** and **highlights**. The particularity of the latter is that there is no column named 'highlight'. The first six highlights of each interaction are extracted and used as variables: columns named **highlight1** to **highlight6**. And to be compatible with the model, they are all encoded as described in the section 3.

The target variable is the **'rating'** corresponding to the score given to a product by a user. **Figure 2** gives an overview of the distribution of ratings. The aim of the model is to predict this rating for products not purchased and rated by customers, based on products and customers characteristics.

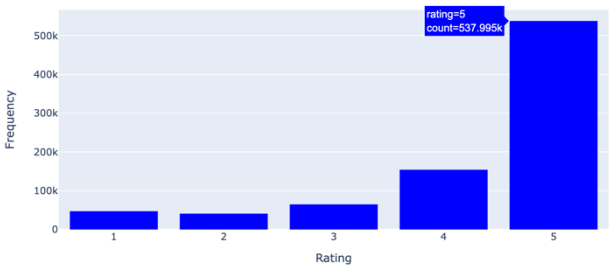

***Figure 2** - Products Rating Distribution*

4.3.2 MovieLens

The characteristics selected for training are as follows: **userId**, **movieId**, **genres** and **tags**. Each **genre** is a separate column, with a binary value indicating whether the film belongs to that genre or not. So, we can't find the 'genre' column but 'Sci-Fi', Thriller' and 17 other columns. And the first five **tags** of each interaction are extracted and used as variables: tag1 to tag5.

The target variable 'rating' is the score given to a film by a user. **Figure 3** gives an overview of the distribution of ratings. The aim of the model is to predict this rating for films not seen by users, based on films and users' characteristics.

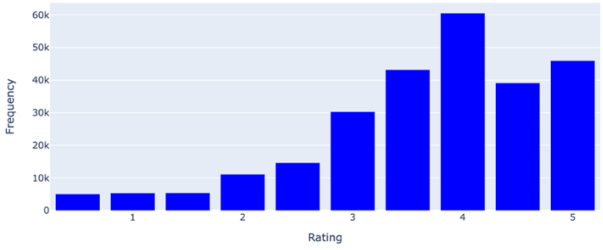

***Figure 3** - Movies Rating Distribution*

4.3.3 Food.com

The features selected for training correspond to all the features in Tables 6 and 7, except for "name" and "grade". It is important to note that the tag and ingredient columns each correspond to a set of tags and a set of ingredients. The first ten tags and ingredients of each user-recipe interaction are extracted and used as variables: columns named tag1 to tag10 and ingredient1 to ingredient10.

And the target variable 'rating' is the score given to a recipe by a user. **Figure 4** gives an overview of the distribution of ratings. The aim of the model is to predict this score and, consequently, to determine whether a recipe is recommendable.

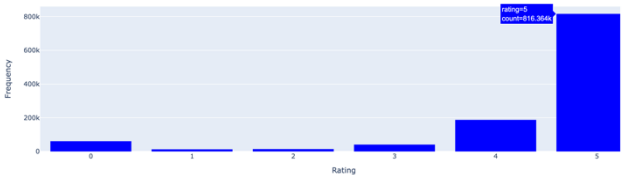

***Figure 4** - Recipes Rating Distribution*

5 Results

5.1 Sephora

First, we can examine a sample of the actual and predicted value data in **Figure 5** and then, for greater precision and guidance, we can analyze the various metrics obtained during model training. The measurements are presented in Table 8, **Figure 6** and **Figure 7**.

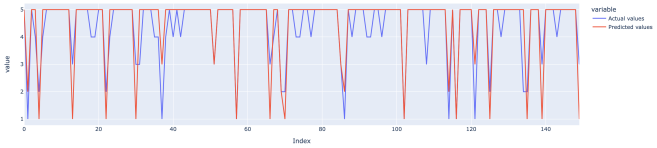

***Figure 5** - Actual vs Predicted Value Rating Products*

At first glance, **Figure 5** shows that most of the time the model predicts the class to be either 1 or 5, with 5 being more prominent. This is also illustrated in the confusion matrix (**Figure 6**), where we can see that 1 and 5 are the most predicted rates.

