POLITECNICO DI TORINO

Master's degree course in Computer Engineering

Master's Degree Thesis

Hotel in a Clickout: a Session-based RNN Prediction Approach

Supervisor
prof. Maurizio Morisio

Candidate
Giorgio Crepaldi
Student ID: 242108

Internship Tutor
LINKS
dott. ing. phd. Giuseppe Rizzo

Academic Year 2018 - 2019

Abstract

This thesis aims at analyzing and experimenting on a Session-based Recommender System problem, which dataset is provided by Trivago for the Recsys Challenge 2019. In order to extract features we develop 2 solutions based on Recurrent Neural Networks, experimenting this recent technology in a field yet to be entirely explored. The diversity of the results obtained are interesting for analyzing how good this approach works in this field. Along with the solution described we analyze an ensemble of the results with a different algorithm (Matrix Factorization), which is discussed in another thesis. This comparison made it possible to make conclusions about how good different approaches are and why.

Acknowledgements

A few words are for Andrea, Giuseppe and Diego. Working with them was a real pleasure and I did not expect I could find such open and careful people for my thesis. Especially Andrea, as working with him was easy and fun and he helped me so much in the final part of my journey.
I thank my friends, old and new, who let me spend the most enjoyable times when everything else was not enjoyable. Especially Andrea G. as he always listened to me. A special thank is for those who were an important part of my student life and more: Jeanpierre, who was the pillar of my university experience, Isabeau, who encouraged me for the future and Giacomo, who always understood me.
I thank my family for the support given to me in these years and the lack of pressure for my studies, they made everything easier. My parents always encouraged every decision I made and my sister was like a friend.
For Chiara, who is very important to me as she always believed in me more than myself, no matter what, and thanks to her I'm who I am now.

Summary

In this thesis we present our solution for the Recsys 2019 challenge which consists of 2 different RNN solutions: **Pure-Classification** and **Last-N**, with the aim of predicting the hotel a user will choose to *clickout* during his navigation on the Trivago platform, in a **session-based** scenario.

First we introduce the world of recommender systems showing examples of fields where they are being successfully used as well as their growing importance. We then describe how they have been classified through the years and the different technologies that have been experimented for solving recommending problems in different ways.

Starting from this knowledge we describe our solution for a specific problem belonging to the session-based category. The proposed solution is composed of a cold-start approach for sessions of 1 step, while the rest of the dataset is fed to 2 different approaches: **Recurrent Neural Network** and **Matrix Factorization**. The main idea is to complement the MF strenght on sparse datasets with the sequential knowledge extraction of the RNN, using an ensemble based on a tree_based boosting algorithm: **XGBoost**. The experiments are done on the dataset provided by Trivago, which consists of a collection of user's sessions on the web site with the goal of predicting which hotel item the user will click (clickout). We initially do an analysis of the dataset to make some assumptions that are essential for the experimental setup choices. This analysis showed that a lot of sessions are 1 step long, making it impossible for the RNN to compute them. The solution to this subset is a **cold-start** approach which consists of recommending the impression_list field, which is the list of hotels

showed to the user at the moment of the clickout.

To feed the RNNs we need to encode the input data in some sort of numerical format. That is why after cleaning the dataset we create the input by taking only the actions related to an item and encoding them with a w2vec algorithm with the use of the *Gensim* library. We do this to represent similar hotels close in a multi-dimensional space, as the algorithm associates hotels by their proximity in the session sequences.
We experiment 2 different RNN solutions.

The first one is the **Pure-Classification**, which is based on a RNN which tries to select the right item in the whole list of existing hotels by having as output a confidence score for each item, the higher score the higher probability for it to be the right one. This is done by using a LogSoftmax function at the end of the RNN, along with a **NLLLoss** for the training part.

The second solution is the **Last-N**, which is configured like the one in the first solution but selects the right hotel among the last visited in a single session or decides that the clickout hotel is not present in this list, and so recommends the impression_list instead.

As expected the results for the Pure-Classification solution shows that an RNN approach based on a pure classification does not get good results when facing such a sparse problem when compared to other ones, but the goal of this solution is to complement the lack of the MF solution with its study on the sequence features. The ensemble with the MF showed a slightly score improvement, which can be related to the hypothesis of the RNN succeeding in the sequence learning.

For what concerns the Last-N solution we observe that the training does not need much time to train as the number of classes is very little, resulting in a very clear loss shape after a little bit of tuning. The score is much higher that the first solution as the last visited items are very likely to be those the user is more interested to. In the ensemble with the MF we notice that there is no noticeable overall score increase, because the XGBoost yet considers the contribute of the last-n items.

Even if the score of the solution is not close to the leaderboard

winners, the importance is not only in the score obtained for the challenge dataset but more importantly in the behaviour of such technology when facing a session-based recommender problem. The last-visited items confirm to be very important in the final user decision, despite the solution they are included into. The need of a hybrid approach for these problems seems necessary to overcome these lacks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The purpose of this thesis is to develop a solution to a Recommender System problem, more precisely the one proposed by the RecSys Challenge 2019, sponsored by Trivago. Our goal is to try different and complementary technologies (Matrix Factorization and Recurrent Neural Network) on a session-based problem and ensemble them in order to study not only their singularity but also their strong points in a multi-approach matter.

1.1 Team

We are a team composed of 4 members joining the forces of Politecnico di Torino and LINKS Foundation, under the supervision of Professor Maurizio Morisio from the DAUIN department. Me and my colleague Andrea Fiandro are the students who developed the complementary solutions for the experiment. We were hugely supervisioned and helped in doing that by more expert figures in the field: Giuseppe Rizzo, a senior researcher employed at Links working in NLP and recommender fields, and Diego Monti, a Phd student who mainly focuses his research in recommender systems.

1.2 Recommender system

A recommender system is an AI software which elaborates huge amounts of data to suggest any kind of product to the customer, ranging from real objects to media products. This permits to profile used preferences and suggest the best item.

In recent years recommendation systems showed positive results in both economic and research matters. Companies adopting this new technology encountered an increment in their income, due to the increasing satisfaction of the users and the affection that they develop for the system. A noticeable case is the one involving media services such as Netflix or Spotify, where the user taste are a top priority. In terms of research this field is good for trying new AI technologies and for exploring the modeling of people behaviours.

1.2.1 Applications

Recommender systems are applied in different fields. They help people decide which object to buy, film to see or song to listen when using web platforms, which are becoming more and more used. From buying a product on Amazon to watching a tv web series on Netflix, those services continuously keep recommending another product to consume and users are happy about that and when users like a platform they are encouraged to spend more time on it, increasing the company incomes. Recommenations may be considered even in less consumistic way as not only products are recommended, instead there are examples like Facebook that recommends friends, Linkedin for jobs. As we can see the fields of application are endless. These companies are investing more and more money in recommender system as they understood the great capabilities and incomes they may provide.

1.2.2 Technologies

The technologies used for recommendation purposes are those based on the extraction of features from a large set of data which characterize

the users and match those features with the ones belonging to the items to be recommended.

This leads to the use of Matrix Factorization techniques, which are characterized by direct user-item matches. But the recent development of AI technologies (i.e. Machine Learning and Deep Learning) showed new ways to extract features with very good results in fields such as computer vision. The availability of Big Data opened the way to Deep Learning solutions as we explore in this thesis.

1.3 Why Deep Learning?

Deep learning is a new technology that is obtaining more and more interest in recent years, especially in fields like Computer Vision and Natural Language Processing. The problem similarity between NLP and Recommender Systems (considered as sequences of data) rises the interest in testing this methods in the Rec field as we do in this thesis.

1.4 Results

The results of this experiment are considered by single solution and by ensemble of 2 different solutions: Matrix Factorization and Recurrent Neural Network. The RNN approach (the target of this thesis) results are relative to the splitted dataset, which is a subset of the one for the official leaderboard, and considered 2 different solutions scoring very different MRR results: 0.28 and 0.54, while the ensemble best score with the MF is 0.602774. The final submission to the Recsys Challenge did not include the RNN results for time issues, but we obtained a score of 0.62.

Chapter 2

State of the art

In the following chapter we describe how recommender systems have been studied and classified, with an additional analysis of the technologies that are obtaining good results in this field.

2.1 Recommender Systems

Since the advent of Internet more and more users faced the opportunity of having access to a constant growing number of information. Nowadays this phenomenon is so big that everyone can do it in a matter of seconds with the use of a cellphone, from reading a new article to booking a new hotel. Accessing data implies the creation of efficient research methods on the net and with them the need of a system that suggests the best fitting data to the user. As this need spread Recommender Systems were created to fulfill it, but they started to score good results only after 2000. Since then a great number of classifications and applied methods have been defined [19]. To better analyze and study recommender systems, different classifications were made and we report them in the following sections [20].

2.1.1 Context

Context categorization may depend on the characteristics of the problem and the dataset or simply on the choice of the implementation.

Last-N interactions

This approach consists in considering only the **Last-N** actions of the session to predict the following one. This may be due to the nature of the problem which makes older interactions not important for prediction.

Session-Based

A session-based recommender system can use as information only actions belonging to a **unique session**, which is a limited window of time in which the user interacts with the service. In this case we lack historical information about the user and the analysis is usually a short-term one. This category is typical for services without a needed registration to the service but also news [6], e-commerce, videos.

Session-Aware

A session-aware recommender system can use both **historical** and **session** information [17]. This type of system is great for both short and long-term models.

2.1.2 Approach

Another categorization is due to the data relations that we consider to extract features.

Content-Based

This approach takes into account user-item interactions and recommends items similar to the ones the user has viewed before. It is based on profiling a user by its features or behaviour or interaction with an

item. This item is characterized by its own features which may be used to profile the user and to understand which type of item he may be interested in. Example: A person may be using Amazon platform and looking for a film. It is easy to assume that the same person attempt and interest is to buy something for its collection and the system will suggest similar movies or maybe a new television to watch them.

Collaborative Filtering

This approach is based on the similarity between users and recommends the same items to similar users. People profiling is very important in this field as we try to understand the taste of the user and how it may reflect in other users. It is useful for discovering new items. Example: take into account the Netflix platform. Two different user may watch the very same action and humor films and being considered very similar. When the first one watches a particular horror movie, the system will suggest that to the second user, because it is very likely for him to appreciate it.

Hybrid

It is a mix of the 2 methods, as they are very complementary it is a good idea to consider both sides of the problem and take the best of them.

2.1.3 Feedback

The feedback defines how we extrapolate the knowledge of the relation between a user and an item.

Explicit

An explicit feedback consists of a user interacting **directly** with an item, being the interaction positive, negative or just an interaction. This association creates a user-item feature.

Implicit

Implicit feedback is the one not depending on a direct user-item interaction but deducted from user features which may be common or related to features belonging to the considered item.

2.1.4 Domain

As we said recommendations can operate in diverse fields ranging from e-commerce to music [22]. It is very important to consider the domain in which the system will operate in terms of features to be considered or technologies to use because the problem environment may vary.

E-Commerce

This is the most explored field as when a user watches or buys an object he is very likely to buy another one which is related to it. For instance buying a toothbrush may lead to buying a toothpaste. It is very common for this systems to be Session-Aware as even if user preferences are important, recent session interactions may be even more relevant.

Music

This field shows the relevance of the session and more importantly to the sequence itself as musical tastes rapidly change, even in short time and the system must adapt to this trend. Next track prediction or suggestion are the most important applications.

Video/TV series

In this case the tastes of the user are fundamental as they define the type of TV series they watch and suggestions are picked from that pool. The Netflix competition of 2006 showed a great performance of Matrix Factorization in this context.

2.2 Technologies

Technologies may be divided in 2 main categories: Sequence Learning Algorithms and Sequence-Aware Matrix factorization.

2.2.1 Sequence Learning Algorithms

Sequence based algorithms focus on the **succession** of actions aspect of the problem. It is useful when considering problems where we suggest an item every time an event happens, so that there is a chance that the user will interact with the newly recommmended item.
An example may be a user navigating on Amazon website: every time we inspect an object, other similar or related items pop up in the window to promote further navigation on the platform. This case needs a prediction after every user interaction, but there are other cases (as the one we face) where we only need to predict the last element of a sequence. These methods not only focus on the items occurring in the sequence but also on their **order**, as it may contain useful features on the user intent. So they are the most impactful when we have data which hides many information in the sequence of the actions.
Some of the main implementations are described below.

