Master in Innovation and Research in Informatics - Data Science

Development of a Travel Recommender System

Facultat d'Informàtica de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech
Stayforlong

Author:
Eloy MARÍN CIUDAD

Advisor:
Daniel MADURELL BLANES, Stayforlong

Tutor:
Mario MARTÍN MUÑOZ, UPC - CS

June 2022

Abstract

Nowadays, e-commerce is facing the problem of information overload, where users are exposed to a vast amount of content, making it more difficult for users to effectively take quality decisions. The need for delivering the right items at the right moment to each user has resulted in making recommendation systems one of the hot topics in research and technology trends, where a good recommendation system might give a key advantage to an e-commerce over its competitors. Industry-leading companies such as Youtube or Amazon introduced the concept of large-scale recommendation systems, where the number of candidate items to consider for recommendation is enormous, and efficient techniques must be applied. The most common way to deal with large-scale recommendation systems nowadays is to build a retrieval model that retrieves a subset of relevant items for the user, and a ranking model that scores and ranks the set of retrieved items.

Research on recommendation systems is continuously evolving, where new approaches produce state-of-the-art results, which especially happens thanks to the rise of deep learning. In this thesis, we describe classical and current approaches to recommendation systems, from content-based methods without assuming latent factors and collaborative-filtering methods like matrix factorization, to hybrid approaches, deep learning-based methods, and state-of-the-art approaches. Precisely, we focus on the concept of context-aware methods and multitask methods, which aim to optimize more than one task at a time. In this master thesis, we focus on developing a recommendation system for *Stayforlong* as a proof of concept. Firstly, we analyze their data and see that we can opt for a hybrid context-aware model since we have at our disposal user features, item features and context features. Another thing that characterizes the data set that we work with is the abundance of implicit feedback and the scarcity of explicit feedback. This drives us to experiment with different model architectures and approaches, focusing on developing a hybrid model that performs a retrieval task and another hybrid model that performs a ranking task. In addition, we check in our experiments the benefits of adding context to our models, the benefits of jointly training a model that optimizes multiple tasks, and the benefits of training a model on an abundant data set, like implicit feedback, and applying transfer learning to fine-tune on explicit feedback the learned representations. Results show that, in rich data scenarios, a context-aware multitask hybrid model trained on implicit feedback and fine-tuned with explicit feedback outperforms other approaches such as training separate retrieval and ranking models, disregarding implicit feedback or not including context features in the models.

Finally, we propose as future work a data pipeline for the recommendation system to be used in production, taking into account data freshness and model re-training.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

It is well-known that nowadays we are overwhelmed with the amount of available content that we are exposed to whenever we are navigating online. Over the recent years, many companies from different sectors, such as entertainment or e-commerce sites, have focused on making the user experience more personalized to tackle the problem of "content overwhelming" or "information overload" when searching for any type of content. This is where Recommendation Systems excel, benefiting at the same time users and providers because they are able to filter and deliver the relevant items for every user. As a study from McKinsey [1] shows, already in 2013, 35 percent of purchases on Amazon and 75 percent of content watched on Netflix came from recommendation systems predictions. In addition, Forbes recently listed Recommendation Systems as one of the top 10 technology trends in retail [2]. With regard to the tourism and travel sector, industry-leading companies such as *Booking.com* [3, 4] and *Airbnb* [5, 6] have put a great effort into developing several recommendation systems. All these indicators led to the main motivation of this work: *Can we build a Recommendation System to recommend hotels to users?*

1.2 Scope

One of the main issues when developing machine learning algorithms is the available data. In the context of recommendation systems, the available data at hand can highly influence which approach to take (content-based filtering, collaborative-filtering, and so on). The project will focus on analyzing the available data, modeling a data set that can be used to train a recommendation algorithm, and finally evaluating this algorithm.

To do so, classical and state-of-the-art recommendation system approaches and algorithms will be studied. Finally, some approaches will be chosen according to the data available at hand and will be evaluated on our data set. The steps that will be carried out are the following:

- Explore which data is available and how can it be modeled in order to be used in a recommendation system. Once we model the data, data analysis will be performed.
- Study different algorithms and/or techniques that are commonly used in recommendation systems. Given the available data, decide which approach to take.

- Finally, evaluate the performance of the selected approaches. Experiment with implicit and explicit feedback, context features and retrieval, ranking, and multitask models.

1.3 Objectives

The thesis objectives can be defined as the following:

- Modeling user-item interactions from different sources, or types of feedback, and analyzing the interactions data.
- Evaluate the performance of one or multiple machine learning models on the generated data set. See if combining implicit and explicit feedback can improve models' performance, evaluate the benefits of adding context features, and, lastly, evaluate the benefits of building a multitask model over separate models.
- Design a proof of concept to model how the recommendation system will work in a production environment.

1.4 Thesis outline

The planned procedure for this thesis is the following: Firstly, research on the topics that this thesis works with will be performed in chapter 2. There, different recommendation systems approaches will be described in detail. Following, the datasets, hyperparameters, and architectures used to train the machine learning models will be described in chapter 3 as well as the chosen metrics to evaluate the quality of recommendations. Next, data pre- and post-processing will be described in chapter 4. Finally, the results of the experiments will be shown and analyzed in chapter 5. And lastly, some conclusions and possible future work will be mentioned in chapter 6.

2. Background

This chapter is focused on describing the main concepts that this thesis works with and reviewing the different approaches taken by related works.

2.1 Problem to be solved

Recommender Systems are one of the main applications of machine learning in online companies nowadays, where the vast amount of available content can play a negative role, worsening the user navigation experience. That is where recommender systems come into play because they can act as a kind of content filter that helps users in finding items of their interest. Therefore, recommender systems provide benefits for both users and the companies that provide them. For users, they can help with finding related items, exploring new ones, or improving decision-making by reducing an immense amount of content to only a few items that are relevant to the user, with the aim of providing personalization. From the company's perspective, they can increase user satisfaction, user loyalty and learn more about their users by analyzing their interactions with recommendations. In summary, recommendation systems aim to model users' preferences so companies can deliver them the right items.

In order to model users' preferences, we need to represent the user-item interactions matrix (see example in Figure 2.1). The value of each cell corresponds to the rating given by the user to an item, where a higher rating value means that the user liked the corresponding item more than other items with a lower rating. However, most of the time the amount of *explicit* ratings, or feedback, from users is quite small, and introducing *implicit* ratings from clicks or views can be useful to reduce the user-item interactions sparsity.

Nevertheless, this user-item interaction matrix will still be very sparse, since users only interact with a few items, meaning that their rating for most of the items is unknown. Therefore, the key problem that recommender systems try to solve is to predict these unknown ratings.

Figure 2.1: Example user-item interaction matrix. Source: [1].

2.2 Designing a Recommendation System

When designing a recommendation system, some key points must be taken into account:

- **Which are the personalization needs?**
  Before implementing a recommendation system, it is necessary to define the use case or personalization need where the recommender system will be applied. Defining a specific use case will result in better business understanding and modeling the available data according to that specific challenge.

- **What data is available?**
  After picking a personalization need, we have to decide how to model the recommendation system. This decision is highly influenced by the available data that describes user-item interactions. The fact of having user and/or item features can make the difference between being able to build a hybrid model or a collaborative-filtering one. Another important issue is the amount and types of feedback available. Explicit feedback is usually scarce, so being able to obtain implicit feedback can help on obtaining a better performance for the recommender system. On the other hand, relying only on implicit feedback can have negative effects since some assumptions are made. For instance, we would assume that a user clicking on an item means that they liked it, while it may not be true.

