Ref: 2020/ II3 / RSR July 2020

Manouba University
National School of Computer Science

A THESIS SUBMITTED FOR THE DEGREE
OF
COMPUTER SCIENCE AND ENGINEERING

Revenue Management System Usage Recommendation & Insights

By
YAHYAOUI ILYES

Host Organization : INFOR

Academic Supervisor : *Mr. TAGINA Moncef*
Technical Manager : *Mr. NOURI Marwen*
Technical Supervisor : *Ms. SOUDANI Amira*
Address : Rue du lac Tchad, Tunis, 1053, TN
Phone : +216 22 569 167

Appreciations & Signatures

National School of Computer Science

Infor

PREDICTIX TUNISIA
(INFOR Company)
Département des Ressources Humaines
Imm. Haifa rue lac Tchad
Les Berges du lac 1053
Tél 216 31 393 931

Despite the complexity of the task and the hardships imposed by the pandamic situation, you were able to deliver an exceptionally high-quality work. You exceeded all of our expectations and took hard work and commitment to a whole new level.
Continue to inspire and soar higher.

Acknowledgement

Prima facie, I wish to express my sincere gratitude to my family for the support and great love. They kept me going on and this work would not have been possible without their contribution. Also, I wish to express my sincere appreciation to my friends who have encouraged me along the way and my fellow colleagues for their feedback and cooperation. Many Thanks!

I would like to pay my special regards to ***Mr. Nouri Marwen***, my technical manager, for the precious guidance and the high quality supervision he provided. Likewise, I would like to thank my technical supervisor, ***Ms. Soudani Amira***, for her patience and support in overcoming numerous obstacles I have been facing through this work. Also, I would like to recognize the invaluable assistance and support that ***Mr. Tagina Moncef***, my academic supervisor, provided me with.

I wish to thank all of my co-workers at Infor especially the Hospitality team members. Your assistance and valuable advice were a milestone in the completion of this project. I am indebted to you! I would also like to thank all of my teachers at the National School of Computer Science for their continuous support and invaluable training during my study years.

Finally, a special thanks to the members of the jury who honored me by examining and evaluating this modest work.

Contents

List of Figures

List of Tables

Glossary of Acronyms

RMS : Revenue Management System.

RS : Recommendation System.

KBRS : Knowledge-Based Recommendation System.

CFRS : Collaborative Filtering Recommendation System.

CBRS : Content-Based Recommendation System.

HRS : Hybrid Recommendation System.

SBRS : Session-Based Recommendation System.

RM : Revenue Manager.

AU : Access User.

MHU : Multiple Hotel User.

SHU : Single Hotel User.

MHRM : Multiple Hotel Revenue Manager.

SHRM : Single Hotel Revenue Manager.

MHAU : Multiple Hotel Access User.

SHAU : Single Hotel Access User.

SPOP : Session Popularity.

I-KNN : Item K-Nearest Neighbours.

FPMC : Factorizing Personalized Markov Chains.

RNN : Recurrent Neural Network.

LSTM : Long Short Term Memory.

GRU : Gated Recurrent Units.

GRU4REC : Gated Recurrent Units for Recommendations.

UML : Unified Modeling Language.

SQL : Structured Query Language.

Introduction

Every aspect of our lives has become data-enabled and beyond this exponential growth of data, lie various challenges. Therefore, reliable tools to handle this overflow of information were needed as the already existing ones were crumbling. Prior to the birth of recommendation systems, users heavily relied on search engines. However, even though the latter were quite effective, they lacked efficiency since consumers had to put a lot of effort in order to filter out unwanted content. This lack of efficiency was the driving force for the rise of recommendation systems that required no explicit input from users and provided them with content before even asking for it.

Recommendation systems' market worth was quickly recognized and its research has bloomed since. Studies have proven that such systems provide the ability to skyrocket businesses and ensure stability within the consuming population. So nowadays, nearly every website embeds such engines whether to optimize sales or enhance the user experience. The latter use cases fall directly under scope of this work since Infor, our host organization, is facing fierce competition from new revenue management systems, supporting such feature, being introduced into the market. The American giant has been omnipresent in the hospitality industry. However, recently, it got threatened by new emerging products with much better user experience. Our contribution will be part of the company's strategy to keep up with the fluctuating market needs as we will be building a recommendation module for Infor's revenue management system: EzRMS.

This document is organized in seven chapters. It represents a synthesis of the different phases this project has been through.

In the first chapter, entitled "General Context" , we will expose the general context of this work in order to better frame the problem we are facing. Also, we will present the methodology we adapted in order to reach the expected results.
As for the second chapter, it will be devoted to discuss the state of the art of algorithms in this field and to conduct a study of the existing solutions in order to get better insights on what lies ahead.
In the "Requirement Analysis & Specification", the third chapter, we will present the functional requirements of the system along with its software attributes. In addition, we will expose the development life cycle of our application from two different perspectives: a machine learning practitioner's and a software engineer's.
Furthermore, the fourth chapter will be dedicated to exposing the design of our solution. This chapter serves as a blueprint that specifies details of the recommender's architecture.
The three next chapters fall under the umbrella of the Achievement phase. The "Insights" chapter will cover the conducted exploratory data analysis that provided valuable input for the advancement of this work by exposing behavior trends and hidden patterns.
As for the sixth chapter, "Recommendation", it will address the adapted approach and expose the models we will be implementing in order to solve the recommendation task.
Concerning the "Evaluation" chapter, it will present the achieved results and provide some interpretations explaining why we are observing such results.
Finally, we conclude this work with the future prospects and possible improvements for what has been achieved so far.

Chapter 1

General Context

1.1 Introduction

This chapter will provide a global overview on our project. It will discuss the different contexts of this work and will clearly present its relevance. As a first step, we will expose the academic context. Then, we will move on to the professional one while presenting fundamental key concepts. Also, this chapter will include the problem formulation, the expected results and will explain the methodology adapted during this work.

1.2 Academic context

This project was elaborated in the form of an end-of-studies internship which is the final assessment required in order to obtain the National Degree in Computer Science and Engineering. The internship took place between February 11th and June 10th, 2020, during which we integrated Infor's Hospitality Data Science team.

1.3 Host organization

Infor is a multi-national enterprise software company, headquartered in New York City, United States. It focuses on business applications for organizations delivered via cloud computing as a service. Infor solutions cover various areas ranging from financial systems and enterprise resource planning (ERP) to supply chain and customer

relationship management with over 68,000 organizations worldwide relying on Infor to help overcome market disruptions and achieve business-wide digital transformation. This internship was hosted by Infor's branch in Tunisia that nowadays employs over 100 engineer amongst which are data scientists, data engineers, software developers and cloud architecture consultants.

1.4 Main concepts

In this section, we are going to introduce general concepts related to the hospitality business which is the application domain of our project.

1.4.1 Hospitality industry

As a broad concept, the hospitality industry refers to a variety of businesses and services linked to leisure and customer satisfaction with an exceptional focus on luxury, enjoyment and experiences. In our context, hospitality refers to any kind of services that hotels provide their customers with.

1.4.2 Revenue management

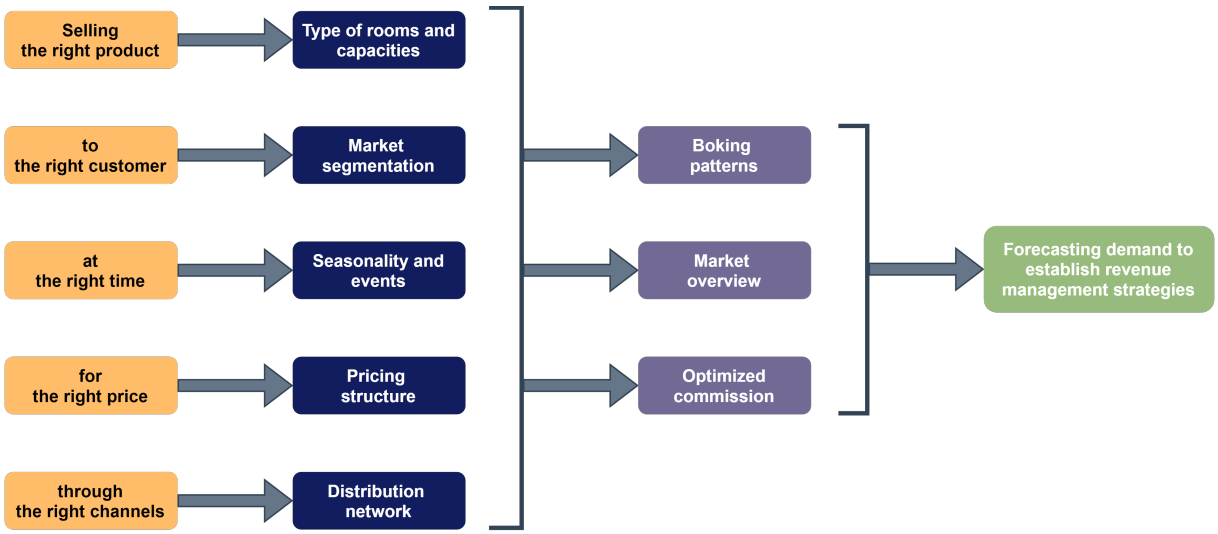

Figure 1.1: **Selling the right product to the right customer at the right time for the right price through the right channels.**

Revenue management is a key concept within the hospitality industry. It is the application of analytics in order to predict the consumer behaviour and optimize product availability. Hotels face various challenges like the perishable inventory, the varying levels of demand or fluctuating prices. Revenue management helps tackle those challenges and provide methods and strategies in order to maximize profit. The essence of this discipline is "***selling the right product to the right customer at the right time for the right price through the right channels***". As shown in the Figure 1.1, revenue management is a complex task that requires precisely answering multiple questions: what to sell? to whom? when to sell? for how much? through what channels? Precisely answering those questions will provide the manager with a perspective that allows him to extract valuable insights from booking patterns and market trends, and thus optimize earnings. This would certainly be a nightmare if the manager had to satisfy all of those tasks by himself especially with the competitiveness of the market and its fluctuations. Therefore, Revenue Management Systems were conceived in order to make it much easier to forecast demand and establish strategies.

1.4.3 Revenue management system

A Revenue Management System, or for short RMS, is a software solution that boosts the efficiency and effectiveness of revenue management tasks. It will make use of the hotel's data, and the market's data in a larger scope, in order to provide the best opportunities and the most adequate strategies for the current situation. By doing so, RMS helps with decision making and future planning. Before coming to the picture, revenue managers had to work on satisfying the business' requirements by trend or by relying on their personal experience. Revenue management systems brought the factor of certainty and denied the existence of the ineffective strategies by looking at the bigger picture and including multiple internal and external variables that affect the business.

1.5 Context of the Project

In this section we will present EzRMS, the issues it faces, our solution's expected results and the methodology we adapted.

1.5.1 Infor EzRMS

Infor hospitality solutions offer highly personalized strategies that fit each hotelier's unique criterion. They provide an in-depth understanding of the customers' needs, and thus an effective and efficient decision making. Among those abundant solutions, comes EzRMS, which stands for "***Easy Revenue Management System***". It is a powerful revenue management software with capabilities to manage all aspects of hotel business, from walk-in to long stays. EzRMS forecasts demand for hotel rooms, and recommends the appropriate selling strategies, such as open/close rates, stay controls, open/close room categories, and overbooking levels. EzRMS is powered by Machine learning algorithms that recognize patterns dynamically to insure the most accurate business forecasts and pricing recommendations.

