POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Data-Driven Recommendations for H&M: A Study on Candidate Generation and Boosting Trees

Laurea Magistrale in Computer Engineering - Ingegneria Informatica

Author: Arcangelo Pisa

Advisor: Prof. Paolo Cremonesi

Co-advisor: Fernando Benjamín Pérez Maurera

Academic year: 2021-2022

1. Introduction

The application of recommender systems in fashion has proven to be an effective way to improve customer satisfaction, increase sales, and create a more engaging shopping experience [1]. Fashion is a highly subjective and dynamic domain [8], where personal taste and fashion trends change rapidly [5]. The thesis work started from a Kaggle Challenge[1] proposed by the H&M brand and aims to explore transaction, customer and article datasets of H&M and build effective recommender systems in that fashion domain. Overall, it contributes to the growing body of knowledge on recommender systems in the fashion industry and provide valuable insights and recommendations for practitioners and researchers working in this field. In the beginning, we participated as a team composed of three students supported by a Ph.D. Student. Past the competition period, I conducted extensive research on fashion-based recommenders and focused the work on proposing a novel and light fashion-based recommender that outperforms the state-of-the-art and publicly available top-scores recommender. We present the state of the art along with the technologies and popular techniques used in the fashion domain; than we analyse the dataset highlighting pattern in the data that we used to build features and candidate generation strategies. Those strategies, built considering common behaviour in the fast fashion domain e.g., repurchase and seasonality, helped to lower the list of possible articles from which the final model take the recommendation. We describe three different experiments we conducted, i.e., *collaborative filtering models*, *heuristics*, and *two-stage recommender*. Than we presented the final gradient boosting decision tree algorithm used, i.e., *Catboost*, along with the results obtained.

[1] Challenge overview `https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations/overview`

2. Our Approach H&M Competition

In this competition H&M invited to develop product recommendations based on data from previous transactions, as well as from customer and articles meta data. The organizers provided a dataset composed by transactions, i.e., the action of buying an item, from *2018-09-20* to *2020-09-22*, made both in the online and physical

stores. Together with the transactions they provided the list of all the users and articles along with their attributes, analysed in the following sections. **The goal of the challenge** was to recommend a list of 12 items to each user in the dataset. The recommendations are scored using the $MAP@12$ with respect to the test week, i.e., the first one after the dataset period, which goes from $2020-09-23$ to $2020-09-30$. The public leaderboard was available during the whole challenge period and is generated using only the 5% of the total test data. The private leaderboard is generated using the other 95% of the test data.

2.1. Articles

Article dataset is composed of 25 features, where 14 of them are text features e.g., *article id* and *product code*, the remaining 11 are numerical features e.g., *garment name* and *description*. There are 105.542 articles in the dataset.

2.2. Customers

This database contains information about the customers. It is composed of 7 features: 4 of them are text features, e.g., *postal code*, *fashion news frequency*, *customer id* and *club member status*, the remaining 3 are numerical features, e.g., *fashion news*, *active to receive newsletter* and *age*. There are 1.371.980 customers in the dataset. *Fashion news* value is missing for 895.050 customers: this means that they are customer not registered online but that they use to buy directly in physical stores. The same apply for *active* attribute. There are no missing values for the *postal code* of the customer, meaning that there is the location information available for each customer. There are a lot of spikes in the number of transaction during holidays or weekend but the usual number of transactions made each day lays in range of 25.000 and 80.000.

2.3. Transactions

This database contains information about the transactions. It is composed of 5 features, where 2 of them are text features, e.g., *customer id* and *article id*, the remaining 3 are numerical features, e.g., *date*, *price* and *sales channel*, i.e., in store or online. There are 31.788.324 transactions in the dataset. No transaction contains missing values.

2.4. Product Sales Seasonality

Apart from some exceptions e.g., accessories and bags, most of the articles sell well depending on the season: articles that sell well in late September represent good candidates to be recommended for the test week, which corresponds to the last week of September i.e., from *2020-09-23* to *2020-09-30*. We check the trend of sales by grouping articles into categories; each category is identified by the unique combination of three of the attributes already available in the article's dataset, i.e., *index group*, *index*, and *product type*. Once generated these categories we plotted for each of them the percentage of monthly sales with respect to the total sold amount of the same category during the entire dataset period of two years. The fig. 1 shows an example of categories that sell better during the autumn season.

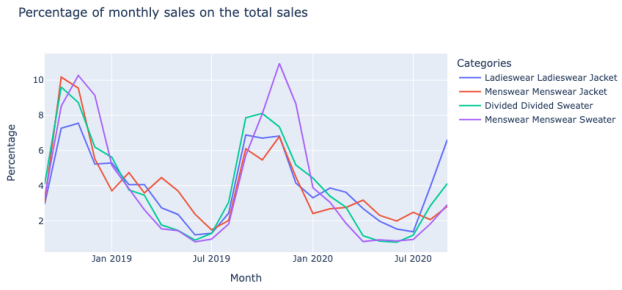

Figure 1: The categories in the legends are only some examples, and are generated combining i.e., *index group*, *index*, and *product type*. *Jackets* and *sweater* sells better during the approaching of winter seasons. Then this value decrease, and is at its lowest during summer.

Gaussian mixture has been used to cluster the categories into 4 groups that represents different types of trends in the monthly sales, over the two years of transactions. Those cluster are not connected to a specific season, they only represents different trends. Anyway, the categories belonging to the fig. 1, that fall into the cluster type 2, are strictly connected to the autumn/winter season. In the same way, the categories that fall into the cluster type 3, are strictly connected to the sprint/summer season. Categories that belong to cluster 0 have not a strong correlation with a specific season of the year. The category *divided divided bracelet* belongs to the cluster 1. This represent a totally different trend

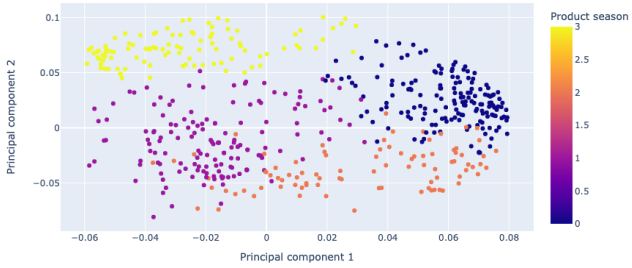

Figure 2: Product seasonal type represent the cluster, between 0 and 3, to which a category belongs to. Categories that belong to cluster 0 have not a strong correlation with a specific season of the year. Categories that belong to cluster 2 have a strong correlation with autumn/winter seasons. Categories that belong to cluster 3 have a strong correlation with spring/summer seasons.

where there are some months where sales are equal to zero. This specific category contains accessories, meaning that articles belonging to this cluster are independent from seasons and unavailable during some periods of the year.

2.5. Out of Stock Product

Considering a specific article, if e.g., the sales before 2019 account for more than 95% of the total sales, we can assume that the article is no longer available by the end of 2020 and exclude it from the list of candidates to be recommended. The fig. 3 shows an example.

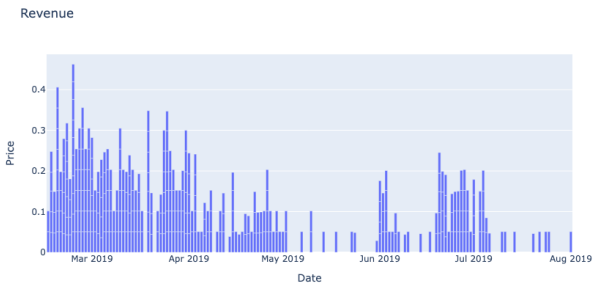

Figure 3: The product, a *swimsuit*, has not been sold anymore after august 2019. We can consider it as no longer available in the stock a remove it from the list of candidates.

2.6. Repurchase

In the fast fashion industry, repurchase behavior is used as a candidate generation technique in recommender systems [6]. Fast fashion companies, H&M is an example, offer a wide range of clothing items that are frequently updated and replaced with new items, and customers may be more likely to repurchase items they have already bought, especially if they are happy with the fit, quality, and style of the item. These items can then be recommended to the user as potential candidates for future purchases. We analysed this behaviour using the available transactions to see is customers decided to buy again the same item, at different level of granularity i.e., same *article id*, *product id* or *category*, i.e., group of items, e.g., t-shirts, that share some characteristics.

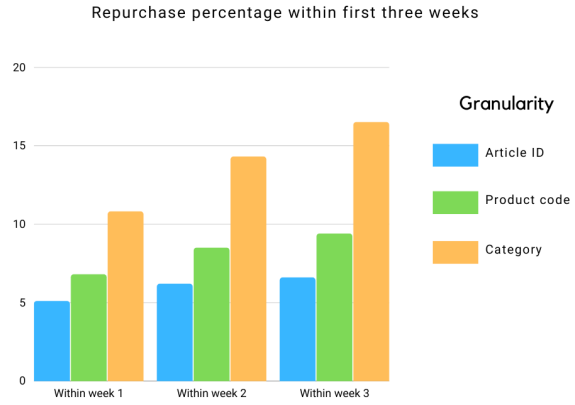

Figure 4: This plot shows the percentage of user that, at least one time during the two years, purchased again an item with the same *article id* (e.g., a blue t-shirt), *product code* (e.g., previous t-shirt of different size or color), and category (e.g., any t-shirt in the dataset). The *category* (see section 2.4) is an attribute generated by unique combination of *index name*, *index group name* and *product group*.

The fig. 4 shows the percentage of customers who repurchased the same item, with exactly the same *article id* (e.g., a blue t-shirt), *product code* (e.g., previous t-shirt of different size or color), and category (e.g., any t-shirt in the dataset). The results obtained shows that there is an increase of percentage repurchase if we increase the time window or the granularity. Con-

sidering the *article id* 5.1% of the customers repurchased the same item within 1 week, 6.2% within 2 weeks and 6.6% within 3 weeks. Instead, considering the *product code* 6.8% of the customers repurchased the same item within 1 week, 8.5% within 2 weeks and 9.4% within 3 weeks. Regarding instead the *category* 10.8% of the customers repurchased the same item within 1 week, 14.3% within 2 weeks and 16.5% within 3 weeks. This percentage increase let us consider repurchasing as candidate generation strategy.

3. Experimental methodology

Our dataset is large and complex with a wide range of attributes and features that need to be preprocessed and cleaned before being used for training and evaluation. First of all we handle missing information in the article and customer datasets by filling them with a 0, i.e., *fashion news newsletter* or *NULL*, i.e., *club member status* and *fashion news frequency*. For all the experiments we use random sampling in order to create 4 different samples, containing different percentages of all the available transactions: 0.1%, 1%, 10% and 100%.

3.1. Hyper-Parameter Tuning

We performed *Bayesian Optimization* using Optuna [2]. Optuna provides a simple and flexible API to define the search space, setting up the objective function, and configure the optimization process. It also supports distributed optimization, visualization of the results, and integration with several machine learning libraries.

3.2. Data Split

We considers only the last 7 weeks of the dataset. The first 6 are used to tune and trains models; the reason is that these weeks share the same context, e.g., weather, products available and fashion trends, with the test week, i.e., the first week after the dataset period. The last week of the dataset is used as validation to validate our model. Since we have no access to test week, we validate our models with the validation week before submitting the recommendations and evaluate our solution.

3.3. Collaborative Filtering Recommenders

In our baseline experiments, we evaluate the accuracy of several collaborative filtering models, including user-based and item-based neighborhood, matrix factorization, graph based, top popular and hybrid models. We perform a variety of experiments to establish the effectiveness of these models, which in previous research works have shown to be competitive and strong baselines in terms of recommendations' quality. The techniques evaluated in this experiment cover a wide variety of recommenders, ranging from non-personalized (top-popular items in august 2020 and top-popular recommenders in September 2020), matrix factorization (PureSVD, ALS), graph-based (P3 Alpha, RP3 Beta), collaborative filtering (ItemKNN, User KNN). We also evaluate hybrid models making an ensemble of models, i.e., top popular, P3Alpha and ItemKNN CF. Recommenders in this experiments are trained with the *train* split of the ICM and URM obtained in the processing phase. URM is a matrix that represents the interactions, e.g., transactions (customer purchases).

3.4. Heuristics

With heuristics and association rules we exploit the outcome of the data analysis made in section 2.4, e.g., seasonality, out of stock products, repurchase, co-occurrence, among the others. This last heuristic represents the idea beyond outfits: users buy two or multiple items together because they compose an outfit recommended by the fashion company itself of because it has been seen somewhere else online. Several experiments are inspired to the concept of fast-fashion [6]. We tested $\sim$ 50 models combining heuristics and associations rules involving consideration on trendy color, new products, ages and top popular items. We tested different combination of trends to see the variation of accuracy and highlights which heuristic the dataset is biased to.

3.5. Two Stage Recommender

During the last experimental phase we apply the two stage recommender. These systems generate recommendations in two phases: first, multiple nominators select a small set of items from

a large pool using cheap-to-compute item embeddings, i.e., candidate generation strategies used to down-sample the list of item; then, a ranker with a richer set of features rearranges the nominated items and presents them to the user. We generate candidates, using strategies listed in the next section, starting from the results of section 3.4. We than add features listed in section 3.5.2 to these candidates. For the model, instead, we make experiments with different *GBDT algorithms*. One of the goals during that phase of experimentation has been to find heuristics which gave good results and use them to generate a pool of candidate to associate to each single user. This has been an important steps because having a pool of $\sim$ 300 candidates for each user means that the final model has to select the recommendations from a lower list of possible items instead of picking them from a pool of $\sim$ 106.000 items for each user. Lowering the pool of candidates from which the model need to select items to recommends increase the model accuracy and the effectiveness of the solutions. We tried different *GBDT algorithms*, e.g., *LightGBM Ranker* and *Catboost* among the others.

3.5.1 Candidate Generation

The goal of candidate generation is to identify a set of items that are likely to be of interest to the user in order to lower the pool of articles from with the final model have to pick up recommendations. The list of candidate generation methods allow us to generate a pool of $\sim$ 300 items for each user. **Once generated, those candidate pools have been used for all of our final experiments**. The strategies adopted are: *repurchase*, *Item-to-item CF*, *popular items*, *age based*, *popularity per department*, *same product code* and *co-occurrence*.

3.5.2 Feature engineering

The goal of feature engineering in fashion recommender systems is to extract the most relevant and informative features from the available data, in order to enable accurate and personalized recommendations to users. I create more than **100** features. We filtered out not informative features for the model and we come up with the following list of features:

1. **User attributes**: *age*.
2. **Item attributes**: *product type number*, *product group name*, *graphical appearance*, *color group code*, *perceived colour value*, *perceived colour master*, *department number*, *index code*, *index group number*, *section number*, *garment group number*.
3. **User featuuhres**: *mean* and *standard deviation* for *prices* and *sales channel id* for all of his transactions.
4. **Item features**: *mean* and *standard deviation* for *prices* and *sales channel id* considering all transactions of that item.
5. **User-Item features**: are the *mean* and *standard deviation* of the *age* of all the users who buy that item.
6. **Item freshness features**: first day the item appears a transaction.
7. **Item volume features**: the number of times the item appear inside the dataset.
8. **User freshness features**: first day he made a transaction.
9. **User volume features**: the number of transactions made by the user.
10. **User-Item freshness features**: the first time the pair appears in a transaction.
11. **User-Item volume features**: the number of times the user-item pair appears inside the dataset.
12. **Item age ranges features**: the age range of people that most likely buy that item.

Using simple features allow to create an effective and lightweight recommender while fastening the training time. This has been possible thanks to the extensive work done on dataset analysis and on building effective candidate generation strategies to down-sample the pool of items associated to each single user of the dataset.

3.6. Resources

To run our experiments we use two cloud computing platforms: Google Colab[2] and Amazon AWS[3]. Google Colab provides a virtual machine with 1 or 2 cores CPU, an NVIDIA Tesla K80 or T4 GPU and 27 GB of RAM. For *AWS* we used the **m6g.16xlarge**[4]. That instance has 64 CPU cores and 256 GB RAM.

[2]Google Colab `https://colab.research.google.com`

[3]Amazon AWS `https://aws.amazon.com`

[4]Amazon EC2 M6g Instances `https://aws.amazon.com/ec2/instance-types/m6/`

4. Results

In the table 1 are listed all the results we got with the 3 different experimental methodologies, i.e., *collaborative filtering recommenders*, *heuristics* and *two-stage recommenders*, both for the private and public leaderboard.

In our strong baseline experiments, we evaluate the accuracy of several collaborative filtering models. The recommenders have low recommendation accuracy with respect to more sophisticated and tailored recommenders, e.g., the one used by the team obtaining the 1st place in the competition. Strong baseline models do not obtain high accuracy in this domain, being less competitive than the best solutions of the challenge. Also, a non-personalized recommender, i.e., *top popular items*, obtains higher accuracy than such baselines. ALS is a strong baseline and obtains the highest accuracy of personalized collaborative filtering recommenders. However, its accuracy is lower than non-personalized approaches. Top Popular Items on August/September 2020 measure whether purchased items in the competition are most popular during the season. We considered only August and September because they are the two months in the same season as the target purchases. The obtained score is lower than the one obtained recommending the top popular items from the entire dataset. These results suggest that the trend of sold items during the test week is not connected with items sold during the same season, i.e., summer of 2020. An additional proof is that the accuracy of Top Popular Items on August/September 2020 is lower that the accuracy of Top Popular Items on September 2020 by 6%.

With heuristics and association rules we exploit the outcome of the data analysis of users' behaviors made in section 2, e.g., seasonality, out of stock products, repurchase, co-occurrence, among others. *Association rules* consider how often two items were sold to the same user among the two years of available dataset. The *co-occurrence* heuristic represents the idea beyond outfits and follows the same idea of the association rule: users buy two or multiple items together because they compose an outfit. Several recommenders are inspired by the concept of fast-fashion [6], i.e., recommender selecting only trendy colors, new products, popularity based on age, repurchases, and co-occurrences. We also combine two or more heuristics or association rules into a single recommender. This new recommender selects items that are selected by each heuristic or association rule. The best recommender in this experiment obtains a relative improvement of 60% with respect to the most accurate personalized collaborative filtering recommender.

In the last experiment, we design, develop, and evaluate a two stage recommender. These systems generate recommendations in two phases: first, multiple nominators select a small set of items from the catalog using lightweight item embeddings, i.e., candidate generation strategies used to down-sample the list of items. Second, a *GBDT* ranker with a richer set of features rearranges the nominated items and presents them to the user. The *GBDT* model that gives better results is **Catboost**; its accuracy is higher than all heuristics and association rules by 34 and 42 %. Also, its accuracy is higher than all collaborative filtering recommenders, with a difference of 41% with respect to top popular 12 items and 114% with respect to ALS. Our most accurate recommender obtains a score of 0.0298 in the private leaderboard, meaning it obtains the 10 place in it. For comparison, the best baseline sits in the 1445 place, the most accurate heuristic sits in the 1260 place, and the most accurate association rule sits in the 1311 place.

Table 1: The table presents 4 sections: the first section contains the score of the first 3 winning solutions as reference points. We analyse these 3 solutions because they train the same *GBDT model*, but present different pipeline's structure and innovative techniques, e.g., *"Two Tower MMoE"*, *SWIM transformer* and *sentence transformer*, among others. Solutions that ranked at 2nd and 3rd place provide a similar pipeline and differ from the 1st because of recall methods and engineered features. The second section contains the results of the experiments made with traditional models. The third section contains the results of the experiments done with heuristics and association rules. The last section shows the result of our best two-stage recommenders, i.e., *GBDT Catboost with YetiRank*, *GBDT LightGBM Ranker*.

Algorithm	Public	Private
First place	0.03716	0.03792
Forth place	0.03544	0.03563
Fifth place	0.03536	0.03553
Top popular 12 items	**0.02163**	**0.02119**
Top popular items on September 2020	0.00407	0.00384
Top popular items August/September 2020	0.00383	0.00362
P3Alpha	0.00431	0.00426
RP3 Beta	0.00425	0.00453
ALS	**0.01413**	**0.01406**
PureSVD	0.00431	0.00426
Item KNN CF	0.00345	0.00357
Ensemble: Top popular 12 items, P3Alpha and ItemKNN CF	0.00457	0.00463
Heuristic Trendy color and top popular items	0.00613	0.00642
Heuristic Age, trendy color and top popular items	0.0064	0.00676
Heuristic Top popular new items	0.0064	0.00676
Heuristic Recommend again product bought last 3 weeks	0.01854	0.0185
Heuristic Trending product based on repurchasing trend	**0.02263**	**0.02291**
Association rule Top popular items based on age and discounted products	0.01478	0.01482
Association rule Top popular items based on age	0.01949	0.01962
Association rule Top popular items from most popular product category (*Trousers*, *Sweater*, *Cardigan*)	0.01973	0.01992
Association rule Items purchased together	**0.02169**	**0.02159**
Two-Stage Recommender with LightGBM	0.0286	0.0279
Two-Stage Recommender with Catboost	**0.0303**	**0.0298**

5. Conclusions

This thesis provides a comprehensive overview of the H&M challenge and presents several solutions to it, the last one ranking among the top 10 in the final leaderboard. The presented solution is a lightweight and scalable model that requires few resources and training time. The best recommender takes 15 hours to train on a 64-core CPU virtual machine using 256 GB RAM. Despite the limited resources needed, our final models yielded competitive results. For future works, we suggest several action items that can be taken in the future to improve the score, e.g., increasing resources and testing models with longer time frames, i.e., to consider as train period more than 6 weeks. Another future direction is to build embedding of items' images and descriptions to boost the effectiveness of the model.

The fashion recommender system developed in this thesis has the potential to enhance the shopping experience for H&M customers by providing personalized and accurate fashion recommendations. Additionally, the system can help H&M to increase customer engagement and loyalty, as well as boost sales and revenue.

In the fashion area, it is recommended to retrain the model after some time, as fashion trends and user preferences can change over time [4, 7]. Not retraining recommenders can lead to a decrease in their accuracy in future interactions with users. In addition, new products may be added to the inventory, which can affect the recommendations made by the model. Many fashion companies are actively retraining their recommender systems to ensure they remain up-to-date and effective. For example, a European online fashion retailer, updates its recommendation algorithms every two weeks, based on customer feedback and new data. Similarly, a US-based online personal styling service updates its algorithms every few weeks to keep up with changes in customer preferences and fashion trends [3].

One limitation of this thesis is that the third-party system evaluating each solution does not simulate an environment when recommenders are retrained. As participants in the competition, we do not have access to the entire dataset, hence it is not possible to reproduce such an evaluation methodology. Even after the deadline it has been not possible to access the entire dataset and the only way to test our solutions has been to submit it on the system.

References

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 2623–2631. ACM, 2019.

[3] Chandadevi Giri and Yan Chen. Deep learning for demand forecasting in the fashion and apparel retail industry. *Forecasting*, 4(2):565–581, 2022.

[4] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[5] Anna Kristjánsdóttir. Moving from fashions to a continuous stream of change: teacher development and IT. In David Tinsley and David C. Johnson, editors, *Information and Communications Technologies in School Mathematics, IFIP TC3/WG3.1 Working Conference on Secondary School Mathematics in the World of Communication Technology: Learning, Teching, and the Curriculum, 26-31 October 1997, Grenoble, France*, volume 119 of *IFIP Conference Proceedings*, pages 165–168. Chapman & Hall, 1997.

[6] Xiaoyang Long and Javad Nasiry. Sustainability in the fast fashion industry. *Manuf. Serv. Oper. Manag.*, 24(3):1276–1293, 2022.

[7] McAfee R. Preston and Brynjolfsson Erik. Hierarchical bayesian modeling of consumer choice with limited information. *Marketing Science*, 31(2):321–340, 2012.

[8] Miroslav Rac, Michal Kompan, and Mária Bieliková. Preference dynamics and behavioral traits in fashion domain. In *14th International Workshop on Semantic and Social Media Adaptation and Personalization, SMAP 2019, Larnaca, Cyprus, June 9-10, 2019*, pages 1–5. IEEE, 2019.

POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Data-Driven Recommendations for H&M: A Study on Candidate Generation and Boosting Trees

Tesi di Laurea Magistrale in
Computer Engineering - Ingegneria Informatica

Author: **Arcangelo Pisa**

Student ID: 947645
Advisor: Prof. Paolo Cremonesi
Co-advisor: Fernando Benjamín Pérez Maurera
Academic Year: 2021-2022

Abstract

The rise of e-commerce has transformed the retail landscape and has created a wealth of opportunities for customers and businesses alike. Customers now have access to a vast array of products and services from the comfort of their own homes. Businesses are able to reach a wider audience than ever before. However, this abundance of choices can also lead to information overload and result in a less-than-optimal shopping experience for customers. To overcome this, companies are constantly looking for ways to improve the customer experience and to provide personalized recommendations that are tailored to each individual customer's preferences, behaviors, and feedback. Recommender systems analyze past customer behavior and preferences to suggest new products that are likely to interest them. They have become an essential tool for companies in the e-commerce industry as they provide a convenient, time-saving, and personalized shopping experience for customers. By providing customers with relevant and targeted recommendations, recommender systems can increase customer engagement, satisfaction, and sales. H&M, a leading fashion retailer, has recognized the importance of recommender systems and proposed a Kaggle Challenge to develop product recommendations based on data from previous transactions, as well as from customer and product metadata. The main objective of this Master's thesis is to propose an effective, scalable, and lightweight recommender system to enhance customer experience. The findings of this thesis provide valuable insights into the accuracy and effectiveness of recommender systems to improve H&M's online shopping experience. Past the competition period, I conduct extensive research on fashion-based recommenders and focus the work on proposing a novel and light fashion-based recommender that outperforms the state-of-the-art and publicly available top-scores recommender. I analyse first the dataset to find patterns in the behavior of the customers and build new features. The thesis proposes a list of candidate generation strategies to down-sample the pool of articles to recommend and an effective, fast and lightweight model to enhance sales.[1]

Keywords: recommender systems, gradient boosting, Neural Network, Fashion

[1] Challenge overview here

Abstract in lingua italiana

L'ascesa dell'e-commerce ha trasformato il commercio al dettaglio, creando opportunità per clienti e aziende. I clienti hanno ora accesso a una vasta gamma di prodotti e servizi dal comfort delle proprie case, mentre le aziende raggiungono facilmente un pubblico più ampio. Tuttavia, il sovraccarico di scelte ed informazioni fornisce un'esperienza di shopping non ottimale. Per superare questo ostacolo, le aziende cercano costantemente modi per migliorare l'esperienza del cliente e fornire raccomandazioni personalizzate che si adattino alle preferenze, ai comportamenti e ai feedback di ognuno. I sistemi di raccomandazione analizzano il comportamento e le preferenze dei clienti per suggerire prodotti che siano di loro interesse. Sono diventati uno strumento essenziale nell'e-commerce, poiché forniscono un'esperienza di shopping comoda, time-saving e personalizzata. H&M ha riconosciuto l'importanza dei sistemi di raccomandazione e ha proposto una challenge Kaggle per sviluppare un sistema di raccomandazione a partire dai dati delle precedenti transazioni, dei clienti e dei prodotti. L'obiettivo principale di questa tesi è proporre un sistema di raccomandazione efficace, scalabile e leggero per migliorare l'esperienza del cliente. I risultati di questa tesi forniscono informazioni preziose sull'accuratezza e l'efficacia dei sistemi di raccomandazione per migliorare l'esperienza di shopping di H&M. Dopo la fine della competizione ho condotto un'ampia ricerca sui sistemi di raccomandazione basati sulla moda, proponendo un nuovo ed efficace modello che supera lo stato dell'arte e i sistemi di raccomandazione attualmnete in uso. Ho analizzato il dataset per trovare pattern nel comportamento dei clienti e creare così nuovi metadati. La tesi propone una serie di strategie di generazione di candidati con l'intento di ridurre il pool di articoli da raccomandare e un modello efficace, veloce e leggero per migliorare l'esperienza di acquisto.

Parole chiave: sistema di raccomandazione, alberi decisionali, reti neurali, moda

Contents

1 | Introduction

Recommendation systems have become an integral part of the e-commerce industry with their ability to provide personalized recommendations for users based on their previous interactions with the platform. According to research by Adomavicius and Tuzhilin [1], the use of recommender systems has been shown to significantly improve customer engagement, satisfaction, and retention. In recent years, the fashion industry has also started leveraging the power of recommender systems to offer personalized product recommendations to their customers.

According to a study by Deldjoo et al. [19], the application of recommender systems in fashion has proven to be an effective way to improve customer satisfaction, increase sales, and create a more engaging shopping experience. Fashion retailers use these systems to recommend products that are more likely to resonate with their customers, based on their past purchases, browsing behavior, and other relevant data. This helps retailers to increase sales and create a more personalized shopping experience.

Developing effective recommender systems for fashion presents unique challenges [57, 93]. Fashion is a highly subjective and dynamic domain, where personal taste and fashion trends change rapidly. The fashion industry is also highly visual, and fashion products often have many different visual attributes, making it more challenging to incorporate meaningful features generated starting from base attributes used to categorize users and items inside the internal storage and capture the nuances of customer preferences. Additionally, fashion is a highly seasonal industry, with new trends and products being introduced frequently during each quarter of the year. This means that recommender systems for fashion need to be constantly updated to remain effective if we want to keep the same results. There are many variables that change over time and based on the fashion trends that may alter the effectiveness of the prediction, even for the same user, over time.

Despite these challenges, there has been significant research on developing recommender systems in the fashion domain [61]. Many researchers have proposed various techniques and methods to develop effective fashion recommender systems [8, 43, 67, 67, 110]. These

techniques range from traditional collaborative filtering methods to more advanced deep learning techniques that can incorporate visual features into the recommendation process.

This thesis explores the transactions dataset of H&M and builds effective recommender systems in the fashion domain. Specifically, this work investigates the challenges of developing recommender systems for fashion, reviews existing approaches and techniques, and proposes a novel framework to build an effective fashion recommender system. The proposed framework is evaluated on a real-world dataset, provided by H&M, which contains two years of transactions, both online and in-store purchasing of fashion clothing.

Overall, this thesis contributes to the growing body of knowledge on recommender systems in the fashion industry and provides valuable insights and recommendations for practitioners and researchers working in this field. For instance, how to inspect the dataset and build effective recall methods to reduce the catalog size to a selection of a subset of it. Or spots trends in the transactions that help to build effective features, both related to seasons, customers, and item type, among others. By developing effective recommender systems, fashion retailers can provide a more engaging and personalized shopping experience, which can help them stay competitive in the fast-paced world of e-commerce.

The thesis work started from a Kaggle Challenge proposed by the H&M brand.[1] In this competition, H&M Group invites researchers and practitioners to develop product recommendations based on data from previous transactions (clothing purchases), as well as from customer and product metadata. The available metadata spans from simple features, such as garment type and customer age, to text data from product descriptions to image data from garment images.

In the beginning, we participated as a team composed of three students supported by a Ph.D. Student. Past the competition period, I conducted extensive research on fashion-based recommenders and focused the work on proposing a novel and light fashion-based recommender that outperforms the state-of-the-art and publicly available top-scores recommender.

This thesis is divided into 6 chapters: chapter 2 presents the state of the art, the technologies, popular techniques, and evaluation of Recommender Systems. It also introduces the evaluation metric specifically used to evaluate the submission files of the challenge. Instead chapter 3 presents the H&M competition and it is divided into two parts. First, we present several analyses of the provided dataset. We highlight patterns in the data we use to build features and the final model. It also describes a selection of publicly avail-

[1] Challenge overview `https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations/overview`

able solutions to the competition by top scorers. Chapter 4 introduces the experimental methodology adopted, both during and after the challenge, with the experiments made. It presents three different experiments, i.e., *collaborative filtering models*, *heuristics*, and *two-stage recommender*. It includes a detailed list of the models we built for each experiment, scores obtained, analysis of the accuracy, the list of features, recall methods, and the final *GBDT* model used to generate recommendations. Chapter 5 presents the results obtained from the three experiments. Chapter 6 summarises the main contribution, resource limitations, and future directions of this work.

2 | State of the Art

Recommender systems are software tools designed to recommend items to users based on previous interactions, context, or similarities between items or users. These systems predict products that the users are most likely to purchase or are interested in. Recommenders share the same objective: to recommend related items to users using the system. Different types of recommenders use distinct data sources to accomplish the thesis goal. Also, their effectiveness depends on the data they have available, their granularity, attributes, among others.

In this thesis the word *user* refers to the buyers, both for the online and physical stores and the word *item* refers to the articles available to be purchased. The word *transaction* refers to the specific type of interaction we have, i.e., a user purchasing an item. *Submission* refers to the predictions generated by a model for users in the dataset provided by the challenge. These predictions are in the form of a ranked list of items that the model believes the user is most likely to be interested in.

The next sections provide a summary of different types of recommender systems experimented with during the thesis' work, as shown in the fig. 2.1. They also presents neural network techniques, e.g., *Two tower MMoE*, *gating network*, *SWIM transformer*, *GBDT algorithms*. At the end of the chapter we provide an overview of the evaluation techniques and of the data processing methods used.

2.1. Non-Personalized Techniques

Non-personalized recommendation models are the simplest type of recommender system, as they do not take into account the preferences or interests of individual users. These models can be effective even with limited or no user data and are often used in conjunction with personalized recommendation models [96]. The most popular are: *Top-Popular*, *Random*, and *Best rated*.

Random It recommends to the users a random sub-sample of items belonging to the catalog.

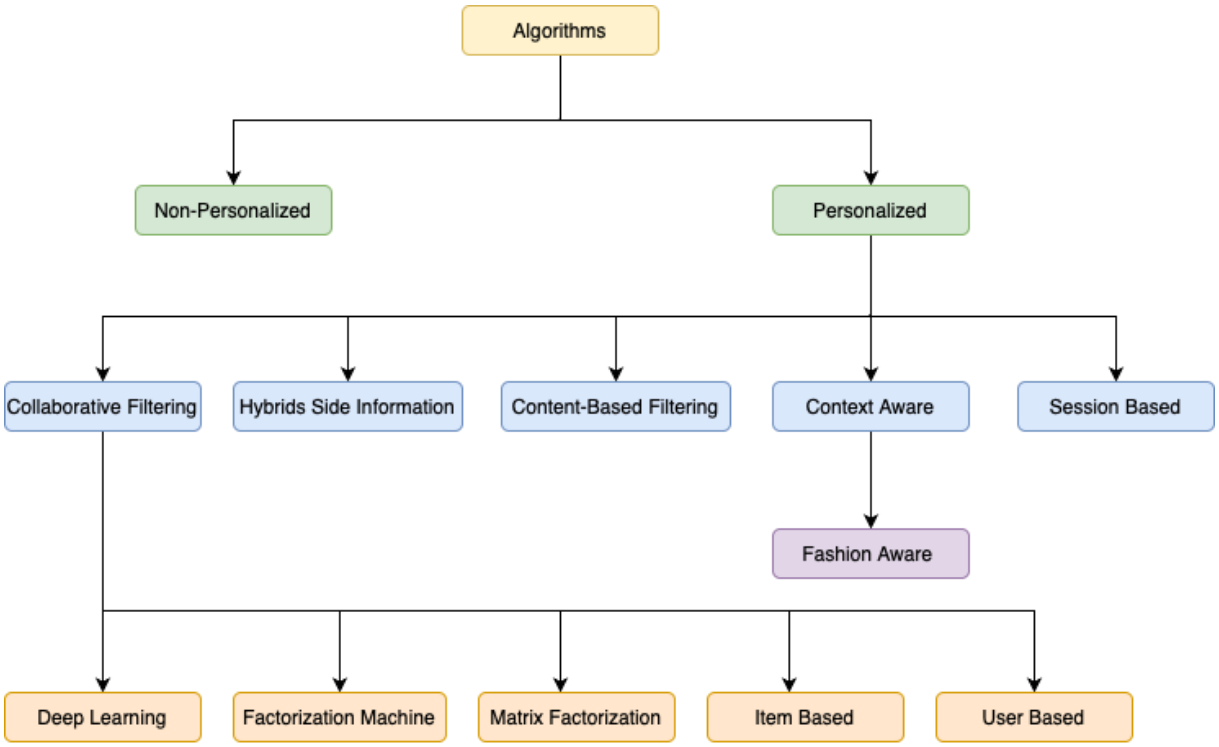

Figure 2.1: The figure provides an overview of different types of recommender systems. The four main types of personalized recommender systems included in the figure are collaborative filtering, content-based filtering, hybrid recommender systems, session-based, and context-aware recommender systems.