- **How will recommender systems' pitfalls be addressed?**
  One of the main pitfalls of recommender systems is the *cold-start* problem [7]. The cold-start

[1] `https://www.researchgate.net/figure/Example-of-user-item-ratings-matrix-in-a-movie-recommendation-scenario_fig2_300646445`

problem happens when a recommendation has to be given to a new user, meaning that this user has not previously interacted with any item. One approach could be to make a default recommendation based on some business rules, while another approach could be to use a content-based or hybrid approach.

Another pitfall is the lack of explicit feedback, which requires making assumptions and getting implicit feedback from users.

- **Recommender system lifecycle**
  Once a model is trained and has a good performance, we need to put it into production. That means that we have to decide if recommendations will be served on-line or off-line, how often the model will be re-trained and how we will maintain it and constantly evaluate its performance.

2.3 Types of Recommendation Systems

Recommendation systems can be categorized into three main categories: content-based, collaborative-filtering and hybrid methods [8]. This section describes these different approaches that can be taken when building a recommender system.

2.3.1 Collaborative filtering methods

Collaborative-filtering methods assume that similar users will like the same type of items. Their main advantage is that they require no information about users and items. However, collaborative-filtering methods suffer the *cold-start* problem, where it is not possible to recommend items to a new user or to recommend a new item because they have no previous interactions.

Within the collaborative-filtering category, we can distinguish two types of approaches: Memory-based and Model-based approaches.

2.3.1.1 Memory-based approaches

Memory-based approaches are based on algorithms like nearest-neighbors [9], where no latent factors are assumed and we work directly with user-item interactions. These approaches can also be divided into two categories: *user-user* and *item-item*.

- **User-user**
  The *user-user* approach computes the similarity between a user of interest and every other user. Then, once we have the k-most similar users, the most popular items between them are suggested.

- **Item-item**
  The *item-item* approach tries to find similar items to the ones the user of interest already interacted with. Two items are seen as similar if users similarly interact with them.

The main drawback of memory-based approaches is that they do not scale because the nearest-neighbor search will be more time-consuming as the number of interactions grows. One possible solution could be to use approximate nearest neighbors [10].

2.3.1.2 Model-based approaches

Model-based approaches assume an underlying latent space that describes the user-item interactions matrix. Models are trained to obtain the users' and items' latent representations that can reconstruct the original user-item matrix. Among these techniques, Matrix Factorization [11] is one of the most popular techniques after *Netflix* held a competition for the best collaborative-filtering method to predict user ratings for films [12].

- **Matrix factorization**
  Matrix factorization algorithms assume that there exists a low-dimensional latent space that can represent both users and items such that a user-item interaction can be obtained by applying the dot product of the corresponding vectors in the latent space.

  The matrix factorization problem can be modeled as the following, where we have $A \in R^{mxn}$ as the interaction matrix, $U \in R^{mxd}$ as the user embedding matrix, and $V \in R^{nxd}$ as the item embedding matrix. The objective is to learn user and item embeddings such that UV^T approximates the interaction matrix A. Therefore,

$$A \approx UV^T \tag{2.1}$$

  Once the matrix is factorized, we reduced the amount of data to manipulate to a set of low-dimensional vectors and we just have to compute dot products between them to obtain a predicted rating.

2.3.2 Content-based methods

Content-based methods assume that users will like items with similar features to the ones they already liked. Therefore, unlike collaborative-filtering methods, content-based methods use additional information about items and/or users and do not rely on other users or items interactions.

The main advantage of these methods is that they mitigate the effect of the *cold-start* problem because new items or users can be described by their features, enabling them to find relevant recommendations based on their characteristics. However, content-based methods have one main drawback, which is that they have a limited ability to expand users' interest in unrated features.

Therefore, recommendations for a user will always fall in the same niche categories. Another disadvantage is that the features that represent items and users are hand-engineered, so this approach requires domain knowledge.

For content-based methods, the recommendation problem can be modeled either as a classification problem or a regression problem. In both cases, we can distinguish between two approaches:

- **Item-centered**
  When the model is based on user features, the approach is *item-centered*. This approach tries to predict the probability of each user to like a specific item. Since, usually, lots of users interact with an item, this approach results in building more robust models, but at the same time, they are less personalized because users that have interacted with this item may have different preferences.

- **User-centered**
  When the model is based on item features, the approach is *user-centered*. This approach tries to predict the probability of a user liking each item. This approach is more personalized since it only takes into account interactions from the user of interest. However, since users only interact with a few items, the model is less robust.

2.3.3 Hybrid methods

Hybrid methods assume that combining both collaborative-filtering and content-based methods results in taking advantage of the best of both methods. This combination can mainly be done with two approaches. One approach would be to train two independent models, one collaborative-filtering model and one content-based model, to then combine their outputs as a model ensemble. The other approach is to directly build a single model that unifies both approaches, usually done with neural networks. In fact, the latter approach achieves state-of-the-art results thanks to the advances in deep learning [13] and it is used in many recommendation systems nowadays.

2.4 Context-aware recommendation systems

In some scenarios, such as the tourism industry, users' preferences can be described by a strong seasonality pattern. Therefore, in some cases, adding context will significantly improve the model performance. For instance, the day of the week or the time of the day can be an important factor to decide which videos or films to recommend to a user, where users may prefer short-length content during the week and longer videos or films during the weekend. In the case of tourism, the types of stay are the locations where users prefer to travel highly depends on the season of the year. For instance, during summer users look for hotels near the sea, while they might look for other characteristics when traveling in winter.

Adomavicius, G. et al. [14] present three different approaches to include context features in a recommendation system, as shown in Figure 2.2. These approaches can be described as the following:

- **Contextual Prefiltering**
  In this approach, data is sampled according to its context, and then predictions are computed like any recommendation system.
- **Contextual Postfiltering**
  Here, recommendations are computed and then they are filtered according to their context.
- **Contextual Modeling**
  This approach consists in incorporating contextual features into the recommendation system model.

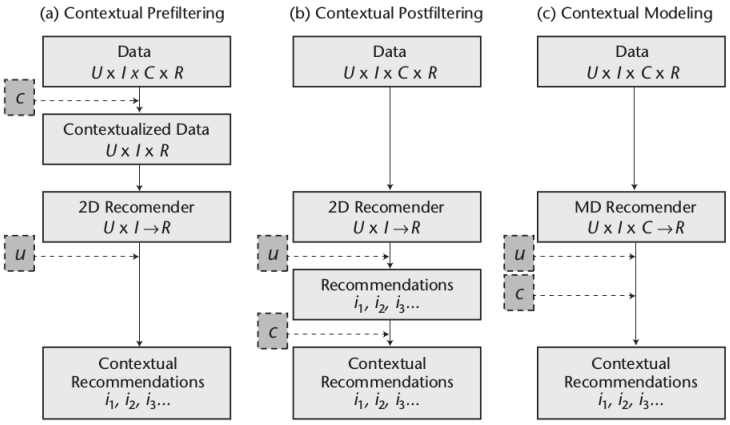


Figure 2.2: Incorporating context approaches. Source: [14].

2.5 Deep-learning based Recommendation Systems

Current state-of-the-art recommendation systems are dominated by deep learning techniques, showing a significant improvement over other state-of-the-art approaches [15]. Within deep learning, there is a great variety of architectures that have been used for recommendation systems, such as Convolutional Neural Networks [16, 17] or Recurrent Neural Networks [18, 19].

This section describes the main deep learning concepts and architectures that the thesis works with.