1.5.2 Problem formulation

Infor EzRMS helps hotels optimize revenue by automatically forecasting demand and recommending appropriate sales strategies based on pre-defined criteria and dynamic algorithms. EzRMS generates recommendations regarding prices, schedules, and associated offers for each hotel. Smoothly presenting such number of features has its own challenges and the user can spend a lot of time exploring the interface rather than efficiently using it. This is especially the case for new users that did not receive any prior training or haven't read the user manual guide. Due to the complexity of the interface, EzRMS is not used at its full potential and with other competitor products launching in the market a decision to enhance the whole interface and integrate a Recommendation System (RS) has been made.

1.5.3 Expected results

The goal of this work is to develop a RS for Infor EzRMS. This system will help with navigation as well as decision making by providing pertinent recommendations. The RS will offer an expert level guidance and minimize the user's effort: during each session, the system shall capture the user's intent, process it and offer recommendations that continuously evolve with the session's evolution and the user's interest drifts. The Figure 1.2 is an illustration of Infor EzRMS ' interface exactly as presented for a *Hoxton,*

Williamsburg hotel staff member. The red square section is how we expect to present the output of our RS.

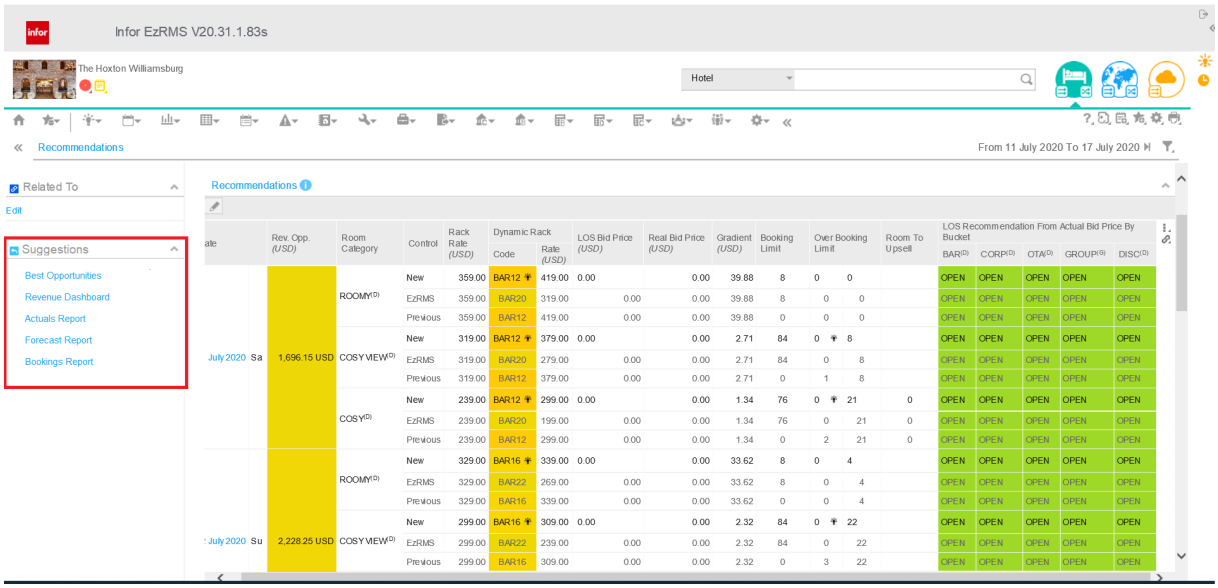

Figure 1.2: **Infor EzRMS' interface.**

1.6 Methodology

This project started by getting deeper insights on the Hospitality industry and the Revenue Management Systems. Then, we moved on to discovering the solution that Infor provides and its shortcomings. After that, we proceeded by getting a deeper understanding of the application domain and shortlisting the features required in order to overcome the EzRMS's shortcomings. Afterwards, we moved on to the research and development phase where we explored and adapted existing solutions related to our project's goals and thoughtfully worked to optimize their outcomes.

1.7 Conclusion

This chapter included an overview for the general and specific contexts of the project. Also, it addressed the problem we wish to solve, what's expected to be achieved and how we proceeded in order to reach the desired outcome.

Chapter 2

State of the art

2.1 Introduction

This chapter will discuss the state of the art of algorithms revolving around this work and provide a comparison between them. In addition, a study of the existing solutions will be conducted in order to provide further insights.

2.2 Recommendation Systems

The Web plays a fundamental role for business transactions and information exchange leading to an overflow of information and a serious need for tools to enhance the navigation experience. This was the driving force for creating Recommendation Systems. The main idea behind these systems is to bring information to the user before he even asks for it. Predicting the intent and recommending relevant elements not only enhances the user experience but also can lead to major increase in revenue especially for the e-commerce-like businesses. Unfortunately, building reliable RS is not an easy task because we need to track the user's past interactions, understand his preferences and adapt to his context and sudden interest drifts. Tracking preferences is achieved by collecting feedback data that can exist in two forms: *implicit* or *explicit*. Implicit feedback is quite abundant and in real life cases this is what we heavily rely on. Implicit feedback can be inferred from where, when and how items were consumed. A simple definition would be that this is the meta-data that the user generates. As for explicit feedback, a straight forward example would be movie ratings, comments on

posts, likes or shares. The challenge here is that users are not always willing to put much effort to satisfy such contextual data making this kind of feedback quite rare.

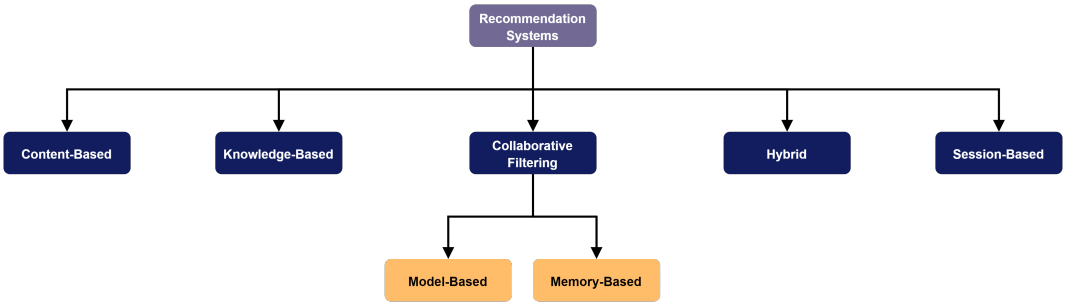

Figure 2.1: **Types of Recommendation Systems**

Obviously, in order to offer relevant recommendations, we need to exploit our user data via certain models. The literature classifies these models based on the algorithmic way they solve the recommendation task. As shown in the Figure 2.1, we can distinguish 5 types of RS: Collaborative Filtering Recommendation System (CFRS), Content-Based Recommendation System (CBRS), Knowledge-Based Recommendation System (KBRS), Hybrid Recommendation System (HRS) and Session-Based Recommendation System (SBRS).

2.2.1 Collaborative Filtering Recommendation System

CFRS is a model based solely on the past interactions recorded between users and items in order to produce new recommendations. These interactions, also called feedback, are stored in a user-item matrix. This approach finds the collaborative traits in the feedback data and builds the model upon those characteristics. Mainly, there are two sub-categories of CFRS: Memory-Based and Model-Based.

2.2.1.1 Memory-Based

The Memory-Based approach directly works with values of recorded interactions, assuming no model. It is essentially based on a nearest neighbours' search (e.g. find the closest users to the user of interest and suggest to him the most popular items among those neighbours). We can distinguish two types in this sub-category of CFRS. The first one is called User-Based CFRS (also known as the user-user approach). It is based on the logic of "***Users who are similar to you also liked ...***".

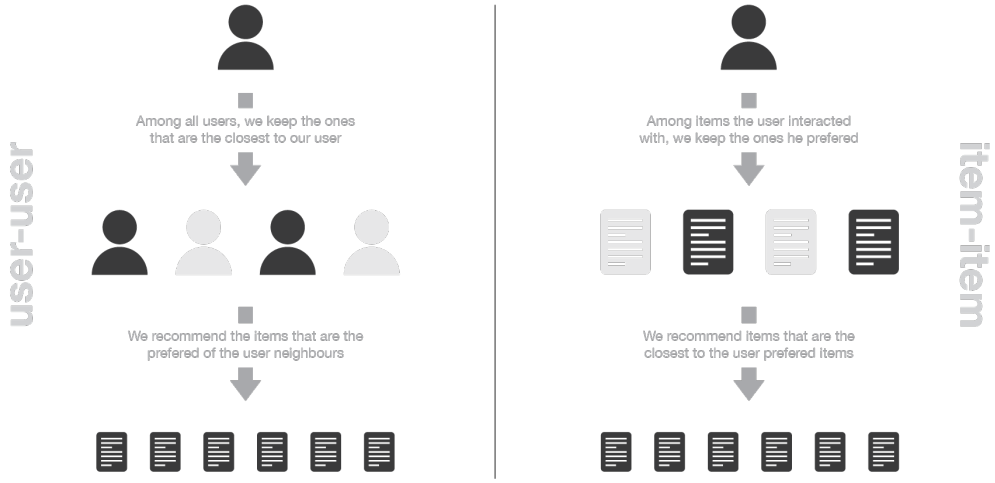

Figure 2.2: **Difference between item-item and user-user models [URL4]**

Basically, we are looking for users with the same interests as our target user and we're trying to guess his rating for the unobserved item by simply computing user similarities. The second approach is called Item-Based CFRS (also known as the item-item approach) and it goes by the logic of "***Users who liked this item also liked ...***". Here, the same logic as the user-user approach is applied but with a slight modification: we are looking for items that were rated by the same set of users instead of users that rated the same items. The Figure 2.2 represents an illustrated summary of the Memory-Based approach that compares the latter's sub-categories.

2.2.1.2 Model-Based

Model based approaches assume an underlying "generative" model that explains the user-item interactions and tries to discover it in order to make new predictions. They are supported by Machine Learning and Data Mining techniques in order to build the predictive models. These models, for example, may include latent factor models like Matrix Factorization [Koren et al., 2009].