Accuracy	Balanced Accuracy	Macro-averaged Precision
70%	42%	48%
Multi-Class Precision		
[0.41930882 0.32772414 0.42606955 0.45884413 0.75915517]		

***Table 8** – Sephora Model Training Performance Metrics*

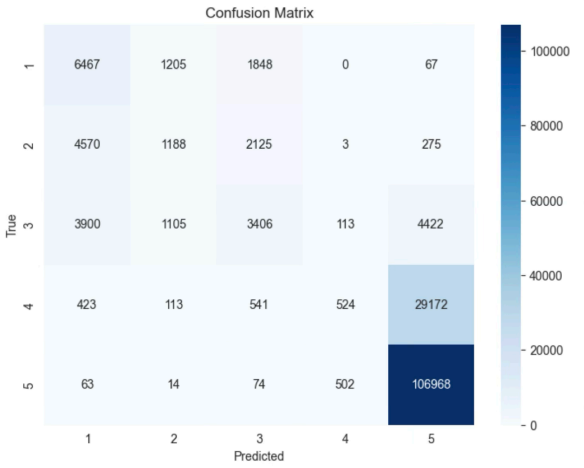

***Figure 6** – Confusion Matrix Skin-Care Products*

	precision	recall	f1-score	support
1	0.42	0.67	0.52	9587
2	0.33	0.15	0.20	8161
3	0.43	0.26	0.33	12946
4	0.46	0.02	0.03	30773
5	0.76	0.99	0.86	107621
accuracy			0.70	169088
macro avg	0.48	0.42	0.39	169088
weighted avg	0.64	0.70	0.62	169088

***Figure 7** – Classification Report Products*

Accuracy indicates the total proportion of correct predictions across all classes. This measure is high here (70%) which shows that the model works well **overall**. But this high accuracy masks an imbalance between classes, with the majority predicted correctly, but the minority neglected. The imbalance is caused by class 5, which represents a large majority (64% of ratings), as shown in **Figure 1** and the classification ratio support value in **Figure 7**. Indeed, the **Balanced Accuracy** metric (42%) gives a better understanding of the under-represented classes. This value reflects the model's average performance for the minor classes: 1, 2, 3 and 4.

The **Macro-Average precision** of 48% also confirms that the model is ineffective for minor classes. If we look at the **multi-class precision** scores, we can see that they are high for class 5 (76%), which contrasts with classes 1, 2 and 3, where accuracies are lower (between 33% and 46%).

The **Recall value** for class 5 is almost perfect (99%): the model detects almost all instances of class 5. For class 1, 67% of ratings were correctly detected. But for the other classes, these values are less than 26%. Only 2% of the predictions in class 4 were well classified, even though this value is the 2nd most awarded. I suppose this is because **class 4 is close to class 5**, so all the predictions were probably classified in class 5 rather than 4. Classes as close as 4 are often absorbed by the dominant class. This can also be seen in the confusion matrix, where 29172 values that should have corresponded to class 4 were instead rated as 5. The same is true of class 1, which is rated slightly higher than class 2. And class 3 is in the middle, so either the class is finally predicted as 1, or 3, or 5. It's balanced between the 3.

5.2 MovieLens

First, we can examine a sample of the actual and predicted value data in **Figure 8**, then, as in the previous analysis, we'll look at the various metrics obtained (Table 9, **Figure 9** and **Figure 10**) during model training.

***Figure 8** - Actual vs Predicted Value Rating Movies*

Accuracy	Balanced Accuracy	Macro-averaged Precision
27%	20%	23%

Multi-Class Precision

[0.30846774 0.18987342
0.13782991 0.20306513
0.17115385 0.21885522
0.24631607 0.26506528
0.22725173 0.35217438]