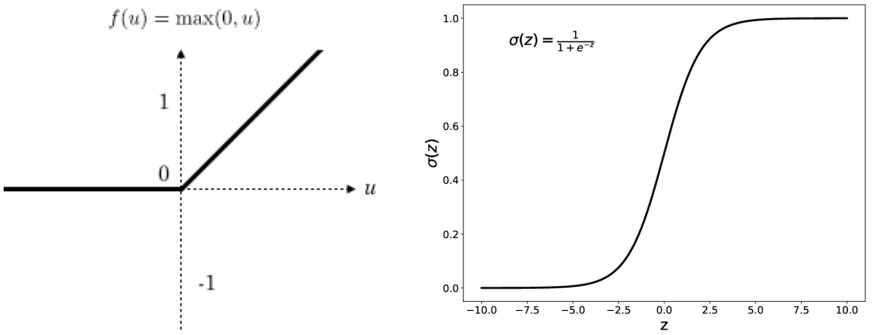
2.5.1 Feed forward neural networks

Feed forward neural networks are one of the basic deep learning architectures. Figure 2.4 shows an example architecture for these types of networks, where we can see that it has an input layer, one or more hidden layers, and an output layer, where neurons between layers are densely connected,

meaning that there are no missing connections. The output of each hidden layer can be defined as the following:

$$h_i^n = f(\sum_j w_{i,j}^n x_j + b_i^n) \tag{2.2}$$

where i refers to the neuron i in the nth hidden layer, $w_{i,j}$ and b_i^n are the weight and bias of neuron i in hidden layer n, x_j is the input that neuron i receives from neuron j, and f is the activation function. Some of the most common activation functions are sigmoid and ReLu (see Figure 2.3b and Figure 2.3a).



(a) ReLU activation function. Source: Researchgate [2].

(b) Sigmoid activation function. Source: Researchgate [3].

Figure 2.3: ReLU and Sigmoid activation functions.

[1] https://www.researchgate.net/figure/ReLU-activation-function_fig3_319235847

[3] https://www.researchgate.net/figure/Sigmoid-activation-function-for-binary-classification-in-neural-networks_fig3_352647580

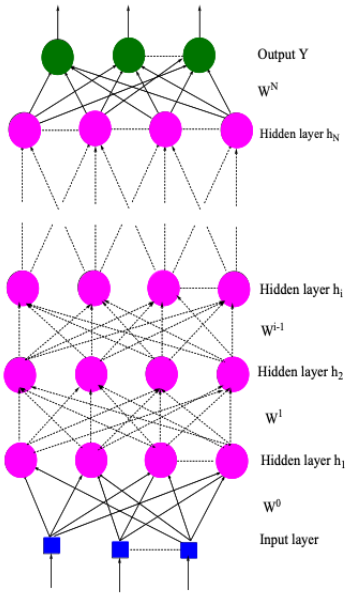


Figure 2.4: Feed forward neural network architecture. Source: [20].

2.5.2 Embeddings

An embedding is a mapping of a categorical variable to a vector in a continuous space. Within the deep learning context, embeddings can be described as low-dimensional representations of categorical variables. Embeddings have been demonstrated to be useful because they are able to reduce categorical variables' dimensionality while meaningfully representing them in the new low-dimensional space [21].

A common practice to obtain a low-dimensional representation of a categorical variable is to use a feed forward neural network as shown in Figure 2.5.

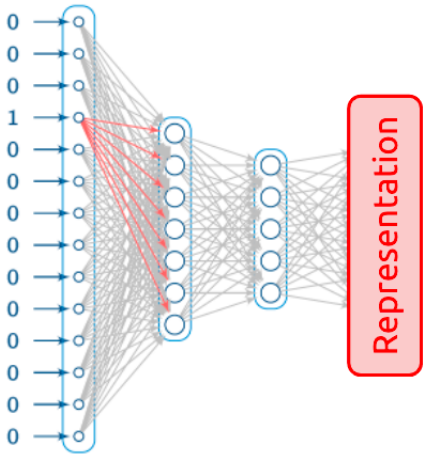


Figure 2.5: Representation of an embedding layer. Source: [4].

2.5.3 Transfer learning and fine-tuning

Transfer learning is based on the idea that a model that is trained on a large data set will be useful as a general model for the same or similar tasks. Then, when you have a small quantity of data, you can take advantage of what a specific model learned on a larger dataset without having to train it yourself.

A common practice when doing transfer learning is to fine-tune the pre-trained model. Fine-tuning consists in unfreezing the last layers of the pre-trained model and re-training them on the new data set, with the assumption that learned features will be slightly adjusted to the new data they are being trained on.

2.5.4 Two-tower networks

Two-tower architectures applied to recommendation systems consist of a candidate tower that learns to represent item embeddings and a query tower that learns to represent user embeddings. Then a scoring function computes the distance between the user and item embeddings, obtaining a measure for their similarity. This type of structure has been widely used in state-of-the-art approaches such as Deep & Cross Network (DCN) [22] and Wide & Deep network [23].

As Figure 2.6 shows, each tower is composed of an embedding layer followed by multiple fully connected layers like in a feed forward neural network.

[4] `https://telecombcn-dl.github.io/2019-idl/`

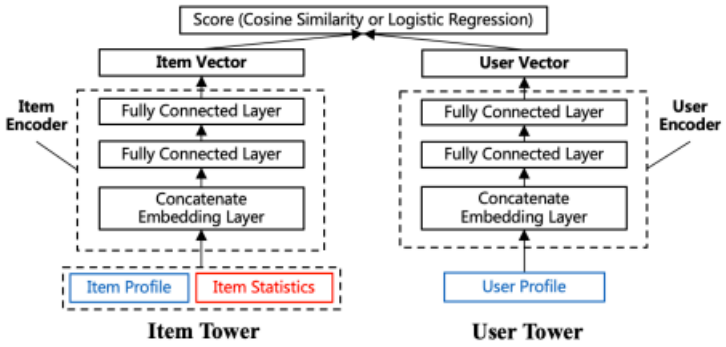


Figure 2.6: Representation of a two-tower neural network. Source: [24].

2.5.5 Two-tower shared-bottom model

Ruoxi Want et al. [25] introduced the two-tower shared-bottom model in the context of multi-task learning. The objective is to take advantage of implicit feedback as auxiliary objectives in retrieval systems, improving the quality of the learned embeddings for queries and items. Specifically, this model is first trained with implicit feedback to learn query and candidate representations, and then these embeddings are fine-tuned using explicit feedback. The knowledge from implicit feedback is transferred to the explicit feedback task thanks to the shared bottom layers, and then each task has its independent top layers, as shown in Figure 2.7.

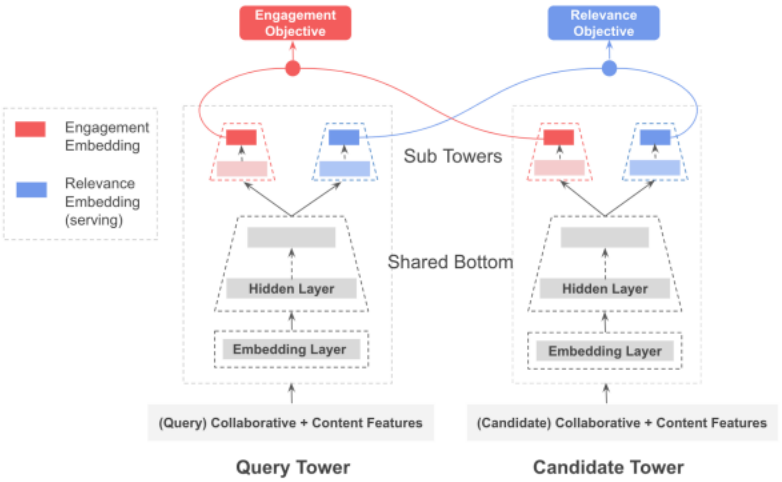


Figure 2.7: Representation of a two-tower shared-bottom neural network. Source: [25].

2.6 Recommendation Systems in production

One of the main challenges that recommendation systems face is to score millions of items in real-time. To tackle this problem, a common practice in large-scale recommender systems, such as the one implemented by Youtube [26], is to design a two-phase recommendation system, where the first phase consists of a retrieval model that reduces the original set of candidate items to a smaller subset that might be of interest for the user. Then the second phase consists of a ranking model that takes the retrieval model output and scores the items.