Top Popular Selects items with the highest number of interactions. It is also called *most popular items*.

Best Rated First considered items are the ones with the highest rating.

$$score(u,i) = \frac{\sum_u r_{ui}}{(N_i + C)} \tag{2.1}$$

Where r_{ui} is the rating given by user u to item i (considering non-zero ratings). N_i is the number of users who have rated item i. C is the shrink term, a constant value.

2.2. Content-Based Filtering

Those algorithms focus on attributes of items, i.e., their content. In the case of **ItemKNN CB**, we calculate the similarity between each pair of items, using the ICM as input. The similarity is calculated using the formula $s_{ij} = \sum_a i_a \cdot j_a = \#common\ attributes$. This

formula counts the number of common attributes between each item, so the values may be greater than 1. If we want a value between 0 and 1, we can normalize it with the norm-2 of the two vectors as shown in eq. (2.2).

$$s_{ij} = \frac{\sum_a i_a \cdot j_a}{\sqrt{\sum_a i_a^2 \cdot \sum_a j_a^2} + C} = \frac{\#common\ attributes}{\sqrt{\#attributes\ of\ i \cdot \#attributes\ of\ j} + C} \tag{2.2}$$

The similarity computed in this way is called *shrunk cosine similarity* since it is exactly the formula to calculate the cosine of the angle between two n-dimensional vectors: the more they are similar, the higher s_{ij} is (and therefore the larger the cosine), the smaller the angle between them. The shrink term give to the formula the *level of trust* and is called *support*. The shrink term is a hyper-parameter of the recommender system and needs to be tuned to find the best model. Once we have the similarity between items we are able to calculate also the estimated ratings of a user for a specific item.

$$\tilde{r}_{ij} = \frac{\sum_j r_{uj} \cdot s_{ji}}{\sum_j s_{ji}} \tag{2.3}$$

It is easy to see that the computation of the estimated ratings is complex in terms of time and memory. We need to manipulate the formula in order to reduce the effort and improve the quality of recommendations. We present two techniques to accomplish that task in section 2.2.1 and section 2.2.2.

Item Content Matrix (ICM) The first possible source of inputs for a recommender system is represented by the "list" of items with their attributes and organized in a matrix called Item Content Matrix or ICM, where **rows** represent items, **columns** represent attributes and each **cells** contain values describing the item-attribute connection. The value of the cell may be binary, indicating the presence or not of the feature or a real number, indicating the importance of the attributes describing the item.

2.2.1. k-Nearest Neighbours (kNN)

When building a content-based recommender system, one key step is manipulating the similarity matrix S to remove outliers and focus on the most relevant items for a given target item [74]. The k-nearest neighbors (kNN) technique is often used to identify the k most similar items to the target in each row of the similarity matrix while setting all other values to 0. This helps to improve the quality of recommendations by filtering out irrelevant items and focusing on those that are most closely related to the target [105].

$$\tilde{r}_{ij} = \frac{\sum_{j \in kNN(i)} r_{uj} \cdot s_{ji}}{\sum_{j \in kNN(i)} s_{ji}} \tag{2.4}$$

where k is a hyper-parameter that influences the quality of recommendations and has to be properly tuned. Indeed, if we choose a value for k that is too small, we will not have enough data to compute a reliable estimation; on the other hand, if we choose a value that is too big, we will use data full of noise tainting our recommendations. The right value must be something in between these two extremes and strongly depends on the dataset [105].

2.2.2. Improving ICM with non-Binary Weights

When building a content-based recommender system, the item-content matrix (ICM) can be manipulated to improve the quality of recommendations by assigning weights to each attribute based on its importance [1]. This allows more important attributes to contribute more heavily to the computation of similarity and can improve the accuracy of recommendations. Additionally, the ICM can include non-binary values to capture more nuanced relationships between items and attributes [88]. This can help to improve the diversity of recommendations and avoid excess reliance on a subset of attributes.

2.3. Collaborative Filtering

Collaborative filtering is a widely used technique in recommender systems that relies on the opinions of users to make recommendations [101]. There are two main approaches to collaborative filtering: user-based and item-based. The user-based approach involves finding users with similar preferences and recommending items that those users have rated highly. The item-based approach, on the other hand, involves finding items that are similar to items the user has rated highly and recommending those similar items [38].

User Rating Matrix (URM) In CF the User Rating Matrix is the main and only source of input and contains interactions between users and items. Interactions refer to the recorded instances of user engagement with the items in the system. These interactions can take various forms, depending on the type of item being recommended and the platform in which the system is deployed. In our case, the interactions represent the transactions made by a buyer (user) for an article (item). In the URM, **rows** represent users, **columns** represent items, and each **cell** represents the interaction. We may have **implicit ratings**, in which 0 means "no interaction" and 1 means that an interaction

occurred. This type of rating is useful when we are looking at the behavior of users without directly asking them for a rating or an opinion. In the case of **Explicit ratings**, we directly ask the opinion of the user with a value on a defined scale. Not every user answers and therefore we assume as 0 *no information* [1].

Many recommender systems use the URM as input data and their goal are to predict missing values in the URM. However, it may happen that each user interacts only with a few items, resulting in a very sparse matrix, with a great number of cells with a value of 0 [56].

2.3.1. Memory Based CF

Memory-Based Collaborative Filtering (CF) is a type of recommendation system that relies on similarities between user preferences to generate recommendations. In Memory-Based CF, the system creates a matrix of user-item ratings, where each row represents a user and each column represents an item, i.e., *URM*.

The system then calculates the similarity between each pair of users based on their ratings of the same items. To generate recommendations for a user, the system looks at the items that the user has not rated and finds the users who are most similar to the target user. The system then recommends items that these similar users have rated highly but the target user has not yet seen.

Memory-Based CF has some advantages over other recommendation systems. It is easy to implement and can provide good results with small datasets. However, it can suffer from the "cold start" problem when there are new users or new items with no ratings yet. Additionally, it can be computationally expensive for large datasets, and the quality of recommendations can suffer if there are not enough overlapping ratings between users.

User-Based CF

The basic idea of User-Based CF is to search for users with similar tastes and recommend to them the items they liked the most [102]. We start computing the similarity between users using the following $s_{uv} = \sum_i u_i \cdot v_i = \#common\ items$

s_{uv} is the value of the similarity, that has a value greater than 1. We can improve our metric by normalizing it using the norms-2 of the two vectors, in order to obtain a value between 0 and 1:

$$s_{uv} = \frac{\sum_i v_i \cdot u_i}{\sqrt{\sum_i u_i^2 \cdot \sum_i v_i^2 + C}} = \frac{\#common\ opinions}{\sqrt{\#opinions\ of\ u \cdot \#opinions\ of\ v + C}} \tag{2.5}$$

The similarity computed in this way is called *shrunk cosine similarity* since it is exactly the formula to calculate the cosine of the angle between two n-dimensional vectors: the more they are similar, the higher s_{uv} will be (and therefore the larger the cosine), the smaller the angle between them. The shrink term gives to the generated values the *level of trust*.

This matrix is symmetric because we are using the cosine similarity, whose formula is commutative with respect to the users ($s_{uv} = s_{vu}$). Once obtained the similarity matrix we can use the kNN techniques to make it sparser and obtain all the benefits we explained in section 2.2.1

To estimate the rating of the user u for item i we can simply compute the weighted mean of all the known ratings of user u, where the weights are the corresponding similarities:

$$\tilde{r}_{ui} = \frac{\sum_{v \in kNN(u)} r_{vi} \cdot s_{vu}}{\sum_{v \in kNN(u)} s_{vu}} \tag{2.6}$$

From the eq. (2.6), the more user u is similar to user v, the more the opinions of user v will influence the estimated ratings for user u and vice versa. This works only for implicit ratings since with explicit ratings we have a deeper opinion of users, with more shades. We can adopt a different strategy to consider the different ways a user can express a rating. This is called user bias, a phenomenon in which some users are more generous than others in giving ratings. The trick to overcoming this problem is to normalize the ratings before computing the similarity, removing from each rating of user u his corresponding bias [9].

$$b_u = \frac{\sum_{v,i \in T} r''_{ui}}{N_T} \tag{2.7}$$

r''_{ui} is the re-normalization of each rating of the URM, obtained by removing the item bias from the rating of the URM ($r''_{ui} = r'_{ui} - b_i$). b_i, i.e., item bias, is the shrunk average rating for each item.

$$b_i = \frac{\sum_{u,i \in T} r'_{ui}}{N_T + C} \tag{2.8}$$

r'_{ui} is the normalization of each rating of the URM, obtained by removing the global bias

($r'_{ui} = r_{ui} - \mu$). The eq. (2.9) is the global bias, which is simply the average rating for all items and users (sum of non-zero ratings over the number of non-zero ratings). In the equation T indicates the set of non-zero ratings, and N_T indicates their number:

$$\mu = \frac{\sum_{u,i \in T} r_{ui}}{N_T} \tag{2.9}$$

$$s_{uv} = \frac{\sum_i (r_{ui} - \tilde{r_u}) \cdot (r_{vi} - \tilde{r_v})}{\sqrt{\sum_i (r_{ui} - \tilde{r_u})^2 \cdot \sum_i (r_{vi} - \tilde{r_v})^2} + C} \tag{2.10}$$

The eq. (2.10), called Pearson correlation, is the result of this operation on the cosine similarity. If $r_{ui} - \tilde{r}_u$ is positive, means that user u likes item i more than his average rating: globally user u likes item i. From the eq. (2.11) is possible to calculate the similarity used to compute the estimated rating.

$$\tilde{r}_{ui} = \tilde{r}_u + \frac{\sum_{v \in kNN(u)} (r_{vi} - \tilde{r}_v) \cdot s_{vu}}{\sum_{v \in kNN(u)} s_{vu}} \tag{2.11}$$

Item Based CF

The basic idea of item-based collaborative filtering is to calculate the similarity between each pair of items considering how many users have the same opinion about them. Starts by calculating the similarity between each pair of items:

$$s_{ij} = \sum_u r_{ui} \cdot r_{uj} = \#common\ opinions = \vec{i} \cdot \vec{j} \tag{2.12}$$

Since this value can be greater than 1, we can normalize it, as previously done with the user similarity, to get values between 0 and 1. We can do that using, as usual, the shrunk cosine similarity:

$$s_{ij} = \frac{\sum_u r_{ui} \cdot r_{uj}}{\sqrt{\sum_u r_{ui}^2 \cdot \sum_u r_{uj}^2} + C} \tag{2.13}$$

Once computed the similarity between each pair of items, we put all those values in a similarity square matrix. This matrix is symmetric because we are using the cosine similarity, whose formula is commutative with respect to the items ($s_{ij} = s_{ji} - \mu$). To compute the estimate of the rating of user u for item i, we compute the weighted mean

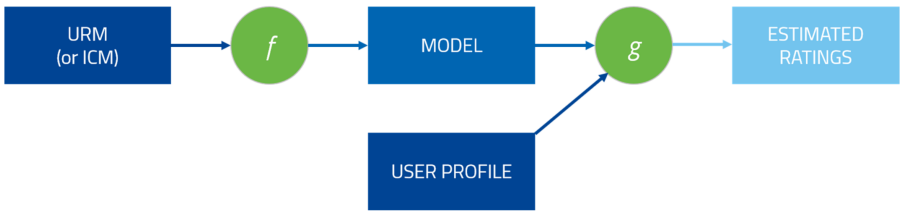

Figure 2.2: First we build the model starting from the URM and ICM, then we calculate the estimated ratings as a function g of both model and user profile

of all the known ratings of user u, where the weights are the corresponding similarities, as shown in eq. (2.14). The higher the similarity between item i and j, the more the opinions about item j will influence the estimated ratings for item i. These concepts are similar to ones in section 2.3.1, but they are applied on the columns rather than on the rows.

$$\tilde{r}_{ui} = \frac{\sum_{j \in kNN(i)} r_{uj} \cdot s_{ji}}{\sum_{j \in kNN(i)} s_{ji}} \tag{2.14}$$

Memory-Based vs Model-Based

In this section, we go over a further distinction between different types of recommender systems, based on the method of implementation and its usage. The process to generate estimated ratings is composed of two steps as shown in fig. 2.2: in the first one, we build the model (using the URM in CF techniques or the ICM in CBF), while in the second one, we use the model alongside the user profile (information about the user's tastes) to provide recommendations. The user profile can be seen as a vector of ratings (the corresponding row in the URM) and it divides recommender systems into two categories.

Memory-Based: these collaborative filtering techniques require that the user profile belongs to the URM used to build the model [97]. This means that these techniques can only provide recommendations to "known" users whose opinions have been used during the model-building process. These methods predict ratings based on users' neighborhoods, using the URM in both the construction of the model and the prediction. Memory-based techniques are easier to implement compared to model-based techniques [37].

Model-Based: techniques of this type do not impose restrictions on where the user profile has to be taken. This means that model-based techniques can provide recommendations both to "known" and "unknown" users. This freedom of action is due to how the model has been built: starting from the items to build the model means that we do not care

about "known" or "unknown" users because we simply use what we have, without having to recompute the model every time we add a new user or provide recommendations to him. In other words, model-based techniques extract information from the dataset to build a model, without relying on the users' neighborhood. They are a bit harder to implement, and they also require a URM that is big enough to extract the model; nevertheless, the huge advantage is that we can provide recommendations also to "unknown" users, without having to recompute the model [56].

2.3.2. Machine Learning Approaches

In an ML context, the quality of a recommender system is evaluated through a loss function, which measures the discrepancy between the predicted ratings and the observed ones. To increase the precision the predicted ratings need to be as close as possible to the real ones, which corresponds to both the highest quality and the lowest value of the loss function. For this reason, minimizing the loss function is equivalent to maximizing the quality of the system. Therefore, the goal of building a recommender system can be framed as finding the similarity matrix, i.e., the model that minimizes the error between the predicted and observed ratings [37].

$$E(S) = comparison(R, \tilde{R}(S)) \tag{2.15}$$

The eq. (2.15) represents the expected similarity between the real ratings R and the predicted ratings $\tilde{R}$ for a set of items S. *Comparison* is the function that measures the similarity between two sets of ratings. One possible definition of the *comparison* function is the cosine similarity between the two vectors, as shown in the eq. (2.16).

$$E(S) = \frac{R \cdot \tilde{R}(S)}{|R||\tilde{R}(S)|} \tag{2.16}$$

where $\cdot$ represents the dot product between two vectors, and $|\cdot|$ represents the L2 norm (Euclidean distance) of a vector. The expected similarity E(S) is a measure of how well the predicted ratings match the real ratings for the items in the set S. By minimizing the difference between the real and predicted ratings, we improve the quality of the recommendations provided by the system.

SLIM

SLIM [118] stands for Sparse Linear Method, and it is an Item-Based CF technique. Once defined the model (the item-item similarity) we have to also define the loss function we use. In the case of SLIM, we use the MSE because it is very intuitive and differentiable [28].

$$E(S) = \sum_{u,i \in R^+} [r_{ui} - \tilde{r}_{ui}]^2 \hookrightarrow ||R - RS||_2 \tag{2.17}$$

where $\tilde{r}_{ui} = \sum_j r_{uj} s_{ji}$ is the sum of the ratings that user u gave to items j, weighted by the similarity between item i and j. The loss function $E(S)$ is the norm-2 of the difference between the URM (R) and the estimated ratings (RS). Using norm-2 means computing the error over all the non-zero ratings of the starting URM.

In the eq. (2.18) S^* is the matrix that, among all other similarity matrices, minimizes the loss function. Nevertheless, the best solution is for $S^* = I$ that, although reduces the loss function to zero. This is an uninformative solution since it only states that every item is perfectly similar to itself and to anything else: it is useless but also a perfect example of the inability to generalize.

$$S^* = \min_S ||R - RS||_2 = \min_S E(S) \tag{2.18}$$

To achieve the same results in a machine learning approach, we have to introduce the concept of regularization. It refers to a set of techniques used to prevent the overfitting of a model which occurs when a model becomes too complex and starts to fit the noise or random fluctuations in the training data, rather than the underlying patterns. This can lead to low accuracy when the model is applied to new data [32]. Regularization techniques involve adding a penalty term to the objective function that the model is trying to minimize. This penalty term encourages the model to have smaller weights or coefficients, which can reduce the complexity of the model and improve its generalization accuracy. The most common regularization techniques are L1 regularization (also known as Lasso regularization) and L2 regularization (also known as Ridge regularization) [25].

BPR

The BPR algorithm exploits the Bayesian approach to rank items according to the preferences of users expressed through implicit ratings maximizing the posterior probability [94].

$$P(item\ i\ higher\ in\ ranking\ than\ item\ j|user\ u) \tag{2.19}$$

Implicit ratings are useful to catch raw interactions of users, and their binary form allows to classify as *relevant* the items with which the user interacts, marked with a 1. Items identified as *non-relevant* are ones the user has never interacted with and are marked with a 0. In this chapter, relevant items are identified with the letter i while non-relevant items are with the letter j. It is true that we cannot distinguish between positive and negative ratings using implicit ratings, but, it is also true that relevant items must be ranked higher than non-relevant ones [87]. Following this statement, the formula of the posterior probability can be rewritten as:

$$P(\tilde{r}_{ui} > \tilde{r}_{uj}|u) \tag{2.20}$$

We want to predict the rating of relevant item i for user $u(\tilde{r}_{ui})$ in such a way that its estimated rating is greater than the estimated rating for non-relevant item $j(\tilde{r}_{uj})$.

One function to estimate the probability that the rating of item i is greater than the one for item j is the sigmoid function:

$$\alpha(x) = \frac{1}{1 + e^{-x}} \tag{2.21}$$

A high value of x corresponds to a value of the probability function $\alpha(x)$ that is close to 1. Moreover, we want that the higher the rating of item i is with respect to item j, the closer to 1 the value of $\alpha(x)$ is. Therefore, to maximize the difference between the ratings of relevant and non-relevant items we refer to the eq. (2.23).

$$x_{uij}\tilde{r}_{ui} > \tilde{r}_{uj} \tag{2.22}$$

$$\alpha(x) = \frac{1}{1 + e^{-x_{uij}}} \tag{2.23}$$

where $\tilde{r}_{ui}$ is the predicted rating of relevant items for user u, $\tilde{r}_{uj}$ is the predicted rating of non-relevant items for user u and x_{uij} is the pairwise difference between items i and j.

2.3.3. Matrix Factorization

User Rating Matrix (URM) is a fundamental component of Recommender Systems and is used in many algorithms, such as Collaborative Filtering (CF) [97]. However, the URM is typically a large matrix, making it computationally expensive to learn an accurate model from it. Most techniques used in CF are based on similarities between items or users. Therefore, we need to extract from the URM something that acts as a counterpart of a "similarity" between users and items and reduce the number of parameters we need to learn during the training process. This approach is known as Matrix Factorization [56].

To compute this new type of "similarity" we cannot categorize users and items using the same attributes as it would result in a conceptual error, similar to adding a string to an integer value. Instead, we need to introduce something that connects the preferences of a user to the items that will satisfy their tastes: features or latent factors. Items are described using features with high values indicating a higher presence of that feature in describing them, while users express their preferences through features with high values indicating a higher preference. Therefore, users will like items that contain the features they prefer the most [56]. Overall, Matrix Factorization allows to the extraction of meaningful information from the URM reducing the number of parameters and leveraging the relationship between users and items through latent factors.

In the eq. (2.24) $\tilde{x}_{uk}$ represents user's preference, i.e., how much user u likes feature k. $\tilde{y}_{ik}$ represents the item's description, where its value tells us how much feature k describes item i. The product of these two values represents how much user u likes item i because of the presence of feature k. However, from a mathematical point of view, features are only an abstraction made to connect users and items and have nothing to do with real attributes. To calculate the rating user u would give to item i, we have to iterate this product over each feature.

$$\tilde{r}_{ui} = \sum_{k} \tilde{x}_{uk} \cdot \tilde{y}_{ki} \tag{2.24}$$

Extending this computation to all the possible combinations of user-item, we obtain a matrix product (see eq. (2.25)), where X is the matrix containing the preferences of all the users: on the rows, we have users (N_u) while on columns we have features (N_k); its dimensions are $N_u x N_k$. Y is the matrix containing the description of all the items: on the rows, we have features (N_u) while on columns we have items (N_k); its dimension is N_i x N_k. $\tilde{R}$ is the matrix of predicted ratings and its dimensions are N_i x N_u.

$$\tilde{R} = X \cdot Y \tag{2.25}$$

Starting from the matrix $\tilde{R}$, to understand how to learn the parameters of the two matrices X and Y we make use of ML techniques and loss function. One possible approach is using the MSE as a loss function.

$$X^*, Y^* = \min_{X,Y} ||R - \tilde{R}||_2 + \lambda_1 \cdot ||X||_2 + \lambda_2 \cdot ||Y||_2 \tag{2.26}$$

Adding the weighted norms of X and Y allows keeping the matrices as sparse as possible, depending on the weights λ_j. If they are near zero, we have no regularization and therefore we do not avoid overfitting. Instead, if they are high, the norms in the loss function will be dominant and, as a result, we obtain very sparse matrices. Since the URM is a sparse matrix with many missing values, it is important to decide the assumption to interpret the meaning of these missing values.

- **Missing As Random (MAR)**: It is assumed that the probability of a user liking or disliking an item they have not yet rated is equal. This approach, while naive, works well for estimating actual ratings. The MAR approach is mainly based on the Norm-2 algorithm [55].
- **Missing As Negative (MAN)**: This approach assumes that a user is more likely to dislike an item they have not yet rated. It is a more conservative approach that works well for optimizing precision and recall. MAN is based on the Frobenius Norm algorithm, which is a modified version of Norm-2 that considers zero elements in the computation [50].

Alternating Least Squares (ALS)

Matrix Factorization models factorize a large matrix into two smaller matrices, with the goal of predicting missing values. However, it is not a simple task to factorize a large matrix into two smaller ones analytically. Alternating Least Squares (ALS) is one such approach that has been widely used in the literature [56]. The idea behind ALS is to alternate between fixing one matrix while learning the other, and vice versa. This is done to avoid the costly process of simultaneously minimizing both matrices. At each iteration, we minimize two loss functions, one for each matrix, using for example Mean Squared Error (MSE) as the loss function.

The optimization process starts by initializing the two matrices with random values and

then we update one matrix while fixing the other. In this way, given the current estimate of one matrix, we can solve for the other matrix by minimizing the loss function. Once updated one matrix, we can fix it and update the other matrix. This process of alternating between fixing one matrix and updating the other continues until convergence. ALS is particularly suited for large, sparse matrices, which are common in many real-world recommender systems. Furthermore, ALS is parallelizable, making it possible to scale to even larger matrices [86]. Despite its simplicity, ALS has been shown to achieve state-of-the-art effectiveness in many benchmark datasets, and it remains a popular choice in the literature [16].

Singular Value Decomposition

SVD works by decomposing a matrix into three other matrices, where the middle matrix contains singular values that represent the most important features of the original matrix [56]. This factorization method states that a complex matrix A of dimensions M x N can be decomposed in the following product:

$$A = U\Sigma V^T \tag{2.27}$$

where U is the orthogonal matrix of the eigenvectors of the square symmetric matrix AA^T and therefore it has dimensions M x M. V is the orthogonal matrix of the eigenvectors of the square symmetric matrix A^TA and therefore it has dimensions N x N. V^T is its transpose. Σ is the matrix of the singular values, i.e., the square root of the eigenvalues, of the square symmetric matrix AA^T (or A^TA), since a matrix and its transpose have the same eigenvalues. Those singular values are placed in the diagonal of Σ. As the product suggests, the dimension of Σ is M x N. This method has been applied for example to movie recommendation [77].

Funk SVD

Funk SVD was proposed by Funk [26] and the main idea is to decompose a user-item rating matrix into two smaller matrices, one representing users and the other representing items. The factorization is done by minimizing the difference between the original rating matrix and the product of the two smaller matrices. The objective function is:

$$\min_{U,V} \sum_{i,j} (R_{i,j} - \sum_{k=1}^{K} U_{i,k} V_{j,k})^2 + \lambda(||U||^2 + ||V||^2) \tag{2.28}$$

where R is the original rating matrix, U and V are the two smaller matrices, K is the number of latent factors, and λ is a regularization parameter. The optimization problem can be solved using gradient descent or other optimization algorithms. Once the two smaller matrices are obtained, the missing ratings are predicted by computing the dot product of the corresponding user and item vectors. Even with this method, regularization can be used to mitigate overfitting.

Asymmetric SVD

Funk SVD is Memory Based technique and thus it is required to retrain the model and tune all the hyper-parameters whenever we have to add a new user. Indeed, adding a new user means modifying the dimensions of the matrix X, which becomes $(N_u + 1) \cdot N_k$. To find a Model-Based Matrix Factorization technique we can manipulate the users-features matrix X and find a way to isolate the user profile.

In the eq. (2.29) we approximate the value of x_{uk} as the sum of a user-related rating, r_{uj}, i.e., the rating of user u for item j weighted by a feature related value, z_{jk}, i.e., how much feature k is important describing item j.

$$x_{uk} = \sum_j (r_{u_j} \cdot z_{jk}) \hookrightarrow X = R \cdot Z \tag{2.29}$$

The proposed approach involves a shift towards a Model-Based approach, where the rating given by a user to an item is separated from the importance of the feature in that item. This simplifies the process of adding new users since their preferences can be easily incorporated into the system. To achieve this, matrix X can be represented as the product of two new matrices: R, which stores the ratings of users for items, and Z, which stores the importance of features in those items. For example, in a study by Koren et al. [56], this approach was used in a Netflix Prize competition to predict user ratings.

2.3.4. Association Rules

Is a technique that helps discover all the relationships between items in the dataset. From the recommender systems' point of view, association rules are used to explore the dataset, in order to estimate the probability that something will happen (a user will interact with an item), knowing that something else happened in the past (a user interacted with another item) [69].

For example to calculate the probability that a user like an item i, if he previously liked

item j (eq. (2.31)), we count the number of times other users liked both items i and j with respect to the number of times other users liked item j: in case of implicit ratings this turns into an alternative definition of similarity between items i and j. This similarity is asymmetric because by inverting i and j, as shown in the eq. (2.30), we obtain a different probability and thus a different similarity. Adding a shrink term C increase the *trust level* and avoid bias.

$$P(i|j) \neq P(j|i) \hookrightarrow s_{ji} \neq s_{ij} \tag{2.30}$$

$$P(i|j) = \frac{\#of\ appearances\ of\ i\ and\ j}{(\#of\ appearances\ of\ j) + C} = s_{ij} \tag{2.31}$$

2.4. Hybrid Recommender Systems

The traditional approaches to building recommender systems have their strengths and weaknesses. Collaborative Filtering techniques rely on existing ratings and may struggle to provide recommendations for items with few ratings, leading to cold start issues. On the other hand, Content-Based techniques are not susceptible to this issue, as they rely on attributes of items, but they lack the ability to exploit the existing opinions of a community of users.

Hybrid recommender systems have been developed to overcome these limitations. In this chapter, we present five types of approaches: *Linear Combination*, *List Combination*, *Pipeline*, *Merging Models*, and *Co-Training*. These techniques combine the strengths of different recommendation algorithms to provide better recommendations.

Burke [11] provides a comprehensive survey of hybrid recommender systems, while Adomavicius and Tuzhilin [1] presents a general framework for building hybrid recommendation models. Additionally, Kunaver and Gregoric [59] presents a recent survey of hybrid recommender systems and their applications in various domains.

2.4.1. Linear Combination

The linear combination comes from the world of linear algebra: it combines the recommendations of two (or more) algorithms through a weighted sum. As shown in the fig. 2.3 the training phases are completely independent. The predictions are computed independently, but they use the same user profile as input. Finally, the outputs of each model are combined to provide a recommendation that considers the points of view of all the models.

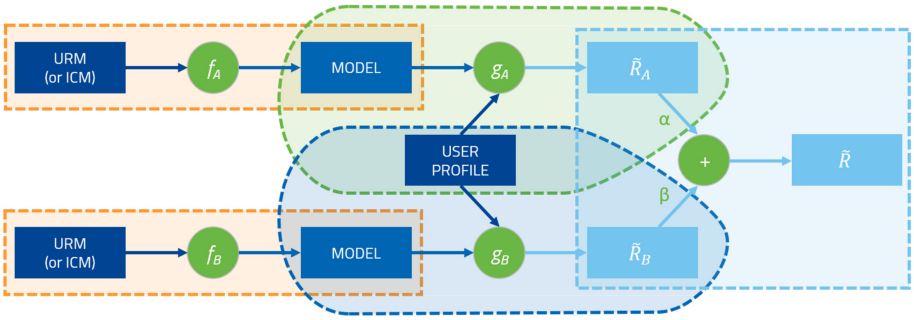

Figure 2.3: In the linear combination technique models are trained independently, then the outputs are combined to provide the final recommendation.

Therefore, we can highlight two independent chains (one for model A and one for model B) that merge together only at the end. Those models are used in an "off-the-shelves" mode, as they allow, without any modification of the structure of their algorithms, a high level of parallelization in the computation of the single models [74].

Weights provide the level of trust and importance to the models: the higher the weight, the more relevant in the computation of the final recommendation the corresponding recommender system is. Therefore, they have to be carefully tuned to obtain an effective hybrid recommender system. A bad tuning of the hyper-parameters results in a very weak and ineffective model. On the other hand, good tuning results in a strong and effective hybrid model.

An example of a possible combination is the one between Context-Based Filtering and a Collaborative Filtering approach. This is because CF suffers from cold items, but we have also CBF that can provide a better recommendation for those items. On the other hand, CBF lacks a way to consider the opinions of a community of users, but this is a task CF can perfectly perform. There are also some disadvantages that must be considered, for example, Hyper-parameters strongly depend on the dataset, and they have to be carefully tuned each time the dataset changes. If the estimated ratings are on different scales, providing good recommendations would be very difficult: we have to take care of properly weighing the different components if we want to use this method [74].

2.4.2. List Combination

The list combination technique merges recommendations from two distinct recommender systems and is particularly well-suited for Top-N recommendations since it aims to combine the most highly-rated items from two ranked lists to create a unique Top-N recom-

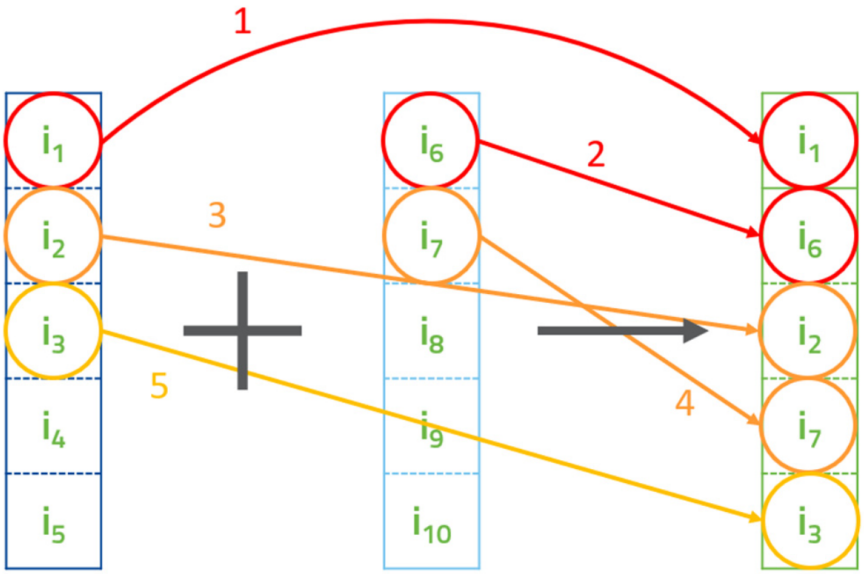

Figure 2.4: How to combine different recommendation lists in circular ways using a round-robin technique.

mendation list. While the concept behind list combination is simple, the question of how to merge two lists can be more complicated. There are a variety of techniques that can be used: one approach uses the round-robin method, in which items are alternately chosen from the first and second list, as shown in fig. 2.4 which can be easily extended to multiple lists.

The order in which the lists are merged is predetermined, with a common approach being to select items first from the list generated by the most successful algorithm and then from the others. The algorithms used to create the starting lists are left untouched, as only their outputs are combined to form the final merged list. And, the final list contains unique elements, they are never repeated.

One advantage of list combination over linear combination is that it eliminates the need for a bound on the rating scale. Instead, it focuses on the relative rankings of the items, which is particularly useful when working with algorithms that produce ratings with vastly different scales. However, there are some trade-offs with this approach. Choosing the method for building the merged list can be challenging since there is no one-size-fits-all solution. Furthermore, once the fusion technique is selected, it can be difficult to weigh the relative importance of the lists provided by each recommendation algorithm [75].

2.4.3. Pipelining

The first types of hybrid recommender systems discussed earlier, namely linear combination and list combination, employ a parallel approach. In this approach, each algorithm can be computed independently without any modification to its structure, and the results are combined at the end when all algorithms have produced their outputs. However, if tasks can be completed in parallel, they can also be completed in serial. Instead, in the pipelining approaches the output of one algorithm is fed as the input of the next algorithm as shown in the fig. 2.5.

- Algorithm A takes as input a URM or ICM and uses it to build a model.
- Now that we have a first model, we can also make some predictions: therefore, we take the user profiles from the URM we want to use in algorithm B, and we compute the matrix of estimated ratings $\tilde{R}_{PARTIAL}$. It is important to use the user profiles coming from the URM we will use algorithm B because otherwise, we will not be able to enrich that URM.
- $\tilde{R}_{PARTIAL}$ becomes the new input for algorithm B, which uses it to build the model. It is important to underline that from now on, the main input required by algorithms is a URM, e.g., we cannot use as a second algorithm the CBF.
- Once the model is trained, it can be used alongside a user profile to make predictions, obtaining the matrix of estimated ratings $\tilde{R}_{FINAL}$.

One example of a paper discussing this parallel versus serial approach in hybrid recommender systems is [35]. The paper presents a novel approach to building hybrid recommender systems using neural attentive item similarity. In their approach, the authors use a serial pipeline of algorithms to generate recommendations. The first algorithm generates an item similarity matrix, which is then used as input by the second algorithm to generate embedding for each item. These embeddings are then used by the final algorithm to generate recommendations.