Markov Models

Markov Models are stochastic models used to predict that are based on the knowledge we get from the current state without considering the previous ones. One of the most used Markov Models for recommender systems are Markov Chains. They are based on a stochastic model that takes into account only the last few N actions before the prediction, as we do not want to consider the history.

Reinforcement Learning

Reinforcement Learning algorithms are based on receiving a **feedback** (positive or negative) after a prediction and maximizing the total reward among multiple successive actions. The sequential nature of this

technologies is perfect in this case.

Recurrent Neural Networks

Recurrent Neural Networks are described more deeply in section 2.3.

2.2.2 Matrix Factorization

Matrix Factorization is one of the most used algorithms in Recommender Systems [9], since the Netflix competition which confirmed this supremacy. Those algorithms are based on matrices as the name suggests. More precisely they create 2 different matrices: one describing **user** features and one representing **item** ones. These will be used to understand the relations between user and items. In Figure 2.1 we have user matrix characterized by a yes/no for different genres and a second matrix representing the genres in a multiple feature domain. The core of the problem is that users usually do not interact with every single item, resulting in a very sparse results matrix. The results matrix missing values are the ones we are going to predict, which translates in foreseeing how much interest user **u** has in item **t**.

Also MF models divide into Content-Based and Collaborative Filtering. The second type is very interesting as we can associate very efficiently similar user tastes by comparing similar interacted object using a matrix structure.

SVD

SVD (Singular Value Decomposition) is a commonly used algorithm to decompose matrices. In the recommender filed it was successfully used by Simon Funk for the Netflix challenge and it is based on the decomposition of the initial user-item sparse matrix into 2 smaller matrices, representing respectively users and items. This process is done with a loss function which value is to be minimized. In the prediction phase we multiply the 2 matrices and add a bias to obtain the values referring to which items a user prefers.

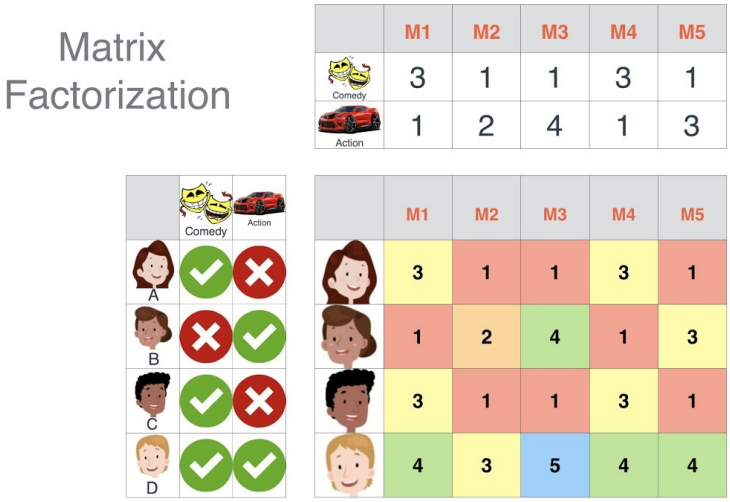

Figure 2.1: User-Item Mf example.

Source: http://primo.ai/index.php?title=Matrix_Factorization

2.2.3 XGBoost

XGBoost[1] (Extreme Gradient Boosting) is an algorithm which tries to boost the performance of **tree based classification systems** [1]. These systems consist of a tree structure which aim is to make decisions and discriminate between different classes. XGBoost analyzes the data and maximizes the decision making of this tree with the use of machine learning techniques. The major contribute of this library is in the scalability it offers, making it a good fit for huge datasets like ours. We use its potential to boost the ensemble between MF, RNN and other significant features we discover in the dataset.

[1] https://xgboost.readthedocs.io/en/latest/index.html

2.3 From AI to Deep Learning

Artificial Intelligence (AI) is the simulation of human intelligence performed by a machine by the use of theory and algorithms. This definition offers a huge variety of possible fields to consider, as it is very general.

2.3.1 Machine Learning

Machine Learning is a subgroup of AI, including a collection of techniques which enable the machine to learn through experience. Machine learning techniques are widely used in today tasks such as image recognition, speech recognition, e-commerce, social networks and much more.

The machine takes as input some vectors of numbers and adjusts internal weights according to the difference between its output and the expected one. The most simple example of this is the perceptron of Figure 2.2.

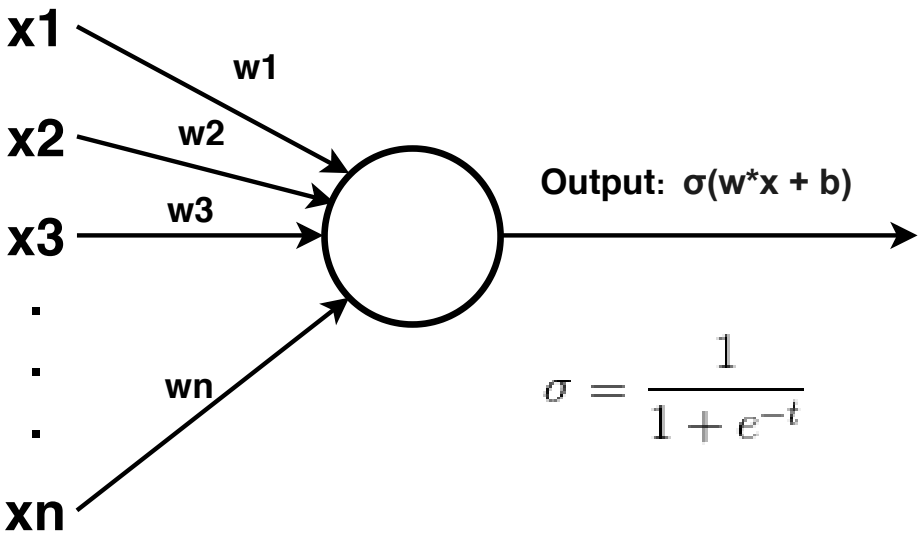

Figure 2.2: Perceptron structure

> *A computer program is said to learn from experience E, with respect to class of tasks T and performance measure P, id its performances at tasks T, as measured by P, improves with experience E.* [Arthur L. Samuel, Some studies in machine learning using the game of Checkers (1959)]

Machine learning has been classified in 2 different subgroups:
Unsupervised Learning is about extracting features from unlabeled items with tools like PCA or clustering.
Supervised Learning are the methods that take labeled data s input and try to predict the correct label by the learning of the items features. This is the most common type and the one that has produced the best results so long.
Another important classification, more relevant for supervised models, given X as the training data and Y as the target:
Discriminative Models estimate the parameters of P(Y|X) from the training data. It just tries to model the data and classifies it.
Generative Models estimate parameters of P(X|Y) and P(Y) from the training data and use Bayes Rule to calculate P(Y|X). It tries to learn the model that generates the data.

2.3.2 Deep Learning

Deep Learning [12] is basically a concatenation of multiple perceptrons where every layer extracts features from the previous one. It has become more and more famous for its increasing success than simple machine learning in tasks where we have a very large amount of data. This has become possible thanks to the progress of computing power of GPUs.
Different Deep Learning architectures have been developed according to specific tasks:
Convolutional Neural Networks (CNN) are perfect for extracting from multiple arrays of data and adapt very well to image processing as they are composed of RGB vectors.
Recurrent Neural Networks (RNN) are used mainly for sequences

of data and work very well in tasks like Natural Language Processing or Recommender Systems. In the following section we go deeper in this architecture description.

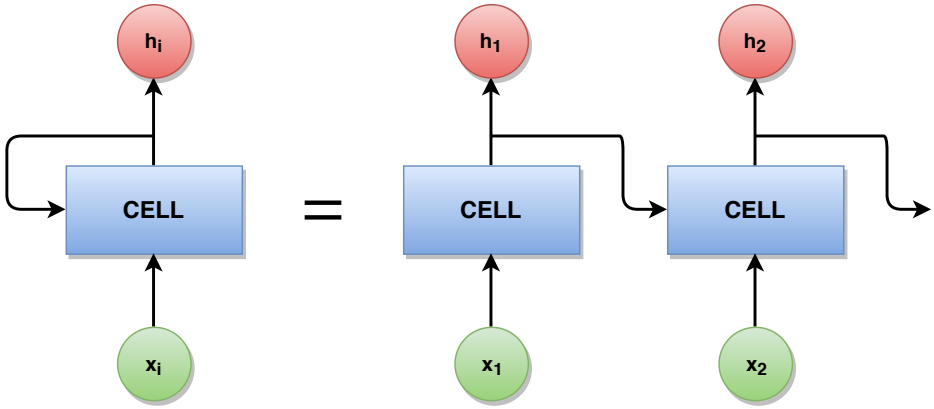

Figure 2.3: Recurrent Neural Networks structure

2.3.3 Recurrent Neural Networks

Recurrent Neural Networks have been tested in session-based scenarios [7] [5] and yet obtained good results by adding some mods to adapt them to more specific recommender problems.

RNN architectures [2] process a sequence of input items (x_i) one at a time and keeps track of the hidden layer (h_i) at each iteration by concatenating it to the successive input as shown in Figure 2.3. The main subclasses of RNN are described below.

Long-Short Term Memory (LSTM)

LSTM uses additional hidden units to better maintain the **memory** of past inputs. Its difference is noticeable in the structure of the cell which presents several arithmetic functions inside with the aim to keep a cell state. The cell is able to select which information is worth to

be kept and which is to be forgotten. In Figure 2.4 we can see the internal structure composed of:

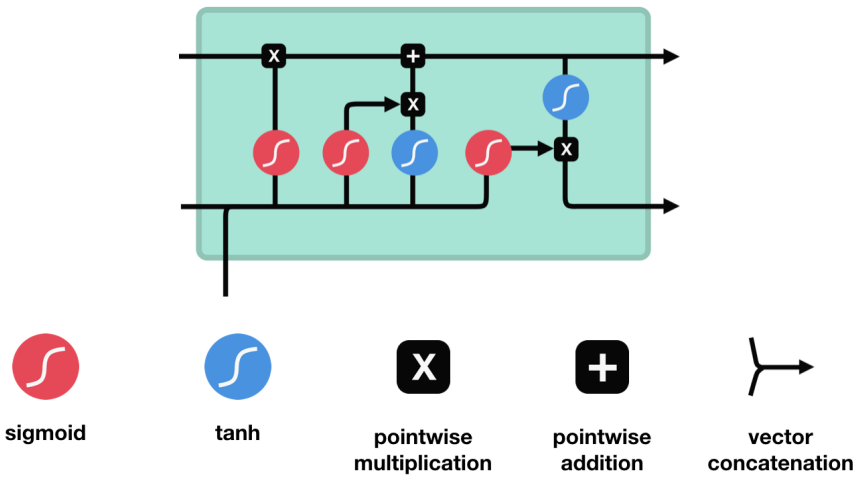

Figure 2.4: LSTM cell structure.

Source: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

- **forget gate**: to select which information to forget, as they are not relevant for the task.
- **input gate**: to select which input features are worth to be kept.
- **output gate**: which decides which information will be kept in the next hidden layer.

This solution proved to outperform vanilla RNN especially in long sequence tasks but it is obviously more time consuming as the architecture complexity increases.

Gated Recurrent Unit (GRU)

GRU is a simplified version of LSTM that does not have separated memory cells but still controls the flow of information, improving time performance while still keeping a good overall score. As we notice in Figure 2.5 the structure has not the cell state any more and it is composed of 2 gates:

- **update gate**: it decides which features are to be kept and which to be forgotten. As the name suggests it makes an update based on the new information.
- **reset gate**: it decides which features from past events are to be forgotten as they are not relevant for the output.

