POLITECNICO DI TORINO
Master degree course in Computer Engineering

Master Degree Thesis

Predict your accommodation: a user centered hybrid recommender system

Supervisor
prof. Maurizio Morisio

Candidate
Andrea Fiandro
Student ID: 246360

Internship Tutor
dott. ing. phd. Giuseppe Rizzo

Academic Year 2019-2020

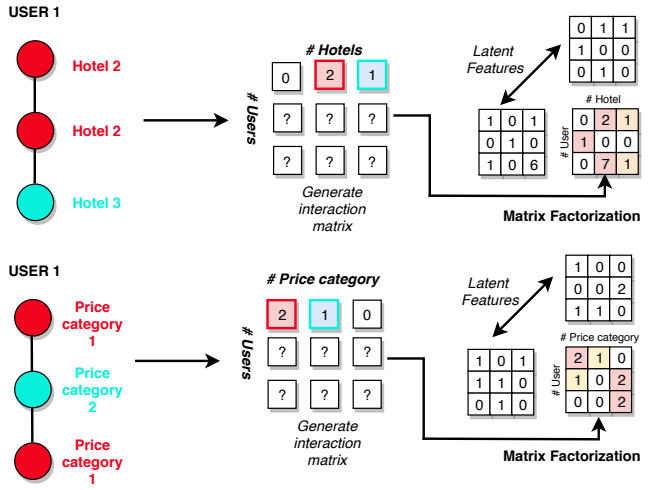
Summary

This work describes how our team faced the task presented in the RecSys Challenge 2019, a competition that this year was hosted by Trivago. The task proposed in the challenge is to understand how a traveler that is looking for an accommodation on the Trivago website is making his choice. To do so it is vital to analyze all the features of the dataset and after an accurate preliminary study we choose to follow two paths:

- A Matrix Factorization algorithm that ideally would detect the tastes of user in terms of the hotels
- A RNN approach that ideally would focus on the sequence of action that leads to the choice

The intuition behind this approach is to use the Recurrent Neural Network to attain the weakness of the Matrix Factorization that cannot take in account the sequence of actions. This thesis is focused on the Matrix Factorization part of the algorithm, while the RNN solution is described in another work. The Matrix Factorization is implemented by means of the LightFM library. It takes as input a matrix that has on the rows the users and on the column all the hotel visited in the time interval. Starting from this input the algorithm calculates two other matrices carrying the latent feature of user and hotels, that are multiplied together to get the prediction for the pair. The most challenging problem of this task was the cold start. In fact, there are some situation where the user immediately choose the hotel, without doing any other action. To attain this issue we use the price category

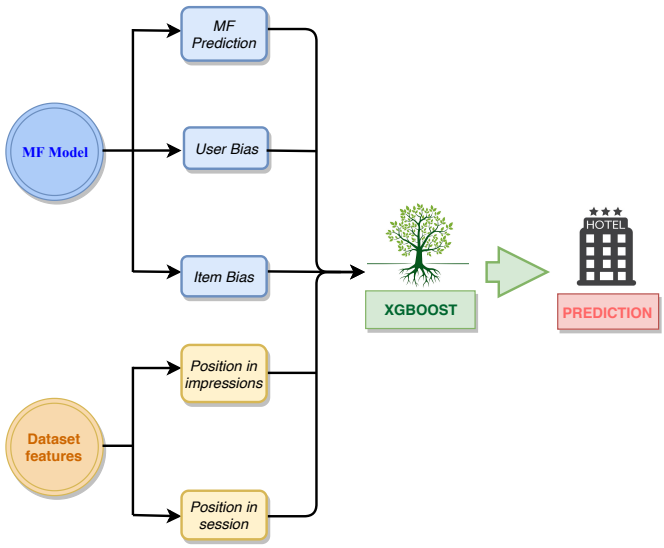
as an item feature, including it in the model by means of LightFM, as outlined in the following picture. This approach produces a score



of 0.52, evaluated by means of the Mean Reciprocal Rank, that is the official statistical evaluation metric of the challenge.

A further improvement is obtained by means of the Gradient Boosting techinque, useful to include two other dataset features: the position of the item in the list of hotels presented to the user (called *Impression lists*) and an index the identify how recent is the interaction between the user and the given item. With this setting the score become 0.61.

The whole algorithm pipeline is layed out in the following image:



We also included the features produced by two different configuration of the Recurrent Neural Network, but we tested it only locally because it was completed after the challenge deadline. In the best case this architecture gave us an additional boost of the MRR score of 0.05%.

Abstract

This master thesis work outlines the approach taken to solve the RecSys Challenge 2019, whose goal is to analyze the data of a user session to predict the hotel that will be clicked at the end. The first part of the thesis focuses on a deep understanding of the given problem, by means of an accurate analysis of the data to extract the most significant features for the prediction. The core of this work is the algorithmic solution, that focuses on a matrix factorization along with a gradient boosting technique. At first will be discussed the Matrix Factorization model alone, and then will be proven the effectiveness of the gradient boosting integration. This approach was effective for the purpose of the challenge, resulting in Mean Reciprocal Rank score of 0.62.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This master thesis work aims to give a thorough explaination of the solution that our team propose to solve the RecSys Challenge 2019. In this chapter will be described the team that collaborate jointly on the project and will be explained generally what is a *Recommender System*, to provide to the reader a better overall picture.

1.1 The team

This work was a joint effort between Politecnico di Torino and LINKS Foundation.

In particular this project was faced as a team with **Giorgio Crepaldi** that helps me a lot during his own master thesis work by exploring different relevant solutions for the task.

This work would not have been possible without the help of **Giuseppe Rizzo** from LINKS Foundation and **Diego Monti** from Politecnico di Torino. They shared with us their precious experience in the field and their deep knowledge about Recommender Systems.

It was also vital the supervision of prof. **Maurizio Morisio** from Politecnico di Torino that coordinates the whole process and allows

the team to access a lot of useful tools and structures.

1.2 Recommender Systems

The online commerce in these days is deeply changed. We have access to an overwhelming quantity of data that may provoke a sense of confusion in the customer. To solve this problem a lot of recent researches were focused on the field of Recommender Systems. A Recommender Systems provides a tool that can be used in a lot of context (e-commerce, music streaming, video streaming, advertising, travel, etc...) and is useful both for the customer, that receive more personalized content and avoid the feeling of being lost in the possibilities, but also for the retailer, that increase the probability of selling an item, by predicting the user tastes. This work focuses on the application of these technologies in the field of tourism [16], that has some unique characteristics, such as the shift of tastes based on the season.

Chapter 2

State of the art

In this chapter will be described the current state of research in the *Recommender systems* field. Will be defined a possible classification of the algorithm that are described in literature, with an accurate focus on the technologies we effectively used during the developement process.

2.1 Recommender systems classification

Nowadays Recommender System became one of the most practical application of the well studied machine learning techniques. They provide an useful tool both for retailer and for customer. The retailer can get a better understanding of what are the needs of the customer while the customer will not be lost in the overwhelming dimension of online stocks. The most straightforward application is in the context of e-commerce, but they are spreading almost everywhere. Analyzing the academic researches [14] on this topic is it possible to categorize the algorithms in two different categories:

- Sequence aware recommender systems
- Collaborative filters

Those two categories will provide a recommendation based on different input as outlined in Figure 2.1.

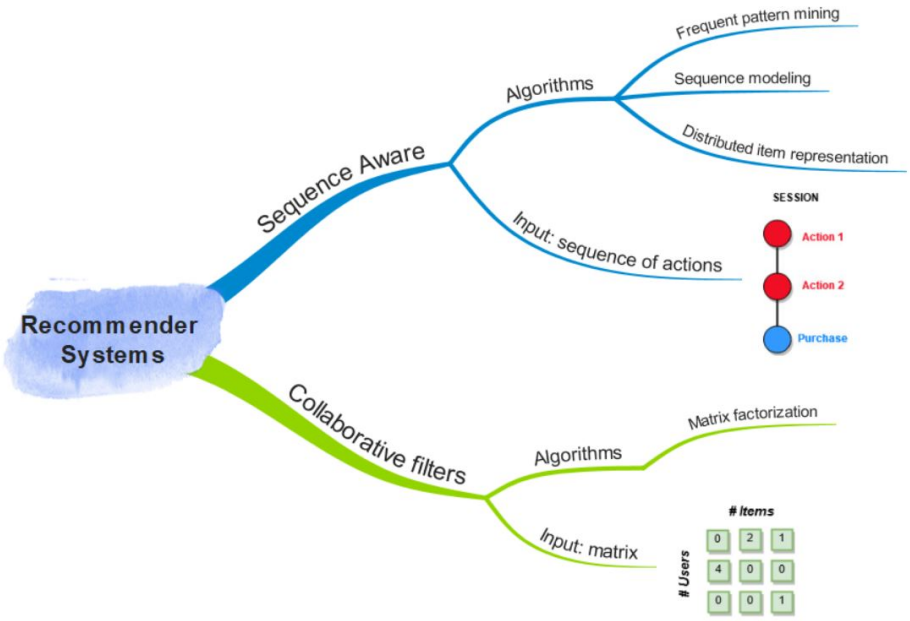


Figure 2.1. A possible classification for recommender systems

2.1.1 Sequence aware recommender systems

In some cases the input given to a specific model is a sequence of actions, performed by a user, in a specific session. This is a very common scenario, because often data are collected using the log files of an application (e.g. click on different items of an e-commerce website). In general this sequence of action gets more value if it brings other useful information, such as the type of action. The algorithms studied in literature in this kind of field can be divided in different categories.

Context adaptation

This category of algorithms takes in account some contextual information in order to get more details about the choice of the user. It is really difficult to understand with a certain grade of confidence if in a specific session there is an intent of buying a product or there is only a preliminary analysis. This kind of information may help to build a better user profile in case of sequence aware recommender systems. Some example of contextual factors are:

- Time of the day (Morning, Afternoon, Evening...)
- Geographical position
- Weather conditions
- Type of device (mobile/desktop)

Trend detection

An other important aspect to take in account in the context of recommender systems is the possibility to detect some behaviour shared by a group of individuals or a shift of interests in a single person. There are two classes of information that might be extracted:

- **Community or Group trends**: the most popular item or the favourite of a specific group of people should have a good relevance in the suggestion generated by the system
- **Individual trends**: the tastes of a consumer can change across the time. This is particularly interesting in the travel context: it is not so common that, after a customer has purchase a specific hotel, the next time he will look for an accommodation with the same characteristics.