Pipelining is used to enhance the sparsity of the user-item rating matrix which is used by the final algorithm. By incorporating the outputs of multiple algorithms, pipelining can generate a more complete URM, providing the final algorithm with more information to generate better recommendations.

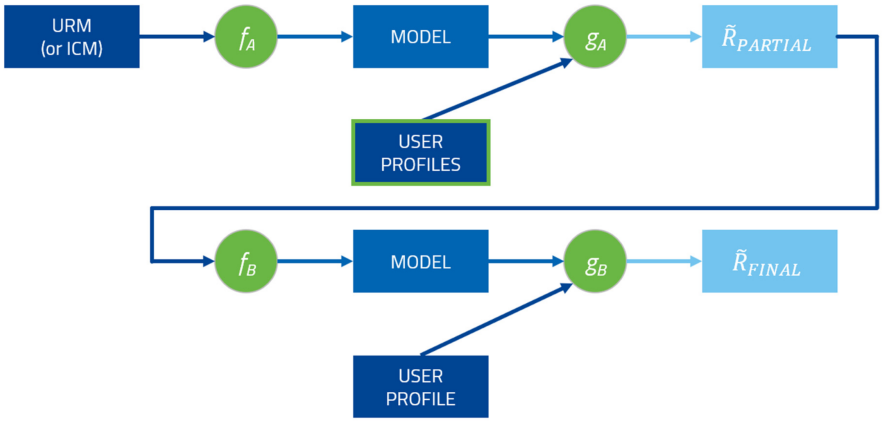

Figure 2.5: In the pipelining techniques models are trained sequentially; each matrix R is given as input to train the next model in the pipeline.

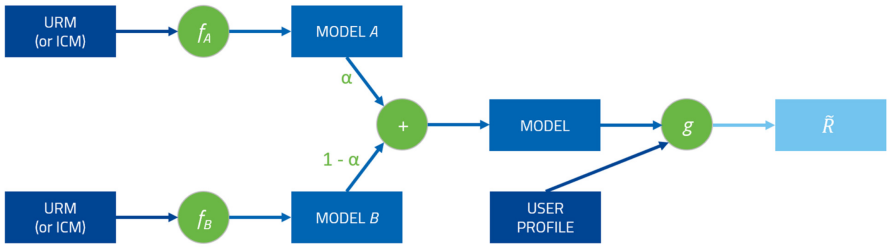

Figure 2.6: The final model is a weighted sum of the models we want to merge together; the merge phase is at the model level.

2.4.4. Merging Models

Merging models is a technique used to merge many recommender system algorithms into a hybrid: they exploit the same principle used in the linear combination but at the "model" level [117]. From the fig. 2.6 two models, in order to be merged together, must have the same structure. This means that an item-item similarity matrix can be merged with another item-item similarity matrix if they have the same dimensions, but it cannot be merged with a user-user similarity matrix. As a further example, a CF model cannot be merged with a Matrix Factorization technique through this hybridization method. This is because of how the merge is done using the parameter α.

$$S_F = \alpha \cdot S_A + (1 - \alpha) \cdot S_B \tag{2.32}$$

where S_A and S_B require the same dimensions and he hyper-parameter α has to be tuned

properly to provide the best recommendations possible since it influences the quality and the goodness of the final model.

2.4.5. Co-Training

So far, we have explored various hybrid techniques such as pipelining, linear combination, and list combination. These techniques involve chaining or merging models at the estimation or model level, without changing the core algorithm used to build the model. However, there is another frontier to explore in hybrid recommender systems, which is the merge at the training level. This technique, called co-training, involves training models together and simultaneously, such that any change during the optimization process is influenced by all the algorithms simultaneously and not separately.

Bella et al. [6] proposes a co-training algorithm for building hybrid recommender systems, which can be used to leverage the strengths of multiple recommendation algorithms. The co-training algorithm works by iteratively training multiple recommendation models using unlabeled data and then using these models to label the data. The labeled data is then used to retrain the models, and the process is repeated until convergence.

2.5. Other Types of Recommender

2.5.1. Two tower MMoE

"Two tower MMoE" stands for Multi-Modal Multi-Task Output Embedding, a deep learning architecture for recommendation systems. The architecture consists of two towers or branches, each one dealing with different modalities of the input data, such as text and images. The two towers are then combined to produce a final prediction by concatenating their output embedding and passing them through a shared fully connected layer. The "multi-task" aspect refers to the ability of the model to perform multiple prediction tasks, such as rating prediction and item ranking, in a single end-to-end trainable model [78]. In recommender systems, item embedding and user embedding are techniques used to represent items and users as fixed-length vectors of numbers. These vectors capture the relationships between items and users in a lower-dimensional space, making it easier to compare and compute similarities between them.

Item embedding is the process of representing the articles as fixed-length vectors of numbers [5] using a trainable neural network. These vectors allow the model to understand the relationships between different items and make recommendations based on similarities.

Similarly, the user embedding technique allows the recommender system to understand users' interests and make personalized recommendations [119]. Mapping items and users to embed the recommender system can effectively capture complex relationships between items and users, and use this information to make more accurate and relevant recommendations [111]. The two embeddings are meshed together using, e.g., the *sampled soft-max* technique (see section 2.5.1). If the result is high then it means that the item is a good match for the user. The next section shows an example of how to combine the *"Two tower"*, i.e., *sampled soft-max*, a technique used even in one of the solutions presented later in section 3.3. The section 2.5.1 presents, instead, the *gating network*, a model used to make sure that the *"user tower"* learn by using different experts for recent active customers and non active customers.

Sampled Soft-Max

Is a technique used in recommender systems to address the scalability issue in training deep neural networks with a large number of classes, e.g., items, in a recommendation system. In a traditional soft-max, the model computes the probability of each item for a given user by normalizing the dot product of the user and item embedding over the sum of the dot products for all items. This becomes computationally infeasible when the number of items is large, as it requires computing the normalization term for every item [64].

$$\text{softmax}(u, i) = \frac{\exp(\boldsymbol{u}^\top \boldsymbol{i})}{\sum_{j \in I_u} \exp(\boldsymbol{u}^\top \boldsymbol{j})} \tag{2.33}$$

where u is the user vector, i is the item vector, I_u is the set of items that the user has interacted with, and j are the vectors of those items. The numerator of the expression is the dot product of the user vector and the item vector, while the denominator is the sum of the dot products of the user vector and the item vectors for all items that the user has interacted with. The result of the *softmax* function is a probability distribution over the items that the user has not interacted with, with each item representing the probability of the user interacting with that item.

Sampled soft-max, on the other hand, only computes the normalization term for a random subset of items, called the sample, for each iteration of training. This can significantly reduce the computational cost of training the model, allowing it to scale to larger datasets.

$$\text{sampled softmax}(u, i) = \frac{\exp(\boldsymbol{u}^\top \boldsymbol{i})}{\sum_{j \in S_u} \exp(\boldsymbol{u}^\top \boldsymbol{j})} \tag{2.34}$$

where u is the user vector, i is the item vector, S_u is a randomly sampled subset of the set of items that the user has interacted with, and j are the vectors of the items in S_u. The numerator of the expression is the dot product of the user vector and the item vector, while the denominator is the sum of the dot products of the user vector and the item vectors for all items in the randomly sampled subset S_u. The result of the *sampled softmax* function is an approximation of the true *softmax* function, but with a reduced computational complexity that speeds up the training process for large-scale recommender systems.

Additionally, it has been shown that training with a sample can still achieve good results, as the sample provides a good approximation of the true distribution. In recommender systems, sampled soft-max can be used in various types of models, such as matrix factorization or neural networks, to reduce the computational cost of training while maintaining the accuracy of the recommendations [114].

Gating Network

A gating network in recommender systems is a type of neural network architecture that is used to filter and prioritize the items to be recommended to a user[47]. The gating network learns to assign a weight to each item, indicating the importance of the item in the recommendation list. The weighted items are then passed to the next layer for final ranking and selection. The gating network helps to improve the relevance and personalization of the recommendations by taking into account various factors such as user behavior, item characteristics, and contextual information.

The gating network is designed to act as a gatekeeper, filtering out irrelevant items and prioritizing the most relevant ones. This helps to ensure that the recommendations are highly personalized and relevant to the user's preferences and needs. The gating network operates by combining various inputs, such as user history, item metadata, and contextual information, to create a representation of each item. This representation is then used to calculate the weight assigned to each item, which determines its importance in the recommendation list. The weighted items are then passed to the next layer of the network, where they are ranked and selected for final recommendation[76]. Overall, the gating network provides a flexible and effective way to incorporate multiple sources of information into the recommendation process, helping to improve the relevance and accuracy of the recommendations.

2.5.2. Two-Stage Recommender

Two-stage recommender systems are scalable and maintainable models [40]. These systems generate recommendations in two phases: first, multiple nominators select a small set of items from a large pool using cheap-to-compute item embeddings, i.e., candidate generation strategies used to down-sample the list of item (see section 2.6.4); then, a ranker with a richer set of features rearranges the nominated items and presents them to the user. However, the challenge with this approach is that optimizing the accuracy of each stage in isolation does not necessarily lead to optimal global accuracy. To address this issue, Ma et al. [79] proposed a nominator training objective that takes into account the ranker's probability of recommending each item. Many of today's largest online platforms, such as YouTube, LinkedIn, and Pinterest, utilize two-stage recommenders due to their scalability. These recommenders have proven to be effective in handling large amounts of data and providing personalized recommendations to users. Thanks to their ability to scale, these platforms are able to provide relevant content and suggestions to their users, leading to increased engagement and user satisfaction [41]. In the current literature the ranker used by a two-stage recommender implement a *GBDT algorithm*, described in the following section.

Gradient Boosting Decision Trees

Gradient boosting machines are a family of powerful machine-learning techniques that have shown considerable success in a wide range of practical applications. They are highly customizable to the particular needs of the application, like being learned with respect to different loss functions.

A common task that appears in different machine learning applications is to build a non-parametric regression or classification model from the data. When designing a model in domain-specific areas, one strategy is to build a model from theory and adjust its parameters based on the observed data. Unfortunately, in most real-life situations such models are not available. The lack of a model can be circumvented if one applies non-parametric machine learning techniques like neural networks, support vector machines, or any other algorithm at one's own discretion, to build a model directly from the data. These models are built in a supervised manner, which means that the data with the desired target variables has to be prepared beforehand [33].

The most frequent approach to data-driven modeling is to build only a single strong predictive model. A different approach would be to build a bucket or an ensemble of models for some particular learning task. One can consider building a set of "strong"

models like neural networks, which can be further combined altogether to produce a better prediction. However, in practice, the ensemble approach relies on combining a large number of relatively weak models to obtain a stronger ensemble prediction. The most prominent examples of such machine-learning ensemble techniques are random forests [10] and neural network ensembles [31], which have found many successful applications in different domains [68].

Common ensemble techniques like random forests rely on averaging of models in the ensemble. The family of boosting methods is based on a different, constructive strategy of ensemble formation. The main idea of boosting is to add new models to the ensemble sequentially. At each particular iteration, a new weak, base-learner model is trained with respect to the error of the whole ensemble learned so far. The first prominent boosting techniques were purely algorithm-driven, which made the detailed analysis of their properties and performance rather difficult [20]. This led to a number of speculations as to why these algorithms either outperformed every other method or on the contrary, were inapplicable due to severe overfitting [17].

To establish a connection with the statistical framework, a gradient-descent-based formulation of boosting methods was derived [24]. This formulation of boosting methods and the corresponding models were called the gradient boosting machines. This framework also provided the essential justifications for the model hyper-parameters and established the methodological base for further gradient-boosting model development.

In gradient boosting machines, or simply, GBMs, the learning procedure consecutively fits new models to provide a more accurate estimate of the response variable. The principle idea behind this algorithm is to construct the new base learners to be maximally correlated with the negative gradient of the loss function, associated with the whole ensemble. The loss functions applied can be arbitrary, but to give a better intuition, if the error function is the classic squared-error loss, the learning procedure would result in consecutive error-fitting. In general, the choice of the loss function is up to the researcher, with both a rich variety of loss functions derived so far and the possibility of implementing one's own task-specific loss.

This high flexibility makes the GBMs highly customizable to any particular data-driven task. It introduces a lot of freedom into the model design thus making the choice of the most appropriate loss function a matter of trial and error. However, boosting algorithms are relatively fast to implement, which allows one to experiment with different model designs. Moreover, the GBMs have shown considerable success in not only practical applications but also in various machine-learning and data-mining challenges [7].

From the viewpoint of neurobotics, ensemble models are a useful practical tool for different predictive tasks, as they can consistently provide higher accuracy results compared to conventional single strong machine learning models [100]. For example, the ensemble models can efficiently map the EMG and EEG sensor readings to human movement tracking and activity recognition. However, these models can also provide valuable insights into the models of neural formation and memory simulations. Whilst artificial neural networks have the memory of the learned patterns distributed within the connections of artificial neurons, in boosted ensembles the base-learners play the role of the memory medium and are forming the captured patterns sequentially, gradually increasing the level of pattern detail [125]. Advances in boosted ensembles can find fruitful applications in the brain simulation domain, as the ensemble formation models can be coupled with the strategies of network growth. In particular, if the base-learners are considered the nodes of the network, it will be possible to construct ensembles with various graph properties and typologies, like small-world networks, which are found in the biological neural networks [18]. In order to proceed with advanced neurobotics applications of boosted ensemble models, it is essential to first define the methodology and algorithmic framework for these models.

2.6. Evaluation and Data Processing

The evaluation of a recommendation model is a crucial aspect of Recommender Systems. Various metrics are available to measure the quality of recommendations provided by a particular algorithm, particularly in the context of a top-n recommendation task. This task involves recommending a specific user an ordered set of items with a limited length. There is a vast array of evaluation metrics that have been explored in the literature to assess the recommendation quality of a recommender system quantitatively. However, due to the multitude of available metrics, there is no universal standard to evaluate all recommenders. In the proposed challenge, the evaluation is done with *MAP@12*, analysed in the next section.

Moreover, data processing is a critical aspect of a Recommender System as it directly impacts the accuracy and effectiveness of the system. Data must be processed and transformed into a suitable format before it can be used for analysis and modeling. We handle missing information in the data and use different sampling techniques to down-sample the pool of items associated to each user to be picked up for the recommendation and to lower the training data to be used to generate candidates and calculate features to be used by models.

2.6.1. Classification Accuracy Metrics

The literature on evaluation methods for Recommender Systems primarily focuses on assessing the accuracy of models. To do so, classification accuracy metrics are often used to determine how often an algorithm provides correct recommendations to users. This evaluation typically involves training a model on a specific dataset and then testing it on a different, separate dataset. The main objective of a recommender system is to generate a recommendation list that includes as many items as possible from the evaluation set, using the information obtained from the training set.

After a model produces a recommendation list for a user, a confusion matrix can be constructed, as shown in table 2.1, which serves as the basis for calculating Precision, Recall, and F1 score. These metrics are essential in determining the effectiveness of a recommender system, and help to assess its ability to generate relevant recommendations for users.

Table 2.1: Confusion matrix serves as the basis for calculating Precision, Recall, and F1 score.

	Recommended	**Not recommended**
Interacted	TP - True positive	FN - False negative
Not interacted	FP - False positive	TN - True negative

True positive are items the user u interacted with and have been recommended.

False positive are items the user u has never interacted with and have been recommended.

True negative are items the user u has never interacted with and have not been recommended.

False negative are items the user u interacted with and have not been recommended.

Precision

Precision is the positive predictive rate. In the case of recommender systems, precision is the number of correct recommendations, i.e., the ones that are of interest to the user, divided by the total number of recommendations [89].

$$Precision = \frac{\#\ of\ correct\ recommendations}{\#\ of\ all\ recommendations} = \frac{\#TP}{\#TP + \#FP} \tag{2.35}$$

$Precision@k$ or, in other words, precision at cutoff k **(P@k)**, is simply the precision calculated by considering only the subset of your predictions from rank 1 through k. To calculate this we take the top k recommendations and find their precision with the ground truth.

$$Precision@k = \frac{\#TP_k}{k} \tag{2.36}$$

where $\#TP_k$ is the number of recommended and relevant items in the list of length k.

Recall

Recall refers to the fraction of relevant items that are recommended to the user, out of all the relevant items in the dataset.

$$Recall = \frac{\#\ of\ correct\ recommendations}{\#\ of\ all\ relevant\ items} = \frac{\#TP}{\#TP + \#FN} \tag{2.37}$$

A high recall value indicates that the model is effective at identifying and recommending relevant items to the user, while a low recall value suggests that the model may be missing out on some relevant items. However, it is important to note that high recall alone may not be sufficient to evaluate the effectiveness of a Recommender System, and other metrics such as precision and F1 score should also be considered. A variant of the recall, extensively used for recommender systems is the recall on a cut-off list of k items (eq. (2.38)).

$$Recall@k = \frac{\#TP_k}{\#\ of\ all\ relevant\ items} \tag{2.38}$$

F1 score

F1 score is the harmonic mean of precision and recall, and provides a balanced assessment of the model's performance in terms of both metrics as showed in eq. (2.39).

$$F1\ score = 2\frac{Precision \cdot Recall}{Precision + Recall} \tag{2.39}$$

Precision is the fraction of relevant items among the recommended items, and recall is the fraction of recommended relevant items out of all the relevant items in the dataset. A high F1 score indicates that the model is effective at both identifying relevant items and recommending them to the user.

2.6.2. Ranking Metrics

One common tasks for a Recommender System is to provide a list of top-n recommendations for users. However, simply recommending relevant items to users is not sufficient, as it is also critical to provide an ordered list in which the most relevant items are ranked higher. This aspect is crucial for enhancing the user experience and can greatly impact the degree of user satisfaction while navigating among the recommended items.

To evaluate how well a Recommender System performs in terms of ranking, ranking metrics are used. These metrics help in assessing the extent to which a recommender system approaches the ideal order in which the items should be presented to the users. The effectiveness of a Recommender System can be determined by measuring how closely the recommended items match the preferences and interests of the users, and how accurately the ranking reflects the users' preferences. The ranking metrics provide important feedback on the performance of the Recommender System and can help to improve the overall user experience.

Mean Average Precision

The work thesis starts from the proposed H&M challenge. We use their systems to evaluate recommendations with respect to the test data, which not publicly available. The ranking metric used for the evaluation is the MAP@k. More specifically the MAP@12, since we recommend 12 items for each user. Specifically, it is used to measure the effectiveness of a system that generates a list of recommendations or search results by comparing the predicted list with a ground truth list of relevant items [81]. It is defined as the mean of the *average precision at k*.

$$AP@k = \frac{1}{min(n,k)} \sum_{k=1}^{min(n,k)} P(k) \times rel(k) \tag{2.40}$$

where, $Rel(k)$ is an indicator function equaling 1 if the item at rank **k** is a relevant (correct) recommendation, zero otherwise.

Once cut at k items in the list of recommendations, their order does not affect the *pre-*

cision at k function but, instead, the *average precision at k* function, since the average precision is a cumulative sum of each iteration of *precision at k*, if the correct predictions come earlier, its score is counted into the sum more times, resulting in a higher value of average precision. The Average Precision function penalizes recommendations on later positions. It rewards relevant recommendations appearing early. To obtain the mean average precision at k we take the **mean** of the average precision for all the users.

$$MAP@k = \frac{1}{N}\sum_{u=1}^{N}\frac{1}{min(n,k)}\sum_{k=1}^{min(n,k)} P@k \times rel(k) \tag{2.41}$$

where n is the number of recommended elements in the list, k is the number of recommendations we consider to score with the precision function, and N is the number of **users**. n can be less or greater than k.

In this thesis we use the **MAP@12**, to evaluate a list of 12 recommendations and score them with respect to the test week, which corresponds to the last week of the available dataset.

2.6.3. Handling Missing Information

Handling missing information is a crucial step when building recommender systems for fashion [46, 106, 123]. Missing data can lead to biased and inaccurate recommendations [71, 116, 120]. Therefore, addressing this issue before training the machine learning models is important. There are various techniques to handle missing data:

- Deletion of values: Removes any entry of the user, item, or any other table of the dataset that contains missing values. While this method is straightforward, it can result in significant loss of data and may not be the best option when working with small datasets [52, 67, 104].
- Mean imputation: Replaces missing values with the mean value of the non-missing data. This approach is effective and works well when the missing values are randomly distributed [21, 65, 116]. However, it can also introduce bias and distort the distribution of the data [62].
- Mode imputation: Replaces missing values with the most frequent value in the non-missing data. This approach is useful when dealing with categorical data, where the mode represents the most common category [21, 65, 116].
- Regression imputation: Predicts the missing values based on the relationships between other features. Regression imputation can be more accurate than the previous

methods, but it requires more complex modeling and may not work well when the relationships between features are weak [98, 103, 121].

- Multiple imputations: Generates multiple plausible values for each missing value, and then creates multiple complete datasets that can be used for analysis. Multiple imputations can be more accurate than single imputation methods [29], but it requires more computational resources and can be more complex to implement [99, 103].

The choice of the method depends on the characteristics of the data and specific goals. By applying appropriate techniques to handle missing data, we ensure that our recommender system is making accurate and relevant recommendations to users [53].

2.6.4. Sampling Techniques

Sampling techniques are used in recommender systems to improve the efficiency and scalability of algorithms. These techniques involve selecting a subset of the data to work with, rather than using the entire dataset [34]. There are different types of sampling techniques:

- Random: Selects a random subset of the data. Random sampling is often used when the dataset is too large to fit into memory or when there are too many users and items to process in a reasonable amount of time [63, 82].
- Stratified: Selects a subset of the data that is representative of the full dataset with respect to some characteristic or feature. For example, in a movie recommendation system, we may use stratified sampling to ensure a balanced representation of different genres in the data [80].
- Importance sampling is a common technique used in recommender systems to address the problem of data sparsity and bias towards popular items [66]. This approach involves assigning weights to users or items based on their importance or relevance to the recommendation task. These weights are then used to sample from the data in a way that prioritizes the most relevant or informative examples [45].
- Adaptive: Dynamically adjusting the sampling strategy based on the effectiveness of the algorithm, in order to maximize its efficiency [44].
- Down-sampling techniques are used in recommender systems to reduce the size of the training dataset by randomly removing some of the less active users and less popular items. The idea behind this technique is to reduce the computational complexity

of the algorithm and speed up the recommendation process, while still maintaining the accuracy of the recommendations [107].

Sampling techniques are useful to reduce the computational complexity of recommender systems, allowing them to process large datasets in a short amount of time. However, it is important to ensure that the samples are representative of the full dataset and that the sampling strategy does not introduce biases that could affect the accuracy of the recommendations. For example, in our case, the precision of the model is calculated with respect to the last week of the dataset, which corresponds to the last week of September 2020. If we train the model using as samples only the transactions during the winter seasons, the model will be biased in recommending winter clothes even for the test week, which falls in the summer season. This generates an ineffective recommendation system.

3 | Our Approach to H&M Competition

H&M is a very popular Swedish clothing company with headquarters in Stockholm. The company was established in 1947 and was originally named Hennes. In 1968 they acquired the hunting and fishing store named Mauritz Widfoss. From this point in time it operated under the name Hennes&Mauritz or simply H&M. Six years later it had a debut on the Stockholm Stock Exchange which allowed them soon after in 1976 to open their first shop outside Sweden in UK, London. In 2000 they entered the US market (see fig. 3.1).[1]. As of 2022, it operates shops in 74 countries as in

Recommender Systems are powerful, successful, and widespread applications for a lot of companies, e.g., H&M. The website of these companies, as soon as an account is created, starts to recommend products, movies, or songs that the algorithm thinks to suit you the best. That's why these systems are precious for business owners. The more interactions the better it gets. Also, the more users the better it gets. However, because this is a dynamic environment and both clients and product change it is quite difficult to tune, manage and maintain these systems [122].

Outline of the competition As mentioned in the chapter 1 the thesis work started from a Kaggle Challenge proposed by the H&M.[2] The organizers provided a dataset composed by the transactions, i.e., the action of buying an item, from *2018-09-20* to *2020-09-22*, which contains transactions made both in the online and physical stores. Along with the transactions they provided the list of all the users and articles along with their attributes, analyzed in the following sections. Past the competition period, I conducted extensive research on fashion-based recommenders and focused the work on proposing a novel and light fashion-based recommender that outperforms the state-of-the-art and publicly available top-scores recommender. One important step has been the

[1]The history of the H&M brand `https://en.wikipedia.org/wiki/H%26M`
[2]Challenge overview here

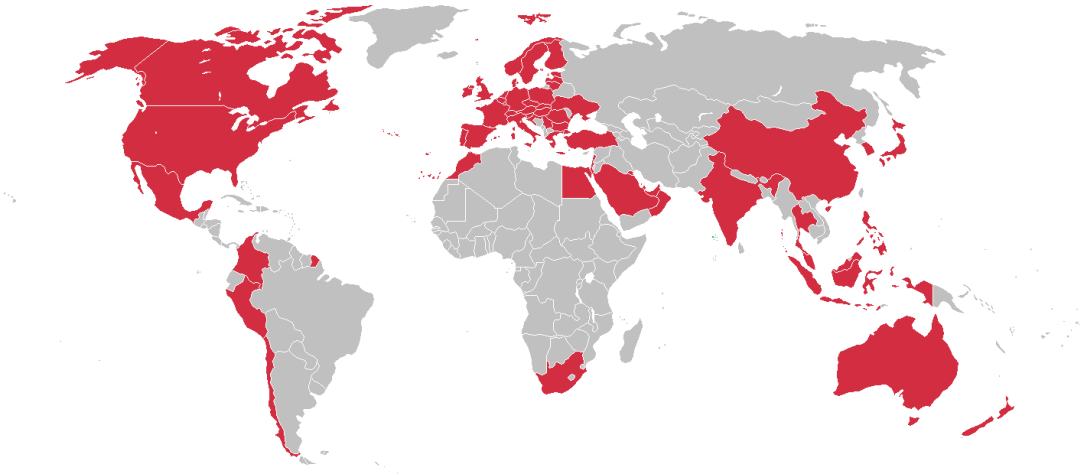

Figure 3.1: Global map of H&M, the red areas are where physical stores are located, and the gray areas are where there are no physical stores.

data analysis and the study of the behaviour of users, combining information of items, users and transactions.

The goal of the challenge was to recommend a list of 12 items to each user in the dataset. The recommendations are scored using the $MAP@12$ (see section 2.6.2) with respect to the test week, i.e., the first one after the dataset period, which goes from $2020-09-23$ to $2020-09-30$. The data of the test week is split in order to generate two leaderboards. The public leaderboard was available during the whole challenge period and is generated using only the 5% of the total test data. The private leaderboard has been released after the deadline and is generated using the other 95% of the test data. The winners of the challenge have been selected considering only the private leaderboard. In this section, we analyze several winning approaches that scored in the top 10 on the private leaderboard. [3]

3.1. Dataset

In this competition, H&M invites to develop product recommendations based on data from previous transactions, as well as from customer and article metadata. The available metadata spans from simple data, e.g., garment type and customer age, to text data e.g., product descriptions, to image data of each product. [4] There is not a unique best approach between just investigating a categorical data type algorithm or dive into NLP

[3]Public leaderboard of the competition here

[4]The competition's website and information are available at `https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations/overview/description`.

and image processing deep learning.

3.1.1. Articles

This database contains information about the articles; Table 3.1 summarizes relevant statistics of the articles dataset, such as the description of each attribute along with the unique values for each one of it. As shown in fig. 3.2 it is composed of 25 features, where 14 of them are text features, and the remaining 11 are numerical features. There are 105.542 articles in the dataset.

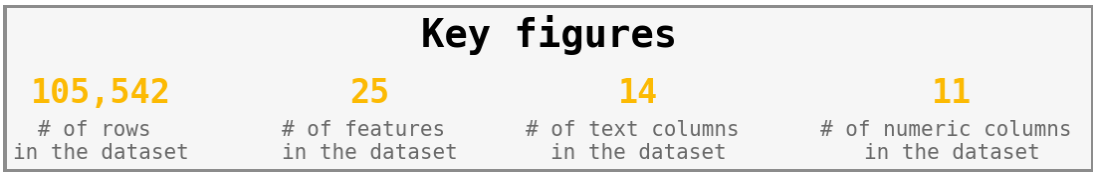

Figure 3.2: Articles dataset is composed of 25 features, where 14 of them are text features, and the remaining (11) are numerical features. There are 105.542 articles in the dataset.

Missing values Only one column of the dataset, called **detail description**, has missing values. However, the number of missing values is small as only 0.4% from the column is missing.

Articles images Along with the article's attributes it has been provided a folder of images too. Those images are placed in sub-folders starting with the first three digits of the *article id*; what we have found is that a leading zero in the name of the folder does not correspond to the first digits of the *article id* and it has to be stripped when looking for a product in the articles database: e.g., *0108775015* is *article id 108775015*. However, as mentioned in the competition description not all articles have an image.

Figure 3.3: These are sample images of some articles picked from the dataset. As we can see all the images contains the article with a background.

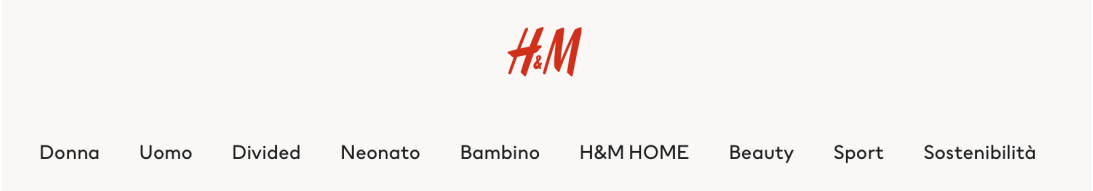

Figure 3.4: Index group name is correlated with options in the principal menu of the website.

Index group name attribute In the fig. 3.5 is evident that most of the articles belong to *ladies swear* (38%) and *baby/children* (32%). The smallest amount of articles (3.5%) belongs to *sport* group. It is correlated with options in the principal menu present on the website as shown in fig. 3.4. Each value contains subcategories described by another attribute: *index name*.

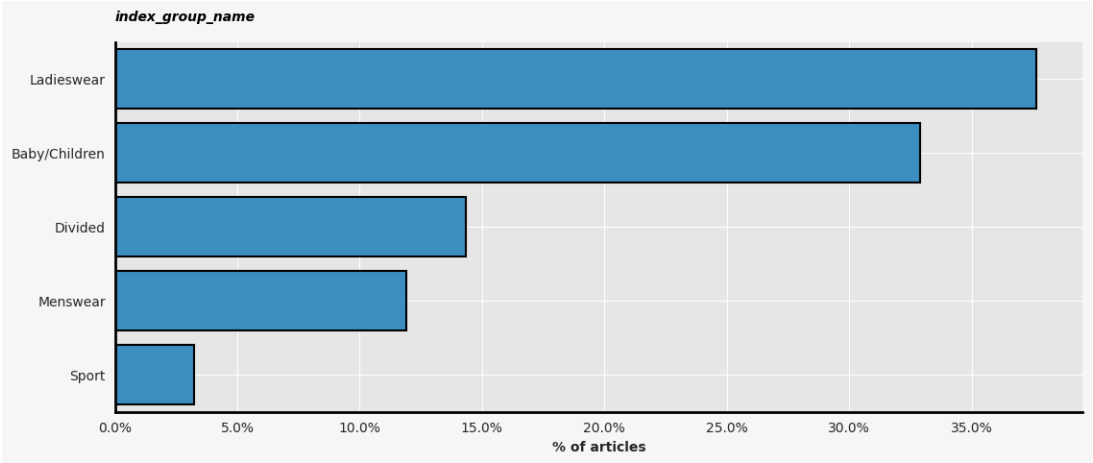

Figure 3.5: Plot of the index group name. We have a total of 5 different groups, the one which contains the most of the articles is the *ladies wear* group that represents more than 32% of the total dataset.

Index name attribute corresponds to subcategories of the *index group name* (Figure 3.6). Once again the dominant group is *ladies swear* (24%) while the second group is *divided* (14%) that is, as showed in fig. 3.7 a category for teenagers.

Product group name attribute it's an even finer category of the *index name*. Since this is a sub-category of the *index name*, that already highlights the gender, this one is instead independent of the gender, and most of the articles are associated with the general group of garments in turn divided into *upper* (41%), *lower* (18%) and *full body* (12.5%). As shown in fig. 3.8 articles like bags, cosmetics, or furniture represent contains a small

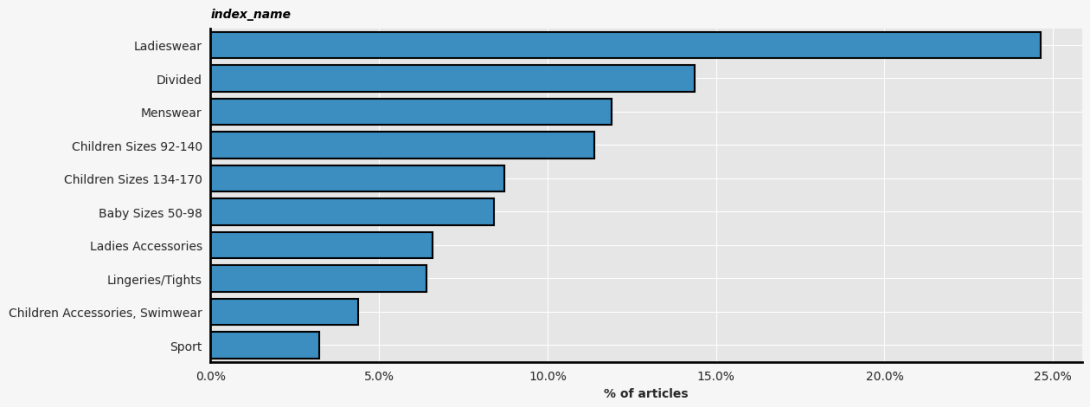

Figure 3.6: Index name represents a subcategory of the group name e.g., different children sizes, or specific accessories ladies items. Most of the articles belong to *ladieswear*.

Figure 3.7: Those are some images representative of the divided category. They are mostly articles for teenagers

number of items with respect to the *garment* category (less than 0.5%). Over 80% of the products lies in 4 out of product groups.

Product type attribute contains subcategories of *product group*. The Table 3.1 shows we have in total 131 different product types; accessories and shoes have the biggest number of these subcategories. From the four attributes analyzed until now, the hierarchy is index group –> index –> group –> type.

Department name attribute There are some *index groups* e.g., *baby/children* and *sport* that are not indexed by gender. The department name, instead, contains keywords like *ladies/men/girl/boy* that help to understand to which gender the article belongs as shown in the table 3.3.