***Table 9** – MovieLens Model Training Performance Metrics*

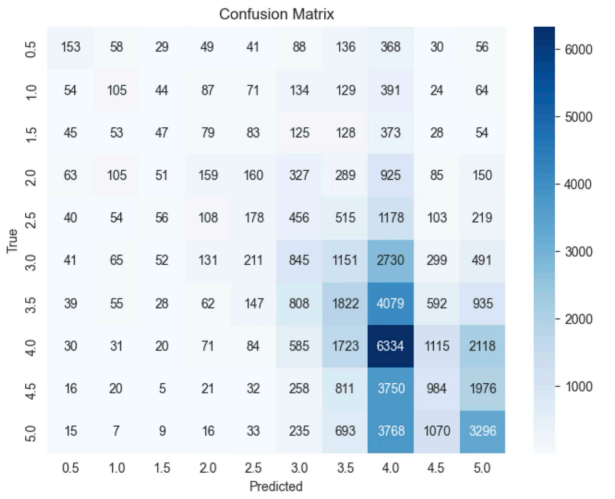

***Figure 9** – Confusion Matrix Movies*

	precision	recall	f1-score	support
0	0.31	0.15	0.20	1008
1	0.19	0.10	0.13	1103
2	0.14	0.05	0.07	1015
3	0.20	0.07	0.10	2314
4	0.17	0.06	0.09	2907
5	0.22	0.14	0.17	6016
6	0.25	0.21	0.23	8567
7	0.27	0.52	0.35	12111
8	0.23	0.12	0.16	7873
9	0.35	0.36	0.36	9142
accuracy			0.27	52056
macro avg	0.23	0.18	0.19	52056
weighted avg	0.25	0.27	0.24	52056

***Figure 10** – Classification Report Movies*

To clarify the classification ratio, the values start from 0 to 9, where 0 corresponds to a score of 0.5, 1 to 1, 2 to 1.5, and rising from 0.5 to 5.

In this experiment, class 4 emerges as the dominant category (**Figure 3**). Consequently, neighboring classes between 3 and 5 are often absorbed by class 4, resulting in significant overlap and confusion. This is probably because the distinctions between these classes are minimal in the dataset.

Low ratings such as 0.5, 1.0 and 1.5 and intermediate scores 2.0, 2.5 and 3 show very low recall (<15%) in **Figure 10**, indicating that the model often overlooks these classes. Similarly, class 4.5 is under-represented, as it is overshadowed by the dominant neighboring classes 4 and 5. The best recall is obviously for class 4, at 52%.

One explanation for these unsatisfactory results may lie in the limited number of features. Genres and a limited subset of tags limit the model's ability to accurately identify user preferences. Unlike the Sephora dataset, which contains 3 columns for product targeting (categories, brand and price), here we only find the gender. What's more, while the tag column contains information such as dates, actors and themes, it does not contain sentiment data. This lack of emotional or qualitative context limits the model's ability to fully capture the complexity of user preferences.

5.3 Food.com

We can examine a sample of the actual and predicted value data in **Figure 11**:

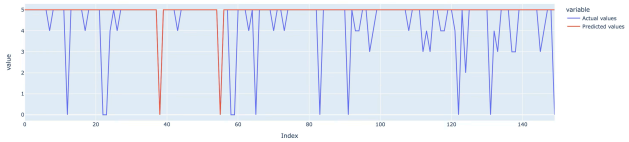

***Figure 11** - Actual vs Predicted Value Rating Recipes*

Then, we can analyze the various metrics shown in Table 10, **Figure 12** and **Figure 13**.

Accuracy	Balanced Accuracy	Macro-averaged Precision
73%	17%	31%

Multi-Class Precision
[0.38249595 0.4 0. 0. 0.37209302 0.72275669]

***Table 10** – Food.com Model Training Performance Metrics*

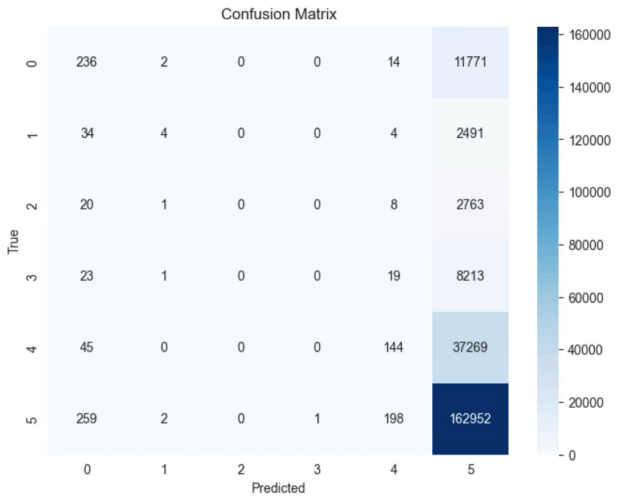

***Figure 12** – Confusion Matrix Recipes*

	precision	recall	f1-score	support
0	0.38	0.02	0.04	12023
1	0.40	0.00	0.00	2533
2	0.00	0.00	0.00	2792
3	0.00	0.00	0.00	8256
4	0.37	0.00	0.01	37458
5	0.72	1.00	0.84	163412
accuracy			0.72	226474
macro avg	0.31	0.17	0.15	226474
weighted avg	0.61	0.72	0.61	226474