Repeated recommendations

In some context it is useful to suggest again an item that has already been bought by the same user in previous sessions. For example, it is

possible to detect if there are recurrent purchasing of the same item in order to recommend it when it will be needed again.

2.1.2 Collaborative filters

This kind of algorithms are the most studied in literature because they are proven to be really effective in a lot of context. However they have some problem to understand the shift of tastes of a user. The main algorithm of this category is the **Matrix factorization** that will be described thoroughly in the next chapter because is one of the main part of the solution described in this master thesis.

2.2 Matrix Factorization

This section will give a thorough description of the Matrix Factorization algorithm because it is a very important part of the research proposed in this thesis.

2.2.1 What is the Matrix Factorization?

The Matrix Factorization is one of the most studied algorithm in literature concerning recommender systems [10]. It is part of the class of *Collaborative filters* that collects some procedure for understanding the long term preferences of users. This algorithm became really famous during the Netflix Prize Challenge [2] and since then was intensively used as a way of modeling the recommender system problems.

Strong point

This approach has a lot of virtues:

- It is an algorithm very well documented
- It is relatively simple in its basic implementation
- It is an effective way to discover the latent features between users and items

- Provides a good way to represent data in case of large datasets

Weakness

The Matrix Factorization has also some limitations. As a consequence, the majority of the solution provided in this kind of challenges uses a hybrid approach, to overcome these drawbacks.

- It does not handle the *Cold Start*, namely the problem that occurs where there are not enough data to make a prediction (e.g new registered user without a previous history)

- It doesn't recognize very well the shift of tastes of the user. This is a huge problem in the context of tourism because it is obvious that a person that has some exigency reserving an hotel, may have other desires in the next one.

2.2.2 Problem definition

The main issue of this approach is the problem definition: it is important to choose a computationally feasible encoding. The most common way to represent the problem is by defining a matrix composed in the following way:

- Each row represent a user

- Each column represent an item, in this case it is an accommodation

2.2.3 Feature extraction

The Matrix Factorization model learns the latent representations in a high dimensional space for users and items. When multiplied together, these representations produce a score for every item for a given user.

Formal explanation

Let us take into account a matrix of dimensions **n x m**. For example, we take a row of the user latent feature matrix

$$U_1 = (u_{f1}, u_{f2}, ..., u_{fn}) \tag{2.1}$$

and a column of the item latent feature matrix.

$$I_1 = \begin{pmatrix} i_{f1} \\ i_{f2} \\ ... \\ i_{fm} \end{pmatrix} \tag{2.2}$$

In order to get the score S_{11} for the pair (U_1, I_1), it is necessary to compute the dot product:

$$S_{11} = U_1 \cdot I_1 = (u_{f1}, u_{f2}, ..., u_{fn}) \cdot \begin{pmatrix} i_{f1} \\ i_{f2} \\ ... \\ i_{fm} \end{pmatrix} \tag{2.3}$$

2.2.4 Example

This section describes what does in practice the matrix factorization algorithm in a simple way. In order to get a better understanding of what are the latent features, they are associated with some real features of the hotel but we never get a real correspondence between a latent feature and a real one. In this example the latent features are associated with two common characteristics of accommodations: the presence of a SPA and a bike rental service.

User-Item interaction matrix

The input of the algorithm is the matrix described in the section 2.2.2 where an item is represented by an hotel. An example of this matrix, derived by the interaction between user and hotels is the following:

Starting from this input, the procedure calculates two other matrices, in an higher dimensional space, that can produce the input when

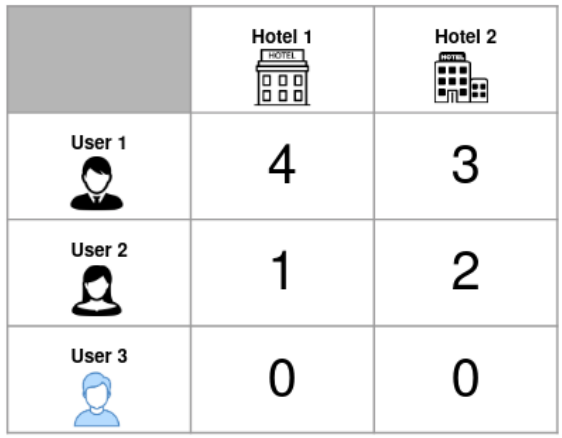


Figure 2.2. Matrix of interaction between user and item

multiplied together. In this way, the rows representing users in the left matrix, identify the tastes. When multiplied with a row referring to an item which have a lot of features required by the given user, the result will be an high compatibility.

2.3 Recurrent Neural Network

In the context of Sequence Based recommender systems often the matrix factorization algorithm decreases the performance. For this reason one of the most widely used system are the Recurrent Neural Networks. Deep neural networks are exploited with success in the field of image and speech recognition while in the Recommender Systems domains they have been introduced recently [7].

While in Matrix factorization the focus is on the interaction between user and item, with RNN we take in account the sequence of action performed in a single session. In particular we take as the first input, the first action done by the user, and so on. The output will be produced using the whole sequence of actions.

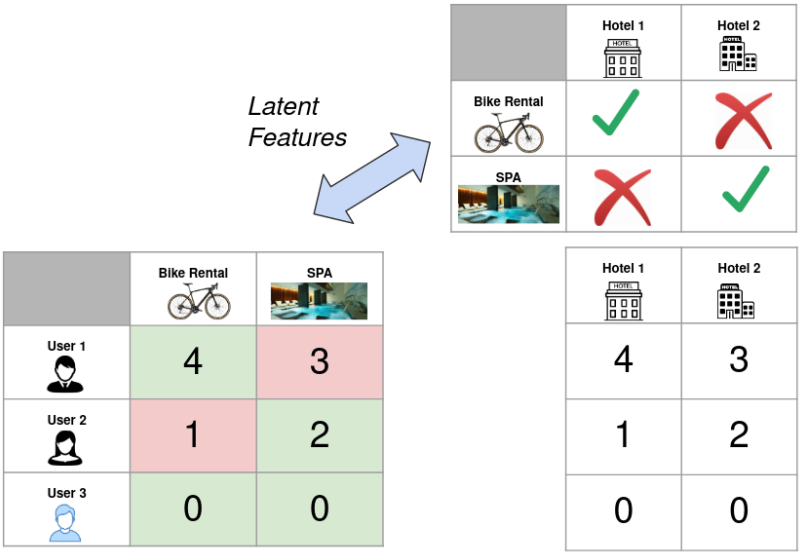


Figure 2.3. Example of matrix factorization

2.3.1 Simple RNN

The neural network [12] is able to take in account for each prediction the consecutive actions by means of the **Hidden Layer**, that is update each time we get a new input of the sequence, and carries the information about the previous steps.

A main idea of the overall structure is outlined in the Figure 2.4

2.3.2 Long Short Term Memory (LSTM)

One of the main objective of the Long Short Term Memory network [9] is to attain one of the problems of the simple RNN: the *Vanishing Gradient* [8]. LSTM may face this phenomenon during the back propagation phase when the sequence is very long. For this reason the gradient update become too small and it doesn't contribute to the learning phase. To solve this problem LSTM networks use some gates that can attain the loss of information during time:

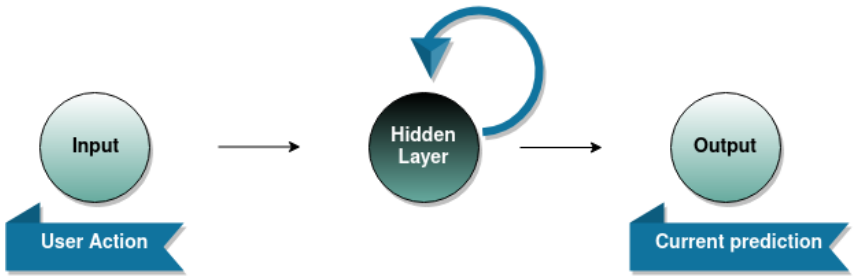


Figure 2.4. Simple RNN structure

- Forget gate
- Cell state
- Input gate
- Output gate

By means of those functions the network is able to understand which information is useful for the prediction and, on the other hand, what can be left out. A straightforward example of a possible application comes from the field of text recognition. In fact in long portion of text there are only few words that are useful for the semantic analysis and all the other words (such as conjuctions, preposition and articles) must be discarded.

2.3.3 Gated Recurrent Unit (GRU)

The Gated Recurrent Unit [4] is another kind of network created to overcome the problem of the vanishing gradient. It is a simplified version of the LSTM architecture that usually have better performances. The architecture has the following gates:

- Update gate: it has the same role of the input gate of the LSTM
- Reset gate: it defines how much information to forget

Chapter 3

Trivago 2019 Recsys Challenge

This chapter will provide a better understanding about how the Trivago website works and how it is useful for them to provide their customers a better recommendations. Then will be given more details about the challenge and how it is evaluated by the organizers.

3.1 Description of the challenge: the task

3.1.1 Trivago

This work aims to solve the problem defined in the RecSys Challenge 2019[1], hosted by Trivago. Trivago is a search platform for hotels that is used by travellers to compare different accomodation in order to choose the most suitable. This platform does not sell directly the accommodation, but it helps to compare offers provided by other booking platforms. In this business model the most important action is the so called **clickout** that is what has to be predicted.

[1] https://recsys.trivago.cloud/challenge/

Item clickout

When the traveller finds an hotel that might be good for him it performs the *clickout action* and it is redirected to a different site, such as booking.com, where it is possible to finalize the purchase. However Trivago is not directly the retailer, it has a huge interest in rising the number of partner's sites visited during a session because his income stream can be enhanced in this way. Indeed, this platform's main revenue is originated by advertising that usually takes in account two different metrics:

- Cost per click: an hotel owner pays a fee for each time a user from Trivago clicks on the sponsored hotel

- Cost per acquisition: the hotel owner pays a fee only if the sponsored hotel is effectively bought by the user from Trivago

By increasing the number of clickout is it possible to increment the revenue given by the cost per acquisition.