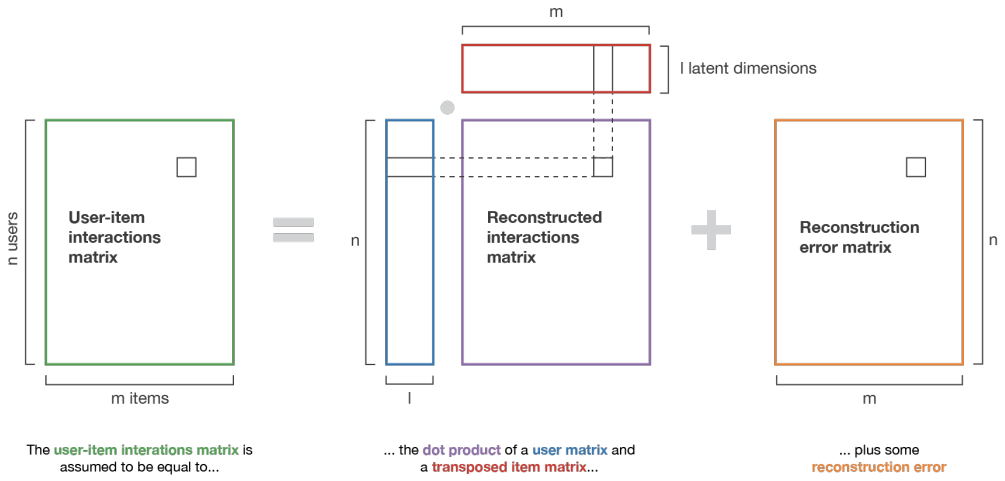

Figure 2.3: **Matrix Factorization [URL4]**

Matrix Factorization algorithms perform a dimensionality reduction operation that consists in decomposing the huge and sparse user-item interaction matrix into a product of two smaller and dense matrices: a user-factor matrix that contains users representations multiplied by a factor-item matrix that contains items representations exactly as shown in the Figure 2.3. CFRS' approach is very efficient because it requires no prior domain knowledge and the data is simply gathered from users interacting with items. As we mentioned before, data can be represented either in an explicit form such as a rating (i.e. the number of stars), a like, a dislike, etc. Or in an implicit form such as time of view, time on screen, etc. In some cases, we have richer user interactions in addition to the classical previously mentioned ones. An example of these rich interactions could be user comments that can be data mined for sentiment analysis. CFRS can solve the problem of safe recommendations that is inherent in the Content-Based approach since it can involve other users in the process and find similarities between them. For instance, user *A* might love action movies but has never even thought about seeing anything outside his genre bubble. With CFRS, user *A* is found to be very similar to user *B* and user *C* due to their shared passion for action movies. However, user *B* and *C* both love fantasy and sci-fi movies as well. So even though those movies genres may be far outside of user *A*'s bubble, it might be a good recommendation for him to check out. Unfortunately, CFRS is not a perfect approach either and does have limitations. It mainly suffers from the cold start problem. This happens when

we have fresh items or users that are introduced to the system. Since there's too little interactions, the collaborative traits are absent and the system fails to provide relevant recommendations. This is really harmful for the system's accuracy and represents the ultimate challenge for CFRS.

2.2.2 Content-Based Recommendation System

The Collaborative Filtering approach, discussed previously, relies on correlations of the ratings in order to make recommendations. On the other hand, this model does not require any data from other users since Content-Based systems use feature attributes in order to offer recommendations. For example, in the case of mobile applications recommendation use case, attributes would be the *categories*, *publisher*, etc.

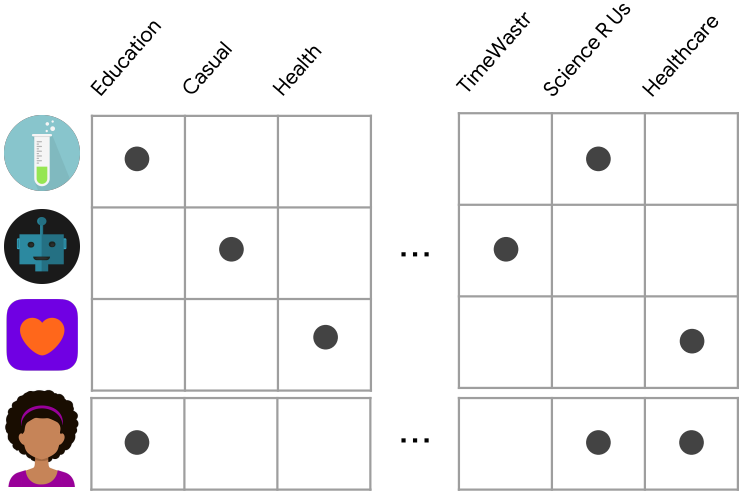

Figure 2.4: **User/item representation [URL1]**

These descriptive attributes are then used to profile both users and items. So, for example let's say our target user is extremely passionate about *Education*, *Science* and *Healthcare* applications. Now that we were able to profile the user, thanks to his previous interactions, we can match his interests with the items and recommend him relevant applications. Relying on attributes is extremely helpful especially when the items are new or have no support (i.e. unpopular items). After building the matrix of item properties and multiplying it with the user profile in order to get the user representation in the item embedding space, we can use one of multiple similarity measures

to recommend relevant items (i.e. nearest neighbours' logic). CBRS have many pros. First there is no need for data about other users than the target in order to recommend similar items from the learnt embedding space. Also, we can provide niche items since we have prior knowledge about our target's specific tastes. Unfortunately, CBRS have limitations too and in order to expose then let's consider the movie recommendation use case: a human has to enter the movie synopsis, a human has to label the movie genres, a human has to put the movie poster, etc. As you can see human experts and hand-engineering are required in this process which is quite troublesome. Furthermore, these systems tend to make only safe recommendations and stay within a safe bubble. CBRS will only recommend highly similar items to the ones already visited which will end up with our users within a never-ending loop, and thus pushing them outside of their usual comfort boundaries to explore new items becomes nearly impossible.

2.2.3 Knowledge-Based Recommendation System

KBRS, also known as Constraint-Based Recommendation Systems, are based on explicit knowledge about the user's preferences or items. They are especially useful when alternative approaches such as Collaborative Filtering or Content-Based models cannot be applied.

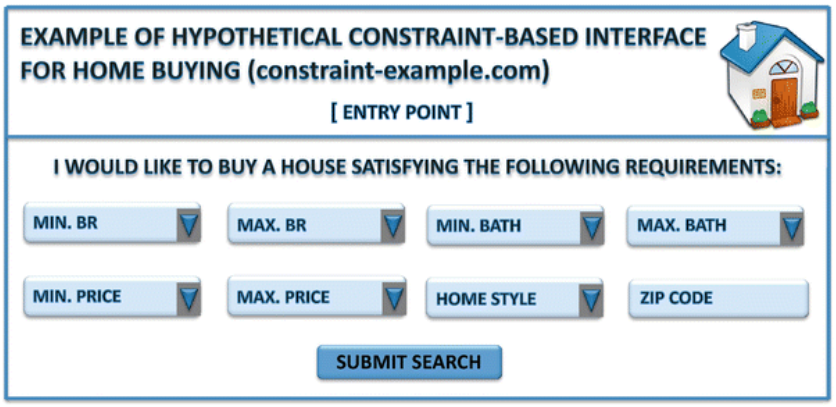

Figure 2.5: **Example of a Knowledge-Based system interface [Aggarwal, 2016]**

This occurs in situations where items are not consumed very often. For example, let's consider the case where we want to build a RS to sell vacation houses. Because most

people don't buy houses often, we probably wouldn't have enough previous house buying information to use either a Content-Based or Collaborative Filtering approach. In this scenario, KBRS will explicitly ask users for their preferences, like shown in the Figure 2.5, and then use that information to begin making recommendations.

2.2.4 Hybrid Recommendation System

HRS can combine KBRS, CFRS and CBRS to achieve state-of-the-art results. Blending all three approaches allows us to get rid of each model's shortcomings and benefit from its advantages.

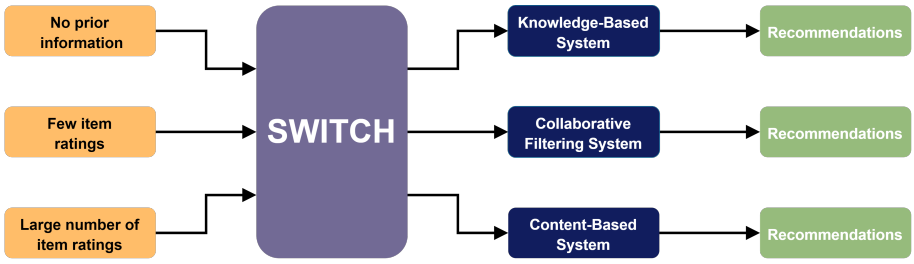

Figure 2.6: **Switching between models as a HRS**

Often, there is value in combining different types of Recommendation Systems into a single Hybrid System. This can be done in several ways. As illustrated in the Figure 2.6 , we could develop a few RS and then use one or the other depending on the scenario. If a user has already rated a large number of items, perhaps we can rely on a Content-Based system. However, if the user has rated only a few items, we may instead prefer to use a Collaborative Filtering Recommendation System. This way, we can fully leverage the information we have about other users and their interactions with items in our database, to gain some insight into what we can recommend. Of course, if we have no information about a user's previous item interactions or we lack any information about a given user, we may instead want to rely on a Knowledge-Based system, and ask the users directly for their preferences via a survey before making recommendations.

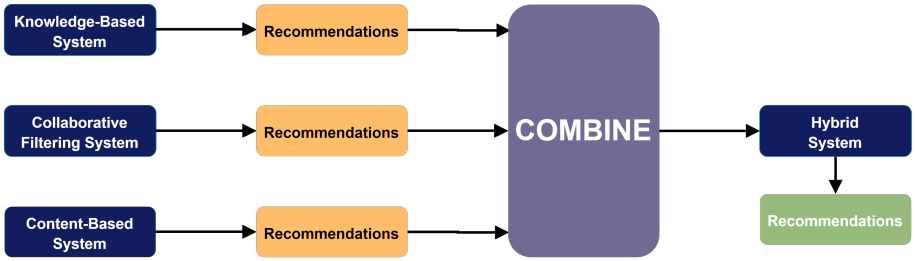

Figure 2.7: **Combining models into a HRS**

Another way to create a Hybrid system is to simply combine the outcomes of more than one of the prior RS. The multiple outcomes could then form the input to a more sophisticated system, as shown in the Figure 2.7, that makes the final recommendation which we then serve to the user. The idea is that the more sophisticated the model is, the more we learn nuanced relationships between the query and the various model outcomes, and thus we will have a much better recommendation. In fact, some research suggests that a hybrid approach combining multiple outcomes like this, can provide more accurate recommendations than a single system on its own [Çano, 2017].

2.2.5 Session-Based Recommendation Systems

The RS we discussed so far, have an effectiveness that has been proved and validated by both research and industry communities. However, these classic RS still have some drawbacks even with the HRS approach. A critical one is that they only focus on a user's long-term preference and ignore his short-term patterns. This results in missing the user's preference shift through time and being easily submerged by his historical behaviours which leads to unreliable recommendations. This is caused by the fact that those systems usually break down a basic session into multiple records at smaller granularity (e.g. user-item interaction pairs) and then mix all these records together. Such a splitting strategy ruins the intrinsic nature of sessions in which the user's interest shift is embedded. For example, let's consider the online shopping use case and the Matrix Factorization approach as a representative model for CFRS. All of the interactions between a user and an item in each navigation session are broken down into a granular level and put into one row of a matrix in form of a rating as shown in the top half of the Figure 2.8.