***Figure 13** – Classification Report Recipes*

The prediction sample (**Figure 11**) shows us directly that the predicted values are mainly 5. Like the other dataset a class is dominante. Looking at **Figure 12**, we can see that none of classes 3 and 2 were predicted, and almost all instances were assigned to class 5.

Here we find a **high accuracy (72%)** that might be considered correct but is not. This is largely due to the over-representation of class 5, which is always predicted correctly: perfect **recall** (100%) and a **high F1-score** (84%). This reflects the massive bias towards this class, which constitutes a large part of the data. Recall is close to 0% for all other classes, showing that the model almost never succeeds in correctly identifying these classes.

Around 72% of the data belong to class 5, as shown in **Figure 4**. This strong predominance encourages the model to learn primarily to recognize this class, to the detriment of the others.

We can say that the features selected and used do not appear to enable the model to effectively differentiate between grades 0 and 4. This may be due to a lack of diversity or relevance in the explanatory variables. For example, 'minutes' is used to capture certain preferences, but it is not always correlated with user satisfaction. Secondly, by selecting only the top 10 tags without assessing their relevance, key information was probably overlooked. Similarly for ingredients, limiting the analysis to the first 10 ingredients for each recipe risks underestimating the importance of ingredient combinations.

The model therefore struggles to correctly predict less frequent classes, probably due to over-learning on the majority class (score 5) and a lack of data on minority classes.

6 CONCLUDING REMARKS

The results of this study demonstrate XGBClassifier's ability to predict user preferences in different domains, while highlighting the challenges associated with minority class management and data richness. A major observation lies in the differences between the domains analyzed, which directly influence the model's performance. In the film domain, users consume a wide variety of content, facilitating a more balanced distribution of scores. This variability, while beneficial, is limited by the quality of the data available. Explanatory columns, such as genres and tags, lack depth and do not incorporate the emotional or contextual dimensions that could enrich predictions. On the other hand, in the product and recipe domains, the predominance of high scores reflects a behavioral bias where users anticipate their satisfaction even before purchasing a product or making a recipe. These biased behaviors, combined with unbalanced data and limited explanatory features, restrict the model's ability to differentiate minority ratings.

To face these challenges, several ideas have been identified, such as using an approach centered on global grade averages. In other words, replacing individual assessments with averages could reduce noise and enable more accurate analysis of general trends. A reformulation of the problem as a regression task would be more appropriate in some cases. Alternatively, an analysis of textual reviews with NLP would capture the emotional and contextual nuances expressed in textual reviews. This could enrich the explanatory columns and offer more accurate and meaningful predictions. And finally, a rebalancing of the data is necessary to mitigate the dominance of majority classes, or an enrichment of minority classes, through oversampling techniques or data transformations may be an alternative solution, would improve overall performance.

References

[1] Daniel Gaspar-Figueiredo, Marta Fernández-Diego, Ruben Nuredini, Silvia Abrahão and Emilio Insfrán. 2024. Reinforcement Learning-Based Framework for the Intelligent Adaptation of User Interfaces. *arXiv:2405.09255*

[2] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. *arXiv:1603.02754*

[3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. *arXiv:1907.10902*

[4] Moshe Sipper. 2022. High Per Parameter: A Large-Scale Study of Hyperparameter Tuning for Machine Learning Algorithms. *arXiv:2207.06028*

[5] Juan Terven, Diana M. Cordova-Esparza, Alfonso Ramirez-Pedraza, Edgar A. Chavez-Urbiola and Julio A. Romero-Gonzalez. 2024. Loss Functions and Metrics in Deep Learning. *arXiv:2307.02694*