Observing the *index group*, *index*, *product type*, and *department* is possible to catch which

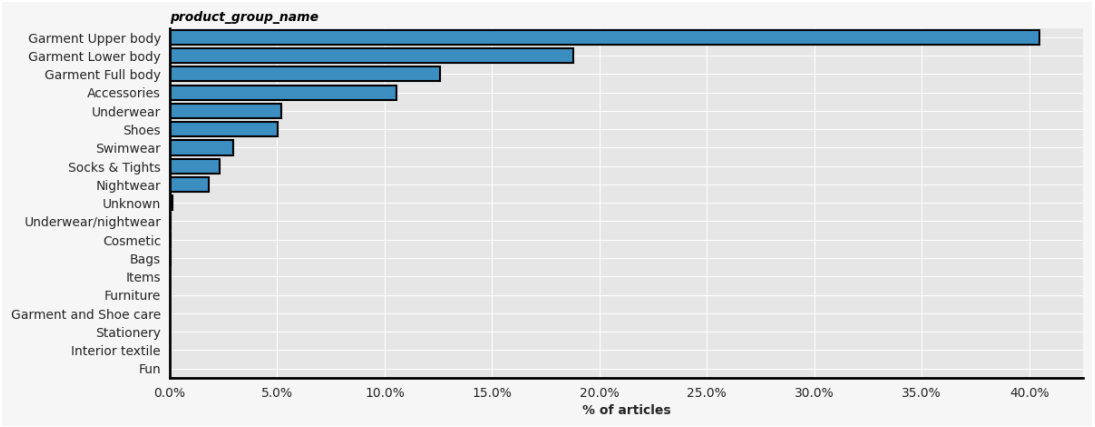

Figure 3.8: The *product group name* does not give information about the gender and most of the articles fall into the group of garment in turn divided into *upper* (41%), *lower* (18%) and *full body* (12.5%). There are some articles in other small categories too, e.g., furniture or bags (0.1%).

articles should be worn by a specific age group, gender, and in a specific season. Considering for example an article with *menswear* as *index group*, *menswear* as *index*, and *jacket* as *product type*. A customer who buys such an article is with high probability a man during the winter season. Furthermore, if a customer frequently buys products from *baby sizes 50-98*, it probably means that he is a parent of a child.

Color group name is the color classification of the article along with its shades and intensity. Most of the articles belong to the *black* and *white* color group, respectively 23% and 9.7% of the total. Some articles have a transparent color group, while for others this information is missing and the corresponding value is marked as *unknown*.

Perceived color value name is related to human color perception and may depend on the type of tissue. As shown in fig. 3.10 most of the articles have as value *dark* (42%). A small percentage of items, i.e., less than 0.1%, have as perceived color *undefined* or *unknown*. The fig. 3.11 shows some examples of *dusty light* perceived color. The last one is misleading since there are two products, one gray and one black, but still, the *dusty light* perceived color is predominant.

Perceived color master name is in line with the *color group* itself but it's more generic; it is not related to the concept of light's intensity but to the association made between the human perception and a specific color group, as showed in the fig. 3.12.

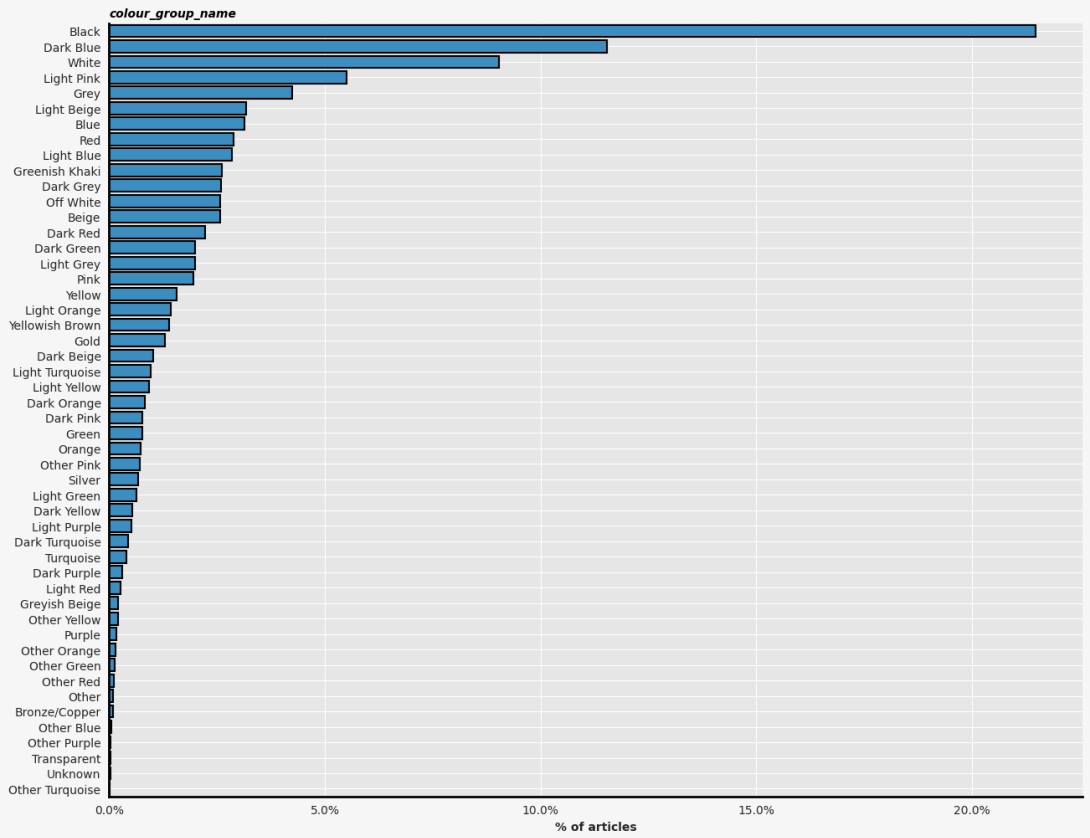

Figure 3.9: Color group name attribute is the color classification of the article along with its shades and intensity. Most of the articles are black or white but there are transparent articles too.

Graphical appearance name attribute represents the classification of the pattern showed upon the clothes e.g., random (fig. 3.14), melange (fig. 3.15) and embroidery (fig. 3.16) among the others. Other examples can be patterns with dots or transparent, but they are less common than others (less than 0.2%) as shown in fig. 3.13.

Description is associated with each article and contains information regarding the already analyzed attributes along with the fit of the article. The fig. 3.17 highlights that are the most common words used to describe the articles. This list of examples shows which is the mean length of the description and which type of information could contain:

- Jersey top with narrow shoulder straps.
- Micro-fibre T-shirt bra with underwired, molded, lightly padded cups that shape the bust and provide good support. Narrow adjustable shoulder straps and a narrow hook-and-eye fastening at the back. Without visible seams for greater comfort.
- Trousers in sweatshirt fabric with an elasticated drawstring waist, side pockets, a

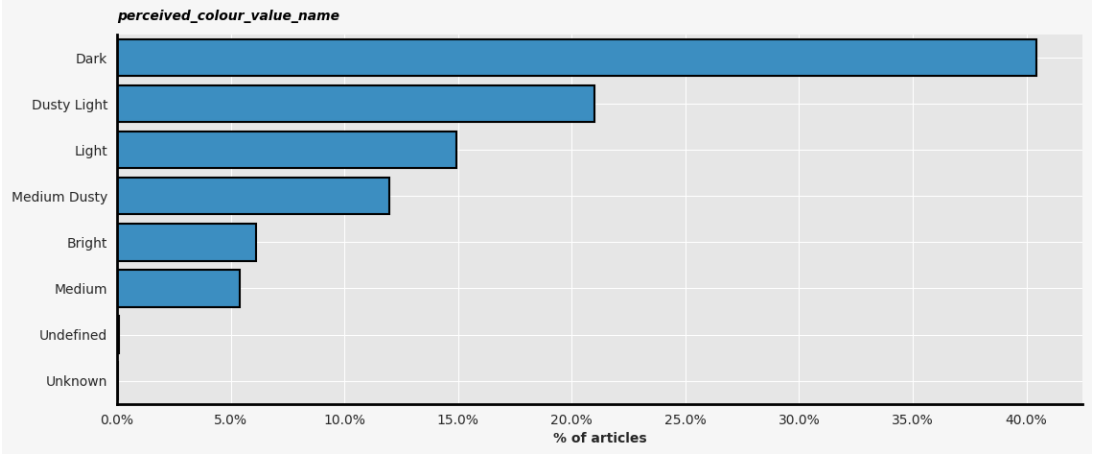

Figure 3.10: The perception of the color is different from the true one; it may depend on the type of tissue used and how the article appears under the lights.

back pocket, and ribbed hems. Soft brushed inside.

- Semi-shiny nylon stockings with a wide, reinforced trim at the top. Use a suspender belt. 20 denier.
- Tights with built-in support to lift the bottom. Black in 30 deniers and light amber in 15 deniers.
- Two soft bandeau bras in soft jersey with side support and a silicone trim at the top.
- Opaque matt tights. 200 denier.
- Semi-shiny tights that shape the tummy, thighs, and calves while also encouraging blood circulation in the legs. Elasticated waist.
- Sweatshirt in soft organic cotton with a press stud on one shoulder (sizes 12-18 months and 18-24 months without a press stud). Brushed inside.

Figure 3.11: Random Dusty light articles visualized. The perceived color is the light tone of the absolute color e.g., light gray, light pink, light blue. There are some strange cases, like the last one, in which we have two articles in just one image, probably they are sold together, the upper one is light gray, and the other one is dark gray.

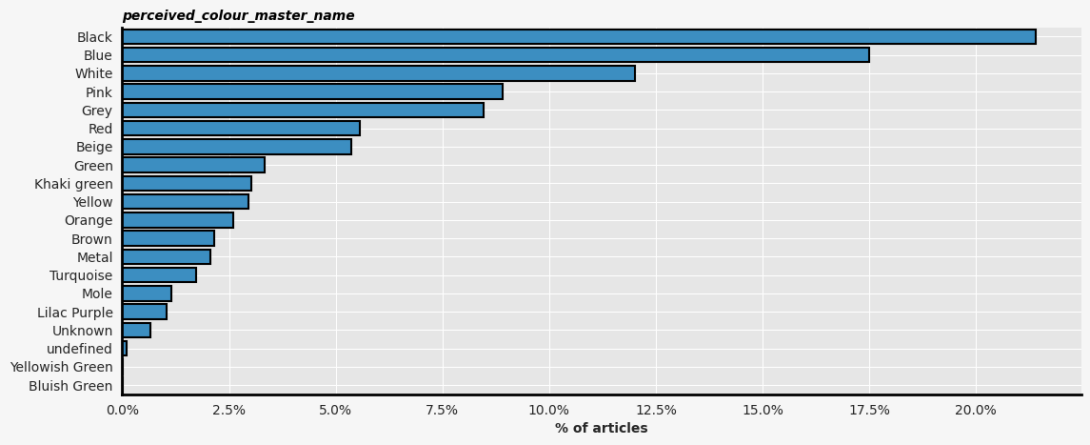

Figure 3.12: Perceived color master name is not related to the light perception or the intensity, but to the association made with a specific color group. Most of the articles are classified as black (23%), blue (17.5%), or white (12%).

Table 3.1: This table shows an attribute analysis for the articles table, highlighting the number of unique values we have for every single column.

Description	Unique values
Article Id a unique 9-digit identifier of the article	105.542
6-digit product code (the first 6 digits of *Article Id*	47.224
Name of a product	45.875
Product type number	131
Name of a product type	131
Name of a product group	19
Code of a pattern, namely graphical appearance	30
Name of a pattern, namely graphical appearance	30
Code of a color group	50
Name of a color group	50
Perceived color id	8
Perceived color value name	8
Perceived master color id	20
Perceived master color name	20
Department number	299
Department name	299
Index code	10
Index name	10
Index group code	5
Index group name	5
Section number	56
Section name	56
Garment group number	56
Garment group name	56
Detailed description of the article	43.404

Table 3.2: Subcategories of product group analysis, named also product type. As we can see each product group contains many subcategories. Accessories contain for example 38 different types of accessories, e.g., bracelets, and rings. Taking into consideration that there are 131 types plotting a bar chart may be useless.

Product group	**N. of subcategories**
Garment Upper body	15
Underwear	11
Socks & Tights	3
Garment Lower body	5
Accessories	38
Items	5
Nightwear	4
Unknown	1
Underwear/nightwear	2
Shoes	16
Swimwear	6
Garment Full body	6
Cosmetic	2
Interior textile	3
Bags	6
Furniture	1
Garment and Shoe care	6
Fun	1
Stationery	1

Table 3.3: The index name does not directly show the gender or any other information about the article, that is instead highlighted in the department name e.g., the gender.

Index name	Department name
Baby Sizes 50-98	Baby **Girl** Jersey Fancy
	Baby **Boy** Woven
	Newborn
	Other
Sport	**Ladies** Sport
	Men Sport
	Bottoms **Girls**
	Bottoms **Boys**

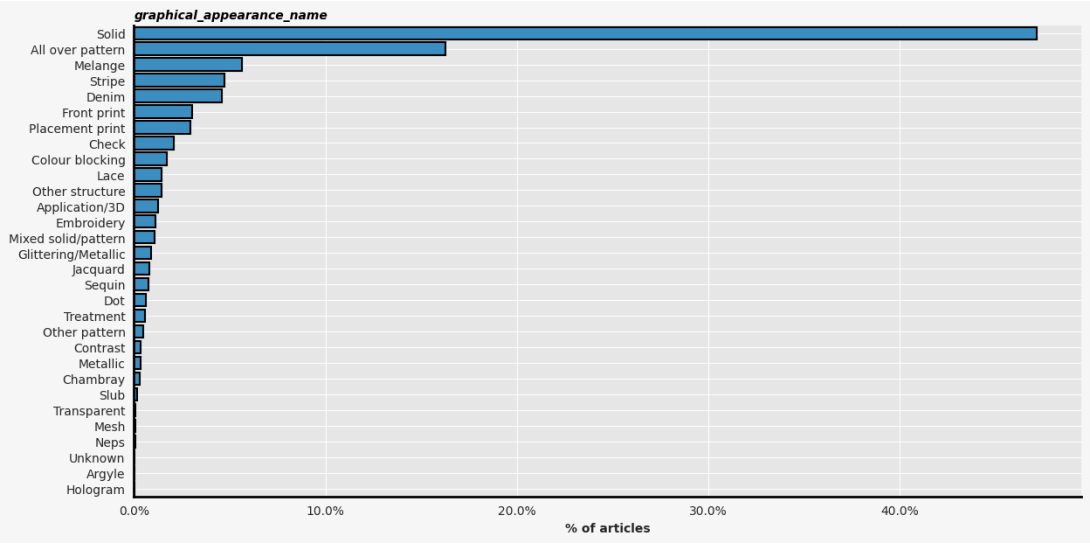

Figure 3.13: Graphical appearance name attribute classifies the pattern showed up on the clothes e.g., solid, stripe, denim, and jacquard. Other examples can be patterns with dots or transparent, but they are less common than others (less than 0.2%).

Figure 3.14: Articles with random patterns have different images printed over them, e.g., stars, pets, or random shapes.

Figure 3.15: Articles with melange patterns have in common a mixture of colors e.g., light gray and dark gray together.

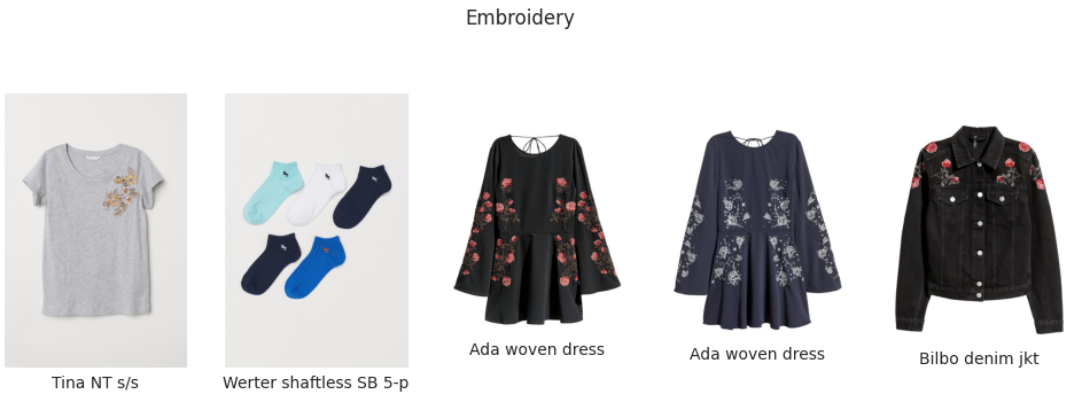

Figure 3.16: Articles with embroidery patter have in common embroideries over them; the most common type of embroidery is flowers.

Figure 3.17: The bigger the word, the bigger the number of times that word appears inside descriptions of articles.

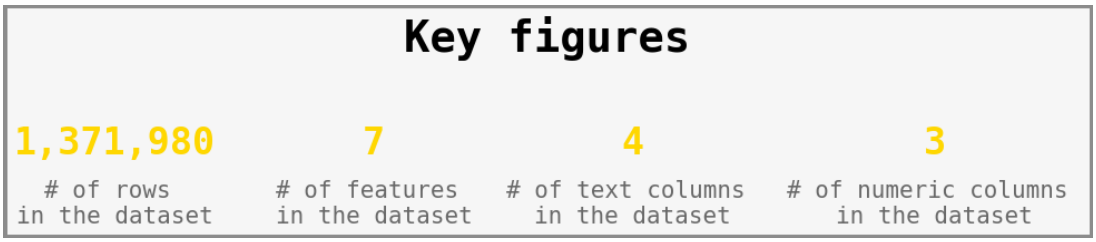

Figure 3.18: Customers dataset is composed of 7 features: 4 of them are text features, and the remaining 3 are numerical features. There are 1.371.980 customers in the dataset.

Table 3.4: Attribute analysis for the customer's table, with the number of unique values we have for every single column and its description.

Description	Unique values
Customer Id is a unique identifier of the customer	1.371.980
FN	1 or NaN
Active	1 or NaN
Customer member status in a club	3
Fashion news frequency of sending a communication to the customer	4
Age of the customer	Value between 0 and 1
Postal code	352.899

3.1.2. Customers

This database contains information about the customers; Table 3.4 summarizes relevant statistics of the customer's dataset, such as the description of each attribute along with the unique values for each one of it. As shown in fig. 3.18 it is composed of 7 features: 4 of them are text features, and the remaining 3 are numerical features. There are 1.371.980 customers in the dataset. The table 3.5 analyses the number of missing values for each attribute. *Fashion news* value is missing for 895.050 customers: this means that they are customers not registered online but that they use to buy directly in physical stores. The same applies to *active* attribute. There are no missing values for the *postal code* of the customer, meaning that there is location information available for each customer.

FN is Fashion News Newsletter, the wish of the user to receive fashion news: 1 means that the user wants to receive fashion news, NaN or 0, otherwise. Looking at the data, the number of customers that have transactions made only in physical stores is equal to the

Table 3.5: Fashion news value is missing for 895.050 customers: this means that they are customers not registered online but that they use to buy directly in physical stores. The same applies to *active* attribute. There are no missing values for the postal code of the customer, meaning that there is location information available for each customer.

Attribute	Missing values
Customer Id	0
FN	895.050
Active	907.576
Club member status	6.062
Fashion news frequency	16.009
Age	15.861
Postal code	0

number of missing values for this attribute. This means that only customers registered online can accept or refuse to receive fashion news.

Active attribute represents the wish of the user to receive communications. It can have two values: 1 or NaN. Even in this case, the number of missing values for that attribute is equal to the number of users that have transactions only on physical stores. This means that only customers registered online can accept or refuse to receive communications. As shown in the fig. 3.20 more than 66% of users refused to receive communications or have a missing value. The percentage of customers that have both *FN* and *active* equal to 1 is the same: 33.85%: all the customers that have *active* have also the *FN* status.

Club member status is a loyalty program that allows members to earn points and redeem rewards for their purchases. Customers can join H&M Club by creating an account on the H&M website and earn points by making purchases, writing reviews, and participating in other activities. H&M Club also offers exclusive promotions and early access to sales for its members. As shown in fig. 3.21 most of the customers have an *active* membership status (92%) while the 7% of them have the *pre-create* status. There is no customer with *left club* status. The remaining 1% of customers have a missing value, marked with N/A.

Fashion news frequency is the frequency with which customers want to receive fashion news. As showed in the fig. 3.22 it contains only 5 different values: *NONE* (63%), *regularly* (34%), N/A (2%), *monthly* (0.8%) and *none* (0.2%). Both *NONE* and *None* represent

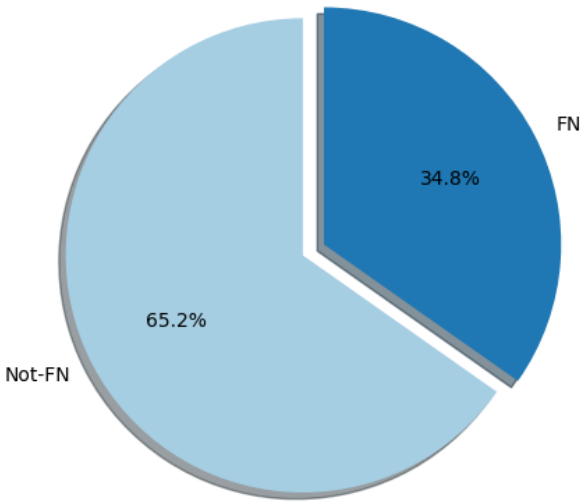

Figure 3.19: More the 65% of users refused to receive fashion news newsletter. NaN and 0 are part of the remaining 35%.

the refusal to receive fashion news and, to improve data quality, they can be merged. It can happen as well that when FN is equal to 1, the corresponding value associated with *fashion news frequency* is equal to *none* or N/A.

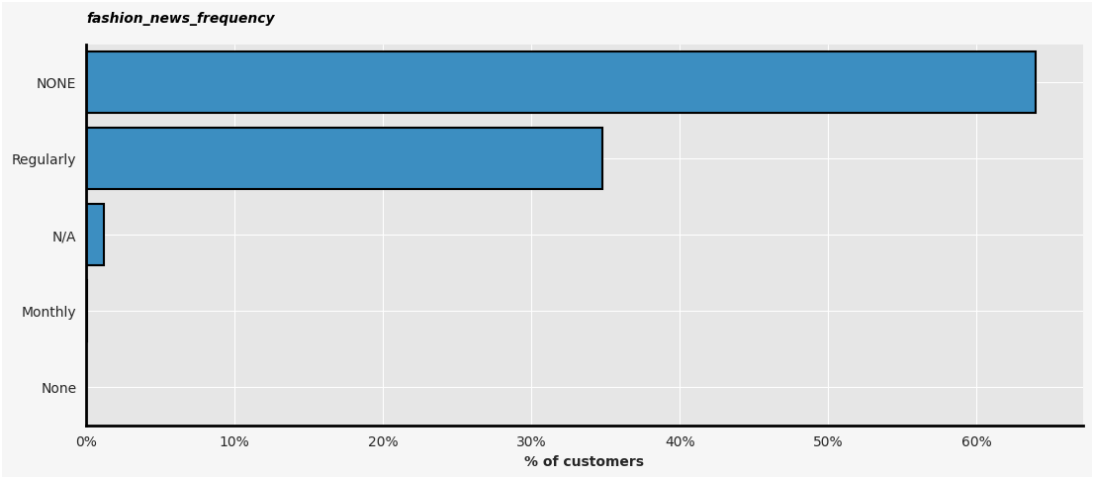

Figure 3.22: It contains only 5 different values: *NONE* (63%), *regularly* (34%), N/A (2%), *monthly* (0.8%) and *none* (0.2%). Most customer refuses to receive fashion news frequently and few customers decide to receive it regularly or monthly. Both *none* and N/A are associated with missing value information.

Customer age is a normalized value representing the age of the customer. As shown in the fig. 3.23, re-scaling that value in a range between 0 and 100, there are two main age

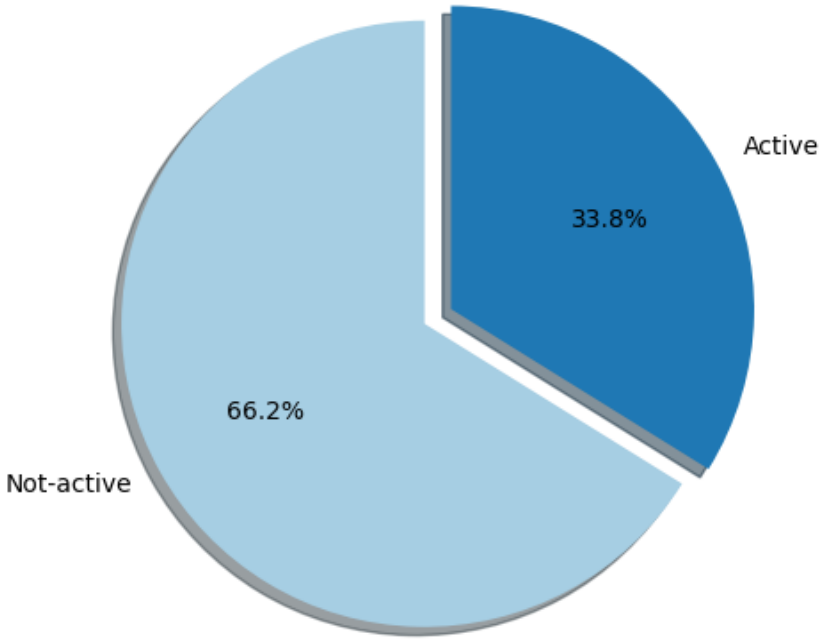

Figure 3.20: More than 66% of users refused to receive communications. This plot considers also the missing values we have as non-active.

groups of customers: around 20-30 years old and 45-55 years old. The oldest customer is 99 years old and the mean age is 32 years old.

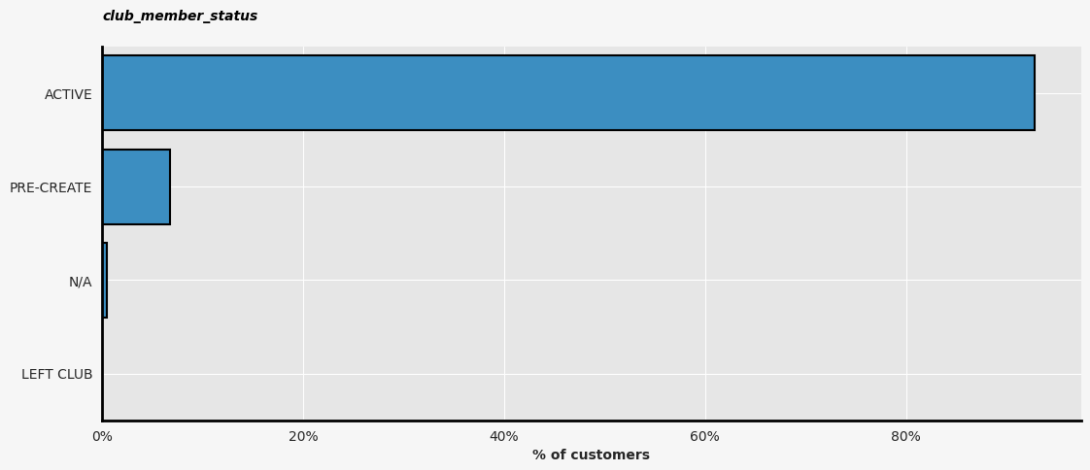

Figure 3.21: It has 4 values: the most common value is *active* (92%), *pre-create* (7%), *left club* and *NaN* (1%).

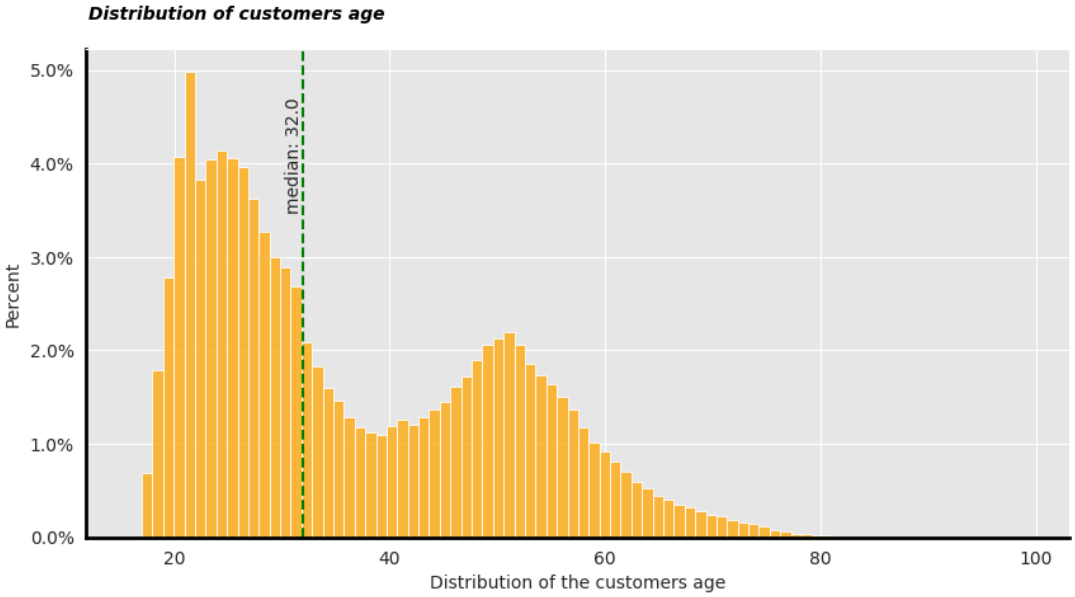

Figure 3.23: The distribution shows that there are two main age groups of customers: around 20-30 years old and 45-55 years old. The oldest customer is 99 years old and the mean age is 32 years old.

The fig. 3.24 shows that there is a similar trend between active and not active users, despite their age. This means that there is no correlation between the age of the customer and the choice to receive or not communicate.

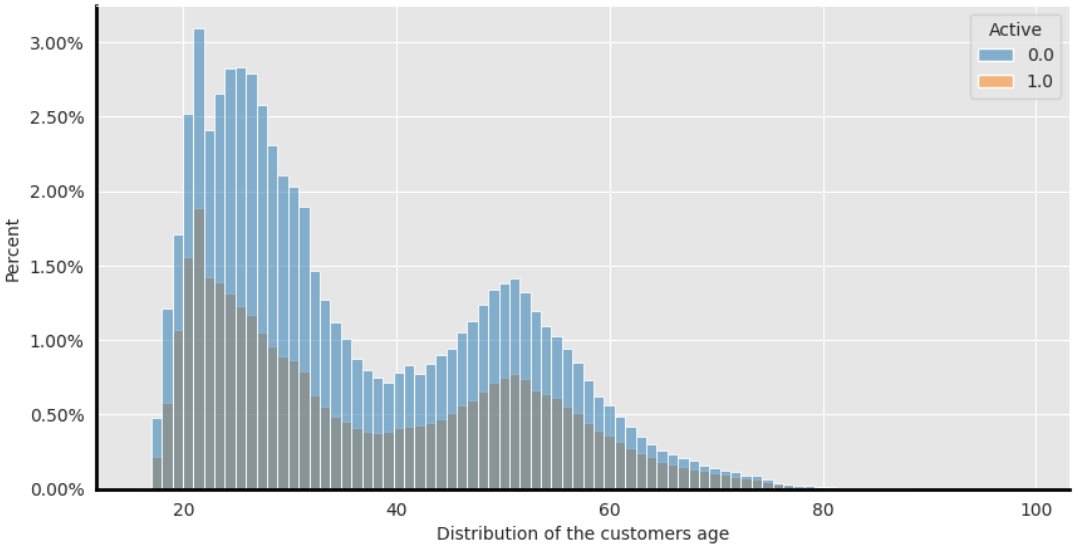

Figure 3.24: The peak between active (light blue) and inactive (grey) are almost around the same ages. That means that the choice trend to be active or not with newsletters does not depend on age.

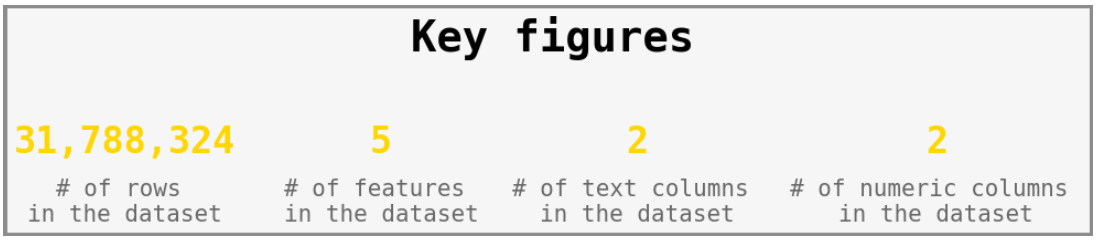

Figure 3.25: Transaction dataset is composed of 5 features, where 2 of them are text features, and the remaining 3 are numerical features. There are 31.788.324 transactions in the dataset.

Table 3.6: This table shows an attribute analysis for the transactions dataset

Description
Date of a transaction in format YYYY-MM-DD but provided as a string
Customer identifier of the customer which can be mapped to the *customer id* column in the *customers* table
Article identifier of the product which can be mapped to the *article id* column in the *articles* table
Price paid
Sales channels which can be online or physical

3.1.3. Transactions

This database contains information about the transactions; Table 3.6 summarizes relevant statistics of the transactions dataset, such as the description of each attribute along with the unique values for each one of it. As shown in fig. 3.25 it is composed of 5 features, where 2 of them are text features, and the remaining 3 are numerical features. There are 31.788.324 transactions in the dataset. No transaction contains missing values. The sales channel has only two possible values: *1*, the online channel, and *2*, the physical store channel.

The transaction table holds all transactions that happened whether returned later or not. If the same customer has made two subsequent transactions for the same item but of different sizes or colors it means that the first transaction is not useful or meaningless for that customer: but this is computationally infeasible.

The fig. 3.26 shows the price distribution considering all the transactions. This value is normalized and has values between 0 and 1. Since most of the items are sold at a price lower than 0.1 is possible, as shown in fig. 3.27 to remove those outliers and focus on the

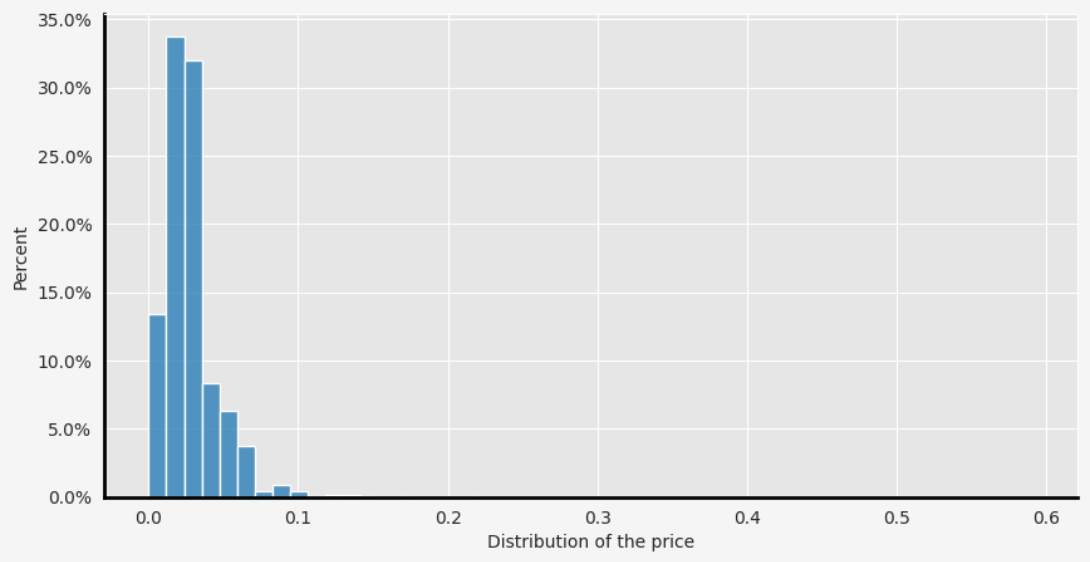

Figure 3.26: The price value is normalized and comprised between 0 and 1. Most of the items are sold at a price lower than 0.1

price range that contains most of the transactions. Even if there are some outliers for the price, as the fig. 3.28, the mean price change in time for the most popular *product groups*, i.e., *shoes*, *garment full body*, *bags*, *garment lower body*, *underwear/nightwear* do not change considerably during the two years.