Gradient Problems

Exploding and Vanishing Gradient [16] are well known problems in deep learning. They are due to the update of the cells when processing long sequences of data. Backpropagating in such a big number of layers may cause the gradient to explode or vanish, making the update too big or too little, meaning it is impossible to learn. The basic RNN cell suffers this phenomeno but both variations of RNN we describe solve the Vanishing Gradient problem [8].

Losses

The loss function is a crucial part of a deep learning algorithms as it is what observes the difference between the obtained result and the desired one, computing the gradient and understanding how to update the net weights. As the training goes on, the loss value should decrease with a hyperbolic rate because lower loss means less difference between result and target, meaning that the net is learning. Different losses have been created for different purposes, but we focus on the ones more related to our problem:

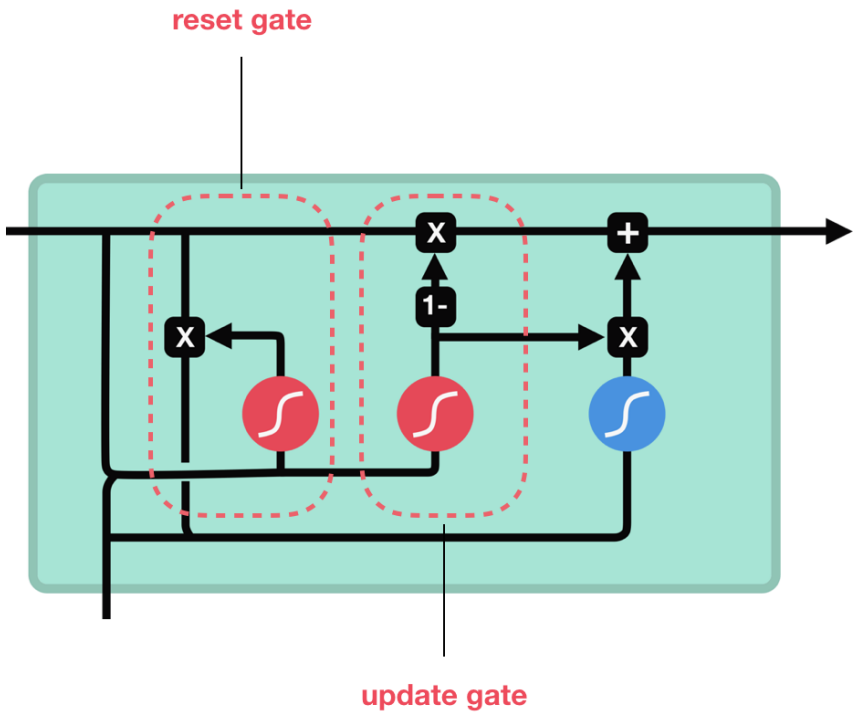

Figure 2.5: GRU cell structure.

CrossEntropy is the best loss for classification problem among a wide range of classes. It expects an output of dimension C, where C is the number of classes, each populated by a number related to the likelyhood that that class is the right one. It is composed of a NLLLoss + LogSoftMax functions.

BPR stands for Bayesian Personalized Ranking [18]. It is a ranking pairwise loss function which aims at promoting positive feedback on items comparing it to negative feedbacks. Even it is possible to use it with multiple items doing pair to pair comparison we thought it was not the best solution for classification purposes.

Chapter 3

Recsys Challenge

Recsys is the premier conference for sharing new solutions in the recommender field. Every year it organizes a challenge sponsored by a different international company. The challenge is open to everybody and it is a good opportunity to try new technologies on real problems, proving their effectiveness not only in the research field, but considering the application side.

3.1 Last year LINKS participation

LINKS Foundation already took part in last year challenge [15], being able to place in the top10 of the leaderboard. The challenge was about filling song playlists for different users starting from no or very little knowledge. It was sponsored by Spotify, being one of the main platforms for music streaming. The solution developed by the team was a multiple deep learning classification approach using different features provided by the dataset (artist, songname, ...) as input which were then ensembled to obtain a better profiling and population of the empty playlists. A similar approach is considered in this thesis work but for a different problem such as the session-based prediction.

3.2 Trivago

As mentioned earlier the sponsor of this year's challenge is Trivago, which is a very famous platform for comparing hotel from different travel platforms by price, location, photos and similar features to recommend the best option to the visiting user. The nature of the service makes easy to understand that it is great for a recommender system. A typical Trivago session is as described in Figure 3.1. The user usually types in the location he wants to travel to and other optional filters such as hotel ratings, hotel type and similar or do some inspection action like opening a hotel image to see the room he will book for. Among the different actions the most important is the clickout, which redirects to the hotel host website. This is the action that creates the income to the company as the hotel platforms pay for this redirection, which increases the amount of visiting users.

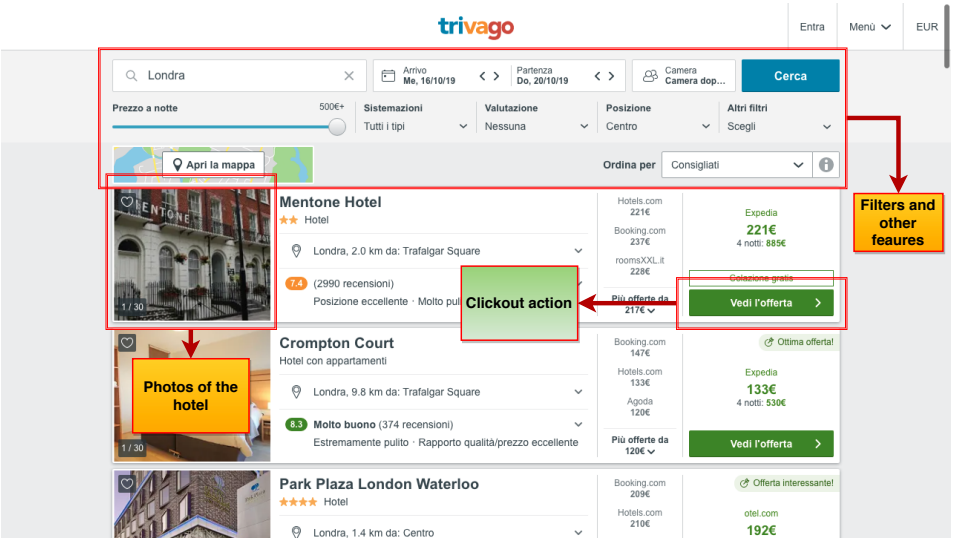

Figure 3.1: Trivago Website.

3.3 Session-based

A session-based problem is characterized by the lack (or minimal presence) of historical details about a single user, meaning that a user is not likely to use the system multiple times or we do not record his identity, this may depend on the service type or choice. The goal is to analyze the succession of actions picked from the logs and model the user preferences only by a single session information rather than his historical choices.

3.4 MRR

The metric used to evaluate the score is Mean Reciprocal Rank (MRR).

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{3.1}$$

As the ouput of the prediction is a reordered impression list, we pick as $rank_i$ the position of the actually clicked item in the recommendation_list the system gave as result.
Example:
query 1:
impressions = [100, 101, 102, 103, 104, 105]
clicked_item_id = 102
submission = [101, 103, 104, 102, 105, 100]
MRR = 0.25

Chapter 4

Dataset

In the Dataset chapter we deeply analyze the Trivago Dataset structure and statistics in order to make the right development decisions to fit its peculiarities.

4.1 Description

We used the Dataset provided by Trivago for the RecSys Challenge 2019. As described in figure 4.1 we are provided with a train and a test set, differentiated by the date in which the logs are taken.

As we can see the dataset is a collection of user sessions collected by logs taken from 3 days of activity on the Trivago platform. Each session is composed of a series of actions, each characterized by the following fields (the most relevant ones are highlighted):

- **user_id**: id of the user
- **session_id**: id of the session
- timestamp: defining the time of the interaction
- step: incremental number defining the sequence of actions in one single session

- **action_type**: it specifies the type of action performed by the user. This is one of the most important ones and it may have a value taken from the following list:
  - **clickout item**: user makes a click-out on the item and gets forwarded to a partner website, this is the most important action and it is the one to be predicted. The reference value for this action is the item_id. Other items that were displayed to the user and their associated prices are listed under the 'impressions' and 'prices' column for this action.
  - interaction item rating: user interacts with a rating or review of an item. The reference value for this action is the item id.
  - interaction item info: user interacts with item information. The reference value for this action is the item id.
  - interaction item image: user interacts with an image of an item. The reference value for this action is the item id.
  - interaction item deals: user clicks on the view more deals button. The reference value for this action is the item id.
  - change of sort order: user changes the sort order. The reference value for this action is the sort order description.
  - filter selection: user selects a filter. The reference value for this action is the filter description.
  - search for item: user searches for an accommodation. The reference value for this action is the item id.
  - search for destination: user searches for a destination. The reference value for this action is the name of the destination.
  - search for poi: user searches for a point of interest (POI). The reference value for this action is the name of the POI
- **reference**: reference value for the action, this is the second most important field as it often specifies the item_id, showing a direct correlation between user and item.

- platform: country platform used by the user
- city: name of the current city of the search context
- device: device used by the user (phone/pc)
- current_filters: list of pipe-separated filters that were active at the given timestamp
- **impressions**: list of pipe-separated items that were displayed to the user at the time of a click-out, this is very important as it always contains the item clicked by the user, the one to be predicted
- prices: list of pipe-separated prices of the items that were displayed to the user at the time of a click-out in the impressions

We are supposed to train our models on the train set and then make a submission on the official website to know the score obtained on the test set.

The test set has the last clickout action reference field set to NULL as it is the one to be predicted. We do not have access to the real value of this field.

The test set is also divided in **validation** and **confirmation** set, the former being the one on which temporary scores are computed when a submission happens, the latter being the one used for the final leaderboard.

In the image vectors are filled by a 'X' or '?' where a clickout happens, 'X' stands for a valued reference while '?' stands for a NULL one.

As showed in Figure 3.1 Trivago site is based on a search tool that shows a collection of accommodations (impressions field) according to the user preferences. As Trivago incomings depend on users clickouts on target hotel as it redirects him to the hotel website, the task here is to reorder the hotels shown on the page from most to least likely to be clicked. This translates in predicting the NULL clickout reference.

A typical user session is structured as shown in Figure 4.2. The user searches for a location on the platform and does different actions ranging from filtering to read hotels descriptions, while the most performed action is the interaction item image, as anybody is used to watch the room photos before booking it. The session eventually encounters the most important action: the clickout item, being the one leading the user to the hotel website. It usually is the last action as it means that the user made his choice, but it may occur even in the middle of a sequence.

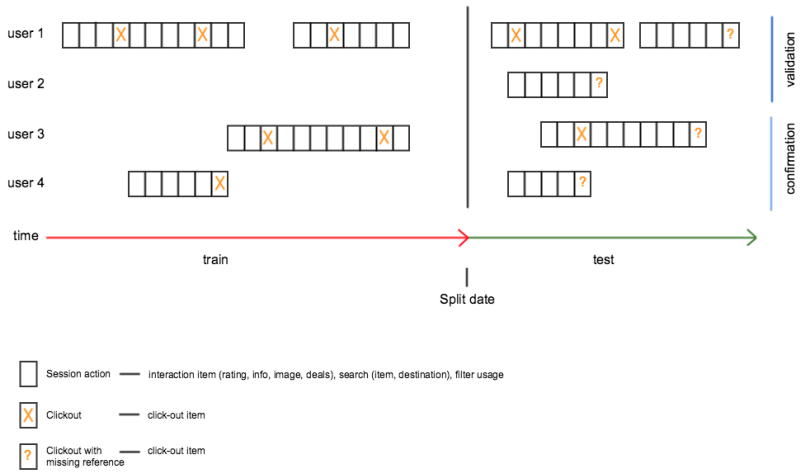

Figure 4.1: Official Trivago Dataset.

4.2 Dataset Manipulation and Cleaning

4.2.1 Filtering

As we want to find user_item interactions we filtered the dataset by keeping only actions with an item_id as reference value, meaning that the action_type must be one of the following: interaction item image, interaction item deals, clickout item, search for item.