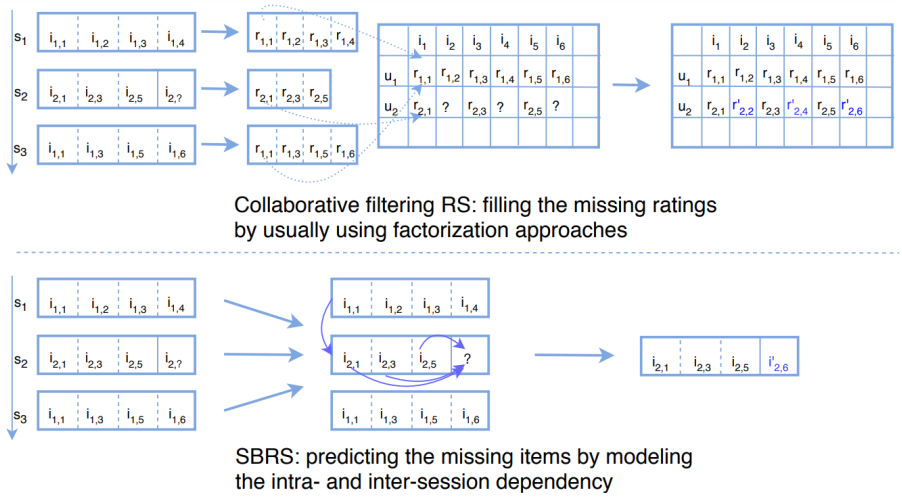

Figure 2.8: **Difference between CFRS and SBRS [Quadrana et al., 2018]**

Another huge limitation is that user identifications are not always available due to privacy concerns. Therefore, the conventional system that require user information are not applicable. In such cases, the recommendations can be only made using the session data. Taking the session's structure into account allows more information gain on behavioral patterns. As shown in the bottom half of the Figure 2.8, SBRS takes a navigation session as its basic unit unlike the classic approach that takes a user-item pair as its basic unit. SBRS has two sub-categories: next-item(s) recommendations that recommend a part of the current session, and next-basket recommendations that recommend partial or whole of future sessions. In both cases, intra and inter-session dependencies are used in order to make the predictions. SBRS are widely applied in domains like next web page recommendations, next song recommendations, and so on.

2.2.6 Summary

The Table 2.1 is a concise summary of what we discussed so far. It exposes the required input, core assumption, work mechanism, pros and cons of each approach.

Model	Input	Core Assumption	Work Mechanism	Pros	Cons
KBRS	User preference	A user likes exactly what he specified	Matching desired features against item content	Simple, can handle cold-start issues	Requires user 's manual input
CFRS	User-item interaction data	A user likes what he liked	Modeling user-item interactions	Effective, relatively simple	Suffers from sparsity and cold-start issues
CBRS	User profile and item content information	A user likes what he liked	Matching up user profile against item content	Simple, straight-forward, can handle cold-start issues	The assumption may not fit real-world cases very well
HRS	User preference, user-item interaction data, user profile and item content information	A user likes exactly what he specified, A user likes what he liked	Combining models outcomes, Switching between models	Can handle cold-start issues, Improved performance	High complexity
SBRS	Session data	User preference changes along with the corresponding session context	Recommending items that have occurred in a similar context	Considering the user preference evolution which fits the real-world cases better	Ignores user's general and long-term preference

Table 2.1: **Comparison between SBRS and other classic RS.**

2.3 Study of existing solutions

Nowadays, Recommendation Systems are quite abundant and have been used in a wide variety of fields. These systems have enriched the user experience and added a huge market value since lots of businesses recorded huge profit increase after relying on these systems [Jannach and Jugovac, 2019]. Among the most iconic ones, comes in the Netflix movie RS. Between 2006 and 2009, Netflix sponsored a competition with a huge prize for whom is able to improve its current RS accuracy with 10% [URL3]. This competition shifted the tides and brought a lot of interest for research in this particular area, and thus the number of papers has drastically increased since.
Even though the internet is overflown with these systems, we cannot directly compare any of them to the one that we are building. The system that we are working on is exclusive for Infor EzRMS and represents a totally new component in it. The closest thing to a RS in EzRMS would be the *Previously visited* feature. As its name implies, this feature simply provides the user with the screens he already visited in the current session. Unlike the static *Previously visited* feature, our solution will be providing relevant recommendations with respect to interest drifts and contextual information.

2.4 Conclusion

In this chapter we presented the state of the art of the algorithms related to the Recommendation System we want to build and we conducted a comparison with the already existing solution within EzRMS. The next chapter will be dedicated to the requirements of our solution and the features it will offer.

Chapter 3

Requirement Analysis & Specification

3.1 Introduction

Throughout the previous chapter, we familiarized ourselves with the set of concepts and algorithms that revolve around our Recommendation System. In this chapter, we will identify the actors, the software and hardware requirements for the application. Then, we will express the functional requirements of the RS along with its software attributes. At last, we will expose the requirement specifications that will be followed by a brief explanation for the solution's development life cycle and a specification of the recommendation workflow.

3.2 Identifying actors

The actors that interact with our system are the users subscribed to EzRMS. These users can have different roles in the hospitality industry : Revenue Managers, Sales Managers, Business Analysts, Room Managers, Desk Officers etc.

3.3 Requirement analysis

3.3.1 Software and hardware requirements

Aside from an internet enabled device, no further hardware is required. Our RS will be embedded into EzRMS' web interface. Therefore, aside from the web navigator, the

user is not required to install any plugins, web-extensions or any third-party software.

3.3.2 Functional requirements

The system must satisfy the following requirements:

- Profile users and capture their interests.
- Provide relevant recommendations.
- Adapt recommendations to interest drifts.

3.3.3 Software attributes

The system should obey the following quality-performance requirements:

- **Quality requirements**
  - **Privacy**
    The software will never collect, use or ask for any personal information.
  - **Compatibility**
    The system will be compatible with other EzRMS components and will have no effect on their behaviour.
  - **Maintainability**
    The source code will be well documented and clearly implemented thanks to the adapted code refactoring process.
- **Performance requirements**
  - **Reliability**
    The software will perfectly meet all of the functional requirements without any unexpected behaviour.
  - **Scalability**
    The software will be able to handle large loads of queries without any issues.
  - **Efficiency**
    Special measures shall be taken in order to ensure the algorithmic efficiency in term of response time and resource utilisation.

3.4 Development life cycle

Throughout this section, we will present the development life cycle of our application from two different perspectives: a machine learning perspective and a software engineering one.

3.4.1 Machine Learning perspective

The Figure 3.1 represents the the typical six-phased Machine Learning pipeline that we adapted during the development phase.

Figure 3.1: **Machine Learning pipeline.**

1. **Problem formulation**
   This is the first step of every project and it is where the problem and the objectives are defined. In a machine-learning-centric project, this phase includes research, problem framing, long discussions about the current workarounds, contemplating the assumptions and adopting appropriate performance metrics.

2. **Getting data**
   This is a data-centric phase where we determine the type, size and the source of the required data.

3. **Exploring data**
   This step is also known as Exploratory Data Analysis (EDA). The objective here is to gain insights from the data and better understand the task at hand. Human experts are extremely valuable in this phase since they have an extensive domain knowledge and the explanation for observed correlations in the data that might not be obvious for the data scientist. This phase is mainly a study of features and their characteristics via visualizations.

4. **Preprocessing data**
   In this phase, data transformations that were identified as worthy by the previous step are applied. Also, this preprocessing includes data cleaning, features selection and feature engineering. Data cleaning concerns the anomalies that we observed in the previous phase. As for the feature selection, it concerns capturing the best characteristics that correlate with our goals. And for feature engineering, it involves creating new features that might be helpful while modelling the data.

5. **Modelling data**
   Here we build models, evaluate and compare them. Also, this phase includes an investigation of the most significant features in each algorithm and a performance analysis. The best performing models should be shortlisted in order to be fine-tuned afterwards.

6. **Fine-tuning models**
   The final phase's objective is to optimize the accuracy of our models. It includes fine-tuning the shortlisted models' hyper-parameters on a separate validation set and then re-evaluating them on the whole training set.

P.S : *Each phase that involves code implementation includes a* ***Code Refactoring*** *phase where a particular attention is paid for the time and performance efficiency of the algorithms.*

3.4.2 Software engineering perspective

Now, if we switch our perspective and look at the problem from another angle, we can define the development cycle of our Recommendation System with a more formal model. There are a wide variety of software engineering models and we believe that the best-fit model that adapts to the Machine Learning pipeline, previously described, is the incremental model. In its original form, the incremental model follows the graph shown in the Figure 3.2. In a typical software engineering situation, the flow starts with a requirement analysis and an architectural design. Then, for each version of the software, we issue a detailed design followed by a development phase. After that, we validate and integrate the increment and we check that the update did not affect the system itself. We perform this loop for each increment until we reach the desired final system. As you may have noticed, there is a huge issue with this model if the

architecture is not carefully defined. Also, there is a certain degree of complexity while integrating new increments due to compatibility issues that may surface during that phase.

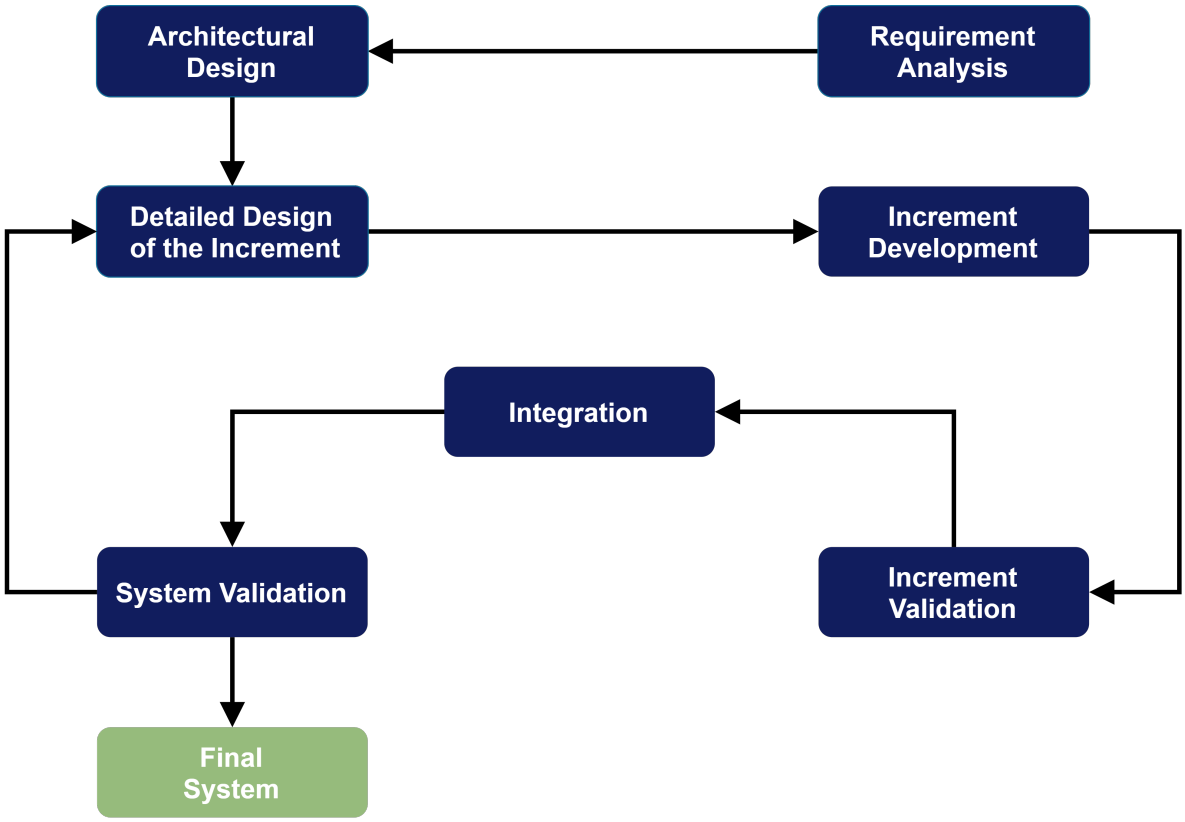

Figure 3.2: **Incremental model graph.**

In order to tackle those issues, we opted to a slightly modified version of the incremental model as shown in the Figure 3.3 where we introduced the framework notion. The framework represents a set of elements that explicitly define or provide an abstraction layer for modules needed by each increment. In our particular use case, the increment-related loop concerns only the modelling phase where we will be implementing several algorithms and then integrating them within the system. We opted to this method in order to address compatibility issues and provide each increment with the same structure: each increment obeys strictly the predefined framework's design. The orange nodes, illustrated in the Figure 3.3, represent the phases that we added on top of the original flow. The data preparation phase includes the Getting , Exploring and Preprocessing steps discussed previously in the Machine Learning perspective.