The dataset contains all the transactions from *2018-09-20* to *2020-09-22*. We have almost 2 years of transactions, most of them during the Covid pandemic, which can be used to extract patterns from the dataset on the common behavior of customers during that period. This also explains why the number of transactions in physical stores is almost 0 during the first months of restriction, caused by a forced closure of stores around the World.

The fig. 3.29 shows the number of transactions made during each day of the dataset. There are a lot of spikes due to the weekend or important periods of the year e.g., discount promotions or holidays. In order to visualize better what is the amount of transactions made during each month of the dataset, the fig. 3.30 aggregate the transaction by months using a box plot. This highlight also that the usual number of transactions made each day lies in the range of 25.000 and 80.000. There are some outliers e.g., spikes during summertime or holidays and special discount periods and drops during winter.

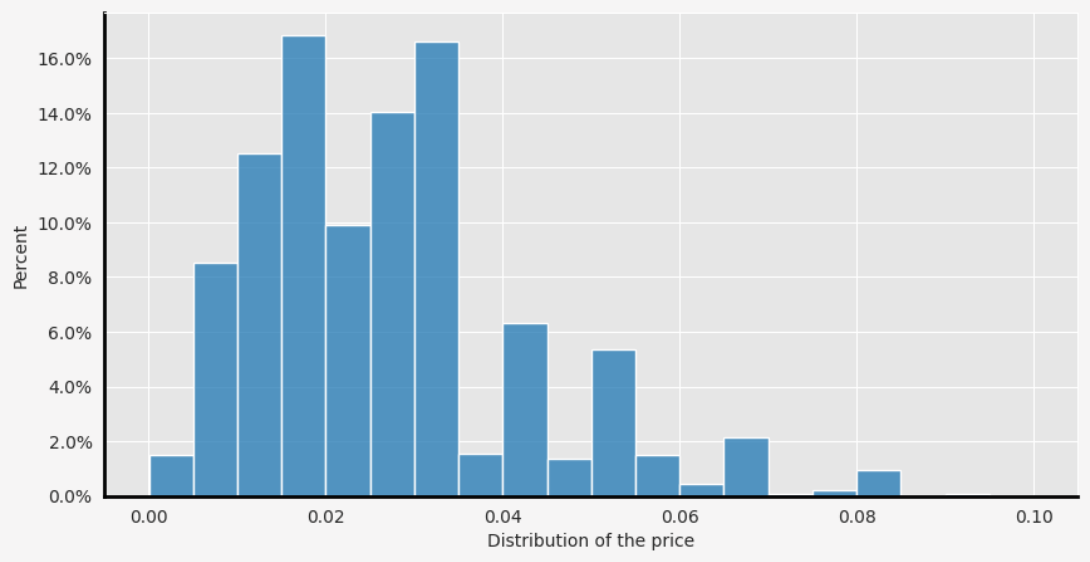

Figure 3.27: This plot focuses on the price range that contains most of the transactions, in order to discard outliers.

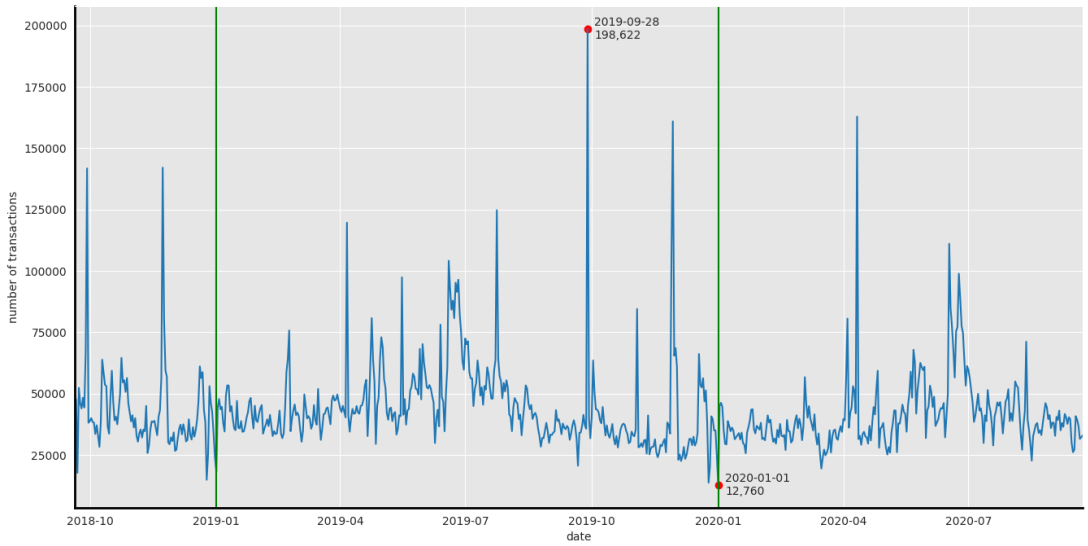

Figure 3.29: This is the plot of the number of transactions made during each day of the dataset. There are a lot of spikes due to the weekend or important periods of the year e.g., discount promotions or holidays.

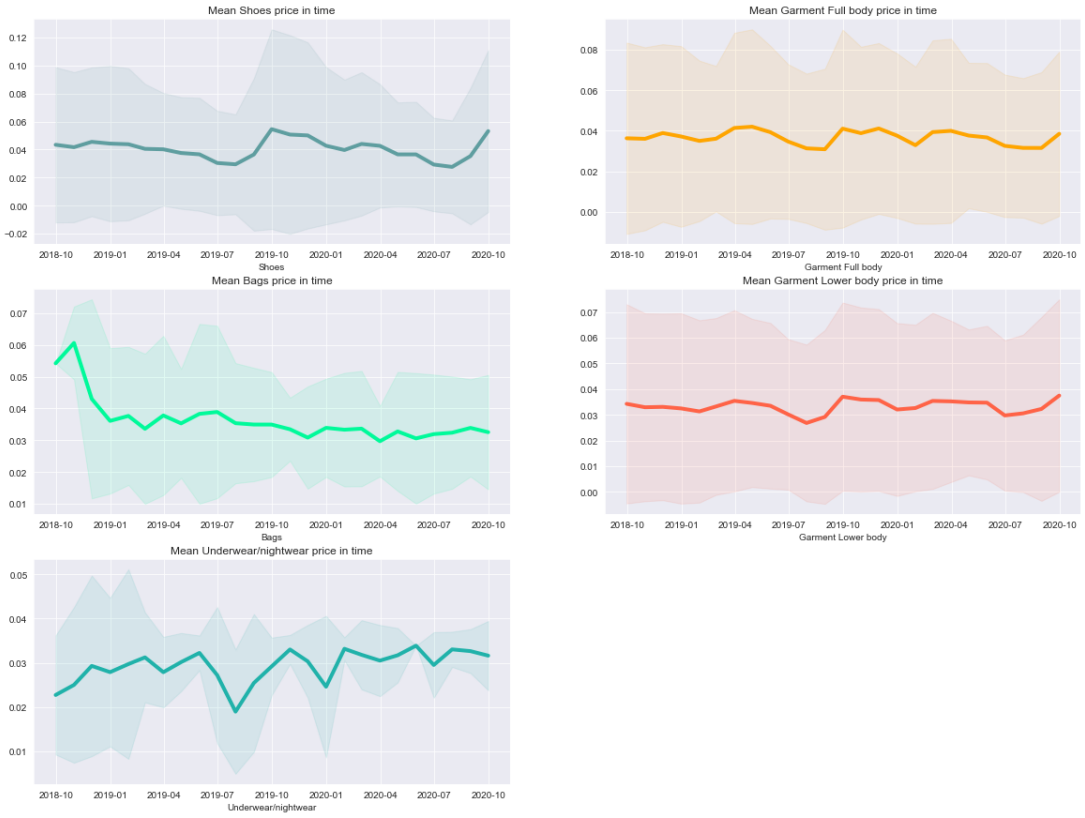

Figure 3.28: There graphs shows the mean price change in time for the most popular *product groups* i.e., *shoes*, *garment full body*, *bags*, *garment lower body*, *underwear/nightwear* do not change considerably during the two years.

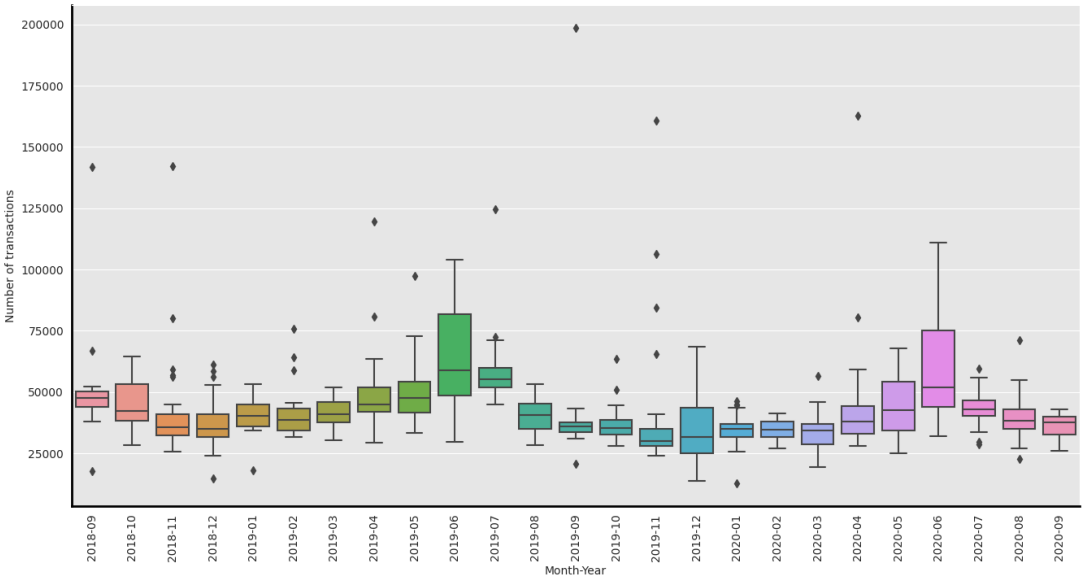

Figure 3.30: Box plot of transactions grouped by month. The number of transactions is very high in both June 2019 and June 2020, when users usually do shopping before going on holiday.

The fig. 3.31 shows how many transactions each customer, on average, customers does. Most of the customers (50% of the total) have done less than 10 transactions during the 2 years of the dataset.

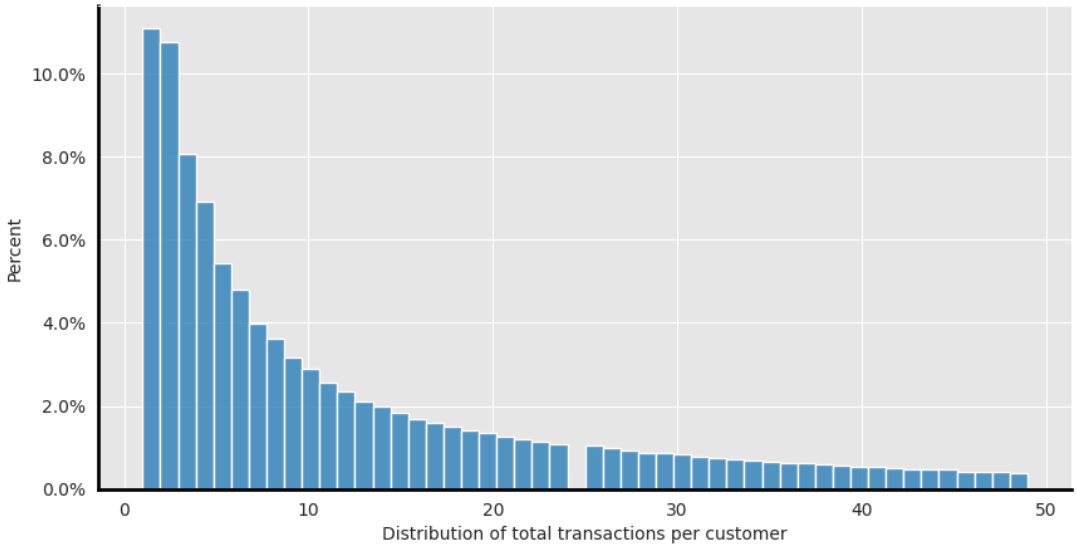

Figure 3.31: Average transactions per user: most of the customers have done less than 10 transactions. These customers are the 50% of all the ones in the dataset.

Table 3.7: Each category is a unique combination of three article's attributes, i.e., *index group*, *index*, and *product type*. These categories have been used to analyze the monthly sales of each category with respect to the total amount of sold articles belonging to the same category during the two years of available transactions.

Index group name	Index name	Product type
Ladieswear	Ladieswear	Jacket
Divided	Divided	Jacket
Menswear	Menswear	Jacket
Divided	Divided	Sweater
Menswear	Menswear	Sweater
Ladieswear	Ladieswear	Sweater

3.2. Extracting Relevant Features

The dataset, as shown in section 3.1, is given along with users and article attributes. From them, we extracted other relevant features from the data. The aim is to spot patterns, association rules, and other characteristics of both users and items not available from the starting dataset. These additional features allowed the final model to increase its effectiveness, as shown in chapter 4. These analysis has been done to create new additional features and candidate generation strategies, in addition to the provided dataset.

3.2.1. Product Sales Seasonality

Apart from some exceptions e.g., accessories and bags, most of the articles sell well depending on the season: articles that sell well in late September represent good candidates to be recommended for the test week, which corresponds to the last week of September i.e., from *2020-09-23* to *2020-09-30*.

We checked the trend of sales by grouping articles into categories; each category is identified by the unique combination of three of the attributes already available in the article's dataset, i.e., *index group*, *index*, and *product type*, analyzed in the section 3.1.1.

Once generated these categories we plotted for each of them the percentage of monthly sales with respect to the total sold amount of the same category during the entire dataset period of two years. The table 3.7 shows some of the generated categories, while the fig. 3.32 show the trends of sales of those categories. The figure highlights the fact that both *jackets* and *sweater* sell better with the approaching of the winter season.

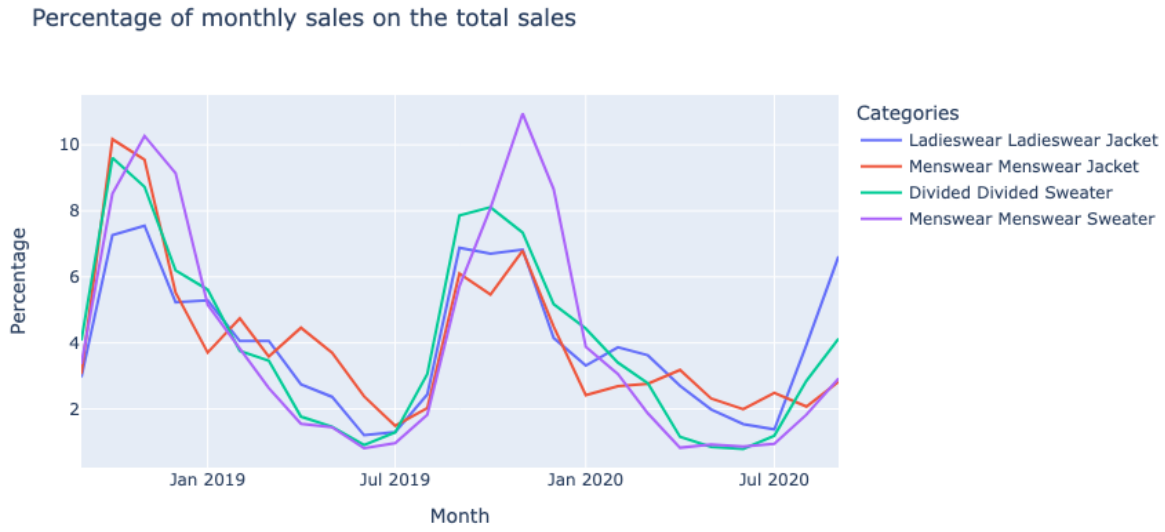

Figure 3.32: The categories in the legends are only some examples and are generated combining i.e., *index group*, *index*, and *product type*. *Jackets* and *sweater* sells better during the approaching of winter seasons. Then this value decrease, and is at its lowest during summer.

The fig. 3.33 shows how different categories of articles have different sales trends. In the case of *shorts*, we have a peak during the summer, with a decrease in sales with the approaching of the winter season. This means that it is possible to associate each category with the season in which that category sees an increase in sales. Knowing that the test week falls in September i.e., summer season, it is possible to use that information to make a first filtering on the list of possible candidates to recommend to the customers during that week.

Correlation of sales between two items is a measure of how closely the sales of the two items are related. Correlation is a statistical concept that measures the strength and direction of the linear relationship between two variables. In the context of sales, if two items have a positive correlation, it means that when sales of one item increase, sales of the other item tend to increase as well. On the other hand, if two items have a negative correlation, it means that when sales of one item increase, sales of the other item tend to decrease [85]. As Lan and Liu [60] says, to calculate the correlation of sales between two items, it is needed to gather data on the sales of both items over a period of time and compute the correlation coefficient. The correlation coefficient can range from -1 to $+1$, where a value of -1 indicates a perfectly negative correlation, 0 indicates no correlation,

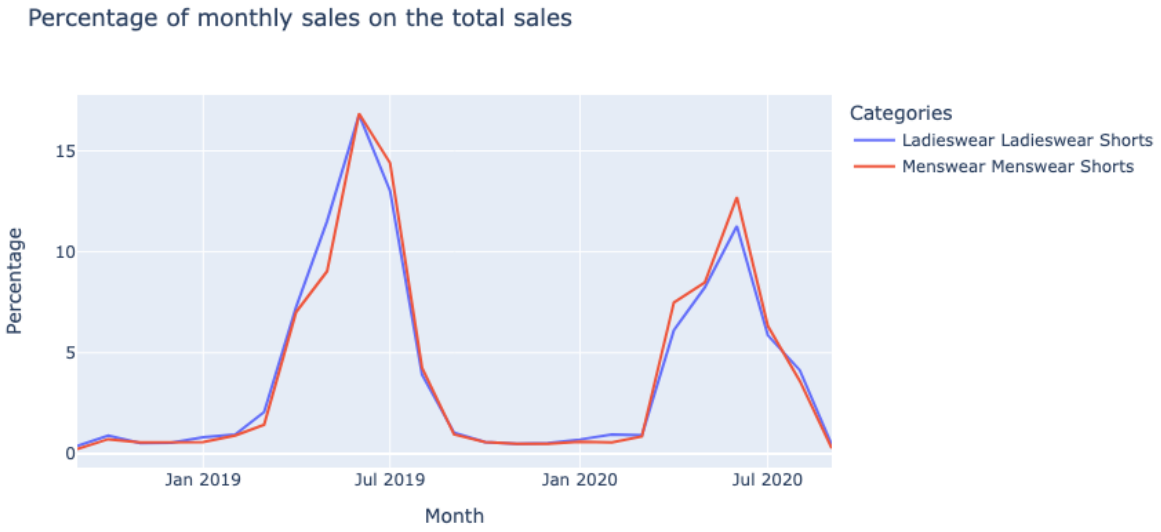

Figure 3.33: Considering *shorts* as *product type* for different *index group* and *index name*, e.g., *ladieswear* and *menswear*, the trend is opposite to the ones in the fig. 3.32. These categories sell better during the summer seasons and the monthly sales reach their lowest point during the winter season.

and +1 indicates a perfectly positive correlation.

The fig. 3.32 has been generated applying the Principal Component Analysis to the monthly sales trend of each category, with respect to the *leadieswear leadieswear jacket* category, used as reference for the autumn seasonality trend. Principal Component Analysis (PCA) is a statistical technique that is often used in data analysis and dimensionality reduction. PCA is a mathematical method that transforms a set of correlated variables into a smaller set of uncorrelated variables, called principal components [39]. The purpose of PCA is to identify the underlying patterns in the data by finding a smaller number of variables that explain most of the variability in the original data set [83]. The principal components are linear combinations of the original variables that are ordered by their ability to explain the variance in the data. PCA can be used to reduce the dimensionality of a large dataset, making it easier to visualize and analyze. PCA can also be used to identify important variables that contribute most to the variation in the data. This can help in developing predictive models or in understanding the underlying structure of the data.

The fig. 3.32 shows that the category *ladieswear ladieswear jacket* reaches its peak of sales during autumn; this means that it can be used as *autumn sales indicator*. Calculating

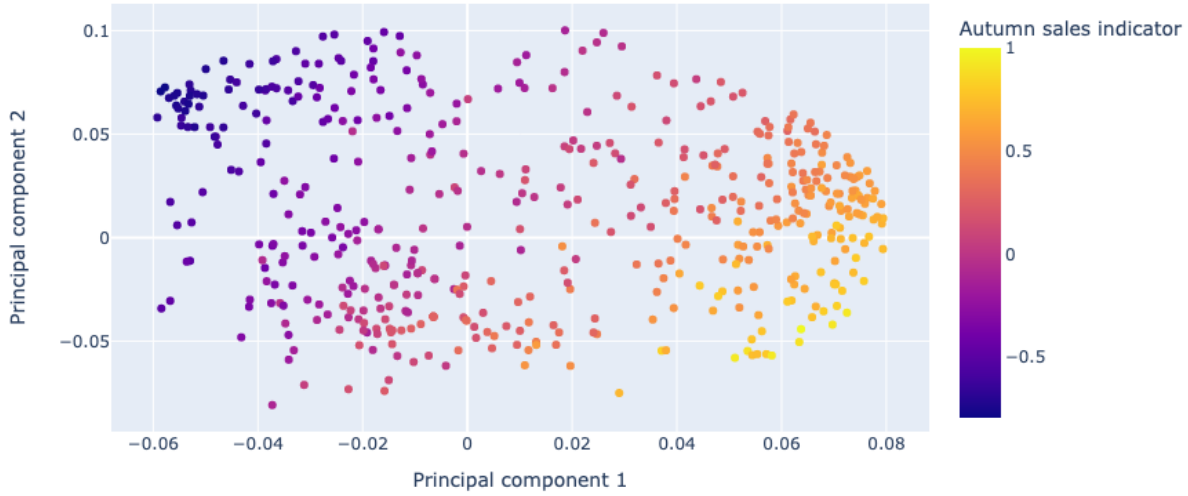

Figure 3.34: Each point is a product category, the higher the value, the higher the correlation of sales with the autumn season. Since the category *ladieswear ladieswear jacket* represents the principal *autumn sales indicator* it has a value of 1 i.e., high correlation. A category like *ladieswear ladieswear sunglasses* instead have instead a very low correlation i.e., blue points, because the sales of sunglasses decrease during autumn and the approaching winter season.

the correlation between the sales of other categories with *ladieswear ladieswear jacket* category is possible to understand if sales of a category are related to the autumn season or not. As expected, in the 3.34, articles with very low correlation, i.e., that are not likely to be bought in autumn and represented with the blue circle, are for example t-shirts, sandals, or shorts. On the other hand, articles like gloves or cardigans, represented with yellow/orange circles, have a strong correlation with the autumn seasons and, as a consequence with the *ladieswear ladieswear jacket* category.

Gaussian mixture has been used to cluster the categories into 4 groups that represent different types of trends in the monthly sales, over the two years of transactions. Those clusters are not connected to a specific season, they only represent different trends. Anyway, the categories belonging to the fig. 3.32, that fall into the cluster type 2, are strictly connected to the autumn/winter season. In the same way, the categories belonging to the fig. 3.33, that fall into the cluster type 3, are strictly connected to the sprint/summer season. Categories that belong to cluster 0 do have not a strong correlation with a specific season of the year. As shown in fig. 3.36 categories like *divided divided belt* have a monthly

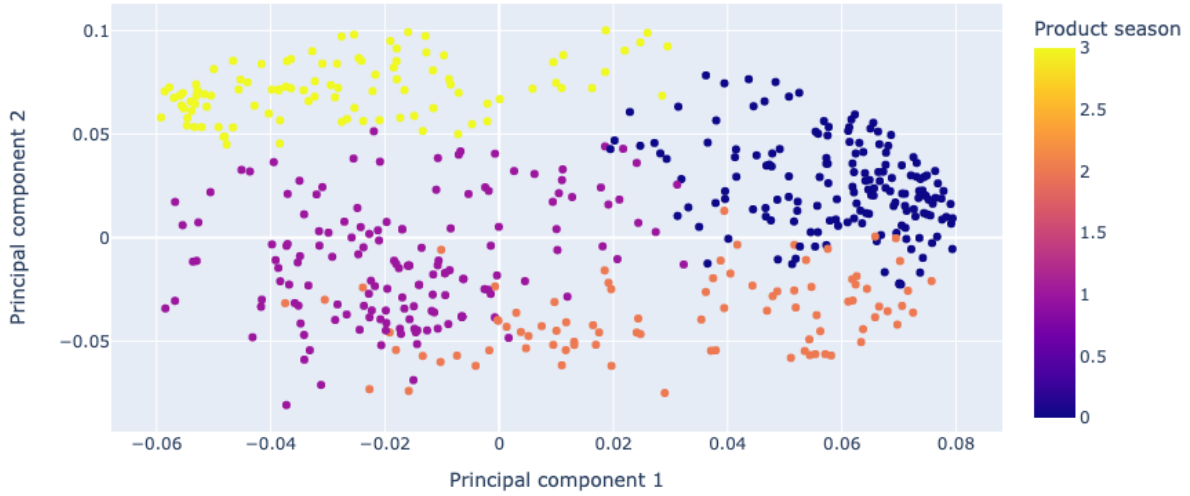

Figure 3.35: Product seasonal type represents the cluster, between 0 and 3, to which a category belongs to. Categories that belong to cluster 0 do have not a strong correlation with a specific season of the year. Categories that belong to cluster 2 have a strong correlation with the autumn/winter seasons. Categories that belong to cluster 3 have a strong correlation with spring/summer seasons.

sales trend uncorrelated with respect to the month. The fig. 3.37 shows instead the trend for the category *divided divided bracelet* belonging to the cluster 1. This represents a totally different trend where there are some months where sales are equal to zero. This specific category contains accessories, meaning that articles belonging to this cluster are not only independent from a specific season but that they are not available during the whole year to be bought from customers.

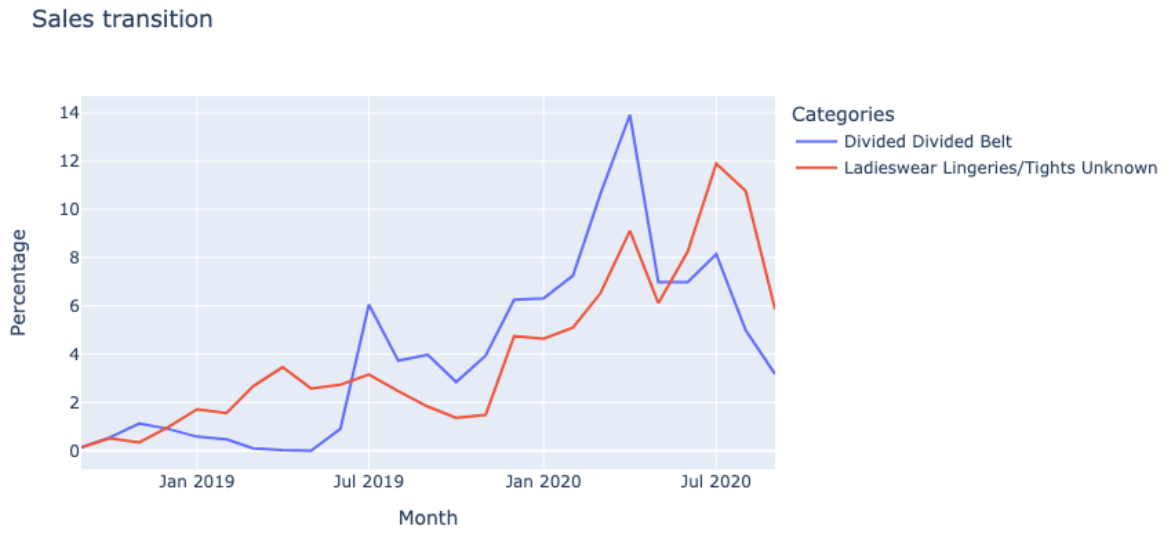

Figure 3.36: Example of categories belonging to cluster 0. The sales of these categories are not connected to the period of the year.

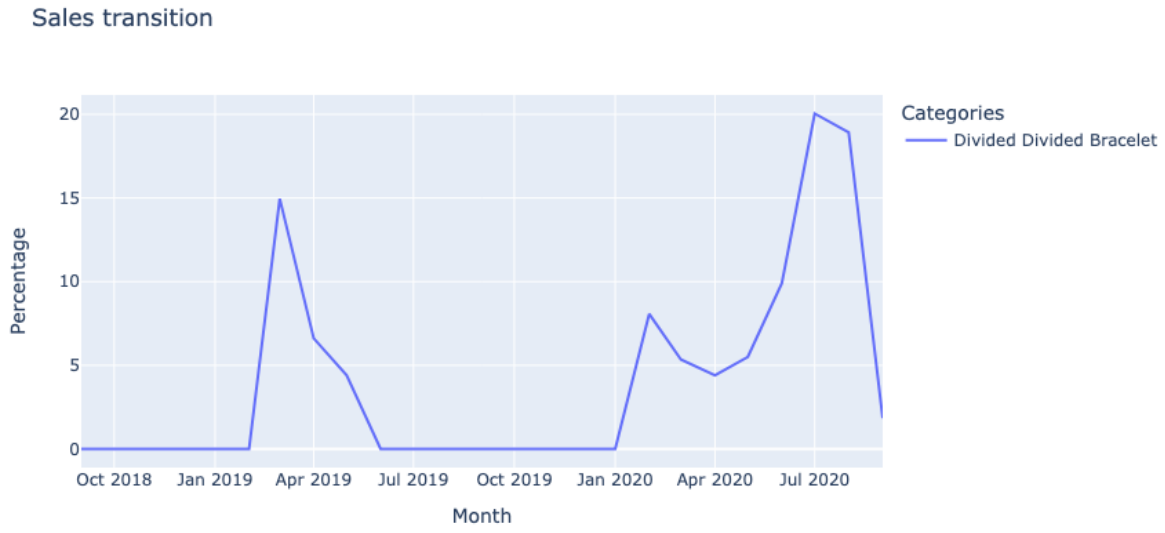

Figure 3.37: Example of categories belonging to cluster 1. There are periods of the year in which the number of sales of that category is null, meaning that items belonging to that category are not available during the entire year.

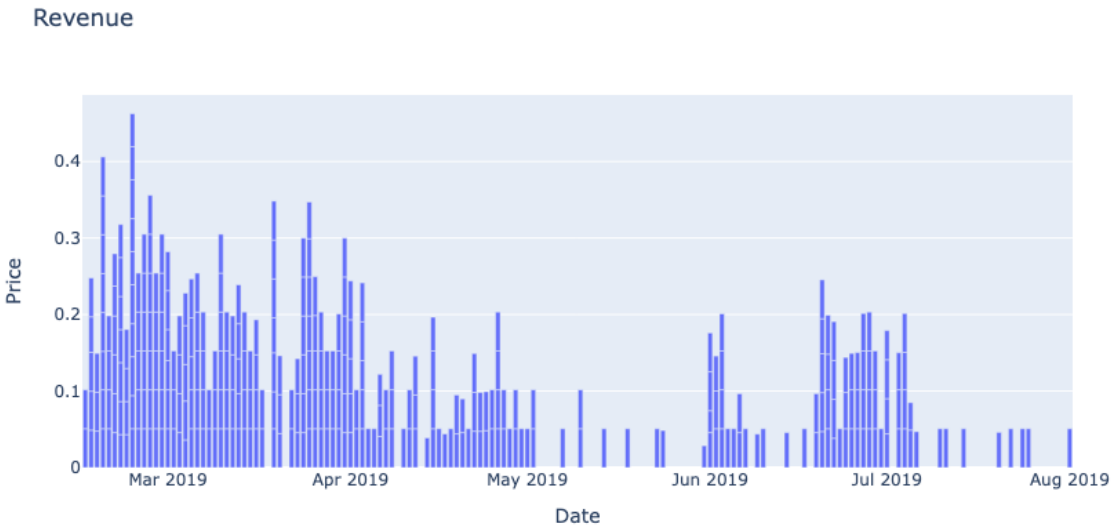

Figure 3.38: The product with article id *698276001*, that corresponds to a *swimsuit*, has not been sold anymore after august 2019. We can consider it as no longer available in the stock a remove it from the list of candidates.

3.2.2. Out of Stock Product

Considering a specific article, if e.g., the sales before 2019 account for more than 95% of the total sales, we can assume that the article is no longer available by the end of 2020 and exclude it from the list of candidates to be recommended. The fig. 3.38 shows that the product with article id *698276001*, that corresponds to a *swimsuit*, has not been sold anymore after August 2019. We can consider it as no longer available in the stock and remove it from the list of candidates.

3.2.3. Repurchase

In the fast fashion industry, repurchase behavior is used as a candidate generation technique in recommender systems [73]. Fast fashion companies, H&M is an example, offer a wide range of clothing items that are frequently updated and replaced with new items, and customers may be more likely to repurchase items they have already bought, especially if they are happy with the fit, quality, and style of the item. Using repurchase behavior as a candidate generation technique helps the recommender system identify items that are likely to be of interest to the user based on their previous purchase history. These items can then be recommended to the user as potential candidates for future purchases. We analyzed this behavior using the available transactions to see if customers decided to buy

again the same item, at different levels of granularity i.e., same *article id*, *product id* or *category* (see section 3.2.1).

The fig. 3.39 shows the percentage of customers who repurchased the same item, with exactly the same *article id* (e.g., a blue t-shirt), *product code* (e.g., previous t-shirt of different size or color), and category (e.g., any t-shirt in the dataset). The results obtained show that there is an increase in percentage repurchases if we increase the time window or the granularity. Considering the *article id* 5.1% of the customers repurchased the same item within 1 week, 6.2% within 2 weeks, and 6.6% within 3 weeks. Instead, considering the *product code* 6.8% of the customers repurchased the same item within 1 week, 8.5% within 2 weeks, and 9.4% within 3 weeks. Regarding instead the *category* 10.8% of the customers repurchased the same item within 1 week, 14.3% within 2 weeks, and 16.5% within 3 weeks. This percentage increase let us consider repurchasing as a candidate generation strategy.