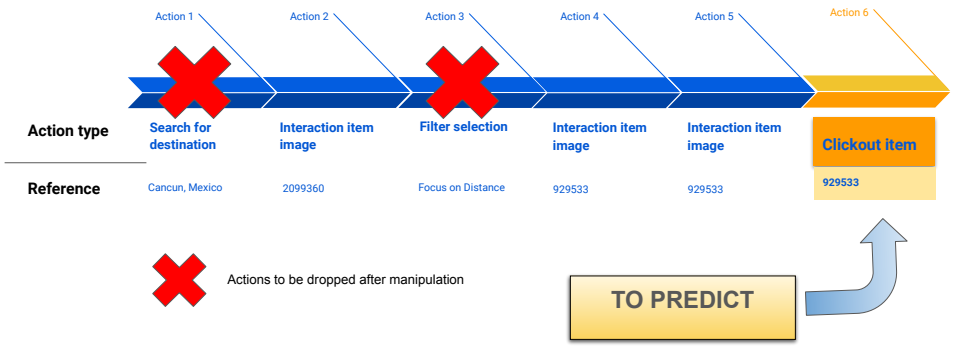

Figure 4.2: Typical user session.

4.2.2 Split

Trivago does not provide the values to be predicted in the test set, so we have to create a local test set to try solutions locally and even after the end of the challenge.

To do so we split the official train set into 2 sub sets: local train (80%) and local test (20%). We obviously set to NULL the reference of last clickout action if each session, while keeping the pre-nullified version as the local ground truth set, useful for computing the score.

4.2.3 Ensemble split

XGBoost ensemble solution requires a training phase. This requirement translates in the need of an additional training and test sets. So we split again the local training set into the so called inner and dev datasets. The different solutions to be ensembled are to be run 2 times, one for the inner dataset to provide a target to the XGBoost to be trained, and the dev dataset, which is the one to be considered for

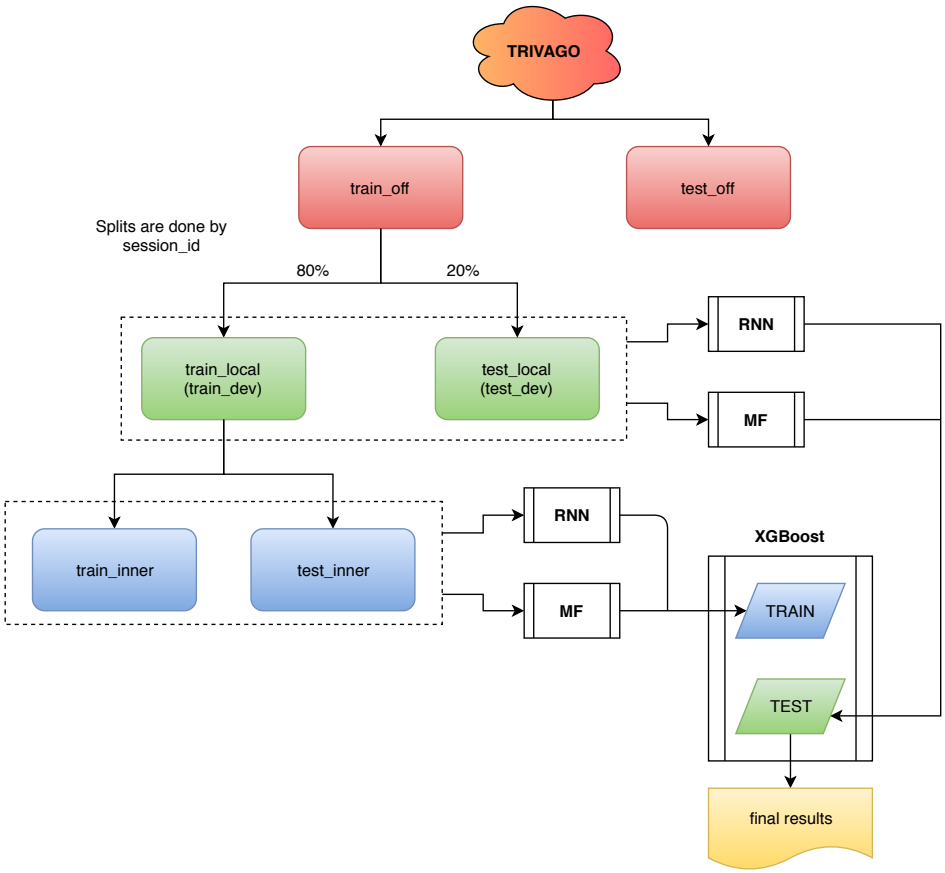

Figure 4.3: Dataset split and usage.

the real solution.

4.3 Statistics and Deductions

The first task and one of the most crucial one to do when working with a huge dataset is to analyze it so that we can think the best solution for our problem.
The provided dataset main features are described in Figure 4.4, where we highlight session and hotel number in the dev dataset, because the

former determines the number of sequences we have to process and the latter defines the output layer dimension of the neural network. The actual output layer has a dimension of **329604** values for the first solution, as the encoding takes into account also hotels that are present in the test set but not in the training set just to make sure they are encoded and can be considered a possible solution. Such a huge number of classes increases the complexity of the solution and leads to some architecture decision for time consumption problems that are discussed later on.

	inner		dev	
	train	test	train	dev
#Users	491546	135223	598385	167357
#Sessions	563567	140817	**704384**	**175944**
#Hotels	290649	116781	**322004**	**133699**

Figure 4.4: Statistical numbers of splitted dataset.

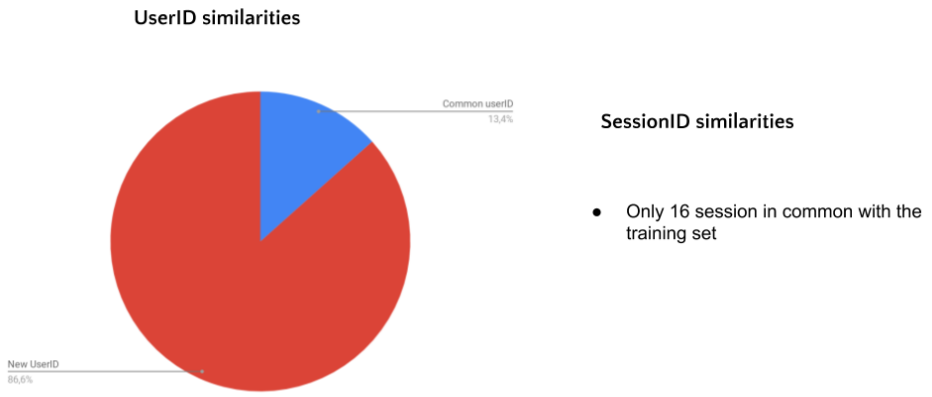

Figure 4.5: User distribution in train and test.

In Figure 4.5 we can see the of users that are in the training set

and appear again in the test set. As we notice the this number is not very high, so the choice of considering this a session-based problem and leave the history feature behind is the right one.

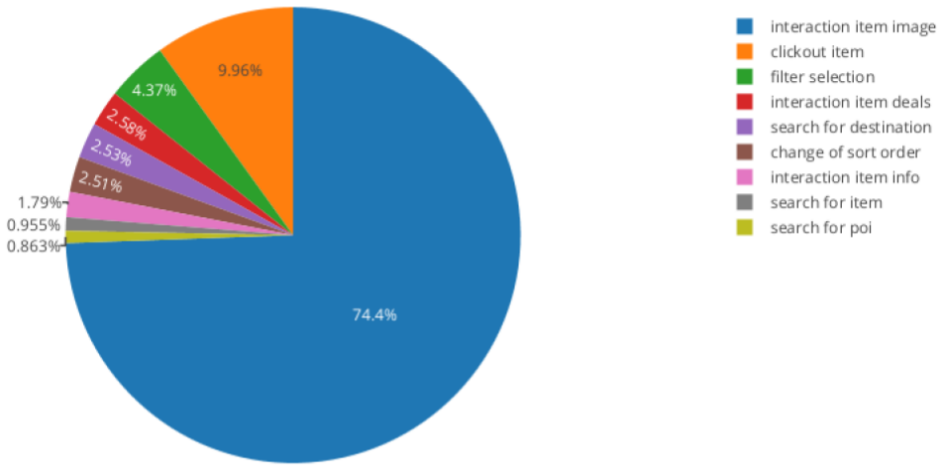

Figure 4.6: Action distribution per type.

In our solution we decide to consider only actions related to a hotel. this choice is supported by Figure 4.6 which shows the number of **actions per type**. As we can see those directly referring to a hotel are the huge majority, and we notice that 'interaction item image' actions are more than half of the dataset. It is indeed likely for a user to look at the hotels room photos before booking.

In Section 4.2 we explained how we split the dataset based on the number of actions in the session. We can visually see the 2 splitted dataset dimensions in Figure 4.7, where we highlight the ratio between **one step sessions** and longer ones. It is important to consider how big are the 2 datasets to estimate the impact of the different solutions on the final score: we can not do much prediction for one step sessions.

The choice of using GRU instead of LSTM cells is justified by Figure 4.8, where we can see that the great majority of sessions have less than 100 actions and an average of 17, meaning that long sequences

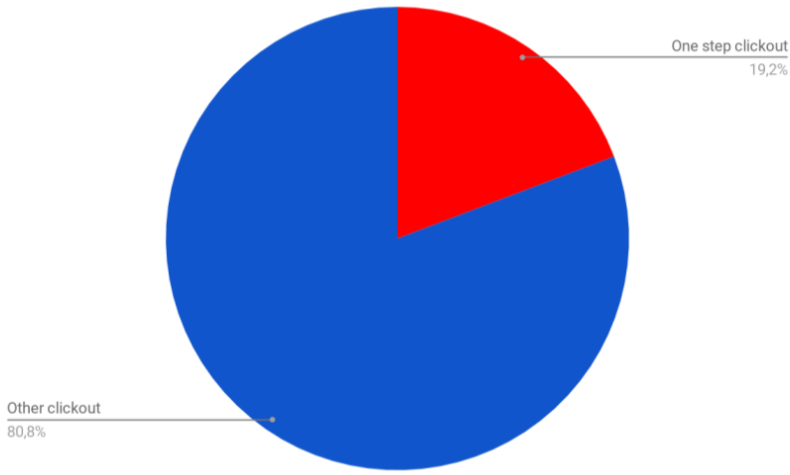

Figure 4.7: One step sessions ratio.

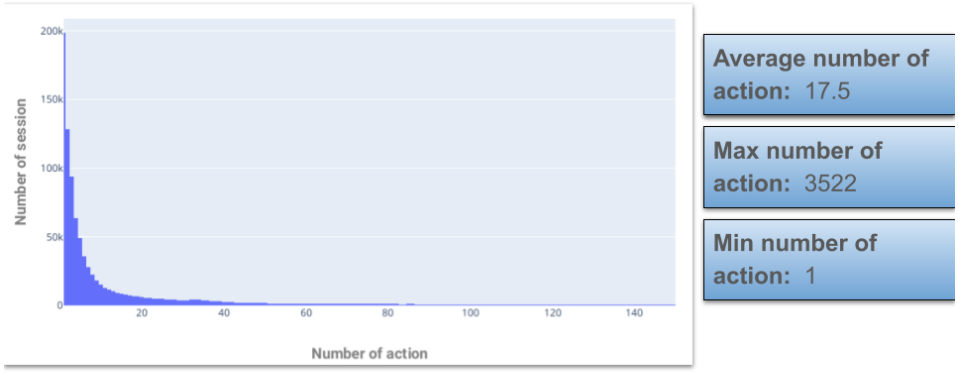

Figure 4.8: Session distribution per length.

analysis is not so crucial.

4.4 Metadata

Along with the training and test datasets we are provided with a **metadata** file, which is a csv containing a list of attributes per hotel. These attributes may include the presence of services like a swimming pool or a mini-bar, useful for characterizing the hotels. Unfortunately the number of hotels included in this file is not comparable to the total, as a result the predicting algorithm tends to always promote those hotels instead of the ones without metadata, influencing too much the final decision. For this reason we decide to exclude this file from the solution.