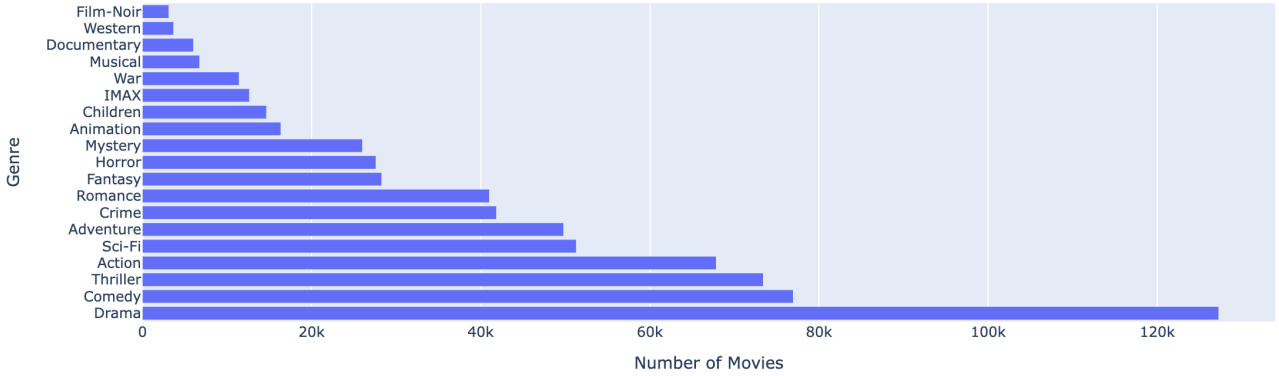

Figure 14 *- Genres distribution*

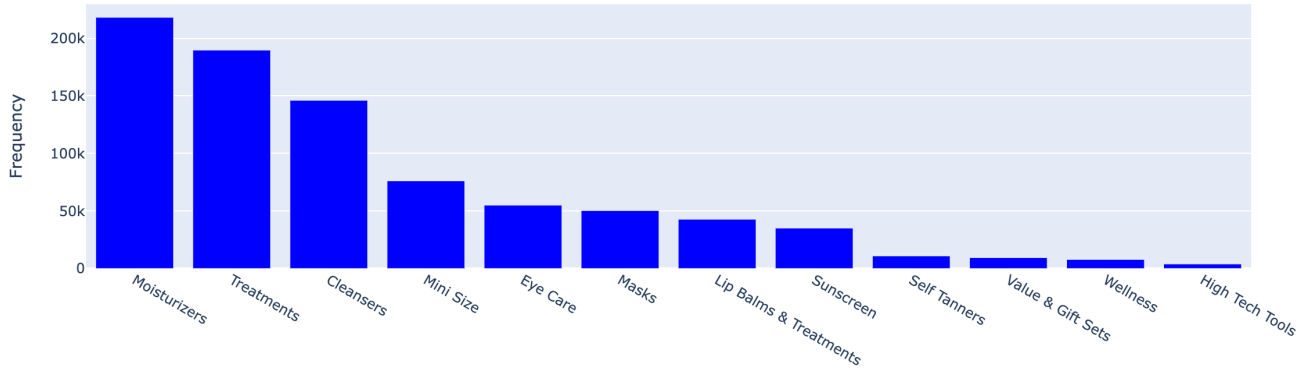

Figure 15 *- Secondary Category Distribution*

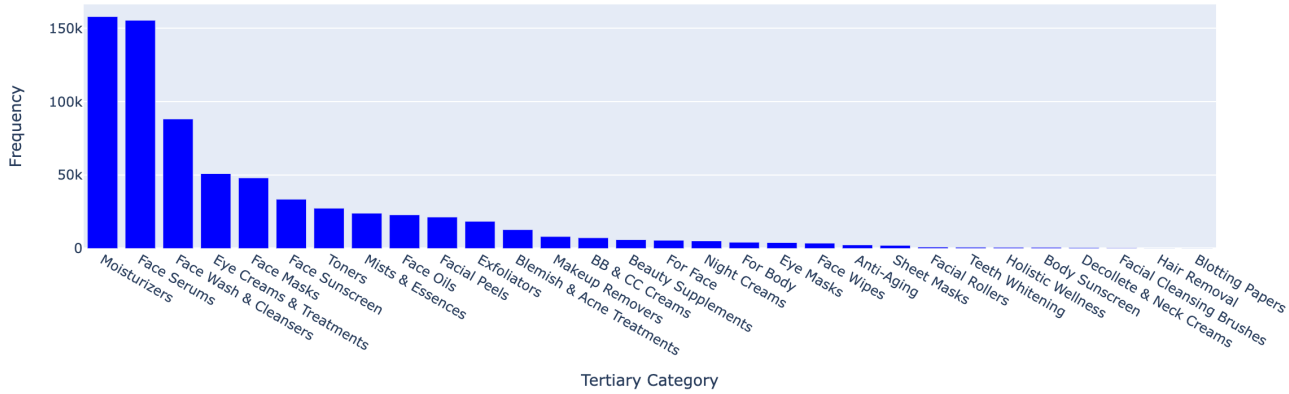

Figure 16 *- Tertiary Category Distribution*