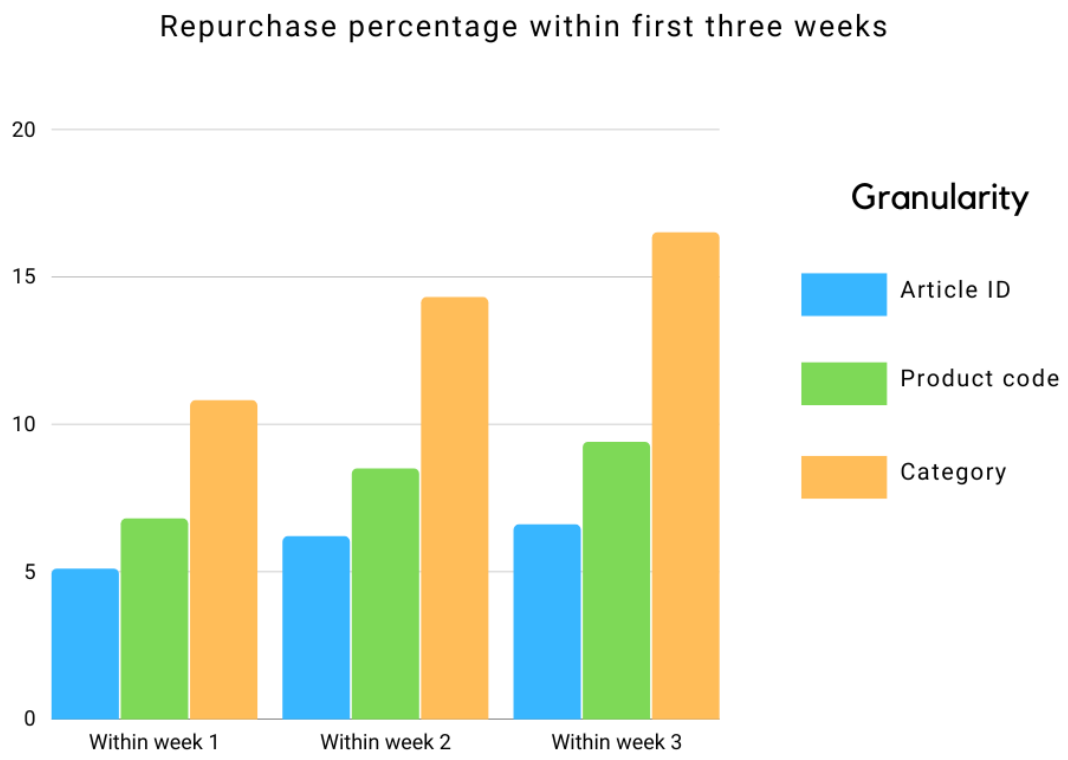

Figure 3.39: This plot shows the percentage of users that, at least one time during the two years, purchased again an item with the same *article id* (e.g., a blue t-shirt), *product code* (e.g., previous t-shirt of different size or color), and category (e.g., any t-shirt in the dataset). The *category* (see section 3.2.1) is an attribute generated by a unique combination of *index name*, *index group name* and *product group*. Considering the *article id* 5.1% of the customers repurchased the same item within 1 week, 6.2% within 2 weeks, and 6.6% within 3 weeks. Instead, considering the *product code* 6.8% of the customers repurchased the same item within 1 week, 8.5% within 2 weeks, and 9.4% within 3 weeks. Regarding instead the *category* 10.8% of the customers repurchased the same item within 1 week, 14.3% within 2 weeks, and 16.5% within 3 weeks. This percentage increase let us consider repurchasing as a candidate generation strategy.

3.3. Leaderboard Public Solutions

The goal of the challenge was to recommend a list of 12 items to each user in the dataset. The recommendations are scored using the $MAP@12$ (see section 2.6.2) with respect to the test week, i.e., the first one after the dataset period, which goes from $2020-09-23$

to $2020-09-30$. The data of the test week is split in order to generate two leaderboards. The public leaderboard was available during the whole challenge period and is generated using only the 5% of the total test data. The private leaderboard has been released after the deadline and is generated using the other 95% of the test data. The winners of the challenge have been selected considering only the private leaderboard. In this section, we analyze several winning approaches that scored in the top 10 on the private leaderboard. [5]

Overviews

The solutions in this section use the same architecture. Specifically, a two-stage recommender (see section 2.5.2). Briefly, a two-stage recommender is one that is composed of two modules. The first module can be a composition of recommenders that provides a curated and reduced list of user-item pairs to the second module. They do this by using past purchases, users, and item features. The second stage is a gradient-boosting decision tree model. This model receives the user-item pairs created by the first stage, i.e., candidates, and label them as a positive sample. This model is a classifier; it assigns a score to each item, and the higher the score the high probability to be of interest to the user if recommended.

From the analysis of these three solutions what we notice is that the gradient boosting algorithms seen in section 2.5.2 gives very good results for this type of task. All the solutions use *LightGBM* (see appendix A.8) as gradient boosting algorithm; this does not mean that it is overall better with respect to other libraries like *Catboost* (see appendix A.9) or *XGBoost* (see appendix A.7); the best algorithm to use depends on many factors e.g., the problem being solved, the size and structure of the data, the computational resources, the attributes of the data, the engineered features, the tuning of hyper-parameters model and the evaluation metric, among other factors [2]. Other than that, the accuracy of a model can greatly depend on the features used as input, e.g., *CatBoost* has been designed to handle categorical features more efficiently, i.e., it performs better on datasets with a large number of categorical variables compared to other algorithms.

We analyse these 3 solutions because they train the same *GBDT model*, but present different pipeline's structure and innovative techniques, e.g., *"Two Tower MMoE"*, *SWIM transformer* and *sentence transformer*, among others. Solutions that ranked at 2nd and 3rd place provide a similar pipeline and differ from the 1st because of recall methods and engineered features.[6]

[5] Public leaderboard of the competition here

[6] 2nd place solution here. 3rd place solution here.

3.3.1. 1st Place

The authors propose a two-stage recommender. The first stage generates a collection of 100 candidate articles for every user. It does this by learning the user's preferences from past purchases and applying some heuristics. The authors do not share the specific recall methods used. The second stage recommender receives the sequence of 100 articles and labels them as positive interactions. The other articles, instead, are labeled as negative interactions. [7] Than they attached generated features (see table 3.8) to transactions passed to the model. These features are called *interaction features* since they connect users and items using the transactions made by each user.

In order to tune the *GBDT* model and classify each article as a negative or positive sample, since the number of candidates for each user is low ($\sim$ 100) with respect to the total number of articles ($\sim$ 106.000), the authors apply the technique of the down-sampling (see section 2.6.4), a technique used in recommender systems to balance the data distribution between positive and negative samples.

Model The best single model they trained is a *GBDT*, i.e., *LightGBM*, which gave them a public score of **0.0367**. Then they also made an ensemble of classifiers composed of several models, i.e., 5 LightGBM and 7 Catboost, which gave them a score of **0.0371** on the private leaderboard.

3.3.2. 4th Place

The authors propose a two-stage recommender. The first stage generates a sequence of candidates using 4 recall methods: Item2item CF to spot the relationship between pair of items, top popular items (see section 2.1), repurchase considering the last 20 purchased products of each customer and *"Two Tower MMoE"* (see section 2.5.1), which is able to generate candidates of arbitrary length for all users. To assign importance score to items based on each user, considering recent active customers versus non-active customers the authors used a *gating network* (see section 2.5.1). The gating network learns to assign a weight to each item, indicating the importance of the item in the recommendation list. The weighted items are then passed to the next layer for final ranking and selection.

Model The best single model they present is a *GBDT*, specifically a *LightGBM*, which gives them a score of **0.0349** on the private leaderboard.[8]

[7] Outline of the 1st place winning solution here.
[8] 4th place solution here.

Table 3.8: Featured generated and used for the final ensemble by the authors of the 1st place solution.

Type	Description
Count	*User-item*, *user-category* of last week/month/season/same week of last year/entire dataset and importance of transaction weighted on the time distance from the test week
Time	First and last days of the transaction, for every single item
Mean/Max/Min	Aggregation based on *age*, *price* and *sales channel*
Difference/Ratio	Difference between age and mean age of who purchased items, the ratio of one user's purchased item count and the item's count.
Similarity	Collaborative filtering score of item2item, cosine similarity of item2item (**word2vec**), cosine similarity of user2item

3.3.3. 5th Place

The authors propose a two-stage recommender.[9] First, to spot similarities between users and items they use several types of embedding:

- For article images, they use *SWIM transformer* to extract the embedding features which is a type of recommendation system that utilizes a transformer neural network architecture to generate recommendations. It is designed to work in a weakly-supervised setting, meaning it only requires partial user-item interaction data e.g., implicit feedback, to generate recommendations, and is scalable to large datasets [72].
- For the article's description, they concatenated all text values of each article and then they used *sentence transformer* to extract the embedding which is a library that uses transformer-based models to embed sentences into a high-dimensional vector space [113]. These sentence embeddings can then be used to represent user-item interactions or item descriptions. The embedding can then be compared using

[9] 5th place solution here.

a distance metric, e.g., cosine similarity, to generate recommendations by finding the items with the most similar embedding to the target user or item. Sentence Transformer can also be fine-tuned on specific domains or languages to better fit the needs of a particular recommendation task [112].

In the second instance they apply multiple recall strategies to generate the list of candidate items to associate with each user e.g., customer's last bought items, *user based CF*, *item based CF*, pair of items bought together and popular items for each bucket. Each bucket is identified by age or gender.

Model The best single model they trained is a *GBDT*, i.e., *LightGBM Ranker* (see A.8), which gives them a score of **0.0350** on the private leaderboard.

4 | Experimental Methodology

This chapter describes the steps followed to obtain the final model. It starts analyzing the processing made on the dataset, i.e., hyper-parameter tuning, data sampling, dataset-splitting, and handling missing information. In the second instance it provides an overview of several experiments, i.e., experiment with strong baselines, i.e., collaborative filtering recommenders (see chapter 2), *heuristics* and, along with the list of chosen *candidate generation strategies*, *two-stage recommender* (see section 2.5.1). At the end of the chapter, we show the computational resources, libraries, and programming languages used to make our experiments and tune the final model. In the first set of experiments, i.e., *collaborative filter recommenders* the goal is to evaluate the accuracy of these models and establish their effectiveness as trained with our dataset. The experiments conducted with *heuristics* make use of the trends seen in section 3.2 to evaluate their effectiveness. The last set of experiments with the *two stages recommender* serves to exploit the effectiveness of a more complex model, i.e., *GBDT*, along with engineered features and selected candidate generation strategies.

4.1. Dataset Processing

Our dataset is large and complex with a wide range of attributes and features that need to be processed and cleaned before being used for training and evaluation. For that reason, processing has been a critical step in this thesis [42]. The result of these steps is the ICM (see section 2.2) which stores item-attribute values and URM (see section 2.3), which stores user-item interactions' value.

Handling missing information As we have seen in section 3.1.1 we have some missing information in both article and user dataset. As shown in section 2.6.3 there are different techniques to handle missing information. We applied the following processing to handle these missing values for the customer: *fashion news newsletter* has been filled with 0, *active* has been filled with 0, *fashion news frequency* has been filled with $NULL$ and *club member status* has been filled with $NULL$. Missing *description* for articles has been filled

with an empty string.

Sampling techniques As seen in section 2.6.4, sampling techniques are important during every experimentation phase, for each model or feature we build. Making use of these sampling techniques allowed us to ease the process of experimentation, lowering also the resources needed while testing the effectiveness of our features and/or models. We have used random sampling in order to create 4 different samples, containing different percentages of all the available transactions: 0.1%, 1%, 10%, and 100%.

4.2. Hyper-Parameter Tuning

Hyper-parameter tuning is the process of selecting the optimal hyper-parameters of a machine learning algorithm to maximize its accuracy on a specific task or dataset. Hyper-parameters are set before training a model and control its behavior during training, i.e., learning rate, regularization strength, and all the parameters required by the *GBDT algorithm* [12].

We performed *Bayesian Optimization* using Optuna [3]. Bayesian optimization algorithm uses the Gaussian process regression model as the surrogate model to approximate the objective function to balance exploration and exploitation in the search space. Optuna provides a flexible API for defining the search space, and setting up the objective function. It also supports distributed optimization, visualization of the results, and integration with several machine learning libraries such as PyTorch, TensorFlow, and Scikit-learn.[1] More specifically, we adopted the *step-wise algorithm* that tunes important hyper-parameters sequentially, resulting in a compact search space [92]. A search space refers to the set of all possible solutions that can be generated by an algorithm to solve a given problem [4]. A compact search space is a search space that has relatively fewer possible solutions compared to other search spaces that solve the same problem. [54].

4.3. Data Split

The fig. 4.1 shows how we split the dataset for all of our experiments. We consider only the last 7 weeks of the dataset. The first 6 are used to train and tune models; the reason is that these weeks share the same context, e.g., weather, products available, and fashion trends with the test week, i.e., the first week after the dataset period. The last week of the dataset is used to validate our model, because it is, considering the period and the

[1]Scikit-learn `https://scikit-learn.org/stable/`. PyTorch `https://pytorch.org`

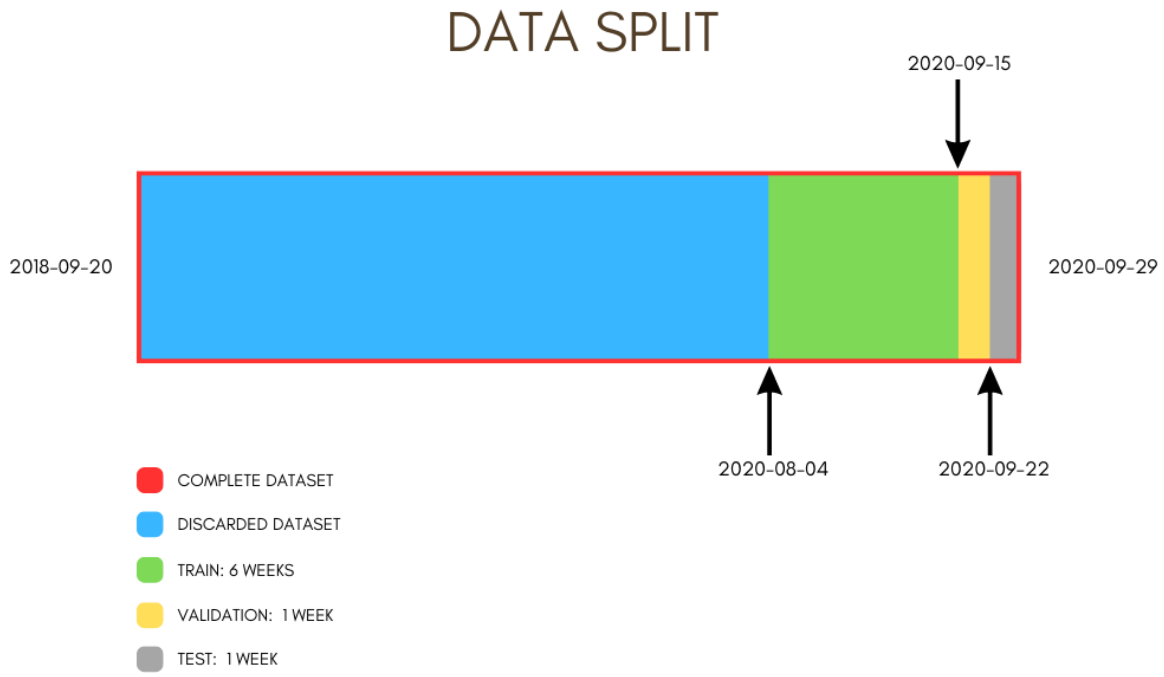

Figure 4.1: The red box represents the whole dataset which goes from $2018-09-20$ to $2020-09-29$. In blue is the discarded dataset. In green is the train split, in yellow is the validation split, and in gray is the test split. We use as training and validation data the last 7 weeks of the dataset since they are the most similar to the test week used for the evaluation.

selling trend, the most similar one to the test week used to evaluate the submission files of the challenge (see chapter 3). Since we have no access to this test week we validate our models using the validation week before submitting the recommendations to evaluate our solution.

4.4. Collaborative Filtering Recommenders

In our baseline experiments, we evaluate the accuracy of several collaborative filtering models, including user-based and item-based neighborhoods, matrix factorization, graph-based, top popular items, and hybrid models. We perform a variety of experiments to establish the effectiveness of these models, which in previous research works have shown

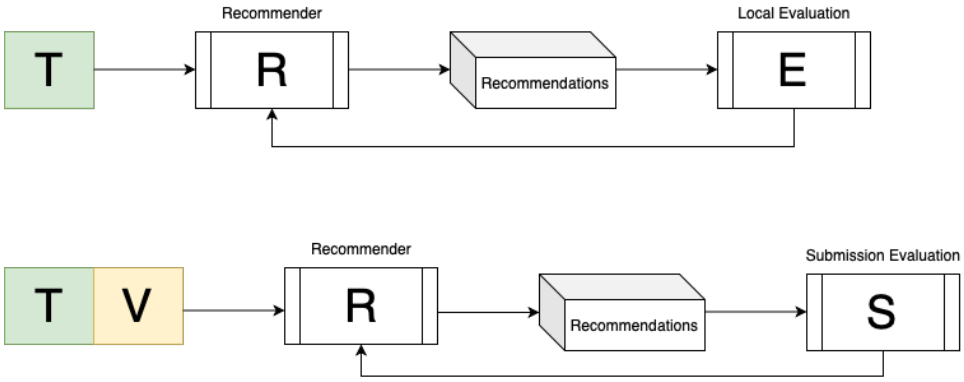

Figure 4.2: To build strong baseline recommenders we tune the recommender using the train data, generate recommendations and evaluate them locally. We iterate the phase of hyper-parameters tuning using the Optuna library. Once tuned the hyper-parameters we trained the model using both the train and validation data. Then we generate recommendations and submit them to be evaluated using MAP@12 (see eq. (2.41)) on test data for both private (95% of the test data) and public (5% of the test data) leaderboard.

to be competitive and strong baselines in terms of recommendations' quality [23]. The results of this experiment serve as a reference point for more sophisticated models that are evaluated later in the thesis [23, 49]. By establishing the recommendation's quality of strong baseline models we gain a better understanding of the challenge and opportunities to build effective recommender systems in the fashion domain [51]. This information is crucial to guide the development of more advanced models that can address these challenges and provide better recommendations to users. The techniques evaluated in this experiment cover a wide variety of recommenders, ranging from non-personalized (top-popular items in August 2020 and top-popular recommenders in September 2020), matrix factorization (PureSVD, ALS), graph-based (P3 Alpha, RP3 Beta), collaborative filtering (ItemKNN, User KNN). We also evaluate hybrid models (see section 2.4) making an ensemble of models, i.e., top popular items, P3Alpha and ItemKNN CF. Recommenders in this experiment are trained with the *train* split of the ICM and URM obtained in the processing phase. URM is a matrix that represents the interactions, which are transactions in our specific case, between users and items. Each row in the matrix represents a user, each column represents an item, and each entry in the matrix represents the user's transaction with the item. ICM is a matrix that represents the content attributes of each item. Each row in the matrix represents an item, each column represents a content attribute, and each entry in the matrix represents the presence or absence of the attribute for the item (see section 3.1.1).

4.5. Heuristics

With heuristics and association rules, we exploit the outcome of the data analysis made in section 3.2, e.g., seasonality, out-of-stock products, repurchase, co-occurrence, among others. This last heuristic represents the idea beyond outfits: users buy two or multiple items together because they compose an outfit recommended by the fashion company itself or because it has been seen somewhere else online. Several experiments are inspired by the concept of fast fashion [73]. We made around ~ 50 combination of heuristics and associations rules involving consideration of trendy color, new products, ages, and top popular items. Some of the experiments conducted and the trend combinations did are: (i) heuristic with trendy colors and top popular items. (ii) heuristic with age, trendy color, and top popular items. (iii) heuristic with top popular new items. (iv) heuristic recommend again product bought last 3 weeks. (v) heuristic on trending product based on repurchasing trend. (vi) association rule on items purchased together. (vii) association rule on top popular items based on age and discounted products. (viii) association rule on top popular items based on age. (ix) association rule on product and on top popular items from most popular product category, e.g., *trousers*, *sweater* and *cardigan*. Training different combinations of trends allows analyzing the variation of accuracy and highlights which heuristic the dataset is biased to.

4.6. Two Stage Recommender

During the last experimental phase, we apply the two-stage recommender (see section 2.5.2). These systems generate recommendations in two phases: first, multiple nominators select a small set of items from a large pool using cheap-to-compute item embeddings, i.e., candidate generation strategies used to down-sample the list of item (see section 2.6.4); then, a ranker with a richer set of features rearranges the nominated items and presents them to the user [40]. We generate candidates, using strategies listed in the next section, starting from the results of section 4.5. We then add features listed in section 4.6.2 to these candidates. For the model, instead, we make experiments with different *GBDT algorithms*. All the experiments use the same hyper-parameter tuning library presented in section 4.2 and the same data split of the section 4.3. One of the goals during that phase of experimentation has been to find heuristics that gave good results and use them to generate a pool of candidates to associate with every single user. This has been an important step because having a pool of ~ 300 candidates for each user means that the final model has to select the recommendations from a lower list of possible items instead of picking them from a pool of ~ 106.000 items for each user. Lowering the pool of can-

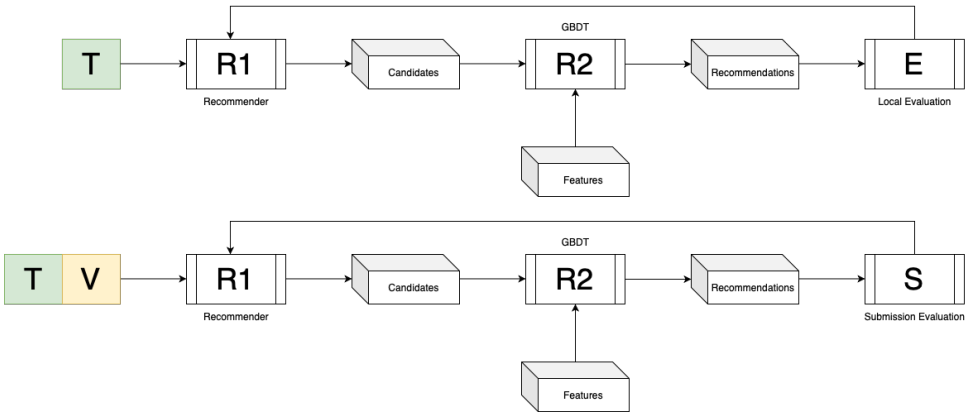

Figure 4.3: To build a two-stage recommender we first generate the candidates for each user using selected candidate generation strategies. These candidates are used, along with features, to train the *GBDT* model. Then we generate recommendations and evaluate them locally. We iterate the phase of hyper-parameters tuning using the Optuna library. Once tuned the hyper-parameters we train the model using both the train and validation data. Then we submit recommendations to be evaluated using MAP@12 (see eq. (2.41)) on test data for both private (95% of the test data) and public (5% of the test data) leaderboards.

didates from which the model needs to select items to recommend increases the model's accuracy and the effectiveness of the solutions. We tried different *GBDT algorithms*, e.g., *LightGBM Ranker*, and *Catboost* among the others.

4.6.1. Candidate Generation

The goal of candidate generation is to identify a set of items that are likely to be of interest to the user in order to lower the pool of articles from which the final model have to pick up recommendations for the user. Starting from the list of heuristics we presented in the section 4.5, we fine-tuned them and come up with a list of candidate generation methods, in order to generate a pool of $\sim$ 300 items for each user. **Once generated, those candidate pools have been used for all of our final experiments**.

Repurchase In the fast fashion industry, repurchase behavior is used as a candidate generation technique in recommender systems [73]. Fast fashion companies, H&M is an example, typically offer a wide range of clothing items that are frequently updated and replaced with new items, and customers may be more likely to repurchase items they have already bought, especially if they are happy with the fit, quality, and style of the item. Using repurchase behavior as a candidate generation technique helps the

recommender system identify items that are likely to be of interest to the user based on their previous purchase history. These items can then be recommended to the user as potential candidates for future purchases. The reason why we choose user repurchasing as a candidate generation strategy is that a considerable percentage of customers used to repurchase the same item again, as seen in section 3.2.3.

Item-to-item (or item-item) collaborative filtering Item-to-item collaborative filtering is a technique that uses the customer's purchase history to identify other items that are similar to the items they have purchased in the past (see section 2.3).

The system creates a user-item matrix where each row corresponds to a customer, and each column corresponds to an item. The cells in the matrix represent the customer's interaction with each item, in our case is there has been a transaction or not. The system then calculates the similarity between pairs of items using a similarity metric, e.g., cosine similarity. To generate recommendations, the system identifies the items that the user has already interacted with and selects the most similar items to these items in the matrix. These similar items are then recommended to the user as potential candidates for future purchases. In the fashion industry, item-to-item collaborative filtering is particularly effective because fashion items have many attributes that can be used to calculate similarities, such as color, style, brand, material, and seasonality[48]. By considering these attributes, the system can identify items that are similar in terms of their overall look and feel, as well as more specific features such as sleeve length, neckline, or hemline. For every single user, 36 items, based on collaborative-filtering techniques have been selected as possible candidates to recommend.

Top popular items This technique identifies the most popular items that have been purchased or viewed by many customers and recommends them to new customers or customers with limited purchase history. Popular items are items that have a high number of interactions, in our case transactions.

However, the popular item candidate generation technique has limitations, as it may not provide personalized recommendations that are tailored to the user's individual tastes and preferences. In addition, popular items may not be suitable for all users, as users have different needs, styles, and budgets. Anyway, as seen in section 4.4, non-personalized techniques give promising results, which is why for each user we associate 60 top popular items as possible candidates.

Age based One approach for age-based candidate generation is to use demographic information about the user, e.g., their age, to identify items that are popular among users of the same age. For example, if a user is in their twenties, the system may recommend popular items that are currently trending among users in their twenties too.

Another approach for age-based candidate generation is to use contextual information about the user, e.g., the user's purchase history, to identify items that are suitable for their age and style preferences. For example, if a user has previously purchased items that are popular among older adults, such as classic blazers or formal wear, the system may recommend similar items that are popular among that age group. However, age-based recommendations should be combined with other techniques, e.g., collaborative filtering and content-based filtering, to generate a diverse set of recommendations that are tailored to the user's individual preferences and needs. For each single age value, from 16 to 100, i.e., the range of user's age we have in the dataset, we extract the most popular items bought by users with that age. Then we associate this list of items to the users with that specific age. With this recall method, we select a list of 12 items.

Popularity per department We rank, for each single department name (see section 3.1.1), the list of all the articles based on the number of transactions associated. Then we generate a list of 6 candidates for each user, taking the most popular items from the department name from which the user does most of his transactions.

Same product code From section 3.1.1, the product code represents the first 6 digits of the *article id*. This means that items with different *article id* may have the same product code, i.e., same item but with different colors, patterns, or sizes. For that reason, it makes sense to extract some candidates starting from the transactions made by the user with articles with the same product code.

- Generate the item2item collaborative filtering, to extract similarities from items.
- Extract the product code from the transactions made by the user.
- Based on the transactions of the product code of the items bought in the past by the user and on the item2item similarities we then extract candidates.

Co-occurrence We select pairs of items that occur in the same transactions list for a specific user. We consider the last 32 weeks for this strategy and then we count, for each pair of items, how many times they are present in the list of transactions for a specific user. Then we filter pairs' count over a threshold, i.e., 150.

4.6.2. Feature Engineering

To train our models, fashion recommender systems require a set of features that describe the fashion items and the users' preferences. Feature engineering is the process of selecting and transforming the raw data related to fashion items and users' preferences into a set of meaningful features that can be used as input to the machine learning algorithm [124].

The process of feature engineering typically involves several steps, including data processing, feature selection, and feature transformation. Data processing involves cleaning and organizing the raw data, such as product descriptions, user preferences, and user-item interaction data, so that it can be used for feature engineering, as we have seen in section 4.1.

Feature selection involves choosing the most relevant features from the processed data to use in the machine learning algorithm. This step is important because including irrelevant or redundant features can lead to overfitting or a decrease in the model's accuracy [84]. Feature transformation involves transforming the selected features into a format that can be used as input to the machine learning algorithm. This might involve scaling the features to a common range, normalizing them to have zero mean and unit variance, or encoding categorical features as binary or numerical values. Once the features have been engineered, they can be used as input to the algorithm. The algorithm then learns patterns in the data to make predictions about which fashion items a user is likely to be interested in.

Overall, the goal of feature engineering in fashion recommender systems is to extract the most relevant and informative features from the available data, in order to enable accurate and personalized recommendations to users. The success of a fashion recommender system depends on the quality and relevance of the features that are used as input to the machine learning algorithm [70]. In our thesis work this step follows the generation of the candidate pool, since most of the features listed here, and associated to the user, are based on that pool, which is user specific. I create more than **100** features. Considering that we apply candidate generation strategies to lower the pool of items associated to each user, we used very simple features. That features give an high boost in the performance of the model. We filtered out not informative features for the model and we come up with the following list of features.

User attributes are taken from the user dataset and, for our final model, we consider only the user *age*.

Item attributes are taken from the article dataset (see 3.1.1). We use the following attributes of the dataset as input to the model: *product type number*, *product group name*, *graphical appearance*, *color group code*, *perceived color value*, *perceived color master*, *department number*, *index code*, *index group number*, *section number*, *garment group number*.

User features are generated starting from an aggregation of all the transactions, and calculating for every single user the *mean* and the *standard deviation* for both *prices* and *sales channel id*, i.e., if the user tend to buy in the physical or online store. This is to understand is the user is more inclined to buy articles in physical stores instead of online ones or vice versa.

Item features are *mean* and *standard deviation* for *prices* and *sales channel id* considering all transactions of that item.

Item-User features are the *mean* and *standard deviation* of the *age* of all the users who buy that item.

Item freshness features associate with each item the first day the item appears in a transaction.

Item volume features associate to each item its volume, i.e., the number of times the item appears inside the dataset.

User freshness features associate with each user the first day he made a transaction.

User volume features associate to each user his volume, i.e., the number of transactions made by the user.

User-Item freshness features associate to each user-item pair the first time the pair appears in a transaction.

User-Item volume features associate to each user-item pair its volume, i.e., the number of times the user-item pair appears inside the dataset.

Item age ranges features is the age range of users that most likely buy that item. We recommend most likely items with age ranges which include the age of the user we are recommending the item to.

The same list of features has been used for our final phase of experimentation, for all the models and gradient-boosting decision tree tested. The features are very simple since they include, e.g., *mean*, *sum* or *standard deviation* of some attributes, e.g., *price* and *age*, among others. This allows to create an effective and lightweight recommender while fastening the training time. This has been possible thanks to the extensive work done on dataset analysis and on building effective candidate generation strategies to down-sample the pool of items associated to each single user of the dataset.

4.7. Resources

To run our experiments we used two cloud computing platforms: Google Colab and Amazon AWS.[2] [3] Overall, this analysis required to fit and evaluate **500 models** requiring a total computational time of **120 hours**. **Google Colab** + is a cloud-based notebook environment that provides users with access to a virtual machine to run and develop code. With the **plus** subscription to *Google Colab* we had available a virtual machine with 1 or 2 cores CPU, an NVIDIA Tesla K80 or T4 GPU, and 27 GB of RAM. **Amazon Web Services** allows launching a virtual machine instance with the needed resources. For our final experimentation, we used the **m6g.16xlarge** instance which is a type of AWS instance that is powered by Graviton2 processors, 64 CPU cores, and 256 GB RAM.[4] As programming language we used *python* while to make our experiments we used different libraries, e.g., (i) pandas.[5] (ii) numpy.[6] (iii) pickle.[7] (iv) lightgbm.[8] (v) catboost.[9] (vi) scikit.[10] (vii) pytorch.[11]

[2]Google Colab `https://colab.research.google.com`
[3]Amazon AWS `https://aws.amazon.com`
[4]Amazon EC2 M6g Instances `https://aws.amazon.com/ec2/instance-types/m6/`
[5]Pandas `https://pandas.pydata.org`
[6]Numpy `https://numpy.org`
[7]Pickle `https://docs.python.org/3/library/pickle.html`
[8]LightGBM `https://lightgbm.readthedocs.io/en/v3.3.2/index.html`
[9]Catboost `https://catboost.ai`
[10]scikit `https://scikit-learn.org/stable/`
[11]PyTorch `https://pytorch.org`

5 | Results

In this chapter we analyze the results of our experiments, highlighting some of the callouts that guided us in the process of finding an effective and scalable model for fashion-based recommendations. Our most effective solution is described in the last section of the chapter. This recommender obtains the highest score on the leaderboard. We start testing collaborative filtering recommenders because several works show they are simple, effective, and strong baselines in most recommendation scenarios [23, 95, 109]. In a second experimental phase, we train models using heuristics, association rules, and combinations of them, to put in place users' behaviors that we have found during the data analysis, e.g., out of stock products, co-occurrence, seasonality, and repurchase (see section 3.2). Lastly, we build a scalable and lightweight two-stage recommender. This recommender is a *GBDT* model on top of a candidate generation model, features extracted from the dataset, and dataset attributes. The choice of a gradient boosting algorithm came from the study of Jannach, Dietmar et al. [49] in which analyses how the winning solutions of recent recommender system challenges mostly consist of substantial feature engineering efforts and the use of gradient boosting or ensemble techniques.

5.1. Collaborative Filtering Recommenders

In our strong baseline experiments, we evaluate the accuracy of several collaborative filtering models. The techniques evaluated in this experiment cover a wide variety of recommenders, ranging from non-personalized (top-popular items in August 2020 and top-popular recommenders in September 2020), matrix factorization (PureSVD, ALS), graph-based (P3 Alpha, RP3 Beta), collaborative filtering (ItemKNN, User KNN). We also evaluate hybrid models (see section 2.4) making an ensemble of models, i.e., top popular items, P3Alpha and ItemKNN CF.

We evaluate recommenders of several types because previous research works have shown they are competitive however, the best recommender type changes by domain [23, 95, 109].

Recommenders in this experiment are trained with the *train* split of the ICM (see sec-

Table 5.1: Accuracy of strong baseline recommenders (see section 4.4). This experiment was performed as a team during the competition after processing the dataset (see section 4.1). We include the results of the bests public solutions of the competition (see section 3.3), however, their methodology may differ with respect to ours. They are placed here as reference points. The top popular 12 items model considers the popularity of items for the whole dataset, while the top popular items in August/September consider only those months in the year 2020. In bold, we highlight the top-2 recommenders with the highest accuracy.

Recommender	Public Score	Private Score
First place	0.03716	0.03792
Fourth place	0.03544	0.03563
Fifth place	0.03536	0.03553
Top popular 12 items	**0.02163**	**0.02119**
Top popular items on September 2020	0.00407	0.00384
Top popular items August/September 2020	0.00383	0.00362
P3Alpha [115]	0.00431	0.00426
RP3 Beta [36]	0.00425	0.00453
ALS (see section 2.3.3)	**0.01413**	**0.01406**
PureSVD (see section 2.3.3)	0.00431	0.00426
Item KNN CF (section 2.3)	0.00345	0.00357
Ensemble: Top popular 12 items, P3Alpha and ItemKNN CF	0.00457	0.00463

tion 2.2) and URM (see section 2.3) obtained in the processing phase. URM is a matrix that represents the interactions, which are transactions in our specific case, between users and items. Each row in the matrix represents a user, each column represents an item, and each entry in the matrix represents the user's transaction with the item. ICM is a matrix that represents the content attributes of each item. Each row in the matrix represents an item, each column represents a content attribute, and each entry in the matrix represents the presence or absence of the attribute for the item (see section 3.1.1).