Obviously, we discarded the Detailed Design of the Increment phase for compatibility issues and we replaced it with a framework design and a framework implementation phases. As for the fine-tuning step, it is included in the system validation phase.

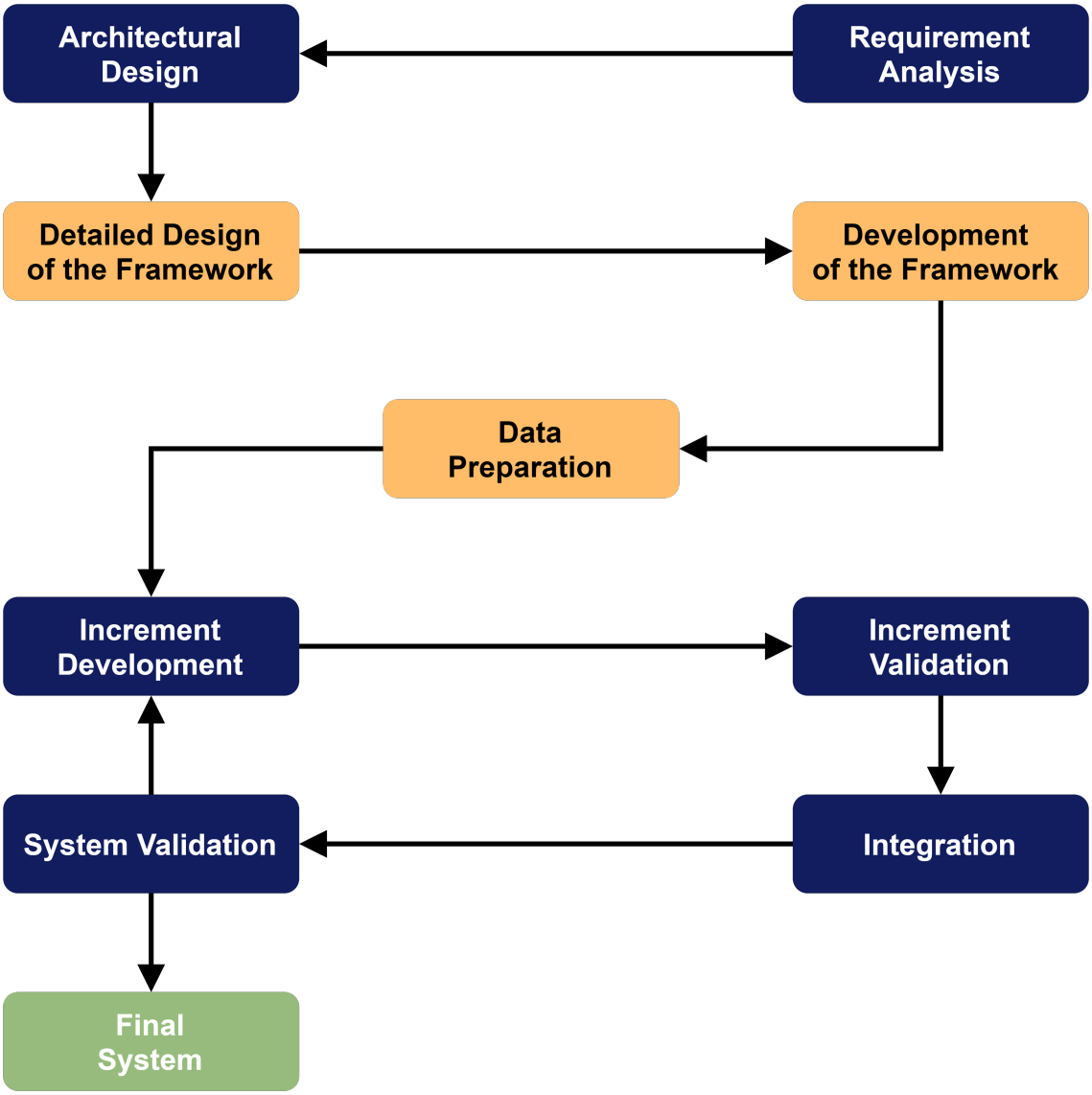

Figure 3.3: **Adapted Incremental model graph.**

P.S : *Concerning the **Code Refactoring**, it takes place in each phase that includes development: Development of the Framework, Data Preparation and Increment Development.*

The Figure 3.4 represents how we adapted the modified incremental model to our specific needs. The project had three versions: **ALPHA, BETA** and **GAMMA**.

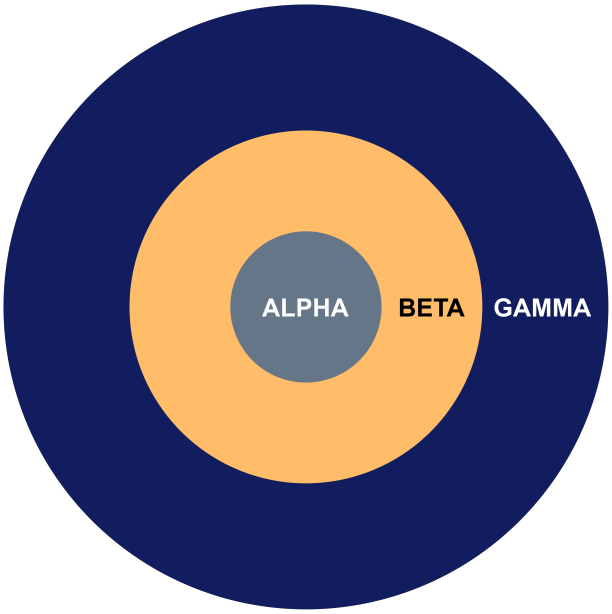

Figure 3.4: **Application development life-cycle.**

1. **ALPHA**
   This increment represents the core version of our application. This is where the data preparation tools, framework modules and evaluation packages are defined. ALPHA also includes the first recommendation model we implemented and evaluated which is a Session Popularity model or SPOP for short.

2. **BETA**
   In the BETA version we performed a few modifications on the data preparation tools. Due to this modification, we had to re-evaluate the SPOP model. Also, we added two other models : Item K-Nearest Neighbours (IKNN) and Product to Vector (PROD2VEC).

3. **GAMMA**
   In this final version, we simply added two other models which are Factorizing Personalized Markov Chains (FPMC) and Gated Recurrent Units for Recommendations (GRU4REC).

P.S : *We will save the discussion of the mentioned algorithms for the achievement chapter.*

3.5 Recommendation System workflow

The Figure 3.5 represents the workflow of our RS. The flow starts when the user navigates through EzRMS' pages. The user will be provided with recommendations for the next pages to visit based on his current navigation sequence. The system will generate new recommendations each time he visits a new page.

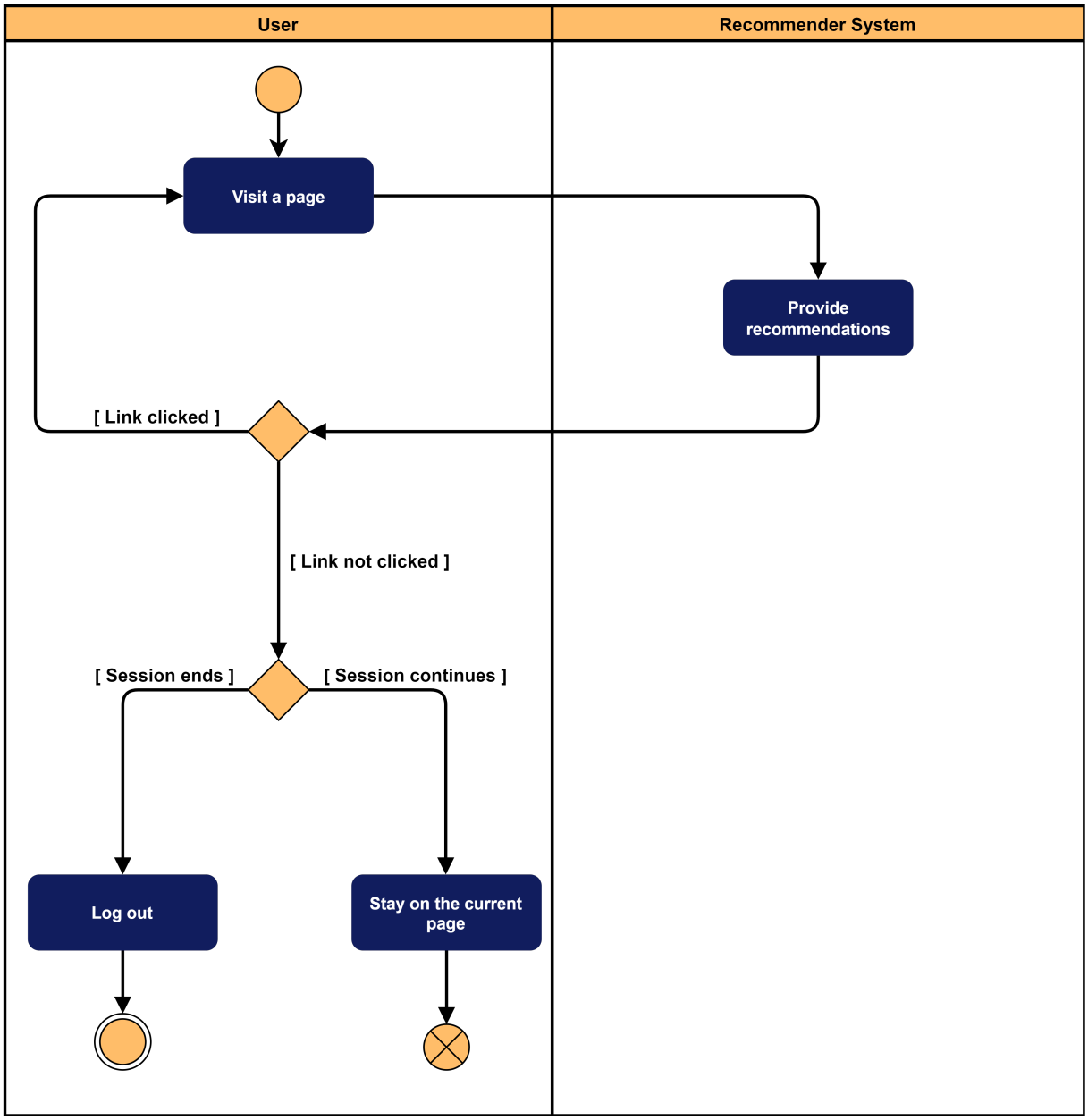

Figure 3.5: **Recommendation System workflow.**

The flow stops when the users decides to stay on the page and the activity can resume as soon as he clicks on a new link. However, if the user decides to log out, the activity gets terminated.

3.6 Conclusion

In this chapter, we presented an in-depth specification for the features of our solution and meticulously described its development life cycle . The next chapter will address the design blueprint of the Recommendation System.