4.5 Official impression list

The impression list is the list of items shown to the user at the moment of the clickout. The order of this list is very important for deciding which hotel is the right one as the user is more likely to consider the items he sees first. Unfortunately this ordering can not be considered in the local set we are using as Trivago shuffled the order of all the impression lists, while in the official dataset they are ordered and it can be considered as a feature. This creates a gap between official training and test set results as the test set can get advantage of this important feature.

Chapter 5

Solution Description

In this chapter we describe the overall idea behind the solution, from the organization of the common part between the different approaches to the description of the expected results of the single parts. The main part is about the structure of the 2 RNN solutions.

5.1 Main Idea

Our idea is a **hybrid approach** to a **session-based** recommender problem. It is hybrid because we try to utilize at best the versatility of a **sequence-based** algorithm as the RNN one, complementing it with the **data-sparsity resistance** of the Matrix Factorization one, trying to overcome the problems of each solution with an **ensemble** using XGBoost library. Other hybrid attempts were made with a multi-approach solution [21] and we want to explore more of similar techniques in a different scenario.

5.2 Overall Approach

Our solution is a multi-stage process, composed by a dataset division part followed by the feature extraction performed on those separately. The 2 subsets are the cold-start subset and the main subset. The

first part is based on dividing the dataset in **Cold-Start** and **Main**, separation based on the problem characteristics. We then opt for an ensemble approach of 2 solutions we suppose to be complementary: **Matrix Factorization** and **Recurrent Neural Network**.

5.3 Cold-start subset

Cold-start is a very common problem in Recommender Systems [11], which arouses when facing the need to recommend in a scenario where we do not really have much information on user recent activity. One of the most common solutions to the problem is using user profiling or collaborative filtering, but we are facing a session-based problem where user history is not so relevant. We decide to separate those cold-start sessions in a different subset for applying a different approach.

This subset consists of sessions of length $len = 1$, meaning that they include only sessions populated by one single action, the clickout one. The lack of information makes it impossible to extract features for the user, as we also miss any kind of historical info as it is a session-based problem. This leads to use the *impression_list* as the recommendation list without any change, as it is the only information we have.

5.4 Main subset

This subset includes sessions of length $len > 1$. This is the interesting part of the dataset as we can extract features from the succession of different user-item interactions.
The solutions for this subset are breefly introduced below.

5.4.1 Matrix Factorization

The matrix factorization approach aim is to extract pure user-item interactions, promoting those items the user interacted the most. It promotes the analysis of pure association between user and item, but it lacks a consideration on the sequence structure of the session. It

is implemented with the use of LightFM[1], which is a library created to make Matrix Factorization implementations in a very simple but effective way, with the extraction of user and item metadata. The solution is based on the product between a user-feature and an item-feature matrix to obtain the best fit between every user and the clickout item. The final solution will be based on how much the user has interacted with a particular hotel.

5.4.2 RNN

The Recurrent Neural Network approach is based on the analysis of the succession of different actions to predict which will be the next item to be clicked, considering which item, in which order and how many times they are interacted with, complementing the lack of sequentiality of the MF solution. It is deeply described in Chapter 6.

5.4.3 Ensemble

The Ensemble takes the output recommendation lists and aims at producing a result which takes the best of the 2 solutions. We consider two ensemble techniques: **Bord-a** and **XGBoost**.

Bord-a is an algorithm based on ranked ordered lists. It takes as input multiple lists of the same objects ordered in different ways and ranks every item with a score related to its position in the list, in a decreasing matter. After that it assigns a score to every item summing them up and recreates a final list ordered by this final score. Unfortunately this method seems not to promote each solution strong points and is not considered in the final results.

XGboost is a gradient boosting algorithm which can be boost different features contribute in a tree based solution. We use it for the ensemble by utilizing the resulting list of recommended hotels of the 2 solutions, each item labeled with a score which ideally represents how

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

confident the algorithm is in recommending target item. As XGBoost is based on Machine Learning techniques it needs a training phase. That is why we split the dataset into inner and dev. Both solutions produce a solution list for the test_inner dataset, which is used as XGB training, while the final evaluation is done by making the prediction on the test_dev dataset and ensembling them for the final solution. All this process is clearly represented in Figure 4.3.

Chapter 6

Experimental Setup

This chapter describes the libraries and the very specific implementation of the two Recurrent Neural Network solution of our approach, enriched by the ensemble with the Matrix Factorization one.

6.1 Hactar

All the experiments we do are possible thanks to the hardware provided by HPC polito. We specifically use the Hactar cluster of the service where we have a powerful node with GPUs at our disposal, interfaced by a scheduler for parallel and automatic computation of the tasks. More details about hardware used is specified in Figure 6.1.

"Computational resources provided by hpc@polito, which is a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (http://hpc.polito.it)"

6.2 Python

We opted for python language as it is one of the easiest and versatile languages for experimenting and it is rich of different libraries for Big Data and DL applications.

Architecture	Linux Infiniband-QDR MIMD Distributed Shared-Memory Cluster
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

Figure 6.1: Hactar Hardware specifications

6.3 Pandas

We use Pandas for the manipulation of the dataset importing the csv files as Pandas Dataframe. A dataframe is a particular structure that wraps the csv file in a column/row view allowing to apply a wide variety of methods.
Among the methods used we highlight the ones we used the most:

GroupBy. It allows us to divide the dataset into groups wrapping together rows that share the same value of a field specified by the user. The most common example may be

```
df_train.groupby(['session_id'])
```

which allows us to group actions of the same session together and apply the same transformations to the whole group.

Apply. Talking applying transformation, the apply method is the one we use to apply whatever piece of code to every row of the dataset (even specifying a condition). We often use the lambda paradigm to describe the applied function. An example may be

```
df_train['reference'].apply(lambda x: x = NULL)
```

for setting all references to NULL.

We opt for this library as it is very easy to use and it is very fast in manipulating very long datasets such as the one we got for this challenge. These methods permit to modify chunks of rows in a parallel way so that it is not necessary to iterate: this greatly reduce the time needed for the pre-modification of the dataset, which is one of the most important parts.

6.4 Pytorch

The main libraries for doing Deep Learning implementations in python are **TensorFlow** and **Pytorch**. We opt for Pytorch as it is much more intuitive than its counterpart and we had some previous experience with this language, making it easier to develop an experimental setup. Pytorch is a very complete and well documented library where we can access all the basic deep learning structures, from net classes to loss functions. The main classes we use are the following.

GRU. This is the class implementing a full GRU net interface. The input of the net is composed by a tensor representing the **input** in the format

$$tensor(seq\_len, batch, input\_size) \tag{6.1}$$

- seq_len: represents every step in a single session
- batch: is for iterating among batches
- input_size: is the dimension containing the encoding of a single action

and the hidden layer h which is the output of the previous step (the first step is initialized). The output of the net is actually the hidden layer, so it is important to remember that they share the same dimension. The **output** vector has shape:

$$tensor(seq\_len, batch, hidden\_size) \tag{6.2}$$

The most important parameters for this class are:

- input_size: size of the input vector, which is the same as our w2vec encoding
- hidden_size: size of the hidden layer and the output one
- num_layers: number of RNN to stack

NLLLoss. Is the class implementing the loss function NLLLoss. The most important things to consider here are the input and output size. **Input** should have shape (N, C) where N is the number of batches and C is the number of classes of our classifier. **Output** has instead shape (N), because it ouputs a single loss value for every batch.

6.5 Encoding

The use of a Deep Learning model implies the encoding of the input data in order to be processed by the network, which means transforming the item_id into some sort of coded sequence.

1-hot encoding is the first attempt we make. This solution is unfeasible because of the huge number of different hotels: transforming every encoded item into a tensor of dimension X, where X is the number of items, is not possible in terms of memory as the input layer is too big.

So we choose to use a **w2vec** encoding as it is explained in the following section.

6.5.1 Word2Vec

Word2vec [14] is a powerful encoding tool initially developed for NLP solutions. After providing a corpus populated by lists of occurring objects, it represents every occurring object in a multi-dimensional space, minimizing the distance between similar ones. The space proximity of similar items depends on how often and how close those items appear in the corpus.
It offers 2 main methods: **Common Bag of Words (CBF)** and **Skip Gram (SG)**.

CBF aim is to take a context as input (which may be one or a collection of words) and tries to predict target word, of which we get the representation. This method is usually better for frequent words.

SG is the opposite. It takes as input a single word and gives a score probability for each word, for each context, going from the word to the context. As explained in Mikolov paper, given a collection of training words $w_1, w_2, w_3, \cdots, w_T$ this method tries to maximize the average log probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c<j<c, j\neq 0} logp(w_{t+j}|w_t) \tag{6.3}$$

referring to c as the size of the training set, the corpus. $p(w_{t+j}|w_t)$ is defined as the softmax function as

$$p(w_O|w_I) = \frac{exp(v'_{w_O}{}^{\top} v_{w_I})}{\sum_{w=1}^{W} exp(v'_w{}^{\top} v_{w_I})} \tag{6.4}$$

where v_w and v'_w are the input and output vector representations of the word w, and W is the number of different words in the vocabulary.

SG works better with less frequent words and as we learn that the dataset is composed of a huge number of items that do not occur often among different sessions, we decide to use this model.

In our implementation the **corpus** is defined as a list of lists: a collection of sessions, each composed by a succession of items, corresponding to the reference values of the actions in that particular session.

We use a model based on Gensim implementation, using the Skip-gram model with its default parameters and the following:

- embedding vector dimension = 60 (we initially try 100 but it turned out to be unfeasible in terms of memory)
- window size = 5
- min_count = 1 (for capturing every single item occurrence as items are very sparse).

6.6 Pure-Classification solution

Recurrent Neural Network is one of the most used neural network models. Its task is to extract features from a sequence of different inputs by predicting the next one. This is possible for its structure which involves a cell that memorizes the previous output as **hidden state** and concatenates it with the successive input, thus meaning that we keep memory of previous data.
We do not really need to predict every single action but only the one at the end of the sequence (session), the clickout action.

The most common implementations of RNN are **basic RNN**, **GRU** and **LSTM**. We choose to use GRU cells over the basic RNN and LSTM because the former has the vanishing gradient problem and the latter are computationally heavy and they do not have much better performance than GRU [13], especially for short sequences.

6.6.1 Input preparation

Preparing the input for the network is one of the most important parts, especially for computational matters.

The input preparation phase consists of 3 main steps:

1. remove every action unrelated to an item, as explained in section 4.2.1.
2. only keep the *reference* field of each action, as it contains the item id.
3. keep the *impression_list* of null reference clickout actions for the prediction part.