3.1.2 Use case: find a hotel

As explained before the objective of the challenge is to predict the hotel that will be clicked by the user at the end of his session. To do so, it is really important to analyze all the actions that lead the user to take the decision. The typical use case scenario of a user session on the Trivago website is illustrated in the Figure 3.1. In this example the user starts by searching the destination and the date of the travel. Then it will see a lot of different accommodations, sorted following the recommendations of Trivago. It can decides the most important features by means of the filter, to finally perform a clickout action and actually reserve the hotel.

Figure 3.1. Use case of Trivago website: example of a session

3.2 Evaluation metric: Mean Reciprocal Rank

The accuracy of the algorithm must be evaluated by using a well known statistic evaluation metric: the **Mean Reciprocal Rank (MRR)**. This method is really useful for evaluating the effectiveness of a list based output, that is the objective of the challenge. In fact, we have to provide a list of suggested accommodations for each clickout action and the MRR gives a full score (1) if the clicked hotel is in first position, half score if it is in the second one (1/2) and so on. In a more structured way:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank(i)}$$

This means that it is not important only to find the most suitable accommodation, but it is vital to consider also the goodness of the recommendations different from the first one.

Chapter 4

Dataset

The dataset of the challenge is provided directly by Trivago. It looks like a session log, which describes all the actions of a specific user that end with the choice of a specific accommodation:

	user_id	session_id	timestamp	step	action_type	reference	platform	city	device	current_filters	impressions	prices
10189209	ZVIDWWE0KWNB	e791605098 0d9	1541531675	1	search for destination	New York, USA	BR	New York, USA	desktop	NaN	NaN	NaN
10189210	ZVIDWWE0KWNB	e791605098 0d9	1541531872	2	search for item	149585	BR	New York, USA	desktop	NaN	NaN	NaN
10189211	ZVIDWWE0KWNB	e791605098 0d9	1541531973	3	interaction item deals	149585	BR	New York, USA	desktop	NaN	NaN	NaN
10189212	ZVIDWWE0KWNB	e791605098 0d9	1541532069	4	clickout item	8985292	BR	New York, USA	desktop	NaN	8640|8816|8791|2298566|1811137|8708|67445|8825...	215|173|188|258|199|133|309|243|211|178|178|18...

Figure 4.1. Example of the dataset provided by the challenge

It is already divided in **Training set** and **Test set**, according to the logic described by the organizer of the challenge. They choose to split the dataset according to a specific date, by keeping together the sessions, as pointed out in the Figure 4.2.

During the competition the evaluation of the solution is calculated (by means of the Mean Reciprocal Rank, described in Section 3.2) on the *Validation Set*. At the end of the challenge the final ranking is based on the result of the *Confirmation Set*.

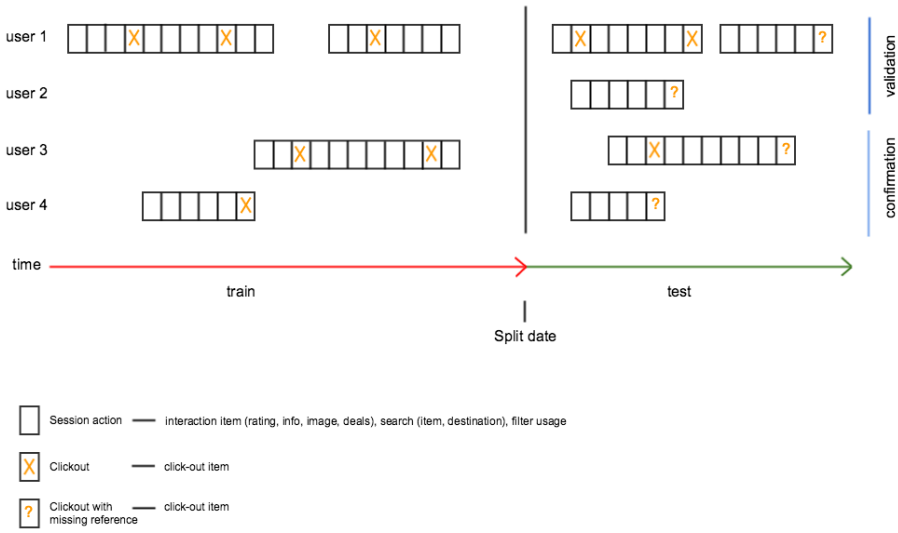


Figure 4.2. Division of the dataset provided by Trivago

4.1 Dataset features

The dataset provided by the organizers of the challenge is very large. In particular:

- Training set: 2.1 GB (80%)
- Test set: 530 MB (20%)

4.1.1 Training set

An initial overview of the training set can be seen in the Table 4.1.1

4.1.2 Test set

We performed the same kind of analysis on the test set. The result are detailed in Table 4.1.2

Number of rows	15.932.992
Number of columns	12
Number of different sessions	910.683
Number of clickout actions	1.586.586

Table 4.1. Training set statistics

Number of rows	3.782.335
Number of columns	12
Number of different sessions	291.381
Number of clickout actions	528.779

Table 4.2. Test set statistics

4.1.3 Dataset structure

The first important step that we took to have a better understanding of the problem was to perform an analysis of all the parameters provided. In this way we found out what affects the choice of the accommodations. The dataset provides different information:

- **User Id**: the identifier of the user looking for the accommodation
- **Session Id**: the identifier of the session
- **Timestamp**: Unix timestamp[1] of the action performed by the user
- **Step**: incremental number used to define the sequence of actions
- **Action Type**: describes the type of the action performed by the user, it can be of different types:
  - Clickout item

[1] https://en.wikipedia.org/wiki/Unix_time

  - Interaction item rating
  - Interaction item info
  - Interaction item image
  - Interaction item deals
  - Change of sort order
  - Filter selection
  - Search for item
  - Search for destination
  - Search for poi (Point of interest)

- **Reference**: this value has different meaning depending on the action type. For example it holds the ID of the clicked hotel in case of *clickout item*
- **Platform**: the country of the user searching for the accommondation
- **City**: the city where the user is searching for the hotel
- **Device**: the device used for searching (mobile/desktop)
- **Current Filters**: all the filters set by the user, separated by pipes
- **Impressions**: the list of all the accommodations, displayed to the user. One of them is the one that will be clicked
- **Prices**: the prices of all the hotels in the impressions, separated by pipes

4.2 Preliminary analysis

4.2.1 Dataset Statistics

We performed a preliminary analysis to have a better understanding of the problem. Not all the statistics calculated were useful for the

solution of the challenge, but it is important to include them because they leads some of the choices made by the team.

Number of session per user

It is useful to define if it is possible to have an history of the choices of the user. According to this idea we calculate the number of sessions for each user. In this context almost all users have only one session, as detailed in Figure 4.3.

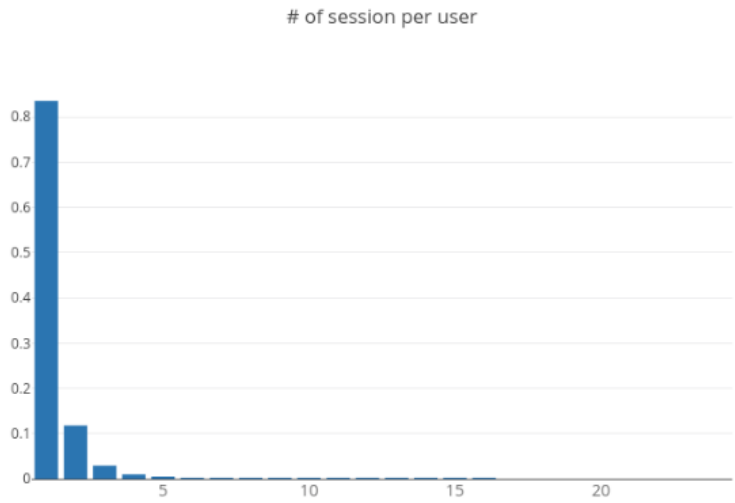


Figure 4.3. Average number of sessions for a single user

Potential differences between desktop and mobile

We analyzed the possibility that a user can be more inclined to perform a specific action on a mobile device rather than on desktop. The Figure 4.4 shows the difference for each action.

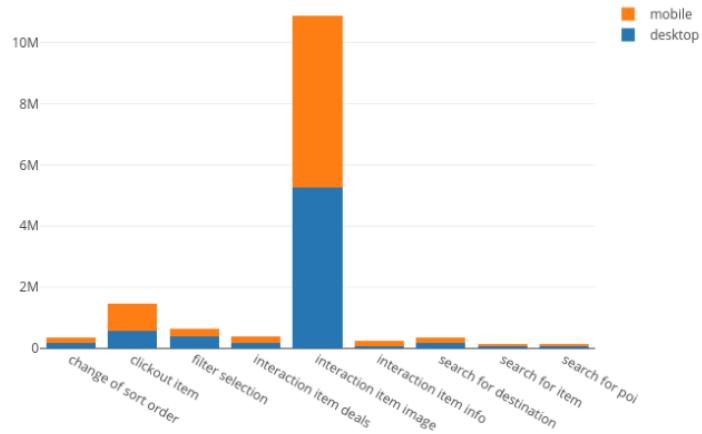


Figure 4.4. Differences between desktop and mobile interactions

Average session time	10 minutes
Max session time	6740 minutes
Min session time	1.2 seconds
Average number of actions	17.5
Max number of actions	3522
Min number of actions	1

Table 4.3. Statistics about the sessions

Statistics relative to sessions

The Table 4.3 sum up some statistics concerning the duration of a session, in order to extract the most common behaviour. As we can see the minimum number of actions in a session is 1. So we have some case where the user simply clicks an hotel without doing nothing relevant: this was one of the main difficulties we have to overcome through the challenge. The number of session with this characteristic is shown in

the Figure 4.5.