2.6.1 Retrieval stage

The retrieval stage can be defined as follows: given a query (user features + context) and a set of candidate items, retrieve the top-k most relevant candidates for the corresponding query. For large-scale recommendation systems where serving time is crucial, the task does not end once the queries and item representations are learned but also an efficient algorithm, such as ANN [10], must be used to find the top-k relevant candidates in an efficient time.

The main objective is to generate a set of relevant candidates from the original set of items. It is a common practice to have different candidate generators at the same time that may take into account other things such as business KPIs to select popular items.

2.6.2 Ranking stage

After generating a set of candidates in the retrieval stage, a ranking model scores and ranks the items to select the ones that will be recommended. One important aspect to consider in ranking models is the objective function for scoring, such as maximizing CTR, since it can drastically influence the ranking and quality of recommended items.

2.6.3 Re-ranking stage

Before serving recommendations, another common step is to re-rank or filter out items from the ranking model output according to some business logic or KPIs, such as popularity or location.

2.6.4 Recommendation Systems Lifecycle

Once a retrieval and a scoring model have been successfully trained, some key decisions must be taken in order to use the recommendation system in a production environment. On the one hand, the system pipeline with its different stages (retrieval, ranking, re-ranking) must be defined to serve recommendations, which might involve the usage of an orchestrator like Airflow [5]. On the other

[5] `https://airflow.apache.org`

hand, freshness is a crucial aspect in recommendation systems to use the latest users and items information, so re-training should be done as often as possible.

3. Methodology proposed

In this chapter, the method used in the conducted experiments and the system proposed are presented. Firstly, the generation and the description of the data set will be presented. Next, the architecture and hyperparameters chosen for each experiment will be described and the metrics used to evaluate the quality of recommendations will be explained. Lastly, the technologies and tools used to implement the recommender system will be presented.

3.1 Dataset

This section describes the available data that this thesis works with and how it has been integrated into a single data set to experiment with different recommendation system approaches. Then an exploratory data analysis to gain understanding and a data cleaning process for the generated base data set are presented.

3.1.1 Dataset generation

The available data for this thesis describes the purchase transactions history from Stayforlong's website since 2018 and describes user navigation data since the year 2020. In order to generate a data set that can be used as an input for a recommender system, we need to integrate user and item features together with user-item interactions' context features. The features corresponding to each one are specified in the following list:

- **Users**
  In any e-commerce, there are two types of users: logged-in users and anonymous users. However, in the thesis context, we will only focus on logged-in users. User features are shown in Table 3.1.
- **Items**
  The items that will be recommended are accommodations (or hotels). Therefore, some key features such as hotel location and hotel star rating are available in the data set. These hotel features are shown in Table 3.2.

- **User-item interactions**
  There are three types of interactions that a user can have with an item in our data set. These interactions correspond to:

  – Booking an accommodation.

  – Clicking on an accommodation.

  – Viewing an accommodation.

  Clicks and views can be seen as a form of implicit feedback, while bookings can be seen as explicit feedback. Combining these types of interactions, we obtain the data set features shown in Table 3.3. The *rating* feature is a boolean (1 or 0) indicating whether the user interacted with an item or not.

Feature index	**Feature name**	**Description**	**Example**
1	user_id	user identifier	17
2	user_country	user's country	US
3	user_language	user's language	en

Table 3.1: User features.

Feature index	**Feature name**	**Description**	**Example**
1	item_id	hotel identifier	2524
2	item_country	hotel country	IT
3	item_city_id	city identifier	12073
4	item_name	hotel name	Barceló Raval
5	item_latitude	hotel latitude	-32.5
6	item_longitude	hotel longitude	14.9
7	item_postal_code	hotel postal code	08020
8	item_stars	hotel star rating	3
9	item_price_night	hotel price per night	57

Table 3.2: Item features.

Feature index	Feature name	Description	Example
1	user_id	user identifier	17
2	item_id	hotel identifier	2524
3	feedback_type	type of feedback	explicit
4	timestamp	timestamp of interaction	1652307638
5	rating	user-item rating	1

Table 3.3: User-item interaction features.

3.1.2 Exploratory data analysis

After obtaining the described features, we proceed to analyze the user-item interactions. Therefore, we perform an exploratory data analysis to examine which features include meaningful data, process them and handle possible outliers or missing values.

We can divide this analysis into three blocks: users, items, and interactions data.

- **Users**
  Within the interactions data set, there are a total of 435072 distinct users. Regarding their features, the amount of null values is minimal w.r.t the total amount of users. There are only 215 null values for the *user_country* feature, and 21 null values for *user_language*, as shown in Table 3.4.

  Since both *user_country* and *user_language* features refer to the market the user belongs to, we will only keep the *user_language* feature and drop the user-item interactions where this feature has a null value. This results in dropping 203 interactions.

Feature	Null values
user_id	0
user_country	215
user_language	21

Table 3.4: Null values for user features.

  When further analyzing the *user_language* feature, we see that some languages are over-represented in our data set, such as the English language, while other languages only appear in a single interaction. This can be seen in Figure 3.1 and Table 3.5.

User_language quartile	Frequency
25%	18
50%	480
75%	51136

Table 3.5: User_language quartile distribution

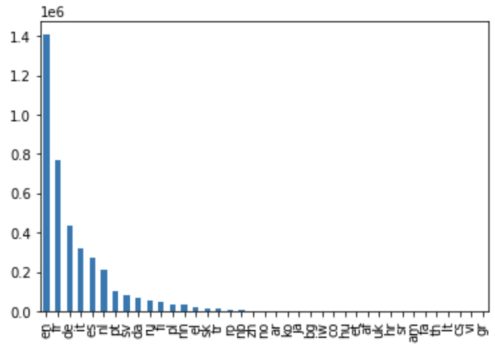


Figure 3.1: User_language frequency plot

- **Items**

  The number of distinct items that users have interacted with in the generated data set is 91680. Regarding their features, Table 3.6 shows that only three features contain null values. With respect to location-related features, we will drop the 12 items with a null *item_country* value and drop the *item_postal_code*, *item_latitude*, and *item_longitude* features, only keeping items with a valid *item_country* and *item_city_id*. As for the *item_price_night* feature, we can try to replace missing values with the average price per night of items located in the same city and with the same star rating.

Feature	Null values
item_id	0
item_country	12
item_city_id	0
item_name	0
item_latitude	0
item_longitude	0
item_postal_code	4320
item_stars	0
item_price_night	35823

Table 3.6: Null values for item features.

Regarding *item_stars* feature, we can see in Figure 3.2 that its values are centered around 3 and 4 stars, resembling a normal distribution. Since there is only one 7-star item, we will merge it into the items with 5-stars.

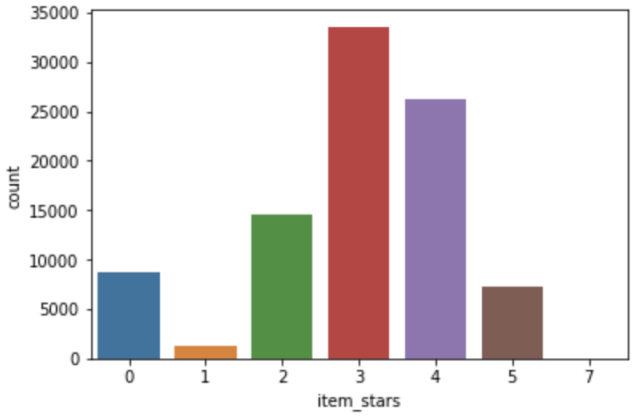


Figure 3.2: Item_stars count plot

When analyzing *item_price_night* non-null values, we can see in Table 3.7 that there are some incorrect values, such as negative prices or over-expensive ones. Therefore, we will treat these values as null ones and we will try to impute them based on the average price per night of same-stars items located in the same city. After imputation, we can see in Figure 3.3 that *item_price_night* follows a long-tailed normal distribution centered around 100 euros. Nevertheless, Table 3.7 shows that we managed to impute less than 50% of the missing and incorrect values after data imputation, so we will drop the *item_price_night* feature.