The input X is the collection of sessions, where each one is composed of a variable list of encoded items (hotels) using w2vec.

$$X = x_0^S, x_1^S, \cdots, x_n^S \tag{6.5}$$

$$x_i = h_0^w, h_1^w, \cdots, h_m^w \tag{6.6}$$

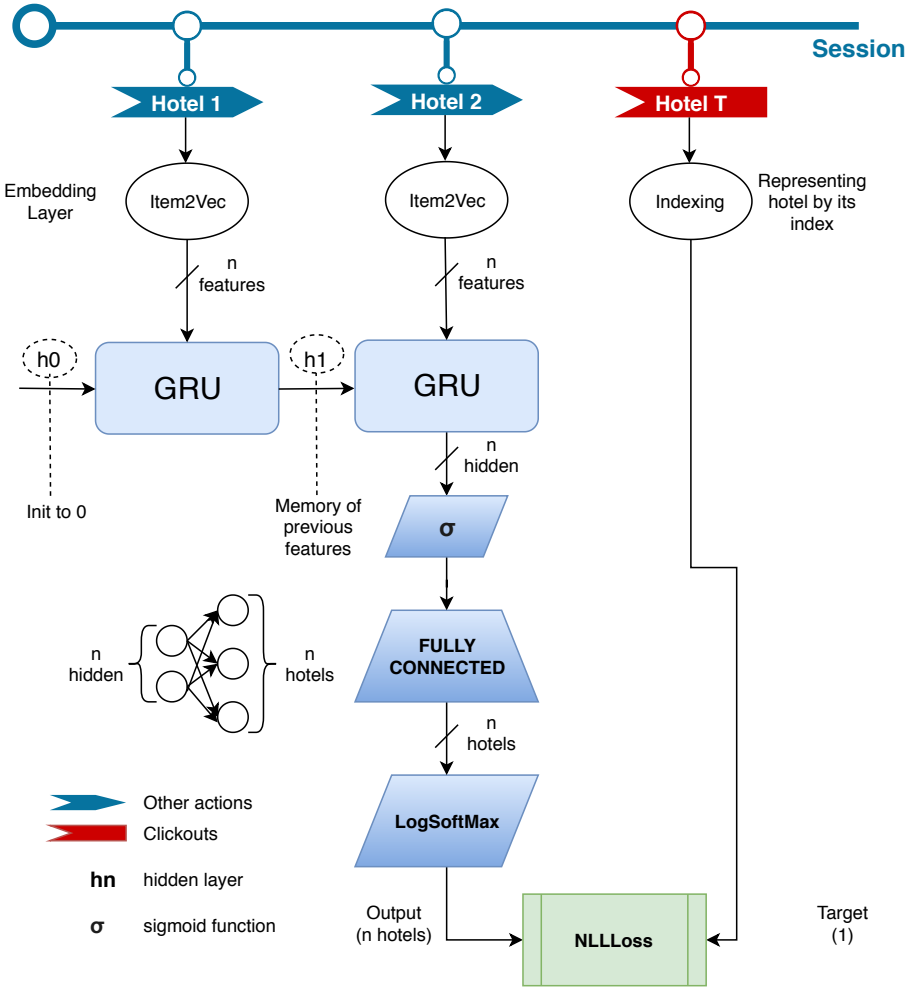

Figure 6.2: Recurrent Neural Network training architecture.

The target Y is the list containing a single item target associated to each session, encoded by its index in the hotel list.

$$Y = y_0, y_1, \cdots, y_n \tag{6.7}$$

Batches

Batching is a technique that consists in wrapping multiple input data (sequences) in the same tensor and process them together. This means that instead of backpropagating after every single sequence is processed, we only backpropagate after b input data, where b is the number of sequences in a single batch. As this operation is the most time consuming one in the learning phase of a deep model we save a lot of time, while keeping the accuracy almost intact as there is so much input data in our dataset.

Batching input data is a must for Big Data processing in Deep Learning. In our case it means that we batch together different sessions. The problem is that our sequences are not the same length, which is a requirement for those which belong to the same batch. We solve this problem by **padding** to 0 every session to the length of the longest sequence in that batch.

Computational problem

The padding of the batches arouses a problem: very few sessions are characterized by a huge number of actions, meaning that every session of the same batch is padded to the same length, wasting a lot of memory and making the problem unfeasible. This problem is solved by cutting sessions to the last 200 actions, as we empirically noticed that user's most recent actions are the most relevant ones.

6.6.2 Net Structure and Training phase

In figure 6.2 we can see the structure of the Neural Network as it goes trough the training phase. The blue line represents the user session from which we extract every hotel and encode each one using Item2Vec (implemented with W2Vec). Each encoded item is sequentially served to the GRU layer. The GRU cells carry previous item features by passing their hidden state to the next step. Before the last step (clickout action), the last hidden layer is sent to a fully connected layer. This is for expanding the latent features to get an output of size equal to the

number of hotels, assigning a *confidence* score to every single existing item. We compare our network output with the hotel target (clickout reference) using a **NLLLoss**, optimal for classification purposes. We use a sigmoid function after the GRU for activation and we use a **LogSoftMax** before the loss function to get the scores in range(-inf, 0). We opt for NLLLoss instead of CrossEntropy because the former does not include the LogSoftMax layer, thus allowing to better analyze the scores for the prediction part.

6.6.3 Loss function

The choice of the loss function for a problem structured as a classification one leads to 2 main alternatives: **NLLLoss** and **CrossEntropy**. Even if CrossEntropy usually shows better results we opt for NLLLoss as we need the output of the LogSoftMax function in order to obtain a score for every item and order them (CrossEntropy include a softmax function by deafult).

LogSoftMax

The LogSoftMax is a function that applies a softmax and computes the Log function over the result. The softmax itself takes as input an array of dimension C, where C is the number of classes and outputs an equal dimension array containing C values ranging [0, 1] which sum is equal to 1 as we can see in Figure 6.3. This output may be interpreted as the probability of the input being classified as target class.

$$LogSoftMax(x_i) = log\left(\frac{exp(x_i)}{\sum_j exp(x_i))}\right) \tag{6.8}$$

Doing the log of this output leaves us with outputs ranging *[-inf, 0]* instead, necessary process for it to be passed to the NLLLoss function.

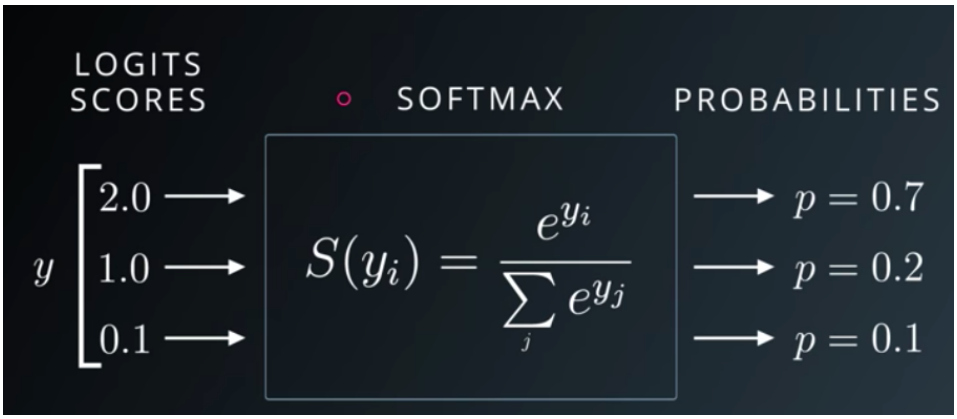

Figure 6.3: Softmax function example.

6.6.4 Test Phase: Prediction

In Figure 6.4 we can see the process for obtaining the final result in the testing phase.
The strategy for obtaining the recommendation list begins by extracting and encoding all items in the session, feeding the neural network and obtaining the confidence score for every item, promoting those sharing similar latent features with the ones present in the session.

The second step is designed to extract the impression_list field from the clickout we want to predict, because it contains the list of items the user can choose from, which we want to order. Using a map, we associate every hotel in this list with its confidence score, thus allowing us to rank it by using a simple sort to obtain the final item_recommendation list.

6.6.5 Test phase: XGBoost feed

For the XGBoost part we have to provide a result for the inner set and dev set, structured in a particular way. The XGBoost algorithm needs as input the collection of recommended hotels as the basic submission but every item has to be enriched by a score which represents the confidentiality of that particular recommended item. So we got an

additional csv output file from the net which represents the model descripted and is visually represented like the output of the GRU net in Figure 6.4. Along with the MF we also considered the order of the visited hotels as a feature for the ensemble, as this feature boosting in Matrix Factorization and similar techniques has been successfully used [10].

6.7 Last-visited solution

This section describes the second approach we try by using a classification among the **last-n** hotels the user interacted with.
The idea is to analyse the user behaviour in the session to make the neural network decide which hotel he will click among the last-n visited or if it is not present in this list, meaning that the clickout hotel will be chosen by using the impression list like in the single-clickout sessions.

The experimental setup is almost equal to the one described previously with slightly differences we are going to describe and the neural network being the same. This solution shows way better scores than the previous one as we cover in section 7.1.

6.7.1 Data extrapolation

The Last-N approaches are based on the importance of the last-visited items: we decide to focus on the last 3 hotels visited by the user by collecting an ordered list of those item's names.

When organizing our dataset and grouping it into sessions we obtain this set by taking the list of encountered hotels, cleaning it of its duplicates (keeping the last one among duplicates) and shrinking it to the last 3 elements in a reverse order. We do this for every session we are going to process.

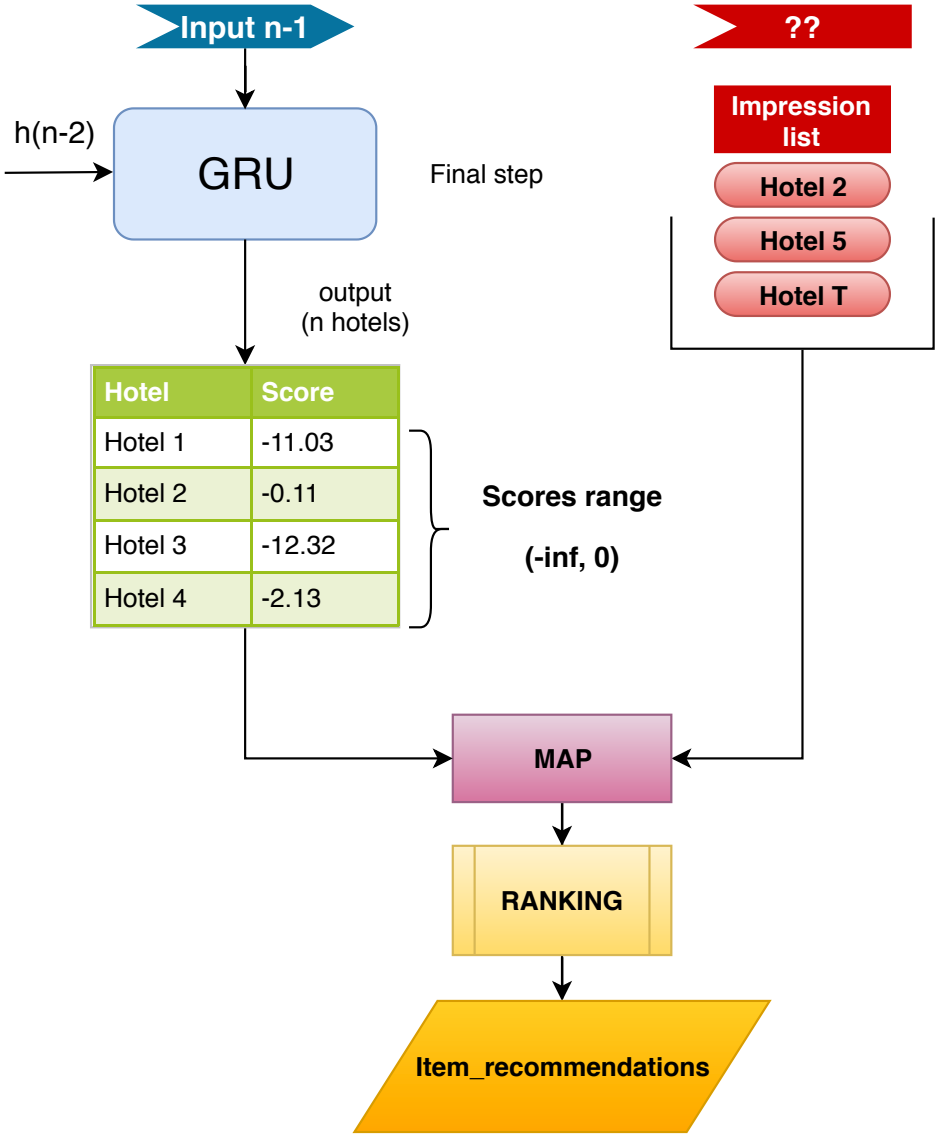

Figure 6.4: Test phase: obtaining the prediction.

6.7.2 Classification

The classification part is the main difference from the previous solution: instead of deciding directly target hotel from the hotel dictionary (which is very big) we now classify between 4 classes:

- **0**: target hotel is not among the the last-n and the recommended list is equal to the impression list
- **1**: the target is the **n** hotel
- **2**: the target is the **n - 1** hotel
- **3**: the target is the **n - 2** hotel

The training phase will be done by converting the last-n list to the classes we showed, meaning that the output encoding of a single hotel depends on the session it is considered in.

6.7.3 Prediction

In the test phase we take the output of the neural network and select the right hotel in the last-n list by using it as an index + 1, the only exception is the 0 class case where none will be taken. We then fill the rest of the recommender list by taking the remaining hotels from the *impressions_list*.