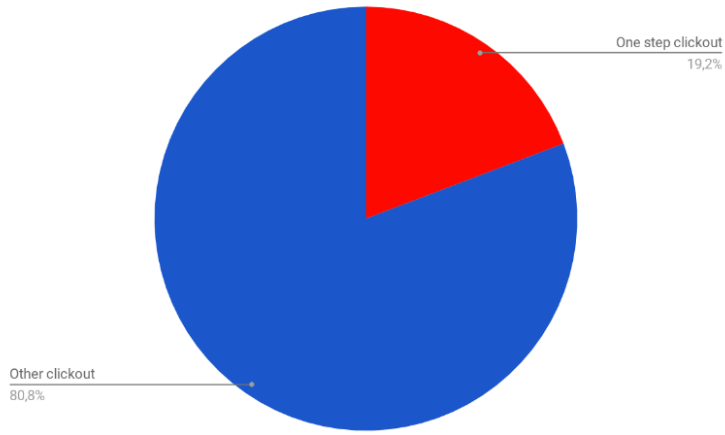


Figure 4.5. Number of single click action (calculated on the test set)

Chapter 5

Overall solution

The purpose of this section is to allow the reader to have a better understanding of the overall picture of the solution proposed for the RecSys Challenge 2019. In particular it will be explained what are the arguments that supports the choices made by the team and how they are reflected in terms of algorithms. There will be also some details of the problems faced during the challenge that force the team to take different and unexpected directions.

5.1 Team's choices

The main idea that leads the choices to the solution we get is to exploit as more dataset feature as possible, by trying various algorithm. During this challenge as a team, were developed different solution:

- Matrix Factorization
- Recurrent neural network

We choose those two procedures because they belongs to different categories of the recommender system classification described in the section 2.1 about the *State of the art* . For this reason, ideally, they would cover different aspects of the data we have:

- Recurrent Neural network -> Sequentiality of the actions

- Matrix Factorization -> User preferences

In this work the focus will be on the Matrix Factorization part of the algorithm.

5.2 Algorithm description

5.2.1 Input matrix

The first challenge was to find a good model for the algorithm because the input data, as described in the Section 4.1.3, is more suitable for a *Sequence aware recommender system* than for a *Collaborative filter* like the Matrix Factorization because it is a log of the user actions.

Interaction matrix

We decide to create the input matrix by using only a specific set of the action described in the dataset provided by the organizer of the challenge. In particular, we select the actions that can be associated, uniquely, with a pair user-item. Those action are:

- Interaction item rating
- Interaction item info
- Clickout item
- Interaction item image
- Interaction item deal
- Search for item

So we used this actions to populate the cells of a traditional user-item matrix. The first attempt use the same weight for each action, but is a parameters that has been decided after a tuning, more details will be provided in the Section 6.2.2.

Algorithm input data

In the Matrix Factorization model, we consider two interaction matrices:

- User - Hotel
- User - Price Category

The first matrix is extensively used in literature. The second matrix uses the same interaction actions, but it takes into account only the price category of the hotel selected by the user. This kind of approach was used only in the cold start scenario, that is one of the weakness of collaborative filtering.

In this way we can predict a score for the pair user-hotel also in the situations where the hotel was not previously seen by the user.

Hotels have not a fixed price but it may fluctuate slightly during time. To define categories we calculate for each hotel the average price during the span of time of the dataset. Then we use a simple formula to define the range of each category:

$$R_{category} = \frac{P_{MAX} - P_{MIN}}{N_{categories}} \tag{5.1}$$

Where:

- $R_{category}$ is the range of each category
- P_{MAX} is the max price across all the hotels
- P_{MIN} is the min price across all the hotels
- $N_{categories}$ is the number of categories defined, it is a parameter of the algorithm and can be defined at runtime

The Figure 5.1 sum up how we passed from the sequence of actions of the dataset to the matrix described before.

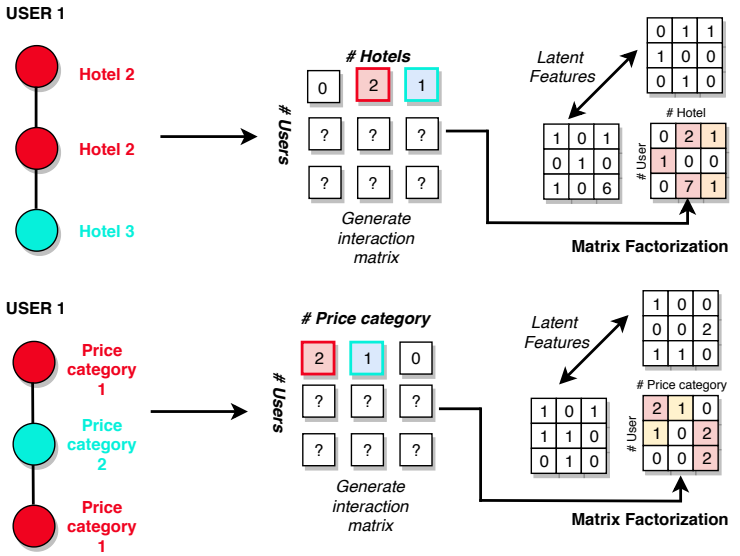


Figure 5.1. Generation of MF model, starting from a sequence

5.2.2 Gradient boosting: including dataset features

Gradient boosting is a machine learning technique used to generate a better outcome, given more than one weak classifier. In our approach, it was useful for adding the contribution of two particular features of the dataset:

- **The position of the hotel in the impression list**. The impression list is an information provided in the dataset and represents the order of the hotel presented to the customer, as seen in the Figure 5.2

  This is vital for giving good suggestions to the users. We found out that in a good amount of cases the customer tends to click on one of the hotel presented in the first positions.

- **The position in user's session**: during a session, the customer

will interact with some different hotel. We defined a number that tell how recent is the interaction between a user and an hotel (e.g. 1 for the most recent action, N for the least recent).

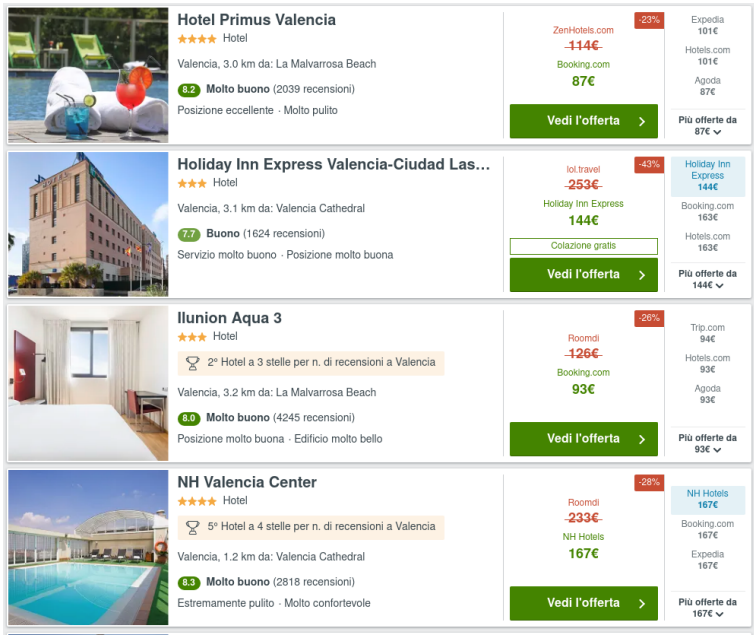


Figure 5.2. Impression list shown on Trivago website

5.2.3 The whole algorithm

A full picture of the algorithm developed is outlined in Figure 5.3.

The decision about the list of recommendations are not provided directly by the Matrix Factorization algorithm. This model produce only some scores, related to the user-hotel pairs, that contributes to the final solution:

- MF Prediction: a score that express the affinity between a user and a specific hotel

- User Bias
- Item Bias

The two biases are features provided by the matrix factorization model that represents a proven way to improve the algorithm and to detect some recurrent behaviours of users that may affect the decision. A typical example is a user that give an high score to the majority of reviews. In this case is very useful to keep track of the difference between the current rating and the average rating of the user, instead of the value itself.

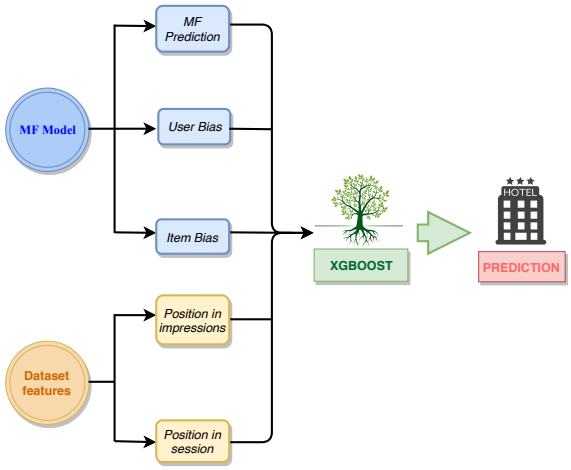


Figure 5.3. Matrix Factorization + Gradient Boosting ensemble

Chapter 6

Matrix Factorization

In this chapter will be described one of the core parts of the algorithm: the Matrix Factorization. In particular the focus will be on the technical implementation, with a wide description of the possibilities given by the library we chose. Finally there will be a discussion about the parameters that performs well on or dataset and the final result.

6.1 LightFM

6.1.1 Introduction

From a technical point of view the matrix factorization algorithm was implemented exploiting a library called **LightFM** [11]. This library provide an efficient implementation of the algorithm that we want to apply to the dataset provided by Trivago.

This library was developed and maintained by Lyst, a company that works in the field of fashion. In this specific domain is very useful to provide to the customers a good recommendations because the catalogue is huge and often they had to face the problem of cold-start. This happens also from the item point of view: every season the catalogue is updated with new dresses that don't have any interaction

with users but must be taken in account because in the fashion context the new collection has always a relevant value.

For these reasons, the most interesting feature of this package is that it provides a good solution in the cold start scenario.

6.1.2 Cold Start

It is called cold start scenario when we have very few data or no data to make the prediction. This is one of the main weaknesses of the matrix factorization model, for more details refers to Section 2.2.1, and it is better to use a content based algorithm, where all the items are represented through features. The goal of this model is to:

- Having a score comparable to the classical matrix factorization model in case of good amount of data
- Having a score comparable to the common content based method in case of data sparsity

To obtain this kind of result they used an hybrid model. User and items are represented as latent vectors, like a matrix factorization model, but those vectors are not a single number, but a function of the item feature or user features.