Chapter 4

Design

4.1 Introduction

Now that we have a clear understanding of the system's functionality, its attributes and the requirements it must adhere to, it is time to dive deep into its blueprint. The design phase is unquestionably fundamental in order to build any reliable software application. In this chapter, we will proceed by explaining the Recommendation System's architecture and how it will be embedded into EzRMS.

4.2 Global design

4.2.1 Layered architecture

The Figure 4.1 shows the global architecture of EzRMS' Recommendation System in its layered form. As an abstraction, the system is represented with three layers: Acquisition, Modelling and Reasoning. The acquisition layer is a built-in layer within EzRMS. It is where data is collected and stored. This layer is responsible for tracking the user actions via a user log interceptor and storing those logs into a log database that is enriched with contextual information to take its final form as a descriptor dataset. The modelling layer's core is the Knowledge Entity. This entity handles the knowledge representation paradigms so in simpler terms we can consider it as a knowledge database rich with features, ontologies, etc. As for the reasoning layer, it is the brain of our RS. It has an embedded Algorithmic Entity that handles the reasoning mechanisms and pro-

vides estimations. Those estimations are pumped to a recommendation service that directly interacts with the user.

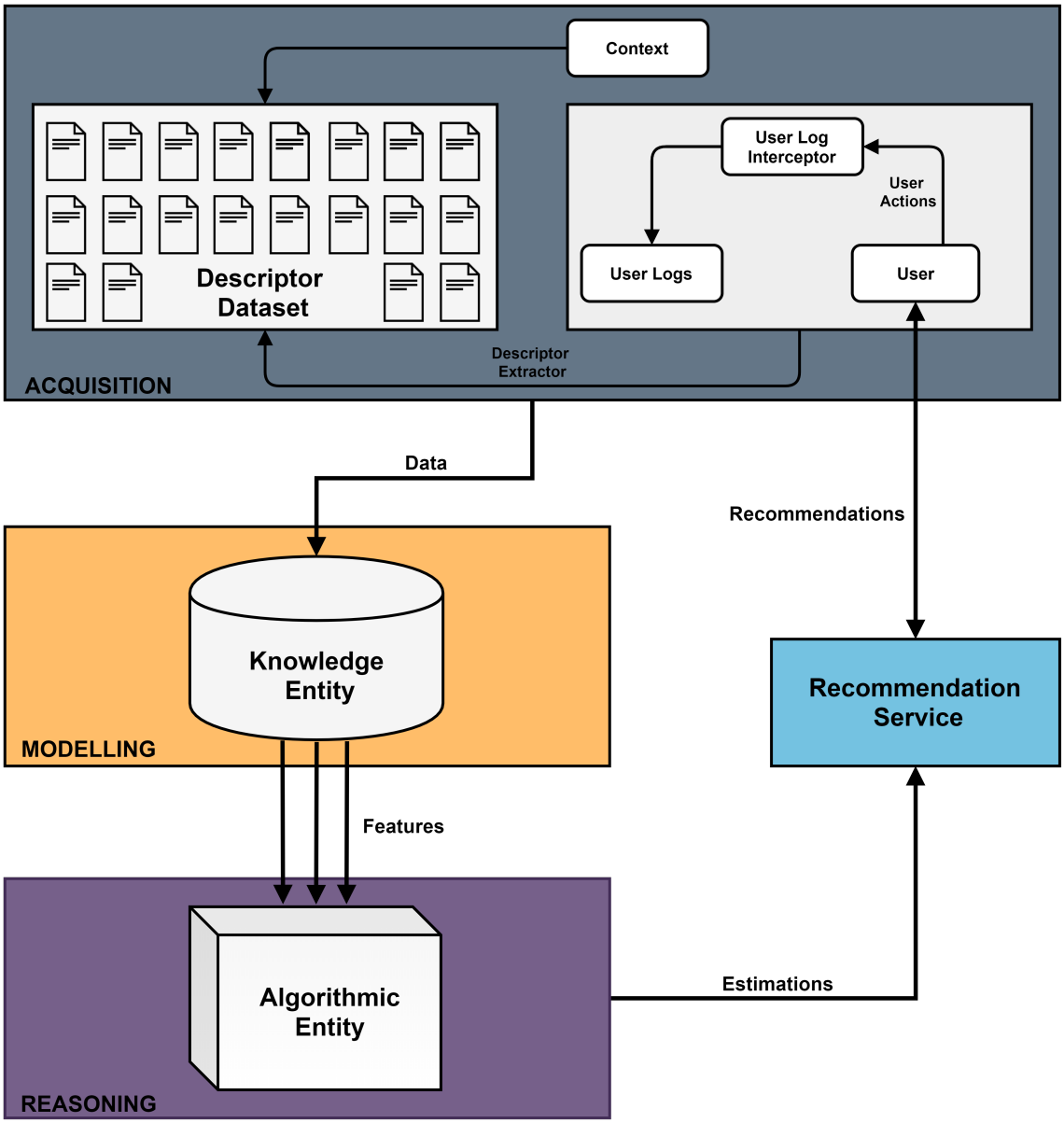

Figure 4.1: **Layered architecture of EzRMS' Recommendation System.**

Discussing the acquisition layer and the recommendation service is out of scope for this work. The acquisition layer, as we already mentioned, is an EzRMS built-in layer that we will simply exploit in order to retrieve data. As for the recommendation service, it is up to the software development team to design it since interacts directly with

the user via EzRMS' interface. Therefore, our project's focus will be on the modelling and reasoning layers. Before diving too deep into the design of those layers, let's first understand how they work.

- ***Modelling layer***

  The modelling layer is fed with raw data, from the previous layer through a pipeline, and its goal is to "***extract the signal from the noise***" by filtering out unnecessary observations and cleaning the necessary ones. Mainly, it has two engines: an Extraction engine and a Preprocessing engine. As their names imply, the first engine is responsible for maintaining a connection with the acquisition layer and extract data from it and the second engine's role is to preprocess the data and shape it into a form that the next layer can ingest. Those two engines form the Knowledge Entity, the core of the modelling layer, which is special form of a knowledge database that holds useful features for the reasoning layers.

- ***Reasoning layer***

  In this layer, the system performs the reasoning process for the recommendation. Different reasoning mechanisms can be used thanks to the modular architecture of this layer that we will be discussing later. This layer is powered by two engines: a Recommendation engine and an Evaluation engine. The Recommendation engine takes as input the features provided by the previous layer, models them and outputs estimations. As for the Evaluation engine, it is responsible for assessing those estimations and helping with deciding which reasoning mechanism is to proceed with. Both of these engines represent the core of the reasoning layer : the Algorithmic Entity.
  P.S : *By reasoning mechanism we mean the algorithmic way to solve the recommendation problem.*

4.2.2 Component-based architecture

The layered architecture provided a clear explanation for EzRMS Recommendation System's design. Four engines are powering it having each a specific role to play in order to provide relevant recommendations. If we change our perspective, we can express the system as a component within EzRMS as shown in Figure 4.2 and if we

dive deeper, we can also express the core entities as separate components exactly as shown in Figure 4.3. Each engine is connected to the next one providing it with its required input.

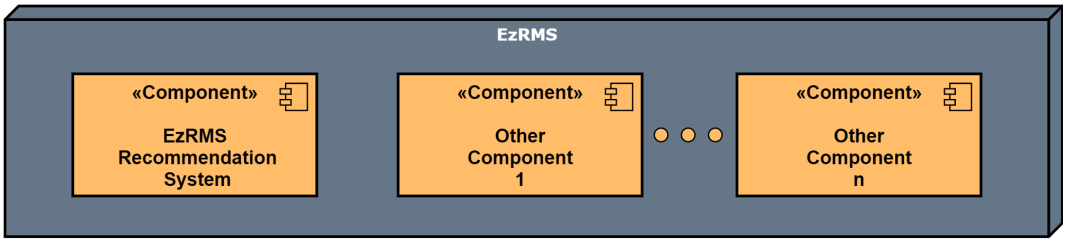

Figure 4.2: **EzRMS' Recommendation System as a component.**

Essentially there are two flows of execution. The first flow concerns the development scope where the Evaluation engine's assessment results interest us in order to choose the optimal reasoning mechanism. However, in the simple usage scope and when the solution is deployed the flow stops at the Recommendation engine's output.

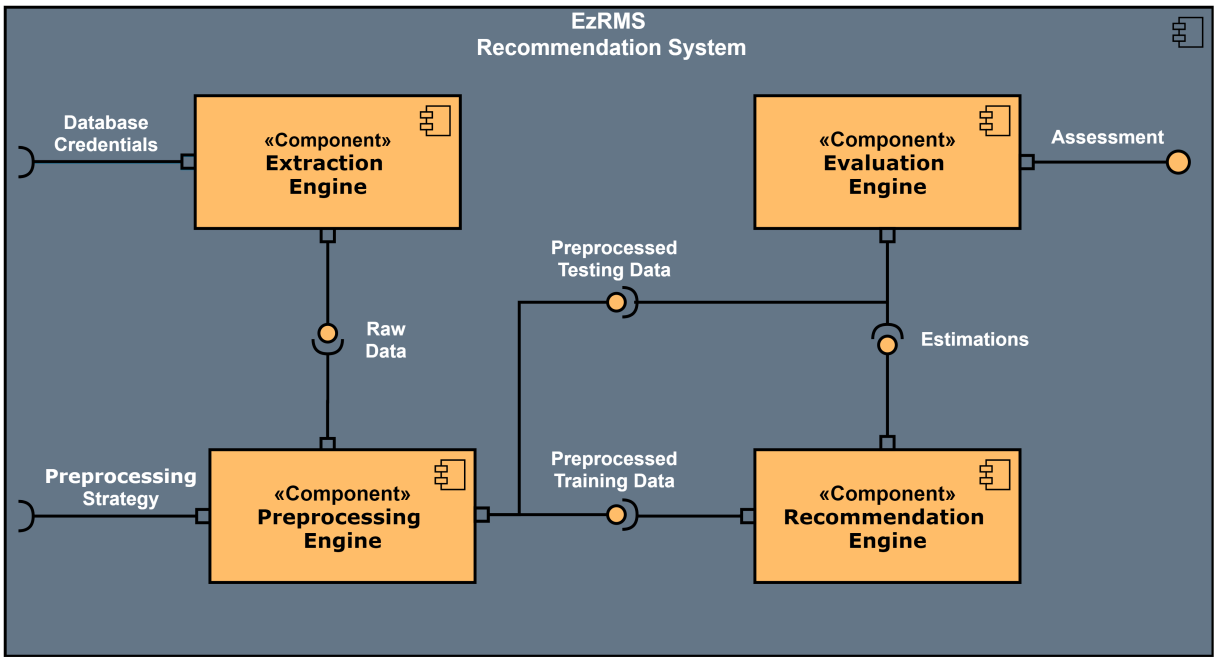

Figure 4.3: **EzRMS' Recommendation System component diagram.**

4.3 Detailed design

The layered architecture gave us a clear intuition about the core elements of the RS. Furthermore, the component perspective enhanced that intuition and gave us a general idea about how the engines are interacting with each other. However, in order to get a better understanding of the system, further details are required. The Figure 4.4 represents the four engines discussed previously with another perspective that expresses the packaging design. As shown below, EzRMS' RS carries four packages, one for each engine, and each engine has packages itself.