6.8 Evaluation Metrics

6.8.1 MAE/RMSE

Mean Absolute Error (MAE) and **Root Mean Squared Error (RMSE)** are used to compare expected and actual results checking how much we are doing mistakes. MAE is the most commonly used but the two methods are aiming at minimizing the value of their formulas. The difference is in the square operation of the RMSE that hugely penalizes greater errors. If we expect to evaluate N predictions and

we label with y_e the expected value and with y_r the real value, the formulas may be written as:

$$MAE = \frac{1}{N}\sum_{i=0}^{N-1}|y_e - y_r| \tag{6.9}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(expected - actual)^2} \tag{6.10}$$

6.8.2 Precision and Recall

Precision and **Recall** [3] are very famous accuracy methodologies and are very important in the recommendation field. They both make a comparison between the items we correctly recommended and another features. In precision the other factor is the number of items we are actually recommending while in recall the comparison is done with the number of items which could be recommended.

$$Precision = \frac{correctly\ recommended\ items}{recommended\ items} \tag{6.11}$$

$$Recall = \frac{correctly\ recommended\ items}{correctly\ recommendable\ items} \tag{6.12}$$

6.8.3 ROC curve

The ROC curve is a graphic used to compare 2 probabilities: **TPR, true positive rating**, and **FPR, false positive rating**. We want the 2 probabilities curve to overlap the less as possible, so that when the TPR is high the FPR is has high as possible and viceversa. This definition defines the ideal situation of a step shape in the plot but a more realistic curve is represented in Figure 6.5, where we can see that the shape tends to a step but is not perfect, while the worst case is the straight line one represented by the dashed line which divides the space equally.

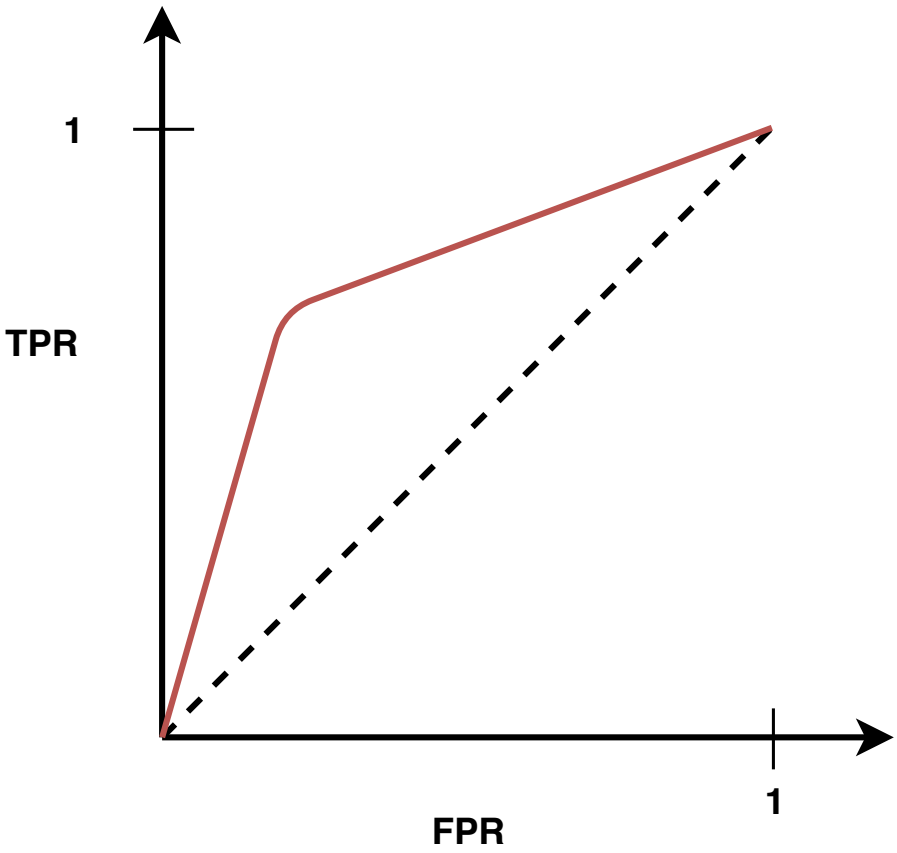

Figure 6.5: Typical ROC curve.

Chapter 7

Results

In this chapter we present the results obtained by the 2 RNN configurations and the ensemble with the Matrix Facotrization, along with our comments on them.

7.1 Pure-Classification Results

The results obtained by the first RNN solution are shown in Table 7.1. Every row shows an experiment with different tuning of the net hyperparameters, focusing on changing one at a time in order to better analyze the influence of each one to the loss function and to the final score. The tuning mainly considers Epochs, learning rate of the net, hidden dimension of the GRU.

Watching the loss plots of Figure 7.1 we can see that they all have a descending trend, even if not very constant, which is very important for understanding if the net is learning, as in this case. A few words are to be said about the 60 hidden dimension case, where the loss function seems to have a really constant descending trend, even if the final score is not the best one. This may be due to the input dimension being equal to the hidden one, leading to a better extraction of the item features as we do not explode the vector values to too many dimensions. A noticeable thing to consider is the average value of the loss value. Without considering minor oscillations, it usually reaches

values <10. Taking into account that we are using a NLLLoss it means that the classifier reduces the window of probable items to 10, which is a great result considering the total of 300k different outputs.

The best score on the test set is obtained by the attempt with a higher learning rate, maybe because the net learns much faster but in a less reliable way. The score difference is not so noticeable.

The distance between the 0.28 best score of the RNN compared to the 0.68 of the winner of the challenge is to consider. The way lower score of this network is to deduct from the solution structure and the dataset shape. The huge number of different accomodations (300k+) is an important limit as the classificator turns out to have hundreds of thousand of outputs to be considered every time we have to decide target hotel.

The most important limit is in the session's structure: items usually occur in a very few number of different sessions, meaning that the encoding of similar hotels is very limited compared to the NLP case, where less words occur multiple times in a document and relations are easier to be defined.

Name	**Epochs**	**W2Vec**	**L. rate**	**Hidden dim**	**MRR(dev)**
s_1	150	60	0.001	100	0.268952
s_2	**200**	60	0.001	100	0.271442
s_3	150	60	**0.005**	100	**0.28277**
s_4	150	60	0.001	**60**	0.262958

Table 7.1: RNN parameters tuning.

7.2 Last-N Results

The Last-N RNN solution results are shown in Table 7.2. The first thing we can notice is the huge increase in the overall attempts with respect to the previously described solution, due to the very important influence of the last visited items feature. A bit of tuning is made

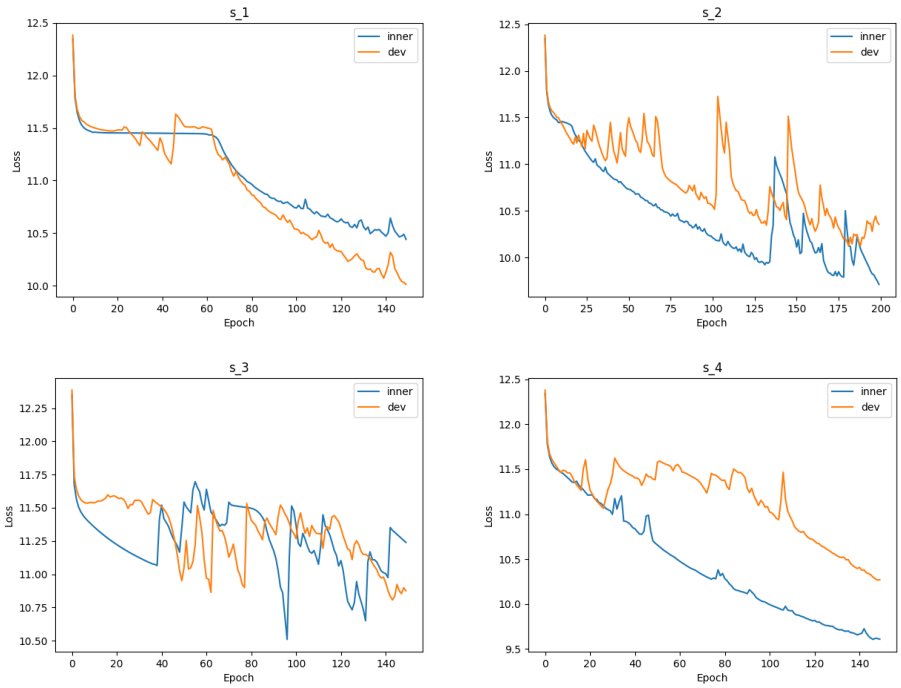

Figure 7.1: First solution loss plots.

after observing that the loss function is rapidly saturating because of the way lower number of output classes (4). This may indicate an overfitting problem , so we decide to lower the learning rate and, more importantly, the number of epochs. With an increase from 0.48 to 0.54 the overfitting hypothesis seems to be the right one, leading to the best score with this solution. We also try to decrease the encoding number of features to 30 to check if it was not necessary to have that many for a good result, saving a lot of computing resources. Watching the loss plots of Figure 7.2 we can see that the 3rd and 4th attempt seem to have the best loss shape, while the first 2 are less constant, meaning the assumptions we make are correct. It is worth to notice that the score does drop with this change, but the difference seems not so noticeable: from 0.5404 to 0.5359.

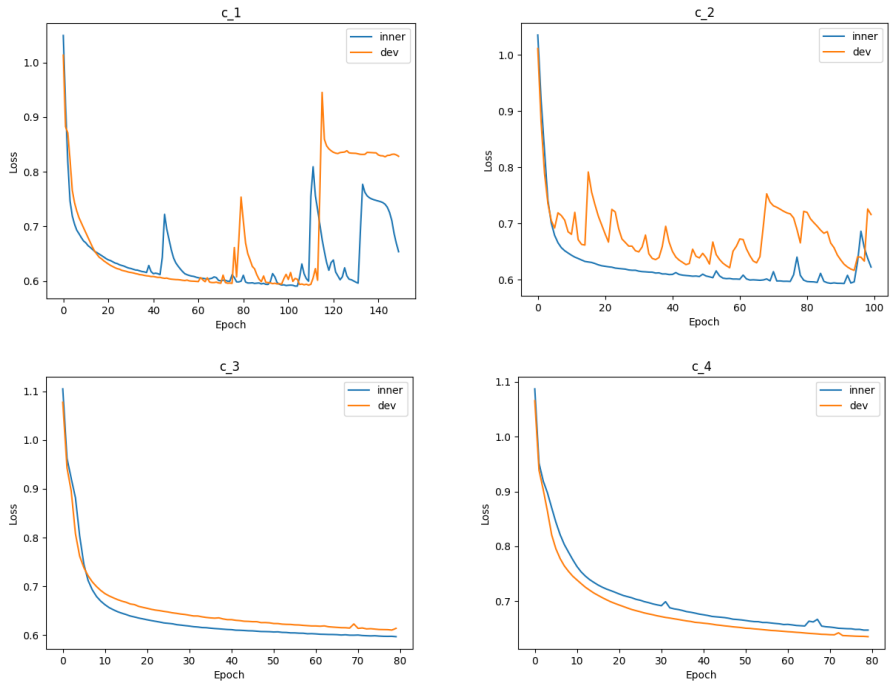

Figure 7.2: Last-n solution loss plots.

Name	Epochs	W2Vec	L. rate	Hidden dim	MRR(dev)
c_1	150	60	0.001	100	0.4823
c_2	150	60	**0.0005**	100	0.5184
c_3	**80**	60	**0.0005**	100	**0.5404**
c_4	150	60	**0.0005**	**60**	0.5359

Table 7.2: Last-n RNN results.

7.3 Solutions breakdown

The first solution we developed was based on a classification over 300k+ and experiment showed that this approach did not lead to good

score results. We think that this was not mainly due to the huge number of classes but to the dataset features, as hotels appeared to be very sparse and it was difficult for the net to learn shared features where there were few or any. Indeed the items occurred in 1 or very few sessions, precluding us from associating them with a suitable number of other items. We can assume that this kind of solution is not proper for a dataset shaped like that.