For example we can take the hotel *NH Valencia Center* from the Trivago website, shown in the Figure 6.1

In LightFM model, this hotel is defined by the sum of the latent representation of his features:

$$I_{rep} = wifi_{enc} + pool_{enc} + spa_{enc} + ... + gym_{enc} \tag{6.1}$$

Problem related to our dataset

The first attempt was to use the informations summarized before to define hotel features. Between the file provided by the organizer of the challenge there is also a file called *item metadata* that contains those data about the accommodations.

The problem of this dataset is that is strictly related to the filter option, for this reason it contains the hotel features just in case they have been filtered by the user in the time span of the dataset (almost 2 days). Thus, the hotel metadata given by that file are really fragmentary to fullfil the requirements of the LightFM model.

We noticed that the algorithm, structured with those data, tends to prefers hotels that have the features with respect to the ones without them. In this case the score was comparable to the random, so we abandoned that metadata file.

Price category

The solution of the problem described before is to use another type of data to define item features. As Trivago is a website that is widely used to compare prices in order to find the cheapest one, we decided to use the price. This feature is defined for each hotel in the dataset so we avoid the problem of having some missing data. The only drawback is that the price of the hotel is not fixed during the time but may vary a little, so we generate our own *item metadata* file, that includes the datas described in Table 6.1.2.

Hotel Id
Average price
Max price
Min price

Table 6.1. Columns of the custom metadata file

The algorithm doesn't consider the price itself, but we divided the total range of price in different categories, that are encoded. For more details about the price categories, refer to Section 5.2.1.

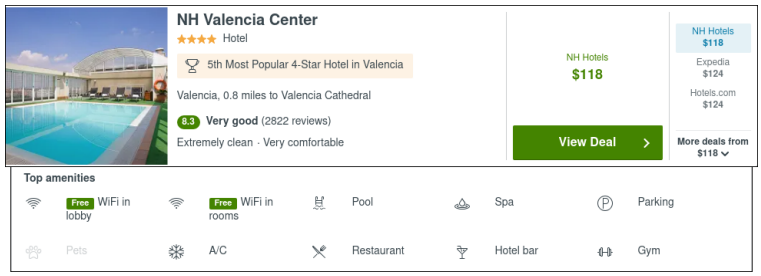


Figure 6.1. An hotel along with the feature that can described it in the LightFM model

6.2 Parameter tuning

6.2.1 Parameters description

The LightFM library allows the developers to change many parameters in order to get a more effective prediction. In this section it will be described each parameter, while in the section 6.2.2 there is a more detailed explaination of the parameters that gave us the best performance, according to the Trivago evaluation method.

Loss functions

The method used to learn the embedding is the *stochastic gradient descent* algorithm [6].

The loss function can be chosen between four different options.

- **Logistic**: this loss function is used when in the model are present positive and negative interactions.
- **Bayesian personalised ranking (BPR)**: this function is a pairwise loss [15]. The peculiarity of this approach is that it doesn't optimize the score given by the a single pair user-item. However, it takes *Item pairs* and performs the optimization over the rank calculated on multiple user-item pairs. This is a Bayesian approach, so we have the following likelihood function that has

to be maximized

$$p(\Theta| >_u) \propto p(>_u |\Theta)p(\Theta) \tag{6.2}$$

Where Θ represent the parameters vector of the model, in our case, the Matrix Factorization, and $>_u$ represents the "desired but latent preference structure for the user u". If we presume that all users make their choices independently from each others, it is possible to reformulate the equation 6.2 as the probability that a user prefers an item i over an item j:

$$p(i >_u j|\Theta) := \sigma(\hat{x}_{uij}(\Theta)) \tag{6.3}$$

where σ is the *logistic sigmoid*:

$$\sigma(x) := \frac{1}{1 + e^{-x}} \tag{6.4}$$

- **Weighted Approximate-Rank Pairwise (WARP)**: this is the most particular loss function implemented by LightFM and it was used for the first time for images [18].
  The empirical evidence is that it generally outperforms the more popular BPR, explained before.

  The WARP starts with an approach similar to the BPR model, it uses triplets (user, positive item, negative item). The difference is that in this case, negative samples are not chosen randomly, but they are selected from a pool of negative items which would violate the desired item ranking. The algorithm is described well in the LightFM documentation [1] and it involves two steps:

  – Take a pair (user, positive item)
  – Sample a negative item for that pair
  – Estimate the prediction for the *positive* item

[1] https://lyst.github.io/lightfm/docs/examples/warp_loss.html

  - Estimate the prediction for the *negative* item.
  - If the negative item's prediction is better than the positive one -> updates the gradient
  - It the positive item's prediction is better than the negative one -> sample another negative item, until a violation is found.

  Notice that if the algorithm has to perform a gradient update in the first steps, that update will be larger, otherwise it means that the model is close to the optimal solution.

- **k-OS WARP**: is a variant of the WARP loss [19]. It bases the pairwise update on the k-th positive item for any given user.

Learning rate schedules

In order to get better performances it has to be optimized the learning rate. This is a very common issue, shared between the majority of machine learning techniques, so it is deeply studied [5]. To solve this problem we use the so called *Learning rate schedules* that simplify this tuning phase when there are a lot of dimensions. LightFM library allows the developer to choose between two possible learning rate schedules.

- adagrad
- adadelta

List of parameters

- **Number of component**: is the number of dimension of the latent space where the features are represented
- **K**: this parameter is useful only with the k-os WARP loss function (more details in Section 6.2.1). It represents the k-th positive example that will be selected from the n positive examples

- **N**: it is again related to the k-os WARP and is the number of positive sample generated for each update
- **Loss**: the loss function chosen
- **Learning rate**: is the initial learning rate for the *adagrad* schedule
- **Rho**: moving average coefficient used in *adadelta* schedule
- **Item alpha**: is the L2 penalty applied on item features
- **User alpha**: is the same penalty, but for users
- **Max Sampled**: this parameters is related to the *WARP loss function* an represent the maximum number of negative samples that can be generated during the WARP fitting.
- **Random state**: is the random seed

6.2.2 Parameters tuning

In this section we report the result given by the MF without the gradient boosting, that will be explained.

General parameters overview

However there are a lot of other parameters (described in Section 6.2.1 that we tested a lot, for the sake of simplicity, we will focus on the parameters that we considered important. More details about the experiments are documented in Section 6.2.2

- Epochs: 150, 200, 300
- Number of components: 200, 300
- Learning rate: 0.01, 0.1, 0.2
- Learning schedule: adagrad, adadelta

As loss function we choose *warp-kos* because it significantly outperformed all the others. The results we obtained on the local validation set are reported in Table 6.2.
We can observe that the *adagrad* learning schedule performs worse than the *adadelta* one. The configuration that we selected is the best one reported.

#Epochs	#Components	Learning Rate	Learning schedule	MRR
200	300	0.1	adadelta	0.577164
150	300	0.1	adadelta	0.577132
300	300	0.1	adadelta	0.577080
200	300	0.2	adadelta	0.577062
200	200	0.1	adadelta	0.576162
200	300	0.01	adagrad	0.431659

Table 6.2. MF parameters tuning overview

Detailed experiments on parameters

As mentioned before, there are a lot of parameters that have been tested in order to obtain the best configuration of the algorithm, which is essential in a competitive scenario like this challenge. The table 6.3 give a better analysis of all the matrix factorization parameters.

#Epochs	#Components	Loss Function	Learning Rate	Learning schedule	K	useralpha	itemalpha	rho	epsilon	maxsampled	Score
200	300	warp-kos	0.1	adadelta	300	1,00E-06	1,00E-06	1.35	1,00E-06	10	0.577164
200	300	warp-kos	0.1	adadelta	300	1,00E+00	1,00E+00	1.35	1,00E-06	10	0.577146
200	300	warp-kos	0.1	adadelta	300	0.01	0.01	1.35	1,00E-06	10	0.577146
150	300	warp-kos	0.1	adadelta	300	1,00E-08	1,00E-08	1.35	1,00E-06	10	0.577132
300	300	warp-kos	0.1	adadelta	300	1,00E-06	1,00E-06	1.35	1,00E-06	10	0.57708
200	300	warp-kos	0.2	adadelta	300	1,00E-06	1,00E-06	1.35	1,00E-06	10	0.577062
200	300	warp-kos	0.1	adadelta	300	0.01	0.01	1.35	1,00E-06	10	0.5769
200	300	warp-kos	0.5	adadelta	300	1,00E-06	1,00E-06	3	1,00E-06	10	0.576826
200	300	warp-kos	0.5	adadelta	300	1,00E-06	1,00E-06	10	1,00E-06	10	0.576826
200	300	warp-kos	0.01	adadelta	300	1,00E-06	1,00E-06	1.35	1,00E-06	10	0.576752
200	300	warp-kos	0.5	adadelta	300	1,00E-06	1,00E-06	0.5	1,00E-06	10	0.576734
150	300	warp-kos	0.1	adadelta	300	0	0	1.35	1,00E-06	1000	0.576626
200	300	warp-kos	0.5	adadelta	300	1,00E-06	1,00E-06	1.35	1,00E-06	10	0.57658
150	300	warp-kos	0.1	adadelta	300	1,00E-06	1,00E-06	1.35	1,00E-06	10	0.576577
150	300	warp-kos	0.1	adadelta	300	0	0	1.35	1,00E-06	100	0.576535
200	300	warp-kos	0.5	adadelta	300	1,00E-06	1,00E-06	10	1,00E-06	10	0.576506
200	300	warp-kos	0.1	adadelta	400	1,00E-06	1,00E-06	1.35	1,00E-06	10	0.576299
200	300	warp-kos	0.5	adadelta	300	1,00E-06	1,00E-06	0.2	1,00E-06	10	0.576242
200	200	warp-kos	0.1	adadelta	300	1,00E-06	1,00E-06	1.35	1,00E-06	10	0.576162
200	300	warp-kos	0.01	adagrad	400	1,00E-06	1,00E-06	1.35	1,00E-06	10	0.431659
200	300	warp-kos	0.01	adagrad	400	1,00E-06	1,00E-06	1.35	1,00E-06	10	0.383908
200	300	warp-kos	0.1	adagrad	400	1,00E-06	1,00E-06	1.35	1,00E-06	10	0.371575

Table 6.3. Detailed parameters tuning analysis

6.3 Order of the hotel's list: another way to handle the cold start problem

As described before, we tried to use the LightFM approach to solve the cold start problem but, however this library was useful to generate a good matrix factorization model, it performs very poorly in this specific context. In order to improve our result we had to get better scores on this specific part of the task, which was the most challenging. To face this challenge we choose to split our dataset in two parts:

- Session with length equal to 1: the *Hard problem* of the challenge
- Session with length greater than 1: the part handle with the Matrix Factorization algorithm, improved with the gradient boosting that will be treated in the section 7.1

In this section we will focus on our efforts to improve the scores relatives to the *hard problem*.