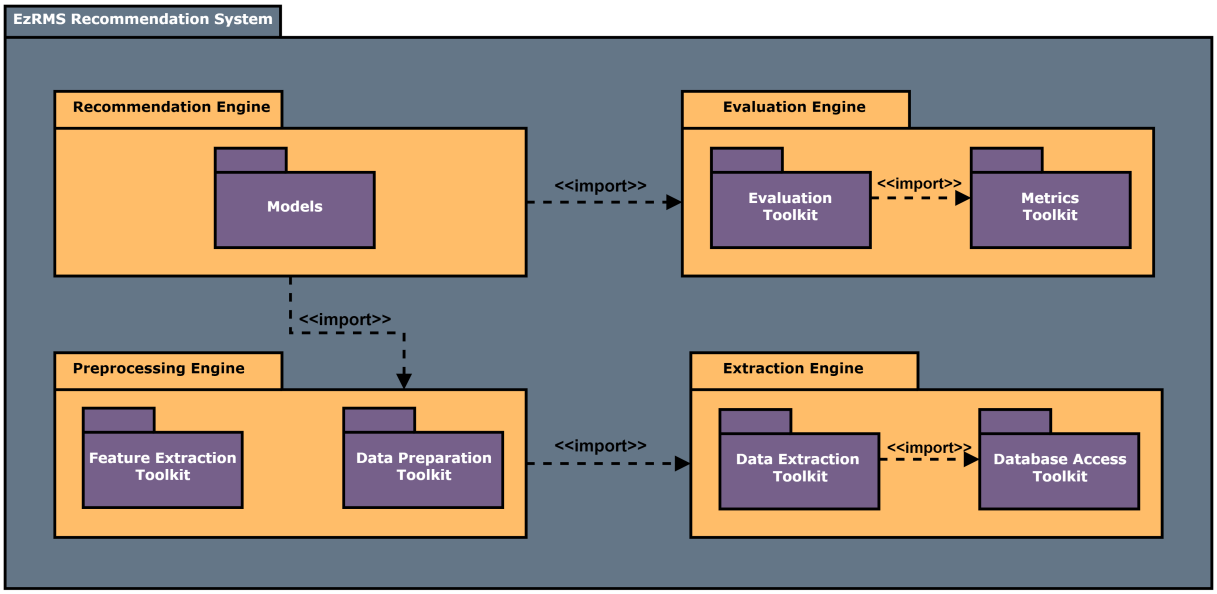

Figure 4.4: **EzRMS' Recommendation System package diagram.**

- **Extraction Engine**
  The Extraction Engine package includes the Database Access and Data Extraction Toolkits.
  - *Database Access Toolkit*
    This toolkit holds all the credential and configuration files required to establish a connection to the database.
  - *Data Extraction Toolkit*
    This package hosts the modules responsible for extracting and merging data.

- **Preprocessing Engine**
  The Preprocessing Engine package includes the Feature Extraction and the Data Preparation Toolkits.

  - ***Feature Extraction Toolkit***
    This package hosts the required modules in order to perform feature extraction from the raw data.
  - ***Data Preparation Toolkit***
    This toolkit contains the modules necessary for the data cleaning process.

- **Recommendation Engine**
  The Recommendation Engine package includes the models adapted in this work.

  - ***Models***
    This package contains the implementation files of the algorithmic models used for recommendation.

- **Evaluation Engine**
  The Evaluation Engine package includes the Evaluation and the Metrics Toolkits.

  - ***Evaluation Toolkit***
    This Toolkit contains the modules responsible for evaluating each and every model from the Models package. Examples of the modules would be session evaluation files, sequence evaluation files, etc.
  - ***Metrics Toolkit***
    This package hosts the implementation of the different metrics used for evaluation.

As you may have noticed, while observing the Figure 4.4, the Recommendation engine is the most dependent entity: it requires inputs from all three other packages in order to produce results. This is not considered as a concerning matter since the dependency exists only when the Recommendation engine is under development : when deployed, it will not require any of those engines to output results unless we want to retrain it.

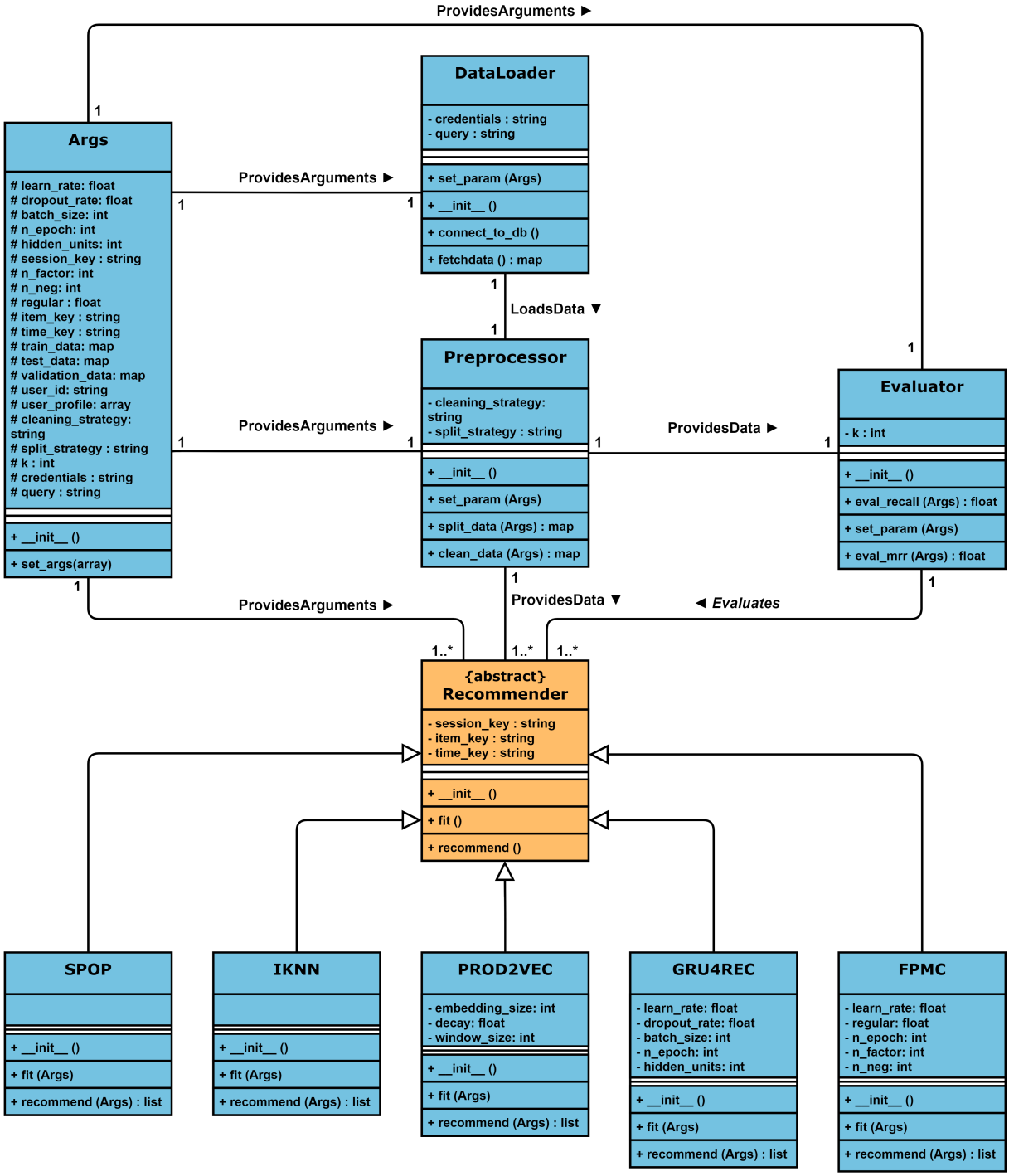

Figure 4.5: **EzRMS' Recommendation System class diagram.**

Now that we built up the intuition into a deeper understanding of the system, let's get more formal in our way of expressing the design. The Figure 4.5 represents the class diagram of EzRMS' Recommendation System. Mainly, there are ten classes interacting with each other. Among those classes comes an abstract Recommender class that makes sure that code sharing is easier among the closely related classes : SPOP, IKNN, PROD2VEC, GRU4REC and FPMC .

- **Args**
  Our solution has an important number of parameters. Therefore, we found it easier to centralize those parameters within this class in order to simplify tweaking them.

- **DataLoader**
  This class is responsible for establishing a connection with the database and extracting the raw data from it.

- **Preprocessor**
  The Preprocessor class takes in the raw data provided by the DataLoader cleans and processes it then provides the Recommender with the training data and the Evaluator with the test or validation data.

- **Evaluator**
  This class is responsible for evaluating each and every Recommender model by providing the adequate metric scores.

- **Recommender**
  This class is abstraction for five models developed in the scope of this work in order to solve the recommendation problem.

4.4 Conclusion

In this chapter, we started by building an intuition over the global design of the system. Then we gradually exposed the key elements of our RS using the Unified Modeling Language (UML) diagrams as a formal representation. The next chapters will concern the realisation of this blueprint and the achieved results.

Chapter 5

Achievement I: Insights

5.1 Introduction

In this chapter, we will present the work environment and explain in detail the insights we were able to extract thanks to an extensive exploratory data analysis that exposed behavioural trends and hidden patterns in our data.

5.2 Work environment

In this section we will expose the software and hardware environments. Also, we will introduce the technologies, frameworks and libraries we used.

5.2.1 Software environment

- **Oracle SQL Developer** is a free to use product provided by Oracle. It uses Java Development Kit and provides an Integrated development environment for working with SQL in Oracle databases.
- **Jupyter Notebook** is an interactive IDE in the form of a web application. Jupyter Notebook is platform-independent as well as language-independent which is why it became one of the most used editors in data science.
- **Visual Studio Code** is an open source code editor built by Microsoft and supports Windows, Linux and mac OS. The editor supports code debugging and intelligent

completion, syntax highlighting and embeds Git.

- **Draw.io** is a free online editor that enables you to create flowcharts, mockups, UML diagrams and more.

- **GitLab** is a web-based version control tool that provides a Git-repository manager that allows issue-tracking and continuous integration/continuous deployment pipeline features and more.

5.2.2 Technologies

- **Python** is an interpreted, high-level, general-purpose programming language created by Guido van Rossum. Nowadays, Python is among the most used programming languages due to the large amount of open-source libraries and modules. It represents the number one choice for data science thanks to the large support and contributions of the community.

- **SQL** is a language used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems.

5.2.3 Frameworks and libraries

- **Numpy** is a Python library that provides built-in methods for multi-dimensional mathematical operations and scientific computations.

- **Pandas** is a library that provides data manipulation, analysis and structuring functionalities.

- **Matplotlib** is a plotting library for Python and Numpy. It provides a large number of functionalities and built-in methods.

- **SciKit Learn** is a python library that provides tools for data mining and analysis. It offers a large amount of optimized models and algorithms ready to use. This library is built on top of Numpy.

- **Numba** is a compiler that translates a subset of Python and NumPy code into av fast lower level machine code. Also, It offers a range of options for parallelising Python code for faster execution.
- **Genism** is an open-source library for natural language processing, and machine learning.
- **Tensorflow** is a library for dataflow and differentiable programming. It is also used for machine learning applications such as neural networks.
- **Keras** is a framework that can work on top of Tensorflow and enable fast experimentation with deep neural networks. It focuses on being user-friendly, modular, and extensible.

5.2.4 Hardware environment

In the development of this project, we only used a computer with a decent configuration, as shown in the Table 5.1, and a reliable internet connection.