Table 5.1 shows the recommendation's accuracy, measured by MAP@12 (see eq. (2.41)), of the recommenders trained and evaluated in this experiment. The results are shown for both public and private leaderboards. The public leaderboard was available during the challenge period and is generated using only the 5% of the total test data. The private

leaderboard has been released after the deadline and is generated using the remaining 95% of the test data. The winners of the challenge have been selected considering only the private leaderboard.

The recommenders have low recommendation accuracy with respect to more sophisticated and tailored recommenders, e.g., the one used by the team obtaining 1st place in the competition.

In addition to this, the results contrast with the research of Ferrari Dacrema et al. [23]. In particular, strong baseline models in previous work do not obtain high accuracy in this domain, being less competitive than the best solutions of the challenge (see section 3.3). Also, a non-personalized recommender, i.e., *top popular items*, obtains higher accuracy than such baselines. ALS is a strong baseline [58] and obtains the highest accuracy of personalized collaborative filtering recommenders. However, its accuracy is lower than non-personalized approaches.

Regarding other recommenders Top Popular Items on August/September measure whether purchased items in the competition are most popular during the season. We considered only August and September because they are the two months in the same season as the target purchases. The obtained score is lower than the one obtained recommending the top popular items from the entire dataset. These results suggest that the trend of sold items during the test week is not connected with items sold during the same season, i.e., summer of 2020. An additional proof is that the accuracy of Top Popular Items on August/September 2020 is lower that the accuracy of Top Popular Items on September 2020 by 6%.

Regarding the ensemble recommenders, we perform evaluations of several ensembles containing combinations of recommenders. As Tsai and Hung [108] state, ensemble techniques have been shown to outperform many single collaborative filtering techniques in the literature. Despite previous success in competitions, an ensemble in this scenario does not translate to higher accuracy. In table 5.1 we show the most accurate ensemble: a combination of the top popular 12 items and strong baselines. This recommender has a low recommendation quality; similar to other strong baselines, meaning that the recommender does not get favored by the top popular 12 items recommender. These results suggest that ensembling these baselines does not translate to higher recommendation accuracy.

Due to non-personalized techniques like *top popular items* obtaining higher accuracy than more advanced and robust techniques like graph-based (RP3 Beta or P3Alpha), SVD, or collaborative filtering, then researchers and practitioners must develop different types of

recommenders to accurately model users' preference and to increase users' satisfaction.

5.2. Heuristics

With heuristics and association rules, we exploit the outcome of the data analysis of users' behaviors made in section 3.2, e.g., seasonality, out-of-stock products, repurchase, co-occurrence, among others. *Association rules* consider how often two items were sold to the same user among the two years of available dataset. During the recommendation phase, we recommended to the user items on the right side of the association rule in case the item on the left side was already present in the list of transactions for the specific user. This happens because in the fashion domains groups of items are usually sold together. The *co-occurrence* heuristic represents the idea beyond outfits and follows the same idea of the association rule: users buy two or multiple items together because they compose an outfit. Several recommenders are inspired by the concept of fast-fashion [73], i.e., recommender selecting only trendy colors, new products, popularity based on age, repurchases, and co-occurrences. We also combine two or more heuristics or association rules into a single recommender. This new recommender selects items that are selected by each heuristic or association rule. We evaluate these combinations:(i) heuristic with trendy colors and top popular items. (ii) heuristic with age, trendy color, and top popular items. (iii) heuristic with top popular new items. (iv) heuristic to recommend again product bought in the last 3 weeks. (v) heuristic on trending product based on repurchasing trend. (vi) association rule on items purchased together. (vii) association rule on top popular items based on age and discounted products. (viii) association rule on top popular items based on age. (ix) association rule on product and on top popular items from most popular product category, e.g., *trousers*, *sweater* and *cardigan*.

Table 5.2 shows the accuracy, measured by MAP@12 (see eq. (2.41)), of the recommenders trained and evaluated in this experiment. The best recommender in this experiment obtains a relative improvement of 60% with respect to the most accurate personalized collaborative filtering recommender of the previous section (see table 5.1). Moreover, all heuristics and association rules obtain higher accuracy than most collaborative filtering recommenders. 2 heuristics and 4 association rules are more accurate than ALS, the most accurate collaborative filtering recommender. With respect to the **recommend again product bought last 3 weeks** recommender, it is the second most accurate heuristic; with higher accuracy than all personalized collaborative filter recommenders. This recommender exploits the *fast fashion* trend seen in the analysis of users' behaviors (see section 3.2.3). In particular, the dataset shows that users tend to buy again the same

item. In some situations, they buy the exact same item, in others, they buy the item in different colors or sizes. In the trend of fast fashion, the price is low and some items remain inside the catalogue for years. Hence, users tend to buy the same item several times in a short period of time [73]. With respect to the **heuristic with top popular items and trendy color** recommender, the result is lower with respect to the top popular items algorithm. The recommender tells if there is a connection between the probability to sell a popular item and the trendy color of the summer season. From the score obtained, we figured out that there is not a strong connection with the trendy color. **Heuristic with age, trendy color, and top popular items** tells that even adding to the previous test the age, the results poorly improved. In any case since the result improved we considered age as an effective attribute to be used during the recommendation phase. **Association rule on items purchased together** is part of the concept of the fast fashion [73]. It gave better results with respect to the other experiments because we added two considerations: the number of times pairs of items are bought together and the fact that customers tend to buy again the same items but of different colors and sizes.

5.3. Two Stage Recommender

In this last experiment, we design, develop, and evaluate a two-stage recommender (see section 2.5.2). These systems generate recommendations in two phases: first, multiple nominators select a small set of items from the catalog using lightweight item embeddings, i.e., candidate generation strategies used to down-sample the list of items (see section 2.6.4). Second, a ranker with a richer set of features rearranges the nominated items and presents them to the user [40]. We generate candidates, using strategies listed in section 4.6.1, starting from the results of section 4.5, i.e., (i) repurchase. (ii) item2item collaborative filtering. (iii) top popular item. (iv) age-based. (v) popularity per department. (vi) same product code. (vii) co-occurrence. We than add features listed in section 4.6.2 to these candidates. For the ranker, instead, we evaluate two different *GBDT* implementations. Table 5.3 shows the best score obtained with *LightGBM* and *Catboost*. Find good candidate generation strategies has been an important step because having a pool of $\sim$ 300 candidates for each user means that the final model has to select the recommendations from a lower list of possible items instead of picking them from a pool of $\sim$ 106.000 items for each user. Lowering the pool of candidates from which the model needs to select items to recommend increases the model accuracy and the effectiveness of the proposed solution. We tried different *GBDT algorithms*: *LightGBM Ranker* and *Catboost Yetirank*.

The *GBDT* model that gives better results is **Catboost**; its accuracy is higher than all heuristics and association rules by 34 and 42 %. Also, its accuracy is higher than all collaborative filtering recommenders, with a difference of 41% with respect to the top popular 12 items and 114% with respect to ALS. Our most accurate recommender obtains a score of 0.0298 in the private leaderboard, meaning it obtains 10th place in it. For comparison, the best baseline sits in the 1445 place, the most accurate heuristic sits in the 1260 place, and the most accurate association rule sits in the 1311 place.

The model with the highest recommendation accuracy is a *Catboost* recommender. This agrees with the studies of Dorogush et al. [22] and Prokhorenkova et al. [91]; they argue that is expected that *Catboost* obtains higher accuracy with the same set of features and datasets than other GBDT implementations. However, this contrasts with the solutions analyzed in the section 3.3 as they use *LightGBM* as *GBDT algorithm*. We remark that those solutions do not describe the dataset used nor the computational resources used to train their respective models.

Among the features used to train our models, we use several categorical features: (i) *product type number*. (ii) *product group name*. (iii) *graphical appearance*. (iv) *color group code*. (v) *perceived colour value*. (vi) *perceived colour master*. (vii) *department number*. (viii) *index code*. (ix) *index group number*. (x) *section number*. (xi) *garment group number*. We generate more than **60 features** (see section 4.6.2). The feature importance gives another important result, i.e., the most important features used by *GBDT algorithms* to rank the pool of candidates and make final recommendations are (i) *user age*. (ii) *item-user mean age* that represents the mean ages of users who buy the item. (iii) *item ages ranges*. (iv) *color group code of the item*. (v) *garment group number*. (vi) *product type*. (vii) *item volumes*. (viii) *user-item volume*.. The combination of *user age* and *item-user mean age* allows the model to give high scores to items whose associated age is equal to the age of the user we are recommending the item to. The combination of *item volumes* and *user-item volume* shows instead that if the user bought that item in the past and that the item has an high sells rate, this means that the probability of the item to be repurchased is high. The remaining listed features, i.e., *product type*, *garment group number* and *color group code of the item* show, instead, which are the most important item's attribute considered by the model to rank items among the pool of candidates. The fact that the *product type* is one of the most important features is related to the high percentage of repurchase of items by the same user with the same product code, i.e., same item but of different color and size (see section 3.2.3).

Table 5.2: Results of experiments done with heuristics and association rules. With heuristics and association rules, we exploit the outcome of the data analysis made in section 3.2, e.g., seasonality, out-of-stock products, repurchase, co-occurrence, among others. All the experiments in the tables have been done using the split shoes in fig. 4.1. The scores are based on the MAP@12 (see eq. (2.41)) applied to the test week for both public and private leaderboards. In bold we highlight the best scores.

	Algorithm definition	**Public Score**	**Private Score**
	First place	0.03716	0.03792
	Forth place	0.03544	0.03563
	Fifth place	0.03536	0.03553
Heuristic	Trendy color and top popular items	0.00613	0.00642
	Age, trendy color and top popular items	0.0064	0.00676
	Top popular new items	0.0064	0.00676
	Recommend again product bought last 3 weeks	0.01854	0.0185
	Trending product based on repurchasing trend	**0.02263**	**0.02291**
Association rule	Top popular items based on age and discounted products	0.01478	0.01482
	Top popular items based on age	0.01949	0.01962
	Top popular items from most popular product category (*Trousers*, *Sweater*, *Cardigan*)	0.01973	0.01992
	Items purchased together	**0.02169**	**0.02159**

Table 5.3: Results of experiments done with the two-stage recommender. The *GBDT* model that gives better results is **Catboost**; as *loss function* we use **YetiRank** which is used in the CatBoost machine learning library to handle ranking problems [30]. From the results obtained there is one interesting call out to highlight, i.e., with respect to the solutions analyzed in the section 3.3 that used *LightGBM* as *GBDT algorithm*, the model that gives the highest accuracy on recommendation is a *Catboost* model. Among the features used to train our models, we use several categorical features. From the studies of Dorogush et al. [22] and Prokhorenkova et al. [91] this is an expected result, since *Catboost* with respect to other algorithms has higher efficiency and accuracy handling categorical data giving also better results. All the experiments in the tables have been done using the split of fig. 4.1. The scores are based on the MAP@12 (see eq. (2.41)) applied to the test week for both public and private leaderboards. In bold we highlight the best score.

Algorithm definition	**Public LB**	**Private LB**
First place	0.03716	0.03792
Fourth place	0.03544	0.03563
Fifth place	0.03536	0.03553
Two-Stage Recommender with LightGBM	0.0286	0.0279
Two-Stage Recommender with Catboost	**0.0303**	**0.0298**

6 | Conclusions and Future Developments

This thesis provides a comprehensive overview of the H&M challenge and presents several solutions to it, the last one ranking among the top 10 in the final leaderboard. The presented solution is a lightweight and scalable model that requires few resources and training time. As described in section 5.3, the recommender takes 15 hours to train on a 64-core CPU virtual machine using 256 GB RAM.

Overall, in this thesis, we analyze the dataset, including customers, articles, and transactions (see chapter 3), to identify and model users' behaviors and correlations between users, missing values, trends, and association rules. As seen in the results, this analysis yields positive results as recommenders based on these analyses obtain higher accuracy than strong state-of-the-art baselines in recommender systems.

In the chapter 4 we present the complete list of processing steps, hyper-parameter tuning, data split, heuristics, candidate generation strategies, and features used to make our experiments, highlighting used resources. In the result chapter 5, we list the outcomes of all the experiments. Despite the limited resources needed, our final models yielded competitive results. For future works, we suggest several action items that can be taken in the future to improve the score, e.g., increasing resources and testing models with longer time frames, i.e., to consider as train period more than 6 weeks. Another future direction is to build embedding of items' images and descriptions to boost the effectiveness of the model.

The fashion recommender system developed in this thesis has the potential to enhance the shopping experience for H&M customers by providing personalized and accurate fashion recommendations. Additionally, the system can help H&M to increase customer engagement and loyalty, as well as boost sales and revenue.

In the fashion area, it is recommended to retrain the model after some time, as fashion trends and user preferences can change over time [56, 90]. Not retraining recommenders can lead to a decrease in their accuracy in future interactions with users. In addition, new

products may be added to the inventory, which can affect the recommendations made by the model. Several studies have shown the importance of retraining recommender systems to maintain their accuracy over time. For example, in a study on the Netflix Prize dataset, Koren et al. [56] found that models need to be updated regularly to maintain their accuracy. They stated that "retraining is necessary to compensate for changes in the rating distribution and to incorporate new users and items". Similarly, in a study on personalized music recommendation systems, Preston and Erik [90] found that retraining the model regularly improved its effectiveness over time. They stated that "retraining a recommender system can improve its recommendations, especially when the user data changes over time". Many fashion companies are actively retraining their recommender systems to ensure they remain up-to-date and effective. For example, a European online fashion retailer, updates its recommendation algorithms every two weeks, based on customer feedback and new data. Similarly, a US-based online personal styling service updates its algorithms every few weeks to keep up with changes in customer preferences and fashion trends [27].

One limitation of this thesis is that the third-party system evaluating each solution does not simulate an environment when recommenders are retrained. As participants in the competition, we do not have access to the entire dataset, hence it is not possible to reproduce such an evaluation methodology. Even after the deadline it has been not possible to access the entire dataset and the only way to test our solutions has been to submit it on the system.

Bibliography

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005. doi: 10.1109/TKDE.2005.99. URL `https://doi.org/10.1109/TKDE.2005.99`.

[2] W. Ahmed, K. Muhammad, and F. I. Siddiqui. Predicting calorific value of thar lignite deposit: A comparison between back-propagation neural networks (bpnn), gradient boosting trees (gbt), and multiple linear regression (MLR). *Appl. Artif. Intell.*, 34(14):1124–1136, 2020. doi: 10.1080/08839514.2020.1824091. URL `https://doi.org/10.1080/08839514.2020.1824091`.

[3] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 2623–2631. ACM, 2019. doi: 10.1145/3292500.3330701. URL `https://doi.org/10.1145/3292500.3330701`.

[4] J. Aljabri, A. L. Michala, J. Singer, and I. Vourganas. mini-elsa: using machine learning to improve space efficiency in edge lightweight searchable attribute-based encryption for industry 4.0. *CoRR*, abs/2209.10896, 2022. doi: 10.48550/arXiv.2209.10896. URL `https://doi.org/10.48550/arXiv.2209.10896`.

[5] G. Behera and N. Nain. Handling data sparsity via item metadata embedding into deep collaborative recommender system. *J. King Saud Univ. Comput. Inf. Sci.*, 34(10 Part B):9953–9963, 2022. doi: 10.1016/j.jksuci.2021.12.021. URL `https://doi.org/10.1016/j.jksuci.2021.12.021`.

[6] F. Bella, F. Fumarola, and S. Pizzutilo. Co-training based ensemble for improving accuracy and coverage of recommender systems. In *Proceedings of the 5th ACM conference on Recommender systems*, pages 315–318. ACM, 2010.

[7] A. Bissacco, M. Yang, and S. Soatto. Fast human pose estimation using appearance

and motion via multi-dimensional boosting regression. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*. IEEE Computer Society, 2007. doi: 10.1109/CVPR.2007.383129. URL https://doi.org/10.1109/CVPR.2007.383129.

[8] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutierrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[9] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[10] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[11] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[12] L. Carminati, G. Lodigiani, P. Maldini, S. Meta, S. Metaj, A. Pisa, A. Sanvito, M. Surricchio, F. B. Pérez Maurera, C. Bernardis, and M. Ferrari Dacrema. Lightweight and scalable model for tweet engagements predictions in a resource-constrained environment. In *Proceedings of the Recommender Systems Challenge 2021*, RecSysChallenge '21, page 28–33, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386937. doi: 10.1145/3487572.3487597. URL https://doi.org/10.1145/3487572.3487597.

[13] B. Cestnik. Estimating probabilities: A crucial task in machine learning. In *Proceedings of the 9th European Conference on Artificial Intelligence*, ECAI'90, page 147–149, USA, 1990. Pitman Publishing, Inc.

[14] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

[15] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August*

13-17, 2016, pages 785–794. ACM, 2016. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

[16] S. Chien, P. Jain, W. Krichene, S. Rendle, S. Song, A. Thakurta, and L. Zhang. Private alternating least squares: Practical private matrix completion with tighter rates. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1877–1887. PMLR, 2021. URL http://proceedings.mlr.press/v139/chien21a.html.

[17] C. Choudhary, I. Singh, and M. Kumar. SARWAS: deep ensemble learning techniques for sentiment based recommendation system. *Expert Syst. Appl.*, 216:119420, 2023. doi: 10.1016/j.eswa.2022.119420. URL https://doi.org/10.1016/j.eswa.2022.119420.

[18] de Garis H., I. T., and H. T. Evolutionary robotics, artificial life, and the autonomy–intelligence–complexity trilemma. *Frontiers in Robotics and AI*, 3:18, 2016.

[19] Y. Deldjoo, F. Nazary, A. Ramisa, J. Mcauley, G. Pellegrini, A. Bellogin, and T. D. Noia. A review of modern fashion recommender systems, 2022.

[20] D. D. Denison, M. H. Hansen, C. C. Holmes, B. Mallick, and B. Yu, editors. *The Boosting Approach to Machine Learning: An Overview*, pages 149–171. Springer New York, New York, NY, 2003. ISBN 978-0-387-21579-2. doi: 10.1007/978-0-387-21579-2_9. URL https://doi.org/10.1007/978-0-387-21579-2_9.

[21] W. Dong, D. Wang, and Y. Liu. An overview of statistical methods for handling missing data. *Journal of Biopharmaceutical Statistics*, 30:430–447, 2020.

[22] A. V. Dorogush, V. Ershov, and A. Gulin. Catboost: gradient boosting with categorical features support. *CoRR*, abs/1810.11363, 2018. URL http://arxiv.org/abs/1810.11363.

[23] M. Ferrari Dacrema, S. Boglio, P. Cremonesi, and D. Jannach. A troubling analysis of reproducibility and progress in recommender systems research. *ACM Trans. Inf. Syst.*, 39(2), jan 2021. ISSN 1046-8188. doi: 10.1145/3434185. URL https://doi.org/10.1145/3434185.

[24] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997. doi: 10.1006/jcss.1997.1504. URL https://doi.org/10.1006/jcss.1997.1504.

[25] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[26] S. Funk. Netflix update: Try this at home. *The ACME Journal*, 4(12):1–3, 2006.

[27] C. Giri and Y. Chen. Deep learning for demand forecasting in the fashion and apparel retail industry. *Forecasting*, 4(2):565–581, 2022. ISSN 2571-9394. doi: 10.3390/forecast4020031. URL `https://www.mdpi.com/2571-9394/4/2/31`.

[28] I. J. Goodfellow, Y. Bengio, and A. C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 978-0-262-03561-3. URL `http://www.deeplearningbook.org/`.

[29] J. W. Graham, A. E. Olchowski, and T. D. Gilreath. How many imputations are really needed? some practical clarifications of multiple imputation theory. *Prevention Science*, 8:206–213, 2007.

[30] A. Gulin, I. Kuralenok, and D. Pavlov. Winning the transfer learning track of yahoo!'s learning to rank challenge with yetirank. In O. Chapelle, Y. Chang, and T. Liu, editors, *Proceedings of the Yahoo! Learning to Rank Challenge, held at ICML 2010, Haifa, Israel, June 25, 2010*, volume 14 of *JMLR Proceedings*, pages 63–76. JMLR.org, 2011. URL `http://proceedings.mlr.press/v14/gulin11a.html`.

[31] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10):993–1001, 1990. doi: 10.1109/34.58871. URL `https://doi.org/10.1109/34.58871`.

[32] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[33] M. He, M. Thottethodi, and T. N. Vijaykumar. Booster: An accelerator for gradient boosting decision trees training and inference. In *2022 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022, Lyon, France, May 30 - June 3, 2022*, pages 1051–1062. IEEE, 2022. doi: 10.1109/IPDPS53621.2022.00106. URL `https://doi.org/10.1109/IPDPS53621.2022.00106`.

[34] X. He, L. Liao, H. Zhang, L. Nie, and X. Hu. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558. ACM, 2016. doi: 10.1145/2911451.2911514.

[35] X. He, L. Liao, H. Zhang, L. Nie, and X. Hu. Nais: Neural attentive item similarity

model for recommendation. In *Proceedings of the 2018 Conference on Recommender Systems*, pages 35–43. ACM, 2018.

[36] X. He, L. Liao, H. Zhang, L. Nie, and X. Hu. Outer product-based neural collaborative filtering. *IEEE Transactions on Knowledge and Data Engineering*, 30(12): 2259–2272, 2018.

[37] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[38] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

[39] T. Horváth, R. G. Mantovani, and A. C. P. L. F. de Carvalho. Hyper-parameter initialization of classification algorithms using dynamic time warping: A perspective on PCA meta-features. *Appl. Soft Comput.*, 134:109969, 2023. doi: 10.1016/j.asoc.2022.109969. URL `https://doi.org/10.1016/j.asoc.2022.109969`.

[40] J. Hron, K. Krauth, M. I. Jordan, and N. Kilbertus. Exploration in two-stage recommender systems. *CoRR*, abs/2009.08956, 2020. URL `https://arxiv.org/abs/2009.08956`.

[41] J. Hron, K. Krauth, M. I. Jordan, and N. Kilbertus. On component interactions in two-stage recommender systems. *CoRR*, abs/2106.14979, 2021. URL `https://arxiv.org/abs/2106.14979`.

[42] Y.-J. Hsu, Y.-W. Liu, Y.-H. Yeh, and H.-H. Lin. Fashion recommendation with missing data using autoencoder. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 616–622. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1066.

[43] X. Hu, W. Zhu, and Q. Li. HCRS: A hybrid clothes recommender system based on user ratings and product features. *CoRR*, abs/1411.6754, 2014. URL `http://arxiv.org/abs/1411.6754`.

[44] X. Hu, X. He, L. Liao, H. Zhang, and L. Nie. Sampling-based matrix factorization for recommender systems with implicit feedback. *ACM Transactions on Information Systems (TOIS)*, 36(4):1–29, 2018. doi: 10.1145/3233700.

[45] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback

datasets. In *Proceedings of the Eighth IEEE International Conference on Data Mining*, pages 263–272. IEEE, 2008.

[46] Y. Hu, X. Chen, and Y. Zhang. A hybrid collaborative filtering model with multi-information sources for fashion recommender systems. *IEEE Access*, 8:212266–212277, 2020.

[47] K. Huang, Y. Du, L. Li, J. Shen, and G. Sun. Pairwise-based hierarchical gating networks for sequential recommendation. In G. Li, H. T. Shen, Y. Yuan, X. Wang, H. Liu, and X. Zhao, editors, *Knowledge Science, Engineering and Management - 13th International Conference, KSEM 2020, Hangzhou, China, August 28-30, 2020, Proceedings, Part II*, volume 12275 of *Lecture Notes in Computer Science*, pages 64–75. Springer, 2020. doi: 10.1007/978-3-030-55393-7_6. URL `https://doi.org/10.1007/978-3-030-55393-7_6`.

[48] T. Ito, I. Nakamura, S. Tanaka, T. Sakai, T. Kato, Y. Fukazawa, and T. Yoshimura. Deep neural network incorporating CNN and MF for item-based fashion recommendation. In H. Uehara, T. Yamaguchi, and Q. Bai, editors, *Knowledge Management and Acquisition for Intelligent Systems - 17th Pacific Rim Knowledge Acquisition Workshop, PKAW 2020, Yokohama, Japan, January 7-8, 2021, Proceedings*, volume 12280 of *Lecture Notes in Computer Science*, pages 46–57. Springer, 2020. doi: 10.1007/978-3-030-69886-7_4. URL `https://doi.org/10.1007/978-3-030-69886-7_4`.

[49] Jannach, Dietmar, G. de Souza P. Moreira, and E. Oldridge. Why are deep learning models not consistently winning recommender systems competitions yet? a position paper. In *Proceedings of the Recommender Systems Challenge 2020*, RecSysChallenge '20, page 44–49, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450388351. doi: 10.1145/3415959.3416001. URL `https://doi.org/10.1145/3415959.3416001`.

[50] S. Kabbur, X. Ning, and G. Karypis. Fism: Factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 659–667, 2013.

[51] B. Kille and A. Lommatzsch. Defining a meaningful baseline for news recommender systems. In Ö. Özgöbek, B. Kille, J. A. Gulla, and A. Lommatzsch, editors, *Proceedings of the 7th International Workshop on News Recommendation and Analytics in conjunction with 13th ACM Conference on Recommender Systems, INRA@RecSys 2019, Copenhagen, Denmark, September 20, 2019*, volume

2554 of *CEUR Workshop Proceedings*, pages 24–28. CEUR-WS.org, 2019. URL https://ceur-ws.org/Vol-2554/paper_04.pdf.

[52] D. Kim, S. Park, and S. Lee. Handling missing data in recommender systems using feature-aware matrix factorization. *Information Sciences*, 483:170–183, 2019.

[53] H. Kim, S. Cho, H. Park, J. Hong, and E. Choi. Handling missing data in machine learning-based fashion recommendation systems. *Journal of Fashion Marketing and Management*, 22(1):39–52, 2018. doi: 10.1108/JFMM-05-2017-0048.

[54] C. Koller, G. Kauermann, and X. X. Zhu. Going beyond one-hot encoding in classification: Can human uncertainty improve model performance? *CoRR*, abs/2205.15265, 2022. doi: 10.48550/arXiv.2205.15265. URL https://doi.org/10.48550/arXiv.2205.15265.

[55] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. *ACM SIGKDD Explorations Newsletter*, 10(2):4–10, 2008.

[56] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. doi: 10.1109/MC.2009.263. URL https://doi.org/10.1109/MC.2009.263.

[57] A. Kristjánsdóttir. Moving from fashions to a continuous stream of change: teacher development and IT. In D. Tinsley and D. C. Johnson, editors, *Information and Communications Technologies in School Mathematics, IFIP TC3/WG3.1 Working Conference on Secondary School Mathematics in the World of Communication Technology: Learning, Teching, and the Curriculum, 26-31 October 1997, Grenoble, France*, volume 119 of *IFIP Conference Proceedings*, pages 165–168. Chapman & Hall, 1997.

[58] R. R. S. R. Kumar, G. A. Rao, and S. Anuradha. Efficient distributed matrix factorization alternating least squares (EDMFALS) for recommendation systems using spark. *J. Inf. Knowl. Manag.*, 21(1):2250012:1–2250012:16, 2022. doi: 10.1142/S0219649222500125. URL https://doi.org/10.1142/S0219649222500125.

[59] M. Kunaver and M. Gregoric. A survey of recommender systems: from traditional to hybrid. *Journal of Information and Organizational Sciences*, 41(1), 2017.

[60] K. Lan and Y. Liu. A spatial-statistics based framework of spatial correlation analysis of retail sales. In *International Conference on Computers, Information Processing and Advanced Education, CIPAE 2022, Ottawa, ON, Canada, August*

26-28, 2022, pages 426–431. IEEE, 2022. doi: 10.1109/CIPAE55637.2022.00095. URL https://doi.org/10.1109/CIPAE55637.2022.00095.

[61] N. Landia. Building recommender systems for fashion: Industry talk abstract. In P. Cremonesi, F. Ricci, S. Berkovsky, and A. Tuzhilin, editors, *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017, Como, Italy, August 27-31, 2017*, page 343. ACM, 2017. doi: 10.1145/3109859.3109929. URL https://doi.org/10.1145/3109859.3109929.

[62] S. Lee, S. Lee, Y. Park, and K.-W. Lee. Fashion item recommendation using deep learning with missing data. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4640–4643. IEEE, 2019. doi: 10.1109/BigData47090.2019.9006283.

[63] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014. ISBN 978-1107077232. URL http://www.mmds.org/.

[64] Y. Lin. Breaking the softmax bottleneck for sequential recommender systems with dropout and decoupling. *CoRR*, abs/2110.05409, 2021. URL https://arxiv.org/abs/2110.05409.

[65] R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. Wiley, 2nd edition, 2019.

[66] J. Liu and X. Liu. Unbiased importance sampling-based collaborative filtering. *Neurocomputing*, 168:516–527, 2015.

[67] Y. Liu and L. Li. A fashion outfit recommender system based on multi-view and multi-modal deep learning. *Journal of Intelligent Information Systems*, 56(3):419–434, 2021.

[68] Y. Liu, Y. Wang, Y. Li, and G. Wu. Earthquake prediction by RBF neural network ensemble. In F. Yin, J. Wang, and C. Guo, editors, *Advances in Neural Networks - ISNN 2004, International Symposium on Neural Networks, Dalian, China, August 19-21, 2004, Proceedings, Part II*, volume 3174 of *Lecture Notes in Computer Science*, pages 962–969. Springer, 2004. doi: 10.1007/978-3-540-28648-6_153. URL https://doi.org/10.1007/978-3-540-28648-6_153.

[69] Y. Liu, Y. Li, and J. Wang. Application of association rules in recommender systems: A survey. In *International Workshop on Data Mining for Business Applications*, pages 219–228. Springer, 2007.

[70] Y. Liu, W. Chen, and W. Chen. Semi-supervised feature engineering for sentiment analysis. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2018.

[71] Y. Liu, W. Li, Y. Li, Y. Liu, and L. Li. Fashion outfit recommendation based on multi-source data with convolutional neural network. *Information Sciences*, 518: 19–33, 2020.

[72] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 9992–10002. IEEE, 2021. doi: 10.1109/ICCV48922.2021.00986. URL `https://doi.org/10.1109/ICCV48922.2021.00986`.

[73] X. Long and J. Nasiry. Sustainability in the fast fashion industry. *Manuf. Serv. Oper. Manag.*, 24(3):1276–1293, 2022. doi: 10.1287/msom.2021.1054. URL `https://doi.org/10.1287/msom.2021.1054`.

[74] P. Lops, M. De Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. *In Recommender systems handbook*, pages 73–105, 2011.

[75] H. Lu, L. Li, A. Swami, and J.-T. Chen. Top-n recommendation via matrix completion. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 721–732. SIAM, 2012.

[76] C. Ma, P. Kang, and X. Liu. Hierarchical gating networks for sequential recommendation. In A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 825–833. ACM, 2019. doi: 10.1145/3292500.3330984. URL `https://doi.org/10.1145/3292500.3330984`.

[77] H. Ma, D. Zhou, C. Liu, and M. R. Lyu. Sorec: Social recommendation using probabilistic matrix factorization. *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 931–940, 2008.

[78] J. Ma, Z. Zhao, X. Yi, J. Chen, L. Hong, and E. H. Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In Y. Guo and F. Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 1930–1939. ACM, 2018. doi: 10.1145/3219819.3220007. URL `https://doi.org/10.1145/3219819.3220007`.

[79] J. Ma, Z. Zhao, X. Yi, J. Yang, M. Chen, J. Tang, L. Hong, and E. H. Chi. Off-policy learning in two-stage recommender systems. In Y. Huang, I. King, T. Liu, and M. van Steen, editors, *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 463–473. ACM / IW3C2, 2020. doi: 10.1145/3366423.3380130. URL https://doi.org/10.1145/3366423.3380130.

[80] Q. Ma, R. Chen, J. Tang, and Y. Ouyang. Stratified sampling for implicit feedback and matrix factorization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 987–996. ACM, 2018. doi: 10.1145/3269206.3271715.

[81] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[82] S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, pages 1097–1101. ACM, 2006.

[83] H. Mohanty, S. Champati, B. L. P. Barik, and A. Panda. Cluster quality analysis based on svd, pca-based k-means and NMF techniques: an online survey data. *Int. J. Reason. based Intell. Syst.*, 15(1):86–96, 2023. doi: 10.1504/IJRIS.2023.10050583. URL https://doi.org/10.1504/IJRIS.2023.10050583.

[84] D. Nguyen, N. Nguyen, D. Le, and B. Nguyen. A survey on feature engineering for machine learning. *Expert Systems with Applications*, 115:491–521, 2019.

[85] M. Nishi and H. Hayashi. Analyzing the correlation between tweets and sales for product brands. In T. Matsuo, K. Takamatsu, Y. Ono, and S. Hirokawa, editors, *9th International Congress on Advanced Applied Informatics, IIAI-AAI 2020, Kitakyushu, Japan, September 1-15, 2020*, pages 481–486. IEEE, 2020. doi: 10.1109/IIAI-AAI50415.2020.00102. URL https://doi.org/10.1109/IIAI-AAI50415.2020.00102.

[86] L. Paleti, P. R. Krishna, and J. V. R. Murthy. Approaching the cold-start problem using community detection based alternating least square factorization in recommendation systems. *Evol. Intell.*, 14(2):835–849, 2021. doi: 10.1007/s12065-020-00464-y. URL https://doi.org/10.1007/s12065-020-00464-y.

[87] R. Pan and L. Chen. A survey of collaborative filtering for recommender systems based on bayesian personalized ranking. *IEEE Transactions on Knowledge and Data Engineering*, 30(4):601–615, 2018. doi: 10.1109/TKDE.2018.2795081.

[88] M. J. Pazzani and D. Billsus. Content-based recommendation systems. *The adaptive web*, pages 325–341, 2007.

[89] D. M. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[90] M. R. Preston and B. Erik. Hierarchical bayesian modeling of consumer choice with limited information. *Marketing Science*, 31(2):321–340, 2012.

[91] L. O. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6639–6649, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/14491b756b3a51daac41c24863285549-Abstract.html`.

[92] J. Qu, N. Hiruta, K. Terai, H. Nosato, M. Murakawa, and H. Sakanashi. Enhanced deep learning for pathology image classification: A knowledge transfer based stepwise fine-tuning scheme. In A. Tomczyk, A. L. N. Fred, and H. Gamboa, editors, *Proceedings of the 12th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2019) - Volume 2: BIOIMAGING, Prague, Czech Republic, February 22-24, 2019*, pages 92–99. SciTePress, 2019. doi: 10.5220/0007356100920099. URL `https://doi.org/10.5220/0007356100920099`.

[93] M. Rac, M. Kompan, and M. Bieliková. Preference dynamics and behavioral traits in fashion domain. In *14th International Workshop on Semantic and Social Media Adaptation and Personalization, SMAP 2019, Larnaca, Cyprus, June 9-10, 2019*, pages 1–5. IEEE, 2019. doi: 10.1109/SMAP.2019.8864802. URL `https://doi.org/10.1109/SMAP.2019.8864802`.