6.3.1 First Attempt: Most popular hotel

The main difficulty of predicting the clickout action of a session where the only action is the clickout itself is obviously the lack of information. For this reason our first attempt was to create a sort of ranking of the most clicked hotels across all the sessions of our dataset and to suggest that in case of the clickout action to predict was performed in the first stop of the session. This approach is very simple and gave us a **MRR = 0,287995** that wasn't really good compared with the **MRR = 0.57** that we get on the other set of data.

6.3.2 Second attempt: Most popular for nation

One of the few information that we have in the context of single click action is the nation of the user that is looking for an hotel. So we define an approach to exploit this information to get some predictions. Instead of collecting the most clicked hotel overall, we create different lists, one for each nation, with the ranking of most popular hotel. In this case we simply suggest the most popular item between the other people that share the same nation with the user that is looking for an accommodation.
The main issue of this solution is that in some cases we don't have enough data, so we have to rely on the most popular item overall. To do so, we define a parameters that represents the weight of the most popular item with respect to the most popular by nation. After some tuning on this parameters we had a little improvement in terms of score (**MRR = 0.292201**) but it was still far from the performance on the other set. The Table 6.3.2 outlines the result related to the tuning phase of the *Most popular by nation*.

Most Popular Weight	MRR
0.00001	0.292181
0.0001	0.292181
0.001	0.292181
0.01	0.292201
0.1	0.291374
0.2	0.290686

Table 6.4. Parameter tuning for the Most Popular by nation

6.3.3 Third attempt: Matrix factorization model for nation

This was probably the most interesting idea to handle to cold start problem. A general overview of the approach we used in this case is outlined in Figure 6.2

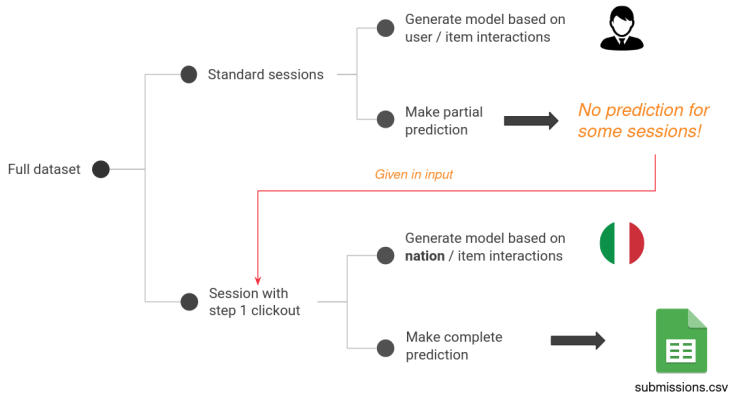


Figure 6.2. The nation fallback approach

We found that the matrix factorization model works well on this kind of task. The idea was to keep the same model for the *Hard problem* but instead of creating a matrix representing the interactions between *Users* and *Hotels*, we replaced *Users* with *Nations*. The new matrix will be composed in the following way:

- On the rows: nations of the users of each session
- On the columns: id of the hotel visited by each user of the given nation

The Figure 6.3 gave a better intuition about this model.

However this model seems really fascinating, it gave us almost the same score of the previously described *Most popular by nation*

6.3.4 Fourth attempt: Impression list order

The most effective idea in this case was also really straightforward. Indeed, Trivago presents to each user a series of hotel, ordered followed some defined rule and we found out that in the majority of session, the user choose one of the hotels located in the first positions, displayed at the top of the page. We also calculates some statistics about this

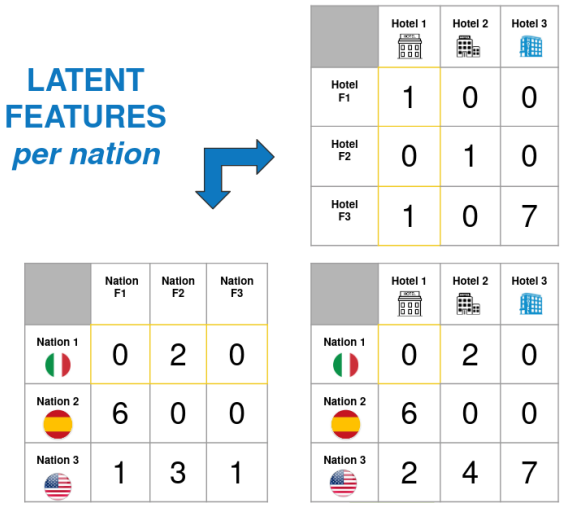


Figure 6.3. The matrix factorization model including nations

observation, by identifying how many time an hotel, displayed in the top positions will be effectively clicked by the user.

- 1st hotel -> **32%** of clickouts
- Top 3 hotels -> **43%** of clickouts
- Top 5 hotels -> **57%** of clickouts

From this data emerges that is very uncommon that a user choose an hotel from the bottom position probably because it doesn't scroll down the page enough to see them.
With this approach we get a good MRR improvement with respect with the other solutions described before: **MRR = 0.39**.

Chapter 7

Ensemble methods

From the previous chapter emerges the necessity to include some other dataset features that are left out from the matrix factorization model. For this reason we decide to include another machine learning technique that simplify this task and that gave us better results in terms of Mean Reciprocal Rank score.

7.1 Gradient Boosting

The Gradient Boosting is a popular machine learning algorithm used in both classification and regressions problem. This model is functional when it is necessary to join the results of some weak prediction model, to produce a better outcome, by means of decision trees.

7.1.1 The intuition behind this approach

The approach of integrating the matrix factorization outcome with other features, by means of gradient boosting was used successfully by the Avito Team in the Recsys Challenge of the previous year [17]. The features that we want to include are:

- **Recent index** is a number that represent how recent is the interaction between the user and the hotel, inside a specific session.

This is a very important feature because in the majority of cases the selected hotel has been visited recently.

- **Position** represents the position of the hotel inside the so called *Impressions List*, better described in Section 5.2.2

In particular it was really interesting the idea of not making the prediction directly with the matrix factorization model but to give the latent feature, calculated for each pair **user - hotel** to a better decision-maker algorithm like the gradient boosting.

7.1.2 XGBoost

The library used to develop this part of the algorithm's pipeline is called XGBoost that stands for e**X**treme **G**radient **Boost**ing [3]. The peculiarity of this implementation of the gradient boosting is that is very focused on the speed and efficiency of the algorithm, and can also be parallelized. Another really important characteristic is that this implementation is one of the most used in the context of Data Science challenges.

7.1.3 XGBoost training

The introduction of a gradient boosting model needs an additional training phase. To test the efficacy of this solution locally we decide to perform an additional split of the dataset. The pipeline of the split is outlined in Figure 7.1.3. The training is divided in two phases:

- **XGBoost training**:
  - The matrix factorization is trained on the blue block of the second split
  - The model produces some features about the pairs **User - Hotel**, using the outcome of the matrix factorization, along with the other features described at the beginning of the section.

  - The prediction is made on the green block, including a label that defines if the hotel has been clicked or not.
  - This dataset is used to train the XGBoost model

- **MF Training**
  - The Matrix Factorization model is trained again using the blue block of the first split
  - This model produces the same feature that were given to the XGBoost training on the orange block
  - The trained XGBoost model produces a score for each pair **User - Item** based on the feature in input
  - The final list of recommendations is produced by sorting the XGBoost scores

7.2 Borda count

Before our most successful attempt with XGBoost we tried another ensemble method, following an approach that was successful in the last year challenge [13].

7.2.1 Description of the method

The Borda method is a voting system that was used since a long time ago, to define the winner by merging the results of more than one output. In our work this method was used to merge the two sequence, that can be seen as a *vote result*, given by each algorithm. The Borda count is composed by the following step:

- Define the length n of the sequence
- Assign n point to the first item of the list, then n-*1* to the second one and so on
- Sum the result of all the different outcomes, rearranged in the described way

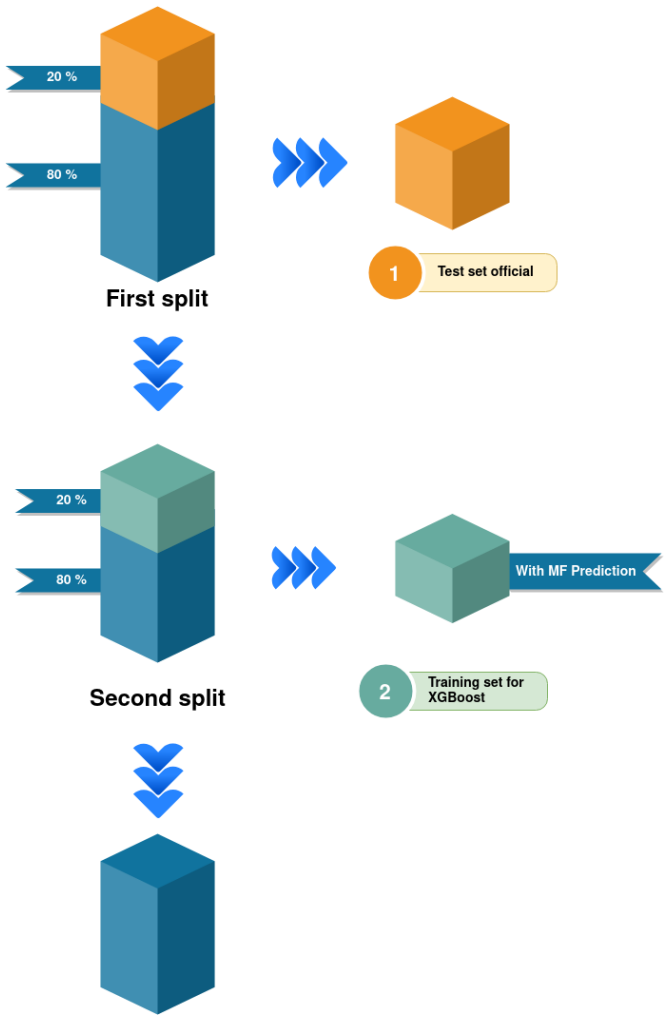


Figure 7.1. The double split of the dataset

- The item with the most number of votes, will be the first recommended

Example

For example, in our context we have two arrays of solutions, one for each algorithm:

$$Y_{MF} = [s_{h1}, s_{h3}, s_{h4}, s_{h2}] \tag{7.1}$$

$$Y_{RNN} = [s_{h3}, s_{h2}, s_{h4}, s_{h1}] \tag{7.2}$$

Where Y represent the outcome of the algorithms and s_{hi} is the score related to the Hotel i.