Measure	item_price_night	item_price_night after imput
mean	126.54	119.73
std	363.619	83.41
min	-23	10
25%	71	71
50%	101	100
75%	147	141.56
max	82086	994
null values	35823	20874

Table 3.7: Item_price_night feature description

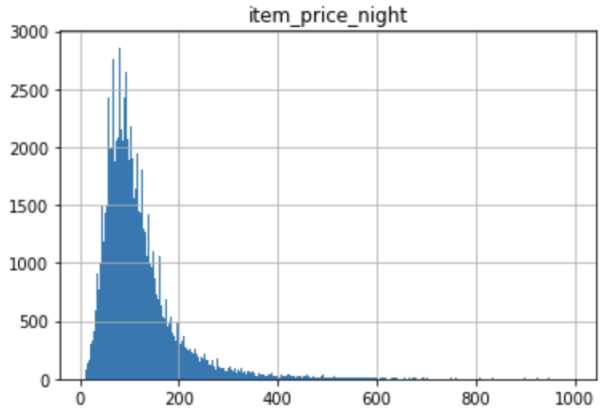


Figure 3.3: Item_price_night histogram

- **User-item interactions**
  There are a total of 3896248 interactions in our data set, 401812 of which correspond to purchases and the other 3494436 interactions correspond to clicks and views. Regarding null values, Table 3.8 shows that there are no null values for user-item interaction features.

Feature	Null values
user_id	0
item_id	0
feedback_type	0
timestamp	0
rating	0

Table 3.8: Null values in user-item interactions data.

Regarding *rating* and *timestamp* features, the former is a constant value set to 1 for every interaction, indicating that a user interacted with an item, while the latter gives us more information. Figure 3.4 shows the number of interactions per date, where we can see that a great part of the period represented in our data set only contains explicit feedback (purchases in this context) and implicit feedback (clicks and views) appears for the first time after the start of the year 2020. Although implicit feedback is missing for 2 years in our data set, web events data is abundant and it can be combined with explicit feedback, which tends to be scarcer, to give better recommendations. Besides that, if we focus on the number of interactions per day, we can notice a seasonal pattern, which is very frequent in the tourism sector. Figure 3.5 shows this pattern from different perspectives. On the left side, we can see the number of interactions per month, which increases from January until May and then starts decreasing. On the right side, we can see the number of interactions per hour, that reach their peak in the afternoon. This makes sense because that is when people get home from work and have time to plan their vacations.

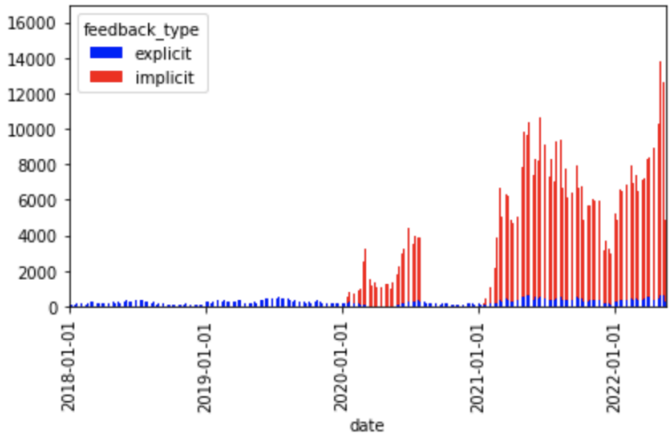
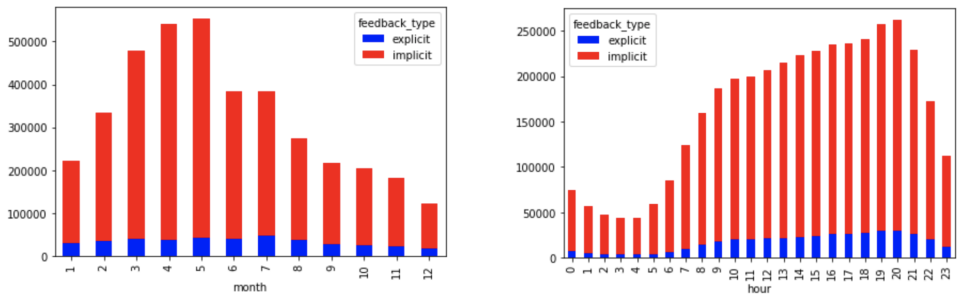


Figure 3.4: Interactions per day



(a) Interactions per month (b) Interactions per hour

Figure 3.5: Interactions count per month and hour

We are building a recommender system for an online travel agency, where the main idea is to recommend every user a set of hotels for their stay. When exploring how users interact with hotels on Stayforlong's website, we can see in Table 3.9 that 50% of users have at least interacted with three items in their history. This may happen because most customers probably book accommodation only once, therefore only navigate through the website a couple of times. We can also see that there are some users with an extremely big amount of interactions, which may correspond to bot navigation or erroneous data. Therefore, we will drop 376 users corresponding to the ones with more than 500 interactions.

Regarding item's frequency, 50% of them have at least 6 interactions or more. Nevertheless, a small number of popular items are the most dominant in the data set.

Measure	**item_id**	**user_id**
count	91680	435072
mean	42.5	8.96
std	179.71	43.72
min	1	1
25%	2	1
50%	6	3
75%	22	7
max	9446	15252

Table 3.9: Description of the number of interactions per user and item in the data set.

3.1.3 Resulting dataset

After exploring our data set, we decided to drop the following features:

- *user_country*

- *item_postal_code*
- *item_latitude*
- *item_longitude*
- *item_price_night*

In addition, we also decided to drop interactions that fulfilled the conditions in the following list, dropping a total of 382372 interactions.

- Users with null *user_language*
- Users with more than 500 interactions
- Items with null *item_country*

The resulting data set contains 3513876 interactions with 11 features and a set of 305941 candidate items. Table 3.10 shows an example instance of the resulting user-item interaction data set.

Feature	**Example**
user_id	22184
item_id	1415
user_language	es
item_country	PT
item_city_id	602838
item_name	Lisbon Hotel
item_stars	4
rating	1
feedback_type	explicit
timestamp	1514779643

Table 3.10: Resulting data set sample.

With regard to the features that will describe our data set, it is worth to mention that they are quite limited. Users are only characterized by their language, making it more difficult to compute relevant recommendations.

3.2 Models

The model experiments will be conducted on deep learning-based recommender system architectures. To do so, neural networks are built on *TensorFlow Recommenders* [27], a library for building

recommender system models using *TensorFlow* [28] and *Keras* [29]. *TensorFlow* is an interface for developing and executing machine learning algorithms, while *Keras* runs on top of *TensorFlow* to enable fast experimentation by providing simplicity and flexibility.

In the thesis context, we want to build models that fulfill two tasks: a retrieval task and a ranking task.

3.2.1 Hyperparameter and Architecture details

The networks are trained with the Adam Optimizer [30] (learning rate $\eta = 1e-03$) and take as input a concatenation of feature embeddings. The architecture details for each model are the following:

- **Retrieval model**
  The retrieval model is trained with learning rate $\eta = 1e-03$ and batch size $= 8192$. It is formed by a *query tower* and a *candidate tower*, where the former outputs a low-dimensional representation of the corresponding query and the latter outputs a low-dimensional representation of the corresponding item. Then, the outputs of the two towers are multiplied together to get a query-candidate similarity score. Figure 3.6 shows the retrieval model architecture and Table 3.11 shows the specific layers that each tower contains. The input of each tower is the concatenation of feature embeddings.