The second solution classification was done on a way smaller number of classes, but the crucial aspect was considering as a key feature the importance of the last visited items. We saw a a huge score improvement with respect to the previous solution and this is due to the use of a yet powerful feature. Indeed we already know that recommending the last visited hotel leads to a good score, but we use the RNN to analyse the session structure and discriminate among the n last hotels to obtain better results.

7.4 Ensemble results

For the ensemble we use XGBoost to match the best attempts of the 2 RNN solutions separately ($MF + RNN_1$ and $MF + RNN_2$), while we have a third case showing the score of the MF + features (MF), excluding the deep learning approach to better compare the results.

In Table 7.3 we can see the obtained results. At first glance we can see that the best score is obtained by the Pure-Classification solution, with a gain of 0.005 from the baseline score of the MF alone. The little gain may be due to the hypothesis of the sequence analysis of the RNN we did, which is extracted by the XGBoost algorithm and added as contribute to the MF solution. Even if the gain is not so relatively high, the slightest gain is important as we saw by the top10 leaderboard scores which differed by very little deltas. In Figure 7.3 we can see the importance of every feature given as input to the XGBoost in the decision tree: score_rnn and score_mf are the fields describing the 2 solutions. The other features we see (user_bias, item_bias, ...) are outputs taken from the LightFM algorithm of the MF solution.

As XGBoost is a tree based algorithm, the F score is based on how many times that features is considered in the decision making of hte algorithm, which may be considered as the importance of that feature. An interesting consideration is the huge contribute of the rnn algorithm despite the low score of the solution by itself. A deeper consideration is to be done on the recent_index feature, which is created by giving a score to the last-n visited items in the session, the bigger the closer to the end of the session. This feature is not complementary with the idea behind the Last-N RNN as they are based on the same concept, that is why the baseline does not increase. Anyway if we consider the second solution we go from 0.5404 to 0.5980 (+0.0576).

Setup	**MRR**	**Offset**
MF	0.598043	0
MF + RNN_1	**0.602774**	**+0.0576**
MF + RNN_2	0.598029	**-0.000014**

Table 7.3: Ensemble results.

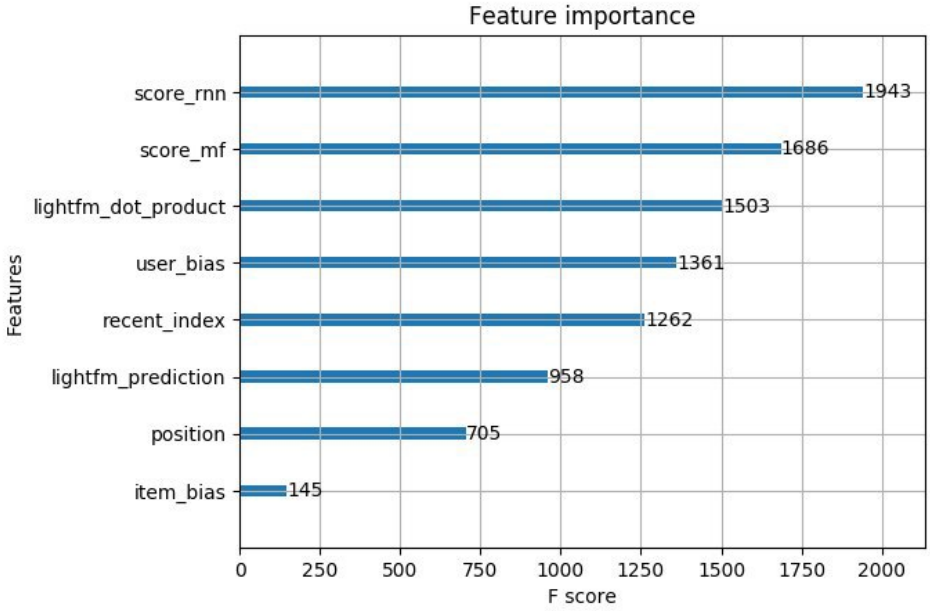

Figure 7.3: XGBoost features importance for MF + RNN_1.

Chapter 8

Conclusions and Future Work

In this thesis we presented our RNN solution for the Recsys 2019 challenge and obtained different results from different approaches. We introduced the world of recommender systems showing examples of fields where they are being used nowadays. We then described how they have been defined through the years and the different technologies that have been experimented for solving recommending problems in different ways.

Starting from this knowledge we tried our solution for a specific problem belonging to the session-based category, practically experimenting on a dataset provided by Trivago for the Recsys challenge 2019, which consists in a collection of user's sessions on the web site with the goal of predicting which hotel item the user will click (click-out). Our proposed solution was a hybrid approach composed of a Matrix Factorization to exploit the user-item direct relation and an RNN approach to analyze the sequence contribute of the dataset, then ensembled using the XGBoost algorithm.

We experimented 2 different RNN solutions: the Pure-Classification one, which tries to select the right item in the whole list of existing ones, and the Last-N, which classifies among the last visited items.

As expected the results for the first solution showed that an RNN approach based on a pure classification does not get good results when facing such a sparse problem when compared to other ones, but the goal of this solution was to complement the lack of the MF solution with its study on the sequence features. For what concerns the second solution we observed that the training did not need much time to train as the number of classes is very little, resulting in a very clear loss shape with a little bit of tuning. The score is much higher that the first solution as the last visited items are very likely to be those the user is more interested to. Even if the score is of the solution is not close to the leaderboard winners, the importance is not only in the score obtained for the challenge dataset but more importantly in the behaviour of such technology when facing a session-based recommender problem.

This experiment showed that extracting latent features from a session based scenario is not an easy task, but more importantly the sparsity of a such a huge number of items along with the lack of additional informations on them makes a prediction very difficult for a Deep Learning Approach. The last-visited items confirm to be very important in the final user decision, despite the solution they are included into. The need of an hybrid approach for these problems seems necessary to overcome these lacks.

8.1 Future Works

For what concerns the Neural Network, trying to create or use new embedding solutions for such a sparse problem might be a possible solution. New concepts are being created like BERT [4] or Attention and we could experiment them in contexts different from NLP. Also thinking of a way to include item features even if they are limited to a bunch of items might be a nice boost. The dataset itself presented minor information about the actions that seemed not so relevant as the main focus was on the item itself, but thinking a way to include those minor contributes for a better learning of the net might be relevant. The net structure could be enriched with parallel nets extracting

features from different fields, then finding a smart way to merge all the knowledge to obtain the best result. The XGBoost algorithm has been used to boost the contribute of dataset features, but experimenting it by merging those parallel RNN is a possible next step. It could be also interesting to slightly modify the net to test the solution on different losses which are good for ranking problems (BPR).

For what concerns the complementarity of sequence and user-item, a good point for the future is the need to go deeply into different solutions to overcome the difficult task of predicting items in a scenario where we lack history and items are so sparse among user sessions.

Appendix A

Parameters

A command line interface is set to test our solution with different parameters. The parameters specification is defined as follows.

python3 ./setup.py {parameters}

- –traininner: training set in a csv format
- –testinner: test set in a csv format
- –gtinner: ground truth set in a csv format, it is basically the test set with non-NULL reference for clickouts. It is needed for score computation
- –hiddendim: hidden dimension for the Recurrent Neural Network, more hidden nodes usually correspond to deeply analisys but slower computation time
- –epochs: number of epochs for the training phase. A higher number means the net goes through the dataset more times, meaning it has more opportunity to learn and improve the score. This is true if we do not fall into overfitting.
- –ncomponents: number of dimensions in which we represent the items using the W2Vec algorithm. A high number quickly escalates to very high RAM usage.

- –window: length of the window of words considered together by the W2Vec algorithm
- –learnrate: learning rate for the training phase, it tunes the weights update. This is a very delicate parameter as a low number may cause the net to learn anything and a high number may cause it to learn so quickly that it is unstable and unreliable.
- –iscuda: if specified it enables the use of GPU instead of doing computation only on CPU. This parameter is always used cause computation time may become unfeasible otherwise.
- –subname: name for the submission files. It is put at the start of every output file so we can also speficy a subfolder.
- –batchsize: batch size in the training phase. It allows a nice computation time reduction as backpropagation is a slow process and batches allow to do that only every x sequences instead of every time.

Bibliography

[1] 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. URL: http://doi.acm.org/10.1145/2939672.2939785.

[2] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[3] Jesse Davis and Mark Goadrich. "The Relationship Between Precision-Recall and ROC Curves". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: ACM, 2006, pp. 233–240. ISBN: 1-59593-383-2. DOI: 10.1145/1143844.1143874. URL: http://doi.acm.org/10.1145/1143844.1143874.

[4] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. arXiv: 1810.04805 [cs.CL].

[5] Balázs Hidasi and Alexandros Karatzoglou. "Recurrent Neural Networks with Top-k Gains for Session-based Recommendations". In: *CoRR* abs/1706.03847 (2017). arXiv: 1706.03847. URL: http://arxiv.org/abs/1706.03847.

[6] Balázs Hidasi et al. “Parallel Recurrent Neural Network Architectures for Feature-rich Session-based Recommendations”. In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys ’16. Boston, Massachusetts, USA: ACM, 2016, pp. 241–248. ISBN: 978-1-4503-4035-9. DOI: 10.1145/2959100.2959167. URL: http://doi.acm.org/10.1145/2959100.2959167.

[7] Balázs Hidasi et al. *Session-based Recommendations with Recurrent Neural Networks*. 2015. arXiv: 1511.06939 [cs.LG].

[8] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. “An Empirical Exploration of Recurrent Network Architectures”. In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, 2015, pp. 2342–2350. URL: http://dl.acm.org/citation.cfm?id=3045118.3045367.

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

[10] Maciej Kula. *Metadata Embeddings for User and Item Cold-start Recommendations*. 2015. arXiv: 1507.08439 [cs.IR].

[11] Xuan Nhat Lam et al. “Addressing Cold-start Problem in Recommendation Systems”. In: *Proceedings of the 2Nd International Conference on Ubiquitous Information Management and Communication*. ICUIMC ’08. Suwon, Korea: ACM, 2008, pp. 208–211. ISBN: 978-1-59593-993-7. DOI: 10.1145/1352793.1352837. URL: http://doi.acm.org/10.1145/1352793.1352837.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[13] Dongqing Liu and Gurbir Singh. “A Recurrent Neural Network Based Recommendation System”. In: 2016.

[14] Tomas Mikolov et al. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[15] 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. URL: http://doi.acm.org/10.1145/3267471.3267484.

[16] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning*. 2013, pp. 1310–1318.

[17] Massimo Quadrana et al. "Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks". In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. RecSys '17. Como, Italy: ACM, 2017, pp. 130–137. ISBN: 978-1-4503-4652-8. DOI: 10.1145/3109859.3109896. URL: http://doi.acm.org/10.1145/3109859.3109896.

[18] 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. URL: http://dl.acm.org/citation.cfm?id=1795114.1795167.

[19] Guy Shani, Ronen I. Brafman, and David Heckerman. "An MDP-based Recommender System". In: *CoRR* abs/1301.0600 (2013). arXiv: 1301.0600. URL: http://arxiv.org/abs/1301.0600.

[20] Guy Shani and Asela Gunawardana. "Evaluating Recommendation Systems". In: vol. 12. Jan. 2011, pp. 257–297. DOI: 10.1007/978-0-387-85820-3_8.

[21] Bartlomiej Twardowski. "Modelling Contextual Information in Session-Aware Recommender Systems with Neural Networks". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys '16. Boston, Massachusetts, USA: ACM, 2016, pp. 273–276. ISBN: 978-1-4503-4035-9. DOI: 10.1145/2959100.

2959162. URL: http://doi.acm.org/10.1145/2959100.2959162.

[22] Bo Xiao and Izak Benbasat. "E-commerce Product Recommendation Agents: Use, Characteristics, and Impact". In: *MIS Q.* 31.1 (Mar. 2007), pp. 137–209. ISSN: 0276-7783. URL: http://dl.acm.org/citation.cfm?id=2017327.2017335.