[94] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461. AUAI Press, 2009. doi: 10.5555/1756006.1756067. URL `https://dl.acm.org/doi/10.5555/1756006.1756067`.

[95] S. Rendle, W. Krichene, L. Zhang, and Y. Koren. Revisiting the performance of ials on item recommendation benchmarks. In J. Golbeck, F. M. Harper, V. Murdock, M. D. Ekstrand, B. Shapira, J. Basilico, K. T. Lundgaard, and E. Oldridge, editors,

RecSys '22: Sixteenth ACM Conference on Recommender Systems, Seattle, WA, USA, September 18 - 23, 2022, pages 427–435. ACM, 2022. doi: 10.1145/3523227.3548486. URL `https://doi.org/10.1145/3523227.3548486`.

[96] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[97] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor. An introduction to recommender systems. In *Recommender Systems Handbook*, pages 1–35. Springer, 2011.

[98] D. B. Rubin. Inference and missing data. *Biometrika*, 63:581–592, 1976.

[99] D. B. Rubin. Multiple imputation for nonresponse in surveys. *Wiley Series in Probability and Mathematical Statistics*, 1987.

[100] C. S., H. S., Z. W., and W. X. Ensemble methods for time series forecasting: A survey. *Information Fusion*, 66:86–109, 2021. doi: https://doi.org/10.48550/arXiv.2104.11475.

[101] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, pages 285–295. ACM, 2001.

[102] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[103] J. L. Schafer. Multiple imputation: a primer. *Statistical Methods in Medical Research*, 7:3–15, 1998.

[104] L. Shao, S. Li, and S. Liu. A unified framework for missing data imputation in collaborative filtering. *Information Fusion*, 76:213–224, 2022.

[105] U. Shardanand and P. Maes. Social information filtering: algorithms for automating word of mouth. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217. ACM, 1995.

[106] X. Shen, H. Liu, W. Wang, and M. Jiang. Fashion recommender system based on improved filling and collaborative filtering. *IEEE Access*, 9:12090–12101, 2021.

[107] D. Shiung and W. Chin. Designing high-performance green filters using downsampling techniques. In *2019 International Symposium on Intelligent Signal Processing and Communication Systems, ISPACS 2019, Taipei, Taiwan, December 3-*

6, 2019, pages 1–2. IEEE, 2019. doi: 10.1109/ISPACS48206.2019.8986383. URL https://doi.org/10.1109/ISPACS48206.2019.8986383.

[108] C.-F. Tsai and C. Hung. Cluster ensembles in collaborative filtering recommendation. *Applied Soft Computing*, 12(4):1417–1425, 2012. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2011.11.016. URL https://www.sciencedirect.com/science/article/pii/S1568494611004583.

[109] R. Verachtert, J. Craps, L. Michiels, and B. Goethals. The impact of a popularity punishing hyperparameter on itemknn recommendation performance. In J. Kamps, L. Goeuriot, F. Crestani, M. Maistro, H. Joho, B. Davis, C. Gurrin, U. Kruschwitz, and A. Caputo, editors, *Advances in Information Retrieval - 45th European Conference on Information Retrieval, ECIR 2023, Dublin, Ireland, April 2-6, 2023, Proceedings, Part II*, volume 13981 of *Lecture Notes in Computer Science*, pages 646–654. Springer, 2023. doi: 10.1007/978-3-031-28238-6_56. URL https://doi.org/10.1007/978-3-031-28238-6_56.

[110] L. Wang and X. Ma. A hybrid recommendation algorithm for fashion e-commerce based on deep learning and collaborative filtering. *Journal of Computational Science*, 57:101492, 2022.

[111] R. Wang, R. Shivanna, D. Z. Cheng, S. Jain, D. Lin, L. Hong, and E. H. Chi. DCN V2: improved deep & cross network and practical lessons for web-scale learning to rank systems. In J. Leskovec, M. Grobelnik, M. Najork, J. Tang, and L. Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 1785–1797. ACM / IW3C2, 2021. doi: 10.1145/3442381.3450078. URL https://doi.org/10.1145/3442381.3450078.

[112] W. Wang, G. Chen, H. Wang, Y. Han, and Y. Chen. Multilingual sentence transformer as A multilingual word aligner. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 2952–2963. Association for Computational Linguistics, 2022. URL https://aclanthology.org/2022.findings-emnlp.215.

[113] W. Wang, G. Chen, H. Wang, Y. Han, and Y. Chen. Multilingual sentence transformer as A multilingual word aligner. *CoRR*, abs/2301.12140, 2023. doi: 10.48550/arXiv.2301.12140. URL https://doi.org/10.48550/arXiv.2301.12140.

[114] J. Wu, X. Wang, X. Gao, J. Chen, H. Fu, T. Qiu, and X. He. On the effectiveness

of sampled softmax loss for item recommendation. *CoRR*, abs/2201.02327, 2022. URL `https://arxiv.org/abs/2201.02327`.

[115] S. Wu, H. Li, Z. Zhang, Z. Wang, and Y. Chen. P3alpha: Large-scale nonlinear personalized ranking with autoencoder for content-based retrieval. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 165–174. ACM, 2019.

[116] Z. Yang, X. Peng, and J. Li. Outfit recommendation via attentional graph convolutional network. *IEEE Access*, 9:51851–51862, 2021.

[117] K. Yehuda, B. Robert, and V. Chris. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456. ACM, 2010.

[118] Z. Yuchen, S. Damien, and L. Hongyan. SLIM: Sparse linear methods for top-n recommender systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 97–104, 2013. doi: 10.1145/2507157.2507163.

[119] K. Zhang, Z. Wang, J. Liang, and X. Zhao. A bayesian matrix factorization model for dynamic user embedding in recommender system. *Frontiers Comput. Sci.*, 16(5): 165346, 2022. doi: 10.1007/s11704-022-1213-7. URL `https://doi.org/10.1007/s11704-022-1213-7`.

[120] W. Zhang, S. Li, and S. Liu. Combining implicit and explicit feedback for fashion outfit recommendation. *Neurocomputing*, 484:123–132, 2022.

[121] X. Zhang, L. Wang, and J. Qin. A new regression imputation method for missing data in categorical variables. *Computational Statistics and Data Analysis*, 101:1–12, 2016.

[122] Y. Zhang, F. Feng, C. Wang, X. He, M. Wang, Y. Li, and Y. Zhang. How to retrain recommender system? A sequential meta-learning method. *CoRR*, abs/2005.13258, 2020. URL `https://arxiv.org/abs/2005.13258`.

[123] Y. Zhao, S. Chen, Y. Lin, and J. Han. Handling missing information in fashion outfit recommendation via auxiliary feature-based collaborative filtering. *Neurocomputing*, 479:66–77, 2022.

[124] A. Zheng and A. Casari. *Feature engineering for machine learning: Principles and techniques*. O'Reilly Media, Sebastopol, CA, 2018.

[125] Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall/CRC, 1st edition, 2012. ISBN 1439830037.

A | Recommendation Techniques

A.1. Funk SVD

```
Set Nk to 0; X and Y are empty matrices (please remember that at any time
     X isNu xNk andYisNk *Ni)
Loop:
    Increment Nk
    Add a column to the first matrix, filled with random values Add a row
        to the second matrix, filled with random values Apply ALS for the
        current value of Nk
Until the process converges (falling below a certain error threshold) or
     we reach the desired Nk
```

A.2. ALS

```
The two matrices (X and Y) are initialized at random, namely, they are
     filled with random values
Loop:
    We fix the newest matrix X, and we learn matrix Y (optimizing its loss
         function) and store it
    We fix the newly obtained matrix Y, and we learn matrix X (optimizing
        its loss function) and store it
    Until the process converges (X and Y do not vary too much, namely
        their variation is comprised in an error threshold)
#immagini fix and learn
```

A.3. List combination

```
Order the set of lists from the most performant to the least performant
```

```
    one. Loop:
   For each list in the ordered set of lists:
        Pick the first untaken item
        If it is not present in the final list yet:
             Add it to the final list
        Remove it from the original list
Until the final list has N items
```

A.4. Rel(k) examples

Example 1:
If **gt=[a,b,c,d,e]** and **pred=[b,c,a,d,e]** then for **rel@1** we only take the first recommendation from **pred**, i.e., **b** and check if it's relevant, i.e., present in **gt**. A.1

$$rel(k) = 1.0 \tag{A.1}$$

Example 2:
If **gt=[a,b,c,d,e]** and **pred=[f,b,c,d,e]** then for **rel@1** we only take the first recommendation from **pred**, i.e., **f** and check if it's relevant, i.e., present in **gt**. A.2

$$rel(k) = 0.0 \tag{A.2}$$

Example 3:
If **gt=[a,b,c,d,e]** and **pred=[a,f,e,g,b]** then for **rel@2** we only take the second recommendation from **pred**, i.e., **f** and check if it's relevant, i.e., present in **gt**. A.3

$$rel(k) = 0.0 \tag{A.3}$$

A.5. Precision@k examples

Example 1:
If **gt=[a,b,c,d,e]** and **pred=[b,c,a,d,e]** then for **P@1** we only take the first recommendation from **pred**, i.e., **b** and find it's **precision** with the **gt**. A.4

$$P = \frac{|\{gt\} \cap \{pred[:1]\}|}{|\{pred[:1]\}|} = \frac{1}{1} \tag{A.4}$$

Example 2:
If **gt=[a,b,c,d,e]** and **pred=[f,b,c,d,e]** then for **P@1** we only take the first recommendation from **pred**, i.e., **f** and find it's **precision** with the **gt**. A.5

$$P = \frac{|\{gt\} \cap \{pred[:1]\}|}{|\{pred[:1]\}|} = \frac{0}{1} \tag{A.5}$$

Example 3:
If **gt=[a,b,c,d,e]** and **pred=[a,f,e,g,b]** then for **P@2** we only take the first recommendation from **pred**, i.e., **[a,f]** and find it's **precision** with the **gt**. A.6

$$P = \frac{|\{gt\} \cap \{pred[:1]\}|}{|\{pred[:1]\}|} = \frac{1}{2} \tag{A.6}$$

A.6. Two Tower MMoE

This section provides a graphical overview of the architecture of the *Two Tower Multi-Modal Multi-Task Output Embedding*. The fig. A.1 shows a trainable neural network used to get the essence of all the information of the item that is relevant to recommendation. The fig. A.2 shows a trainable neural network used to take all the information about the user and make a fixed-size vector from it. In the end, as fig. A.3 shows, the two embeddings are meshed together using a *sampled softmax* (see section 2.5.1). If the value is high then it means that the item is a good match for the user.

A.7. XGBoost: A Scalable Tree Boosting System

XGBoost is an open-source machine learning system that is scalable for tree boosting. Its impact has been widely acknowledged in several machine learning and data mining challenges. One of the most notable competitions is hosted by Kaggle, a machine-learning competition site. Of the 29 challenge-winning solutions, three were published on Kaggle's blog in 2015, and 17 used XGBoost. Out of these solutions, eight solely used XGBoost to train their models, while most others combined XGBoost with neural nets in ensembles. In comparison, the second most popular method, deep neural nets, was only used in 11 solutions. XGBoost's success was further demonstrated in KDDCup 2015, where every winning team in the top 10 used XGBoost [15]. Additionally, the winning teams reported that ensemble methods only slightly outperform a well-configured XGBoost [14].

It achieves state-of-the-art performance on a broad range of problems. These winning

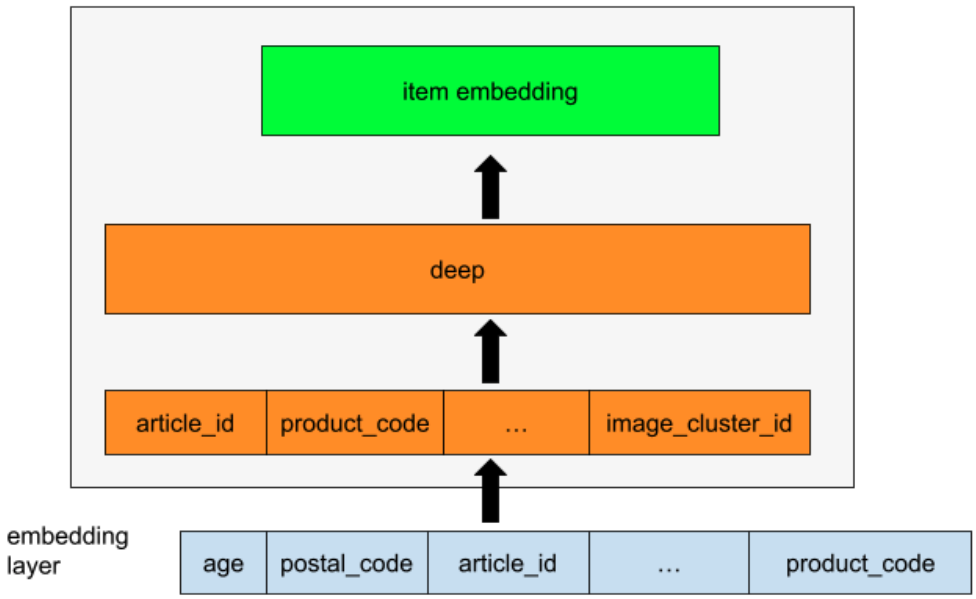

Figure A.1: Item embedding is the process of representing the articles as fixed-length vectors of numbers [5] to allow the model to understand the relationships between different items and to make recommendations based on similarities. In this case, the embedding layer makes use of some of the available attributes e.g., age, article id, and postal code. This neural network learns to get the essence of all the information of the item that is relevant to recommendation.

solutions include predicting store sales, classifying high energy physics events, classifying web text, forecasting customer behavior, detecting motion, predicting ad click-through rates, classifying malware, categorizing products, predicting hazard risks, and forecasting dropout rates for massive online courses. Although domain-specific data analysis and feature engineering are essential components of these solutions, the widespread adoption of XGBoost as the consensus choice of learners indicates the significance and impact of tree boosting.

The most critical factor in XGBoost's success is its scalability across all scenarios. It runs over ten times faster than existing popular solutions on a single machine and can scale to billions of examples in memory-limited or distributed settings. XGBoost's scalability is due to several important algorithmic and system optimizations, including a novel tree learning algorithm for handling sparse data, a weighted quantile sketch procedure that is theoretically justified, and enables the handling of instance weights in approximate tree learning (see fig. A.4). Parallel and distributed computing make learning faster, enabling quicker model exploration. More importantly, XGBoost leverages out-of-core

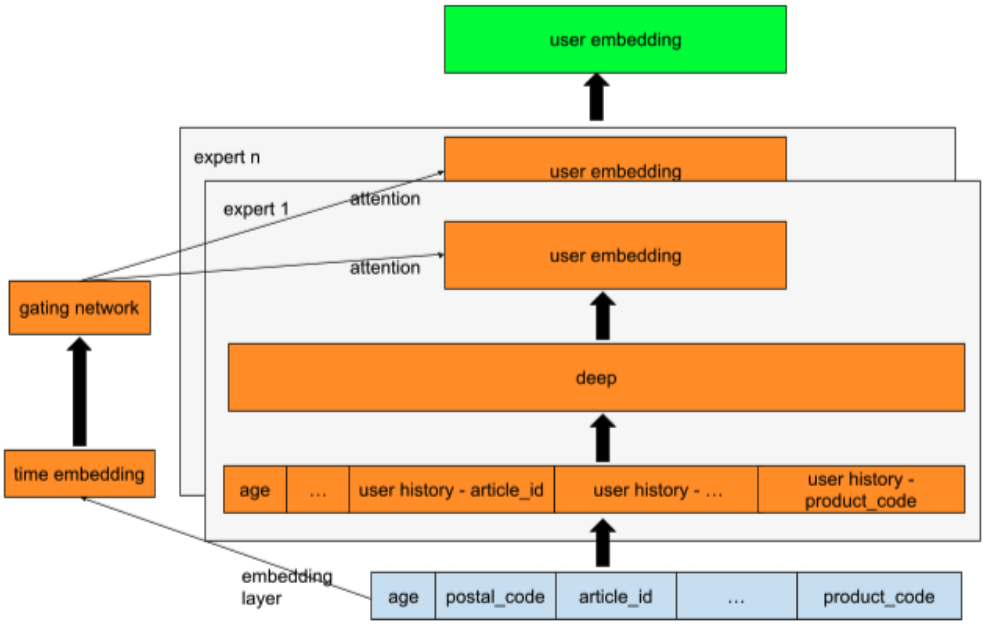

Figure A.2: User embedding is the process of representing the users as fixed-length vectors of numbers [119] to allow the model to understand users' interests and make personalized recommendations. In this case, the embedding layer makes use of some of the available attributes e.g., age and user history. This neural network (encoder) learns to take all the information about the user and make a fixed-size vector from it.

computation, enabling data scientists to process hundreds of millions of examples on a desktop. Combining these techniques makes an end-to-end system that scales to even larger data with the least amount of cluster resources, which is even more exciting.

As demonstrated in fig. A.5, decision trees generate a model that predicts the label by evaluating a tree of true/false feature questions in an if-then-else format. The minimum number of questions required to assess the probability of making a correct decision is also estimated. Decision trees can be used for classification to predict a category or regression to predict a continuous numeric value.

A Gradient Boosting Decision Tree (GBDT) is a decision tree **ensemble learning algorithm** similar to the random forest, for classification and regression. Ensemble learning algorithms combine multiple machine learning algorithms to obtain a better model. Both random forest and GBDT build a model consisting of multiple decision trees. The difference is in how the trees are built and combined.

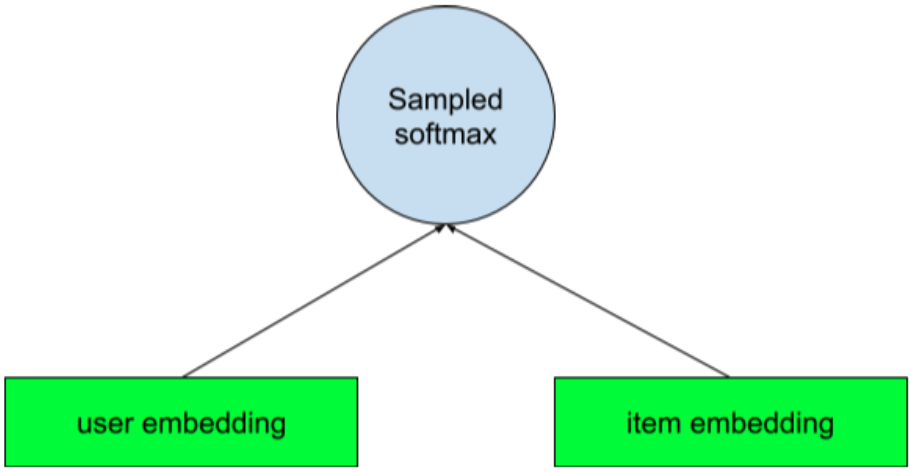

Figure A.3: The two 'towers'/encoders of fig. A.1 and fig. A.2 are joined together. Thus each pair of users and item pass through these towers to get a fixed-size user embedding and an item embedding of the same size. Then it computes a *sampled softmax* (see section 2.5.1) of these two vectors. If the value is high then it means that the item is a good match for the user.

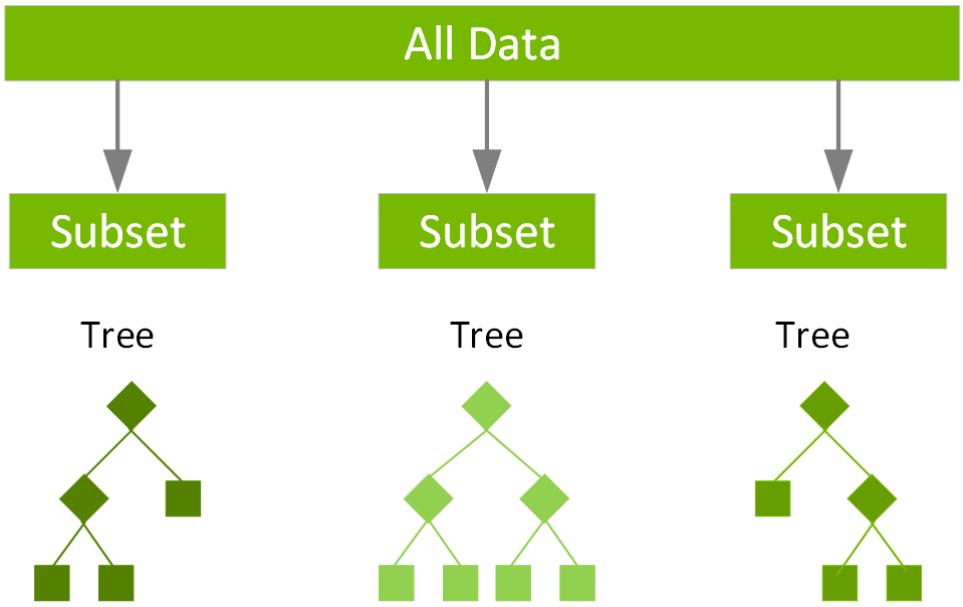

Figure A.6: Random forest uses a technique called bagging to build full decision trees in parallel from random bootstrap samples of the data set. The final prediction is an average of all of the decision tree predictions.

The concept of "gradient boosting" arises from the notion of improving a single weak

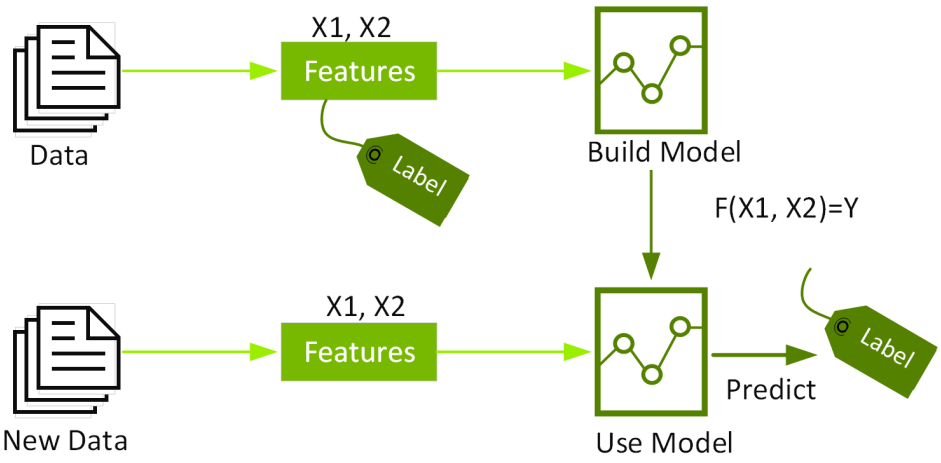

Figure A.4: An XGBoost model is built giving a labeled features dataset as input. Then the model is able to predict the label for unseen data. The label can be a score in the case of the regression model or a class in the case on a classifier

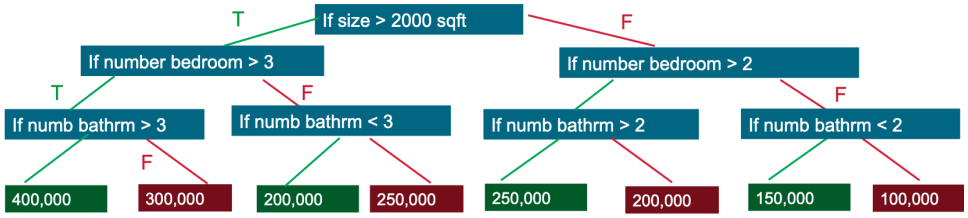

Figure A.5: This is a sample representation of a boosting tree, where at each node, based on some condition, the model chooses which path to follow, until it reaches a leaf.

model by combining it with several other weak models to form a collectively strong model. Gradient boosting is an extension of boosting that formalizes the process of additively generating weak models as a gradient descent algorithm over an objective function. The goal of gradient boosting is to minimize errors by setting targeted outcomes for the next model. These outcomes are based on the gradient of the error (thus the name gradient boosting) with respect to the prediction. Gradient Boosted Decision Trees (GBDTs) iteratively train an ensemble of shallow decision trees. Each iteration employs the error residuals of the previous model to fit the next model. The final prediction is a weighted sum of all tree predictions. While random forest "bagging" reduces the variance and overfitting, GBDT "boosting" reduces the bias and under-fitting. XGBoost is a highly accurate and scalable implementation of gradient boosting that maximizes the computational power of boosted tree algorithms. It is built to increase machine learning model accuracy and computational speed. Unlike GBDT, XGBoost builds trees in parallel, utilizing a level-wise strategy that scans across gradient values and utilizes these partial sums

to evaluate the quality of splits at every possible split in the training set.[1]

The list of benefits and attributes of XGBoost is extensive, and includes the following:

- A large and growing list of data scientists globally that are actively contributing to XGBoost open source development.
- Usage on a wide range of applications, including solving problems in regression, classification, ranking, and user-defined prediction challenges.
- A library that's highly portable and currently runs on OS X, Windows, and Linux platforms.
- Cloud integration that supports AWS, Azure, Yarn clusters, and other ecosystems
- Active production use in multiple organizations across various vertical market areas
- A library that was built from the ground up to be efficient, flexible, and portable

A.8. LightGBM

LightGBM is a gradient boosting framework that uses tree-based learning algorithm. LightGBM grows tree vertically while other algorithm grows trees horizontally. This means that trees generated by this algorithm grow *leaf-wise* A.7 while the ones generated by other algorithm grow *level-wise* A.8

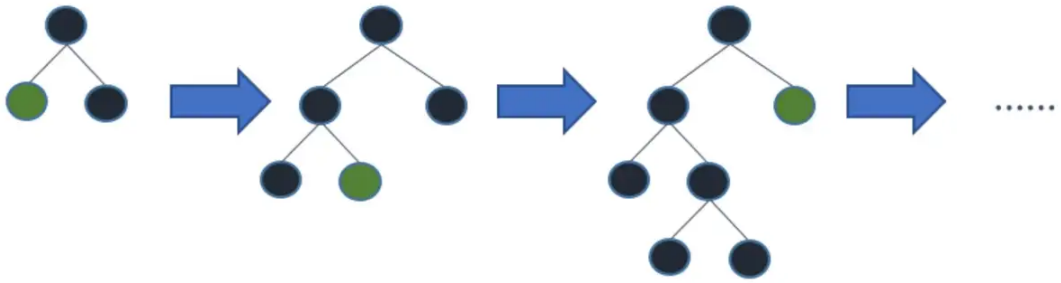

Figure A.7: Trees generated by LightGBM grow leaf-wise.

[1]XGBoost Website `https://www.nvidia.com/en-us/glossary/data-science/xgboost/`

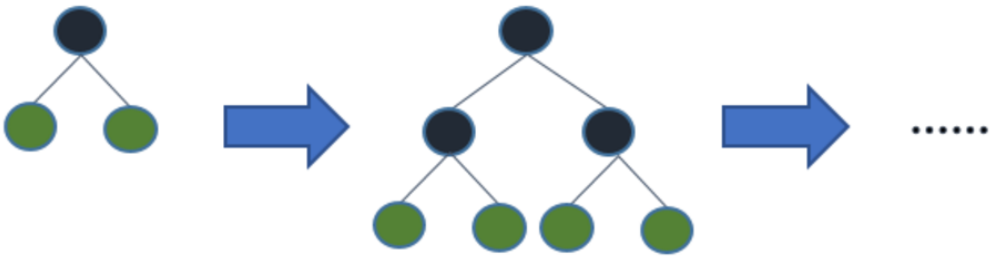

Figure A.8: Trees generated by other gradient boosting trees algorithm grow level-wise.

LightGBM is prefixed as *Light* because of its high speed. LightGBM can handle the large size of data and takes lower memory to run. Another reason why LightGBM is popular is that it focuses on the accuracy of results. LGBM also supports GPU learning and thus data scientists are widely using LGBM for data science application development. But it can also overfit when used with small datasets, that is why the advice is to use it only with a very large dataset.

Implement a model with LightGBM is easy thanks to the provided API with the library, what is difficult is tuning the model, since the library has a lot of control parameters, for example:

- **max_depth**: It describes the maximum depth of the tree. This parameter is used to handle model overfitting. Any time you feel that your model is over-fitted, my first advice is to lower max_depth.
- **min_data_in_leaf**: It is the minimum number of records a leaf may have. The default value is 20, the optimum value. It is also used to deal with overfitting
- **feature_fraction**: Used when your boosting(discussed later) is random forest. 0.8 feature fraction means that LightGBM selects 80% of parameters randomly in each iteration for building trees.
- **bagging_fraction**: specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.
- **early_stopping_round**: This parameter can help you speed up your analysis. The model stops training if one metric of one validation data does not improve in the last early_stopping_round rounds. This reduces excessive iterations.
- **lambda**: lambda specifies regularization. The typical value ranges from 0 to 1.
- **min_gain_to_split**: This parameter describes the minimum gain to make a split.

It can be used to control the number of useful splits in trees.

- **max_cat_group**: When the number of categories is large, finding the split point on it is easily over-fitting. So LightGBM merges them into 'max_cat_group' groups and finds the split points on the group boundaries, default:64
- **application**: This is the most important parameter and specifies the application of your model, whether it is a regression problem or a classification problem. LightGBM considers the model as a regression model by default.
  - regression: for regression
  - binary: for binary classification
  - multiclass: for multiclass classification problem
- **boosting**: defines the type of algorithm you want to run, default=gdbt
  - gbdt: traditional Gradient Boosting Decision Tree
  - rf: random forest
  - dart: Dropouts meet Multiple Additive Regression Trees
  - goss: Gradient-based One-Side Sampling
- **metric**: again one of the important parameters as it specifies loss for model building. Below are few general losses for regression and classification.
  - mae: mean absolute error
  - mse: mean squared error
  - binary_logloss: loss for binary classification
  - multi_logloss: loss for multi-classification

A.9. Catboost

Most popular implementations of gradient boosting use decision trees as base predictors. It is convenient to use decision trees for numerical features, but, in practice, many datasets include categorical features, which are also important for prediction. A categorical feature is a feature having a discrete set of values that are not necessarily comparable with each other (e.g., the user ID or name of a city). The most commonly used practice for dealing with categorical features in gradient boosting is converting them to numbers before training.

Catboost instead is capable of handling those categorical features, without converting them into numerical features, taking advantage of dealing with them during training as opposed to processing time. Another advantage of the algorithm is that it uses a new schema for calculating leaf values when selecting the tree structure, which helps to reduce overfitting.

It also outperforms the existing state-of-the-art implementations of gradient-boosted decision trees (GBDTs) XGBoost and LightGBM, on a diverse set of popular tasks. [2] [3]

CatBoost has both CPU and GPU implementations. The GPU implementation allows for much faster training and is faster than both state-of-the-art open-source GBDT GPU implementations, XGBoost and LightGBM, on ensembles of similar sizes. The library also has a fast CPU scoring implementation, which outperforms XGBoost and LightGBM implementations on ensembles of similar sizes.

Categorical features in Catboost have a discrete set of values called categories which are not necessarily comparable with each other; thus, such features cannot be used in binary decision trees directly. A common practice for dealing with categorical features is converting them to numbers at the processing time, i.e., each category for each example is substituted with one or several numerical values.

The most widely used technique which is usually applied to low-cardinality categorical features is one-hot encoding: the original feature is removed and a new binary variable is added for each category [14]. One-hot encoding can be done during the processing phase or during training, the latter can be implemented more efficiently in terms of training time and is implemented in CatBoost.

Another way to deal with categorical features is to compute some statistics using the label values of the examples. Namely, assume that we are given a dataset of observations $D = (X_i, Y_i)_{i=1..n}$ where $X_i = (x_i, 1, ..., x_{i,m})$ is a vector of m features, some numerical, some categorical, and $Y_i \in R$ is a label value. One possible way is to substitute the category with the average label value on the whole train dataset.

We can than substitute $X_{i,k}$ with the following equation A.7

$$\frac{\sum_{j=1}^{n}[x_{j,k} = x_{i,k} \cdot Y_j]}{\sum_{j=1}^{n}[x_{j,k} = x_{i,k}]} \tag{A.7}$$

[2] XGBoost Doc `https://xgboost.readthedocs.io/en/stable/`
[3] LightGBM Doc `https://lightgbm.readthedocs.io/en/v3.3.2/`

The symbol $[\cdot]$ represent the Iverson brackets. [4] In particular it follows that rule: $[x_{j,k} = x_{i,k}]$ is equal to 1 if $x_{j,k} = x_{i,k}$, 0 otherwise.

This procedure leads to overfitting. For example, if there is a single example from the category $x_{i,k}$ in the whole dataset then the new numeric feature value is equal to the label value on this example. A straightforward way to overcome the problem is to partition the dataset into two parts and use one part only to calculate the statistics and the second part to perform training. This reduces overfitting but it also reduces the amount of data used to train the model and to calculate the statistics.

Catboost uses a more efficient strategy to reduce overfitting using the whole dataset for training. It performs a random permutation of the dataset and, for each example, it computes average label value for the example with the same category value placed before the given one in the permutation.

Let $\sigma = (\sigma_1, ..., \sigma_n)$ be the permutation, then $x_{\sigma_p,k}$ is substituted with the equation A.8

$$\frac{\sum_{j=1}^{p-1}[x_{\sigma_j,k} = x_{\sigma_p,k}]Y_{\sigma_j} + a \cdot P}{\sum_{j=1}^{p-1}[x_{\sigma_j,k} = x_{\sigma_p,k}] + a} \tag{A.8}$$

In the equation, we also add a prior value P and a parameter $a > 0$, which is the weight of the prior. Adding prior is a good practice and it helps to reduce the noise obtained from low-frequency categories [13]. For regression tasks standard technique for calculating prior is to take the average label value in the dataset. For binary classification tasks, a prior is usually a prior probability of encountering a positive class. It is also efficient to use several permutations. However, one can see that a straightforward usage of statistics computed for several permutations would lead to over-fitting. As we discuss in the next section, CatBoost uses a novel schema for calculating leaf values which allows using several permutations without this problem. [22]

[4] Iverson brackets `https://oeis.org/wiki/Iverson_bracket`

List of Symbols

Variable	Description
$\boldsymbol{MAP@12}$	Mean Average Precision at 12
$\boldsymbol{MAP}$	Mean Average Precision
$\boldsymbol{P}$	Positive Predictive Rate
$\boldsymbol{P(k)}$	Precision at k
$\boldsymbol{Rel(k)}$	Relevance at k
$\boldsymbol{r_{ui}}$	Rating of user u to item i
$\boldsymbol{C}$	Shrink term
$\boldsymbol{s_{ji}}$	Items similarity
$\boldsymbol{KNN}$	k-Nearest Neighbours
$\boldsymbol{CF}$	Collaborative filtering
$\boldsymbol{URM}$	User Rating Matrix
$\boldsymbol{ICM}$	Item Content Matrix
$\boldsymbol{b}$	Bias
$\boldsymbol{MAE}$	Mean Average Error
$\boldsymbol{MSE}$	Mean Square Error
$\boldsymbol{AUC}$	Area Under the Curve
$\boldsymbol{SLIM}$	Sparse Linear Method
$\boldsymbol{MAR}$	Missing as Random
$\boldsymbol{MAN}$	Missing as Negative
$\boldsymbol{ALS}$	Alternating Least Square

Acknowledgements

A mia mamma, a mio papà, a mio fratello. Grazie.