In this case we have the following score for each hotel:

- **Hotel 1** $= 3 + 0 = 3$ points
- **Hotel 2** $= 0 + 2 = 2$ points
- **Hotel 3** $= 3 + 2 = 5$ points
- **Hotel 4** $= 1 + 1 = 2$ points

The final list of the recommended items is generated by sorting the previous scores.

7.2.2 Results

Unfortunately this approach, however was really fast and simple, doesn't seem to get the strong point of the two algorithm. The result with this approach is almost an average of the two score produced singularly.

Chapter 8

Experimental Setup

This chapter describes some more technical details about the software and hardware that allow us to get the results described before. It also includes some information about how we managed the dataset in order to have a local validation set, that is vital for evaluating performances.

8.1 Technical details

8.1.1 Software

The software is written in Python and it is divided into three parts:

- Import of training and test set
- Generation of the solution following the algorithm
- Score calculation (Mean Reciprocal Rank)

The software can be executed from command line and receives a different set of parameters depending on the chosen algorithm. An example of a possible execution is the following one:

```
python Setup.py train.csv test.csv basesolution
```

In this case the first parameters is the setup file that has to be executed, than we have the path of the two files (train and test set) and

finally the name of the algorithm that we want to use to produce the submission file.

8.1.2 Hardware

Computational resources were provided by HPC@POLITO, a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino [1]. We used the cluster named *Hactar* whose infrastructure is described in the Table 8.1.2.

Architecture	Linux Infiniband-QDR
Node interconnect	Infiniband QDR 40 Gb/s
Service Network	Gigabit Ethernet 1 Gb/s
CPU Model	2x Intel Xeon E5-2680 v3 2.50 GHz 12 cores
GPU Model	2x nVidia Tesla K40 - 12 GB - 2880 cuda cores
Sustained performance (Rmax)	20.13 TFLOPS (last update: june 2018)
Peak performance (Rpeak)	25.61 TFLOPS (last update: june 2018)
Computing Cores	696
Number of Nodes	29
Total RAM Memory	3.6 TB DDR4 REGISTERED ECC
OS	Centos 7.4.1708 - OpenHPC 1.3.4
Scheduler	SLURM 17.11.5

Table 8.1. Hardware of the Hactar Cluster

8.2 Data Preparation

8.2.1 Local dataset split

The only way to get the result using the dataset provided by the challenge organizers is to perform a submission on the Trivago website [2]. In the interest of providing a better way to get a feedback about the

[1] http://hpc.polito.it/

[2] https://recsys.trivago.cloud/challenge/

techniques experimented we decide to split the training set following the same rule of the challenge. More in detail:

- We defined a training set with 80% of the data, and a test set with 20%
- We kept the actions referred to the same session together

The proficiency of this local spilt was confirmed by the fact that the result obtained on our local test set were vary similar to the official score calculated after a submission.

Differences with respect to the official test set

We discovered that the validation set were Trivago calculates the score for the challenge ranking has a big difference with respect to our local split that is impossible to be replicated. In the training set that we used for the split, the *Impressions list* is randomly shuffled, in order to hide the recommendations provided by Trivago itself. This is not replicated in the validation set, and for this reason we get a little improvement in the submission, with respect to the local result.

Chapter 9

Results

9.1 Pure Matrix Factorization results

The MF model alone gave us a good MRR score. The best thing about this approach is that it provides a relatively good solution with a small training time (roughly 1 hour). To compute the official score, we took the parameters with the best performances on our local validation dataset and we calculate the recommendations on the test set provided by Trivago. In particular we choose:

- **Number of epochs**: 200
- **Number of components**: 300
- **Learning rate**: 0.1
- **Learning schedule**: adadelta
- **K**: 300
- **User Alpha**: 1.00E-06
- **Item Alpha**: 1.00E-06
- **Rho**: 1.35

- **Epsilon**: 1.00E-06
- **Max sampled**: 10

The result given by the submission calculated by Trivago on their own validation set is **MRR = 0.522896**

9.1.1 Comment on results

This solution gave us the first improvement over the simple recommendation of the most popular items (**MRR = 0.28**) that the baseline. However we treat the cold start problem with the method described in section 6.1.2 the performance on this kind of dataset were not as good as we expected and we have to start exploring other solutions.

9.2 Matrix Factorization with Gradient Boosting

For this results we used the best configuration obtained for the matrix factorization, described in the Section 9.1.

9.2.1 Local result

The effectiveness of the algorithm is evaluated on the local test set before the submission, in order to make easier to find the best parameters. One of the most interesting features of the XGBoost library is that it plots, after the training, the importance of each feature given as input, for the prediction. Notice that this value is calculated by using the F score, that simply sum up all the times that the feature is used in a split of the decision tree.

The importance of each feature is shown in Figure 9.2.1.

As we can see, the most important feature is the score produced by the Matrix Factorization model, followed by the *Recent Index*. The position of the hotel in the impression list is not so relevant because the training set, as described before, has a shuffled impression list and is not considered useful in terms of predictions.

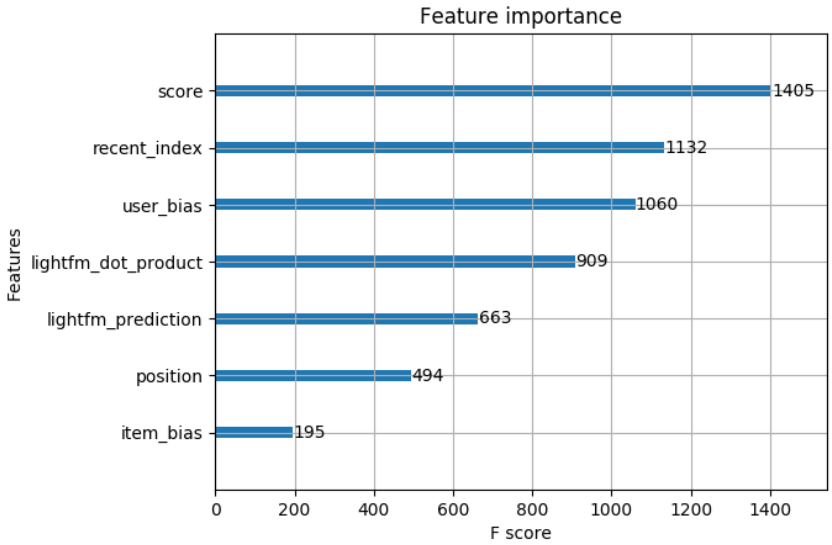


Figure 9.1. Importance of each feature in the matrix factorization model

The result obtain on the local test set are shown in the Table 9.2.1.

MRR	Improvement
0.598042	**+ 0.07**

Table 9.1. XGBoost + MF on local dataset

9.2.2 Official result

There is a small improvement between the official result and the local one. This happens because, as the organizers of the challenge explicitely say that there was a difference between the training set and the test set. In the training set, that we used to calculate the local score, since is the only way to define the efficacy of algorithm, the impression list has been shuffled, while in the test set they are sorted

following the Trivago recommendations.

For this reason, in the submissions, we get a better score for the *Cold Start* part of the algorithm, described in the Section 6.3, that reflects a better overall score.

The result, expressed by means of Mean Reciprocal Rank, is shown in Table 9.2.2, with the improvement from the previously submitted solution that uses only the Matrix Factorization.

MRR	Improvement
0.610525	**+ 0.08**

Table 9.2. Official score of the MF + XGBoost algorithm

9.3 Matrix Factorization with Gradient Boosting and RNN

The results of the ensemble between the Matrix Factorization and the Recurrent Neural Network were taken only on the local test set because unfortunately, the configuration of the RNN wasn't completed before the deadline of the challenge, due to a long training time. There were two different configuration of the network.

RNN: First configuration

The first configuration of the RNN is based on a GRU model, centered on the pure classification task. The hotels inside a session were encoded and given as input to the network that extracts the latent features and selects the more suitable hotel by means of a confidence score.

The prediction given by the Recurrent Neural network on each pair **User - Item** is added to the input given to the XGBoost model.

We can see the impact of the RNN by looking at the importance of each features in the Figure 9.3.

The result, described in Table 9.3 give another little improvement with respect to previous matrix factorization model.

MRR	Improvement
0.602774	**+ 0.005**

Table 9.3. First RNN + XGBoost MRR and improvement from MF

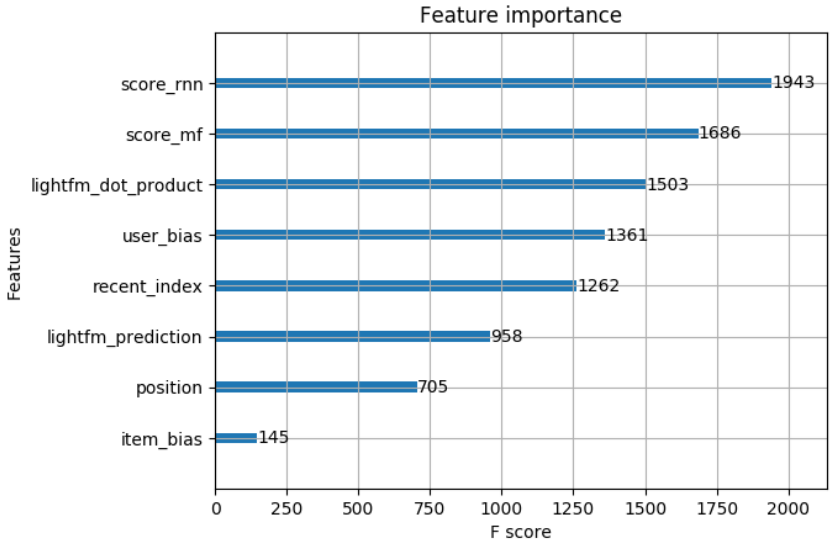


Figure 9.2. Importance of each feature in the first configuration of the RNN

RNN: Second configuration

The second configuration is based as well on the GRU architecture. In this case the focus is on the last-n hotels that have been visited during a session. The RNN tries to identify if the hotel clicked is included in the last visited or is outside this list. In the first scenario

it tries also to understand which of the visited hotels have been clicked.