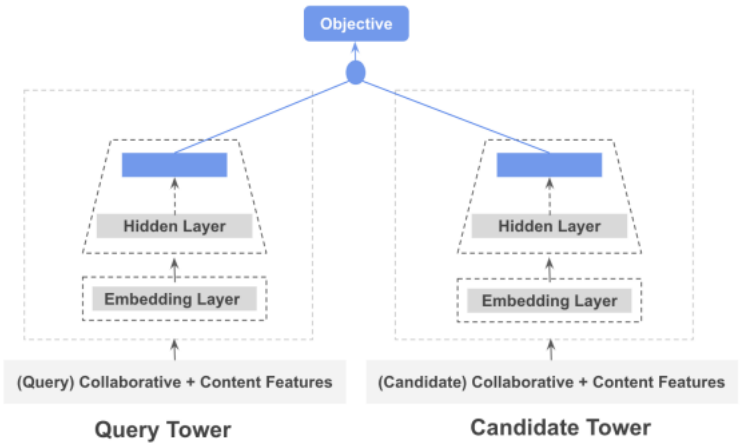


Figure 3.6: Two-tower retrieval model. Source: [25]

Query tower	**Candidate tower**
Input $e = (e_{id}, e_{lang}) \in \mathbb{R}^{d_e}$	Input $e = (e_{id}, e_{country}, e_{city}, e_{name}, e_{stars}) \in \mathbb{R}^{d_e}$
FC 64 ReLU	FC 64 ReLU
FC 32 linear	FC 32 linear

Table 3.11: Retrieval model architecture

- **Ranking model**
  The ranking model is trained with learning rate $\eta = 1e-03$ and batch size = 8192. As the retrieval model, it is formed by a *query tower* and a *candidate tower*, but then some fully connected layers take in query and candidate embeddings and output a predicted rating. Table 3.12 shows the architecture of these extra layers on top of the *query* and *candidate* towers.

Ranking layers
Input $z = (query\_emb, candidate\_emb) \in \mathbb{R}^{d_q+d_c}$
FC 256 ReLU
FC 128 ReLU
FC 1 linear

Table 3.12: Ranking layers architecture

- **Multitask model**
  The multitask model combines both retrieval and ranking model architectures (see Figure 2.7), with the assumption that training a model that optimizes both tasks at the same time can produce better results than training separate tasks. Therefore, *query* and *candidate* towers will have the architecture specified in Table 3.11, and the ranking layers will have the architecture specified in Table 3.12.

3.2.2 Procedure

The three proposed models will be trained with different settings for our experiments. These settings are the following:

- **Explicit feedback vs Implicit + Explicit feedback**
  We want to see which is the benefit of combining implicit and explicit feedback over only using explicit feedback. To do so, we will compare the performance when training our models on explicit feedback and when training our models on implicit data and then applying transfer learning to fine-tune the learned weights with explicit data.

- **No context vs adding context features**
  We want to see which is the benefit of training context-aware models, in our case, adding the *timestamp* feature to the query features. To do so, we will train each model with and without context features.

Finally, we will compare the performance of our models against a matrix factorization approach to evaluate the advantage of using state-of-the-art methods over a baseline method.

3.3 Metrics

Measuring the performance of a Recommender System is not an easy task since we can only compare recommendations to user-item interactions that already happened, which can be seen as "ground-truth" labels. Therefore, in order to measure the accuracy of our Recommender System, we use the following metrics:

- **Root Mean Squared Error**
  Root Mean Squared Error measures the average error magnitude, giving higher weight to large errors. In the thesis context, this metric is well-suited to measure the predicted rating for a user-item interaction.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y_j})^2} \tag{3.1}$$

- **Recall@K**
  Recall@K is a type of accuracy metric that measures the percentage of recommendations for which the targets (or items the user previously interacted with) are in the top K recommendations. To analyze how precise our model is, we will compute the recall at different levels:
  - Recall@1
  - Recall@5
  - Recall@10
  - Recall@50
  - Recall@100

3.4 Recommender System tools and technologies

The proposed recommendation system is formed by different components, each of them involving different technologies and tools. These requirements for each component are described in the following list:

- **Data storage**
  This is where users, items, and user-item interactions data is stored. In order to generate a data set to train or re-train a model, this data has to be processed. Following the current tools and technologies that *Stayforlong* works with, our data will be stored in an *Amazon S3* Data Lake and an *Amazon Redshift* data warehouse. The former is a type of NoSQL database that stores data in key-value pairs, while the latter uses SQL databases.

- **Machine Learning models**
  As described in section 3.2, machine learning models will be implemented in *Tensorflow* [28] and *Keras* [29] since they provide a simple interface, enabling fast experimentation.

- **Recommendations storage**
  Our system will compute the recommendations for each user in an off-line manner, therefore we need to model how these recommendations will be stored. Since efficiency is a must to serve recommendations and the most appropriate way to store these recommendations is using a key-value storage, they will be stored in a *Redis* database. *Redis* [31] is an in-memory key-value database, where each key will correspond to a unique user and its value will be the list of items to recommend.

- **Flow orchestration**
  All the different steps needed to compute recommendations must be executed in a specific order and with some scheduling. Therefore, *Airflow* is well suited to act as the recommendation system orchestrator.

4. Development of the system

In the thesis context, the objective is to design a proof of concept of an end-to-end recommendation system, from data gathering to serving recommendations to users. This proof of concept contains two main challenges:

1. Training a Machine Learning model to infer recommendations.
2. Building a Machine Learning pipeline for model re-training and computing off-line recommendations.

4.1 General System Description

In order to implement and serve the proposed Recommendation System, we need to define its workflow, as shown in Figure 4.1. This workflow is formed by the following steps:

- **Dataset generation**
  This step gathers all users, items, and user-item interaction data in order to update the data set that will be used to re-train our models.
- **Model re-training**
  This step is responsible for performing data preprocessing, and re-training or fine-tuning the retrieval and ranking models.
- **Computing recommendations**
  Once the machine learning models have been updated, we compute recommendations for every user. To compute recommendations, we first have a retrieval model that reduces the number of candidate items that are considered relevant for the user. Then, a ranking stage scores this reduced set of candidates according to the probability of the specific user liking them. Finally, a re-ranking model takes the highest scored items and ranks them according to any KPI of interest, and some post-processing is computed to filter-out possible items that do not fulfill specific conditions.
- **Saving recommendations**
  The last step takes the recommendations made for each user and stores them in *Redis*, so then they can be accessed and served by any service.

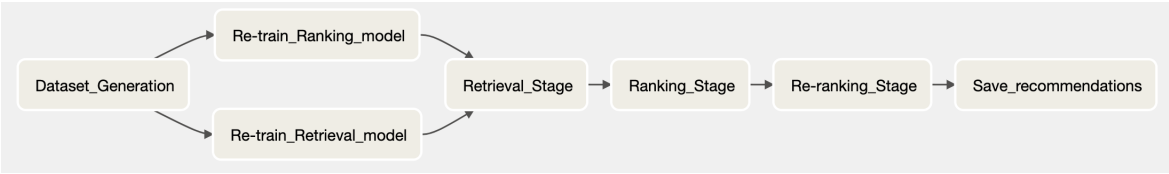


Figure 4.1: Recommendation System orchestration flow.

4.2 Dataset generation

In order to generate our dataset, we need to gather data from three different sources: explicit feedback, implicit feedback, and a feature store (see Figure 4.2).