Figure 17 *- Brand Names WordCloud*

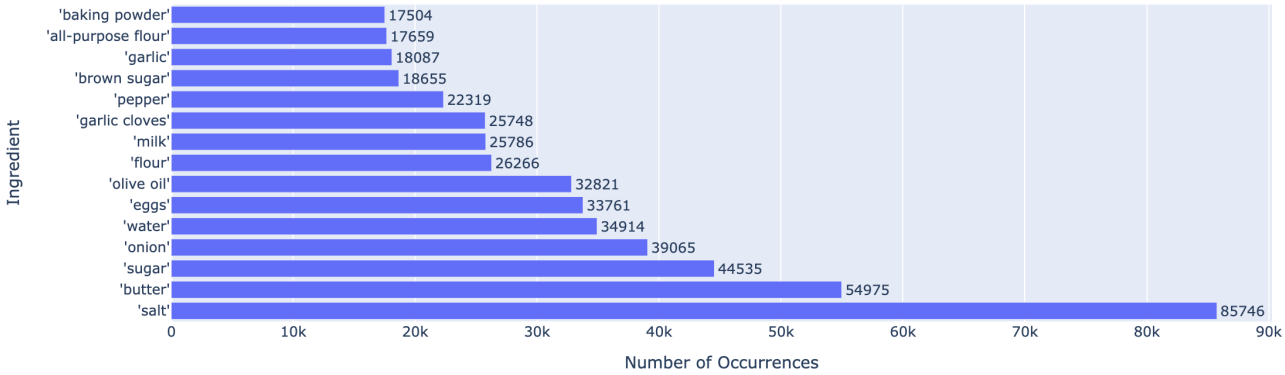

Figure 18 *- Top 15 Ingredients Before Columns Creation*

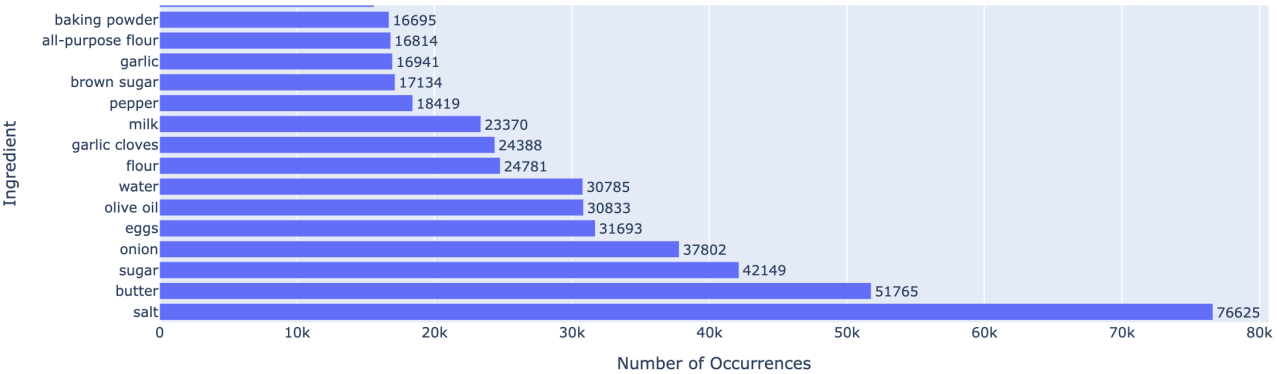

Figure 19 *- Top 15 Ingredients Final Dataframe*