The input given to the XGBoost model is in the same format specified before, with the outcome produced by this different configuration of the Neural Network.

In this case the result is really similar to matrix factorization, just a little bit worse, as outlined in Table 9.3

MRR	Improvement
0.598029	**- 0.0000137**

This behaviour is reflected also in the importance of each feature for XGBoost, as described in the Figure 9.3

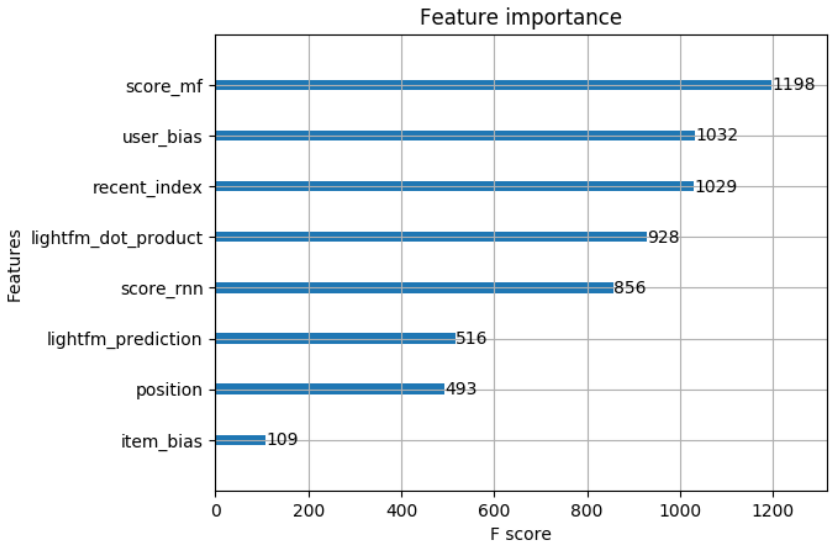


Figure 9.3. Importance of each feature for the second configuration of the RNN

Figure 9.4. Second RNN + XGBoost MRR and improvement from MF

Chapter 10

Conclusions and future works

In this thesis was described the approach that our team took to solve the task presented in the RecSys Challenge 2019, hosted by Trivago. The objective of the challenge was to analyze the actions that a user perfoms during a session, in order to decide which hotel is the most suitable for him.

We start with an accurate analysis of the dataset provided by the organizers of the challenge, where we define some important features that should be taken into account in the model, such as some relevant statistics about the session.

Then in the third chapter was described the state of the art in the context of recommender systems. First of all, the algorithm were classified in two categories: Sequence Aware and Collaborative filter. The most well known algorithm of the first category is the matrix factorization, that represents the core of the solution presented in this master thesis. On the other hand, from the category of sequence aware recommender system, was given a general overview of the Recurrent Neural Network, focusing on the problem of each implementation. Our team decides to explore both solutions in order to exploit different features

of the dataset that will be ideally reflected in an improvement of the score. The score of each submission is calculated using a statistical method called Mean Reciprocal Rank.

The core of the solution is the matrix factorization, which is implemented by means of LightFM. It takes as input a matrix that is very well known in literature: on the row there are the users and on the column the hotels. Starting from this input the algorithm calculates two other matrices carrying the latent feature of user and hotels, that are multiplied together to get the prediction for the pair.

A further improvement was obtained by using the Gradient Boosting technique to include two other dataset features: the position of the item in the list of hotels presented to the user (called *Impression lists*) and an index the identify how recent is the interaction between the user and the given item. This method was used also to include the scores given by the two configuration of the Recurrent Neural Network, developed by an other component of the team.

The eighth chapter describes the technical details of the implementation: the code is written in Python and is executed on the HPC Cluster, powered by Polito. Finally were presented the results for each configuration. The Pure Matrix Factorization gave a MRR score of 0.52, that become 0.61 with the integration of the other features, by means of Gradient Boosting.

10.1 Future Works

The work presented in this master thesis has a lot of possible improvement, due to the fact that some part of the problem were not analyzed thoroughly, because of the strict deadline of the challenge.

10.1.1 Optimization

There are a lot of other possible solutions for this task that will remain unexplored. In particular the part of the algorithm that can be improved at most is the cold start. Due to the problem of the metadata file it wasn't possible to exploit at all the Matrix Factorization model provided by LightFM and even if we made an attempt by using the price category, with more time it would be possible to find other features with more impact.

10.1.2 Portability

One of the problem of this research work is that is deeply focused on the task proposed by the challenge. The score is always calculated on the Trivago validation set, except for our attempt on the local test set. So, we didn't explore a lot how the solution performs on different dataset or on a bigger dataset. In fact, as explained before, one of the main issue in the context of tourism is that the prediction are affected a lot by the change of season for obvious reason. It wasn't possible, due to the limitation of the dataset provided by Trivago, which takes in account only 2 days, to analyze the performances of our algorithm in this scenario. It is also interesting to try the pipeline on different context [1].

10.1.3 Weakness of the Matrix Factorization

The application of the Matrix Factorization algorithm, however it is straightforward and very well studied in literature, presents two main problem:

- Cold start problem
- Difficulties in detecting the shift of tastes

The first problem has been attained a bit by using the user and item embeddings as detailed in Section 6.1.2 but it can surely be improved. The second one remains an open issue. The Matrix Factorization

works very well when it is necessary to define the user tastes, but if that person change interest over time, which is a really common scenario, it requires a lot of interaction to understand that. It would be interesting to handle this problem directly using this algorithm, without using an hybrid approach such as the one described in this work.

Acknowledgements

I would like to thank my family, for the constant support during the whole university path: they give me an invaluable encouragement and assistance to me. A special mention for my girlfriend, she comfort me when the times got rough and was also comprehensive during the periods I cannot spend a lot of time with her because I was working hard.

I offer my sincere appreciation to all my university collegues, for making each difficult day funnier and for the team working that allow us to pass a lot of exams. It is also vital to thank all my friends, for the amusing moment of each weekend spent together.

Finally I would like to thank also the whole team of the challenge. First of all Giorgio: we worked very well together in the delicate part of joining our two solutions, thanks to the continuous process of sharing information about the challenge. This work could not have been accomplished without the contribution of Giuseppe and Diego. I found really useful the way they structured the work process, with constant weekly meeting where they share their precious knowledge.

Bibliography

[1] Linas Baltrunas, Bernd Ludwig, and Francesco Ricci. "Matrix Factorization Techniques for Context Aware Recommendation". In: *Proceedings of the Fifth ACM Conference on Recommender Systems*. RecSys '11. Chicago, Illinois, USA: ACM, 2011, pp. 301–304. ISBN: 978-1-4503-0683-6. DOI: 10.1145/2043932.2043988.

[2] James Bennett, Stan Lanning, et al. "The netflix prize". In: *Proceedings of KDD cup and workshop*. Vol. 2007. New York, NY, USA. 2007, p. 35.

[3] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785.

[4] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. "Sequential user-based recurrent neural network recommendations". In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM. 2017, pp. 152–160.

[5] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[6] Rainer Gemulla et al. "Large-scale matrix factorization with distributed stochastic gradient descent". In: *Proceedings of the 17th*

ACM SIGKDD international conference on Knowledge discovery and data mining. ACM. 2011, pp. 69–77.

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

[8] Sepp Hochreiter. "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[10] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems". In: *Computer* 8 (2009), pp. 30–37.

[11] Maciej Kula. "Metadata Embeddings for User and Item Cold-start Recommendations". In: *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015.* Ed. by Toine Bogers and Marijn Koolen. Vol. 1448. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 14–21.

[12] Tomáš Mikolov et al. "Recurrent neural network based language model". In: *Eleventh annual conference of the international speech communication association*. 2010.

[13] Diego Monti et al. "An Ensemble Approach of Recurrent Neural Networks Using Pre-Trained Embeddings for Playlist Completion". In: *Proceedings of the ACM Recommender Systems Challenge 2018*. RecSys Challenge '18. Vancouver, BC, Canada: ACM, 2018, 13:1–13:6. ISBN: 978-1-4503-6586-4. DOI: 10.1145/3267471.3267484.

[14] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. "Sequence-Aware Recommender Systems". In: *ACM Comput. Surv.* 51.4 (July 2018), 66:1–66:36. ISSN: 0360-0300. DOI: 10.1145/3190616.

[15] Steffen Rendle et al. "BPR: Bayesian Personalized Ranking from Implicit Feedback". In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. UAI '09. Montreal, Quebec, Canada: AUAI Press, 2009, pp. 452–461. ISBN: 978-0-9749039-5-8.

[16] Francesco Ricci. "Travel recommender systems". In: *IEEE Intelligent Systems* 17.6 (2002), pp. 55–57.

[17] Vasiliy Rubtsov et al. "A Hybrid Two-stage Recommender System for Automatic Playlist Continuation". In: *Proceedings of the ACM Recommender Systems Challenge 2018*. RecSys Challenge '18. Vancouver, BC, Canada: ACM, 2018, 16:1–16:4. ISBN: 978-1-4503-6586-4. DOI: 10.1145/3267471.3267488.

[18] Jason Weston, Samy Bengio, and Nicolas Usunier. "Wsabie: Scaling up to large vocabulary image annotation". In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[19] Jason Weston, Hector Yee, and Ron J Weiss. "Learning to rank recommendations with the k-order statistic loss". In: *Proceedings of the 7th ACM conference on Recommender systems*. ACM. 2013, pp. 245–248.