- **Feature store**
  The feature store contains the set of features that describe each user and each item, which are described in section 3.1.
- **Explicit feedback**
  Explicit feedback is collected from user purchase history. This is the most reliable type of feedback since a user purchasing an item is a strong indicator that they like it. Nevertheless, this type of feedback is scarce, since users that have only purchased one item are the most common ones.
- **Implicit feedback**
  Implicit feedback is collected from click and view events on *Stayforlong*'s website. A user viewing or clicking on an item does not really mean that they like the item, so this type of feedback is less reliable because lots of assumptions are made. However, this type of feedback is abundant since users typically interact with multiple items before purchasing one.

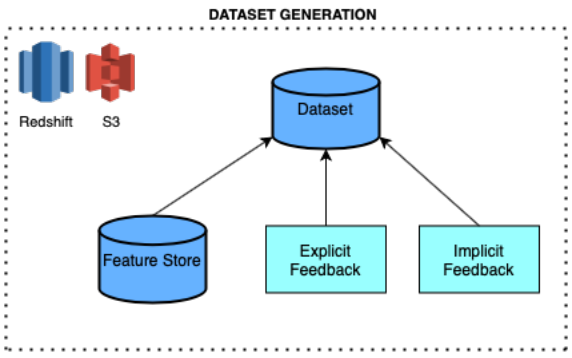


Figure 4.2: Dataset generation description.

4.3 Data pre-processing

Once the dataset is generated, we need to perform some data pre-processing so we can use it as input for the machine learning models. All features, see Table 3.10, can be treated as categorical variables. Therefore, we will use embeddings for each variable, transforming them from a categorical value into a low-dimensional embedding vector. To be able to represent categorical features in an embedding vector, we first need to map raw values to an integer in a continuous range and then perform the embedding. We decide the size of each embedding following the rule-of-thumb [32] that says that the embedding size should be the fourth root of the number of unique values in the variable. The following list describes how each feature is pre-processed:

- **User features**
  - *user_id*: There are a total of 435072 users, so user identifiers will be represented in an embedding space of 26 dimensions.
  - *user_language*: There are a total of 37 languages, so user languages will be represented in an embedding space of 3 dimensions.
- **Item features**
  - *item_id*: There are a total of 85380 items, so item identifiers will be represented in an embedding space of 18 dimensions.
  - *item_country*: There are a total of 188 countries, so item countries will be represented in an embedding space of 4 dimensions.
  - *item_city_id*: There are a total of 15662 cities, so item cities will be represented in an embedding space of 12 dimensions.
  - *item_name*: To process this text feature, we first tokenize the text, obtaining a sequence of tokens for each text. Then, each token is represented in an embedding space of dimension 32. Finally, since we want to obtain a single representation for every text, we average the representations of every token to get the text representation.
  - *item_stars*: Although *item_stars* is a numerical variable, each value represents a category, so it also can be seen as a categorical variable. Therefore, we will treat it as such. There are a total of 6 star categories, so item stars will be represented in an embedding space of 2 dimensions.
- **Interaction features**
  - *rating*: Since this variable is binary, we do not need to process it.
  - *timestamp*: Although *timestamp* can be seen as a continuous numerical feature, we will apply discretization techniques to it. Therefore, we take the maximum and minimum *timestamp* values and divide its values into 1000 buckets. Then, since there are now 1000 unique values for the timestamp category, we will represent it in an embedding space of 6 dimensions.

4.4 Machine Learning Models

As explained in section 3.2, the recommendation system will be formed by machine learning models that are able to perform the retrieval and ranking tasks. Therefore, for every user, we will retrieve a subset of relevant candidates and then rank them according to the probability of the user liking them. Moreover, these models will need to be re-trained on a regular basis. To do so, we can take advantage of previous models and use transfer learning techniques.

4.5 Data post-processing

After computing recommendations for each user, we need to check that every item recommended is a valid item or that we are interested in recommending every item. Therefore, some filters based on item features might be applied. For instance, at some moment we might be interested in only recommending hotels above 2 stars-rating. In addition, since we are recommending hotels, it might happen that they do not have available rooms at the time of recommendation, so only items (hotels) that have available reservations will be recommended.

5. Experimental Evaluation

In this section, the results obtained from experiments are presented. Then, we will compare the results obtained when adding or ignoring context features, in our case only *timestamp*, we will see the effects of training with implicit feedback and applying transfer learning, and we will compare the difference in performance when training the retrieval and training tasks separately and jointly. Finally we will compare our results with a matrix factorization baseline approach.

5.1 Results

Table 5.1 shows the RMSE and loss obtained when evaluating the ranking model in the different experiment settings. Table 5.2 shows the Recall@K categorical accuracy and loss obtained on the validation set for the retrieval model in the different settings, and finally, Table 5.3 shows the Recall@K, RMSE, and loss obtained on the validation set for the multitask model.

	Explicit feedback only		**Transfer learning**	
Metric	**No-context**	**Context-aware**	**No-context**	**Context-aware**
RMSE	0.2023	0.0859	0.0122	0.0038
Loss	0.0416	0.0073	$1.59 * 10^{-4}$	$1.4 * 10^{-5}$

Table 5.1: Ranking performance comparison between experiments.

	Explicit feedback only		**Transfer learning**	
Metric	**No-context**	**Context-aware**	**No-context**	**Context-aware**
Recall@1	$1.55 * 10^{-4}$	$3.47 * 10^{-4}$	$2.32 * 10^{-4}$	0.0133
Recall@5	0.0022	0.0051	0.0012	0.071
Recall@10	0.0048	0.0101	0.0026	0.1265
Recall@50	0.0186	0.0344	0.0196	0.3053
Recall@100	0.0336	0.0545	0.0404	0.3962
Loss	3980.88	16215.09	18785.98	18708.24

Table 5.2: Retrieval performance comparison between experiments.

	Explicit feedback only		Transfer learning	
Metric	**No-context**	**Context-aware**	**No-context**	**Context-aware**
RMSE	0.0431	0.0413	0.0286	0.0511
Recall@1	$5.48 * 10^{-5}$	$6.44 * 10^{-5}$	$3.1 * 10^{-4}$	0.1144
Recall@5	$9.6 * 10^{-4}$	$3.48 * 10^{-4}$	0.0012	0.3391
Recall@10	0.0017	$5 * 10^{-4}$	0.0028	0.4129
Recall@50	0.0074	0.0026	0.0183	0.5723
Recall@100	0.0135	0.0055	0.0393	0.6402
Loss	27101.98	32980.58	18784.52	17195.15

Table 5.3: Multitask performance comparison between experiments.

Table 5.4 shows the performance obtained with a matrix factorization approach.

Metric	**Value**
Recall@1	0.001
Recall@5	0.0135
Recall@10	0.0307
Recall@50	0.1273
Recall@100	0.2
Loss	18200

Table 5.4: Matrix factorization performance.

5.2 Discussion of results

After performing multiple experiments, we can analyze the results obtained from different points of view:

- **How does transferring learned features from implicit feedback improve the performance over training models directly on explicit feedback?**
  When looking at Table 5.1, Table 5.2 and Table 5.3, there is not much difference in performance when training models on explicit feedback or training them on implicit feedback and fine-tuning them. However, there is one case that seems to greatly benefit from applying transfer learning, and that is when combining transfer learning and context features, especially for the retrieval task.

- **How does adding context features help with retrieval and ranking tasks?**
  To answer this question, we trained each model with and without the *timestamp* feature. Table 5.1, Table 5.2 and Table 5.3 show that all models are benefited from adding context

features when also performing transfer learning and fine-tuning. Specifically, the retrieval task shows a significant improvement, where the recall@100 increased from 4% to almost 40% when adding context features, or from 3.9% to 64% recall@100 for the multitask model. Nevertheless, adding context does not seem to have such a significant impact when only training our models on explicit feedback.

- **How does building a joint model improve the performance of building separate models?**
  If we look at Table 5.2 and Table 5.3 metrics of models without context features or models trained on explicit feedback, we do not see any improvement when training a multitask model. In fact, both retrieval and ranking tasks seem to have a better performance than the multitask model. However, when adding context features and using transfer learning, the multitask model outperforms the retrieval one, achieving a 64% recall@100 against a 40% recall@100.

- **What is the improvement of using state-of-the-art methods instead of a baseline method?**
  If we compare the results of our baseline method, see Table 5.4, against the results obtained with our other approaches, we can see that the baseline method obtains a solid performance. Our state-of-the-art approaches only perform better than the baseline method when combining transfer learning and context features. In these cases, state-of-the-art approaches outperform the baseline method performance, achieving a 64% recall@100 against a 20% recall@100.

All in all, our experiments show that training context-aware models and applying transfer learning on implicit feedback can significantly improve the model performance, especially for the retrieval task. In addition to adding context features and transfer learning, training a model that optimizes both retrieval and rating tasks shows an increase in performance with respect to training two separate models. When comparing these results against our baseline method, we see that using transfer learning and context features is the only case where state-of-the-art approaches outperform our baseline method.

6. Conclusion and future development

This master thesis has explored with a practical approach the feasibility of developing and implementing a recommendation system in *Stayforlong*. The first challenge consisted of gathering data and modeling how to integrate this data to generate a data set that represents user-item interactions. When analyzing this data set,we have seen that the set of features was limited but we still could use a diverse set of features: item, user, and context features. This allowed us to experiment with hybrid methods, which address most recommender systems pitfalls, such as the cold-start problem. Within hybrid approaches, we experimented with current state-of-the-art deep learning methods, such as context-aware or multitask models. We also address another common pitfall in recommendation systems, which is the lack of explicit feedback. We demonstrate that by first training a recommendation system model on implicit feedback and then applying transfer learning and fine-tuning on explicit feedback, the models' performance can be improved, leveraging the knowledge learned with the abundant implicit data. In addition, we demonstrate that combining transfer learning with context-aware models significantly improves the performance of a model, outperforming our matrix factorization baseline method. We also explored the effect of training a multitask model, that optimizes retrieval and ranking tasks at the same time, showing that it outperforms the performance of training two separate models for each task when adding context features and transfer learning. To sum up, we can conclude that deep learning models excel in rich data scenarios, and that is the key to good performance when combining a hybrid multitask model with context features and transfer learning.

6.1 Future work

After analyzing the data set, we realized that user features were limited since we can only describe a user by its language. Therefore, some future work could focus on gathering more user features such as age, number of purchases, visited countries, and so on.

With regard to the machine learning models, we experimented with two-tower models but some future work could also experiment with other types of architectures such as deep & cross networks [22] or wide & deep networks [23].

Lastly, the next step is to implement the designed recommendation system flow. This is no easy task since this flow must accomplish efficiency, scalability, and maintainability. Also, we could use more measures such as novelty or coverage to evaluate if the recommender system is generating beneficial

recommendations, or in other words, to evaluate its effectiveness.

Bibliography

[1] Ian MacKenzie, Chris Meyer, and Steve Noble. *How retailers can keep up with consumers*. 2013. URL: https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers.

[2] Bernard Marr. *The Top 10 Technology Trends in Retail: How Tech Will Transform Shopping in 2020*. 2019. URL: https://www.forbes.com/sites/bernardmarr/2019/11/25/the-top-10-technology-trends-in-retail-how-tech-will-transform-shopping-in-2020/?sh=4be6b3ec4e03.

[3] Sarai Mizrachi and Pavel Levin. "Combining Context Features in Sequence-Aware Recommender Systems." In: *RecSys (Late-Breaking Results)*. 2019, pp. 11–15.

[4] Lucas Bernardi et al. "Recommending Accommodation Filters with Online Learning." In: *ORSUM@ RecSys*. 2020.

[5] Mihajlo Grbovic and Haibin Cheng. "Real-time personalization using embeddings for search ranking at airbnb". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 311–320.

[6] Malay Haldar et al. "Applying deep learning to Airbnb search". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 1927–1935.

[7] Soanpet Sree Lakshmi and T Adi Lakshmi. "Recommendation systems: Issues and challenges". In: *International Journal of Computer Science and Information Technologies* 5.4 (2014), pp. 5771–5772.

[8] Jie Lu et al. "Recommender system application developments: A survey". In: *Decision Support Systems* 74 (2015), pp. 12–32. ISSN: 0167-9236. DOI: https://doi.org/10.1016/j.dss.2015.03.008. URL: https://www.sciencedirect.com/science/article/pii/S0167923615000627.

[9] Thomas M. Cover and Peter E. Hart. "Nearest neighbor pattern classification". In: *IEEE Trans. Inf. Theory* 13 (1967), pp. 21–27.

[10] Piotr Indyk and Rajeev Motwani. "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. STOC '98. Dallas, Texas, USA: Association for Computing Machinery, 1998, pp. 604–613. ISBN: 0897919629. DOI: 10.1145/276698.276876. URL: https://doi.org/10.1145/276698.276876.

[11] Yehuda Koren, Robert M. Bell, and Chris Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42 (2009).

[12] *The Netflix Prize*. https://web.archive.org/web/20090924184639/http://www.netflixprize.com/community/viewtopic.php?id=1537. Accessed: 2022-05-22.

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[14] Gediminas Adomavicius and Alexander Tuzhilin. "Context-aware recommender systems". In: *Recommender systems handbook*. Springer, 2011, pp. 217–253.

[15] Ayush Singhal, Pradeep Sinha, and Rakesh Pant. "Use of deep learning in modern recommendation system: A summary of recent works". In: *arXiv preprint arXiv:1712.07525* (2017).

[16] Donghyun Kim et al. "Convolutional matrix factorization for document context-aware recommendation". In: *Proceedings of the 10th ACM conference on recommender systems*. 2016, pp. 233–240.

[17] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. "Deep content-based music recommendation". In: *Advances in neural information processing systems* 26 (2013).

[18] Balázs Hidasi et al. "Session-based recommendations with recurrent neural networks". In: *arXiv preprint arXiv:1511.06939* (2015).

[19] Veronika Bogina and Tsvi Kuflik. "Incorporating Dwell Time in Session-Based Recommendations with Recurrent Neural Networks." In: *RecTemp@ RecSys*. 2017, pp. 57–59.

[20] Hassan Ramchoun et al. "Multilayer perceptron: Architecture optimization and training". In: (2016).

[21] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

[22] Ruoxi Wang et al. "DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems". In: *Proceedings of the Web Conference 2021*. 2021, pp. 1785–1797.

[23] Heng-Tze Cheng et al. *Wide amp; Deep Learning for Recommender Systems*. 2016. DOI: 10.48550/ARXIV.1606.07792. URL: https://arxiv.org/abs/1606.07792.

[24] SHEN Xin et al. "ATNN: Adversarial Two-Tower Neural Network for New Item's Popularity Prediction in E-commerce". In: *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE. 2021, pp. 2499–2510.

[25] Ruoxi Wang et al. “Improving Relevance Prediction with Transfer Learning in Large-scale Retrieval Systems”. In: *Proceedings of the 1st Adaptive & Multitask Learning Workshop*. 2019.

[26] Paul Covington, Jay Adams, and Emre Sargin. “Deep neural networks for youtube recommendations”. In: *Proceedings of the 10th ACM conference on recommender systems*. 2016, pp. 191–198.

[27] Maciej Kula et al. *TensorFlow Recommenders*. 2020. URL: `https://github.com/tensorflow/recommenders`.

[28] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[29] Francois Chollet et al. *Keras*. 2015. URL: `https://github.com/keras-team/keras`.

[30] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[31] Salvatore Sanfilippo. *Redis In-memory Data Structure Server*. 2009. URL: `https://redis.io`.

[32] Lak V Lakshmanan, Michael Munn, and Sara Robinson. *Machine Learning Design Patterns*. 2020.