Central processing unit	8th Generation Intel Core^{TM} i7-8665U
Graphics processing unit	Intel UHD 620 Graphics
RAM	16 GB 2666 MHz DDR4
Hard Drive	1 TB 7200 RPM
Operating system	Windows 10 Enterprise Edition

Table 5.1: **Development machine specifications**

5.3 Data preprocessing

In real world, nothing is perfect nor complete: this is a general truth that applies to every aspect of our lives. In this modern era, every aspect of our lives is data-enabled and those imperfections are projected to the digital world via a phenomenon called noise. Noisy data means that the collected observations are affected by some anomalies that can be represented as some missing values or wrong ones. Data preprocessing, also called data cleaning, is a critical phase in any machine learning project because

the correctness of the final results is strongly correlated with the quality of the data produced by this phase. Therefore, we invested a large amount of time and effort in this step as proven in the following sections.

5.3.1 Raw data

Our starting point is the navigation logs of **2019**. Since it is impossible to perform a full tracking of the users' activity, the best we can do is to intercept his clicking actions. Click logs are the kind of datasets that describe the activity of users while navigating web pages. Initially, our dataset describes the activity of **4009** users that performed **637,394** session with a total of **5,153,481** clicking event.

5.3.2 Cleaning phase

As a first step, we performed a community level split. What we mean by a community level split is that we discarded some users and this is mainly due to their inactivity. Users having less than two sessions per month were excluded as we judged them as inactive. The next operation we performed is getting rid of missclicks. Misclicks are events where users unwillingly clicked on a certain link. We were able to detect missclicks by computing the time on screen. If the time on screen is too low (i.e. less than 3 seconds) then the event is more likely to be a missclick. As we previously mentioned, we are doing a full click tracking over our users' activity, so as a final phase of cleaning, we need to to get rid of pages that we will not be recommending for our users to visit. (e.g. interface configuration, home page, etc.).

P.S : *The threshold of 3 seconds is set by the business analysts of EzRMS.*

5.3.3 Final data

The outcome of the cleaning phase is **1121** users that performed **470,332** session with a total of **3,829,418** clicking event. As you may have noticed, these active users hold the majority of the sessions. Therefore, we did not fall for the sampling bias trap. As for the pages that we are to recommend they were reduced from **140** page to **82** page.

5.4 Insights

The only source of confidence is data, therefore, each and every successful business decisions must be based on data analytics. Data Analysis is the process of applying statistical techniques in order to illustrate information from what appears to be meaningless piles of data. This process is important in order to understand and discover hidden patterns that might be of high market value. Therefore, in order to extract insights and patterns, we conducted an extensive exploratory data analysis.

5.4.1 User behaviour

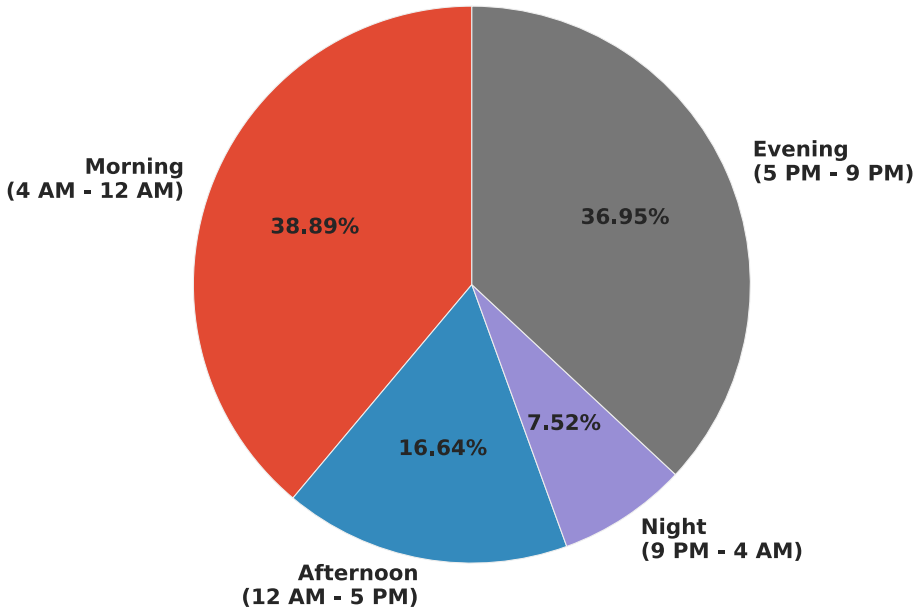

Figure 5.1: **Proportion of sessions per time of the day**

Recommendation Systems are built around user behaviour and as a first step, we wanted to understand when users prefer to navigate. The Figure 5.1 is a representation of the proportion of sessions per time of the day. The pie chart shows that users prefer using the system in the morning and during the evening, with respectively **38.98%** and **36.95%** of the total number of sessions, rather than in the afternoon or at night, with only **16.64%** and **7.52%**. This kind of information is extremely valuable especially when system updates or maintenance operations need to be scheduled. When we dived deeper and observed the total activity during each day, as illustrated in the Figure 5.2, we noticed a typical pattern where the activity is intense at the beginning of the week and drops drastically by the end of it. However, it is never null. This is mainly due to the fact that hotels operate 7 days a week and rarely shutdown their services. We obtained the previous illustration by simply computing the total number of clicking events for each day of the week, that we encoded from zero to six starting from Monday.

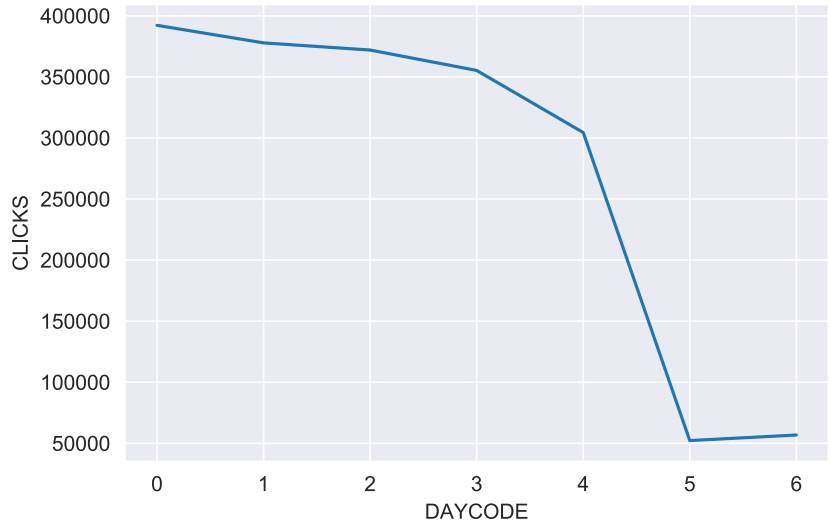

Figure 5.2: **Overall weekly activity**

Now that we understood the global trend, we decided to be more specific and observe the behaviour on a wider scale exactly as shown in the Figure 5.3. By paying attention to the plot, we can easily notice that the trend we discussed previously is omnipresent. In each month, there is 4 noticeable drops in the activity and this simply due to the

fact that typically each month has 4 weekends and as shown in the Figure 5.2, the navigation drops in that period of time. In some months we can observe more than 4 drops, like in June for example, this is simply due to the fact that there's a possibility that a month has more than 4 weekends or that it has a holiday exactly like December: between December the 20th and 30th the drop of the system usage is quite noticeable and this is mainly due to Christmas. EzRMS' business analysts explained the important spike of activity by the end of 8th month with the fact that in late August, revenue managers work on their annual reports, and thus the system usage is higher than usual.

P.S : *The explanations of the business analysts of certain observations that we cannot understand is quite valuable since they have the domain knowledge that we lack.*

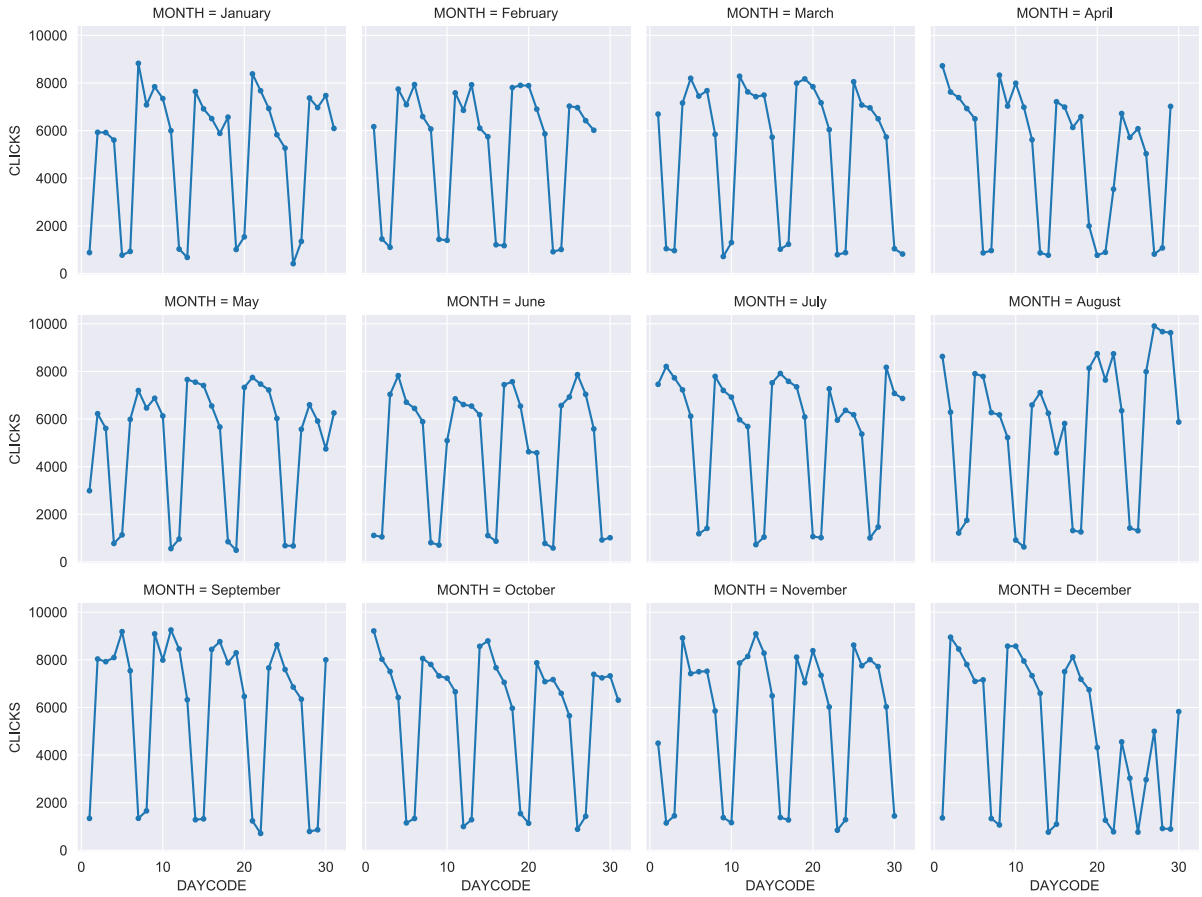

Figure 5.3: **Monthly activity**

5.4.2 Hidden pattern

Further exploratory data analysis was conducted in order to better understand the users' behaviour and discover patterns that might impact building our RS. The Figure 5.4, represents the most clicked pages in our data sample.

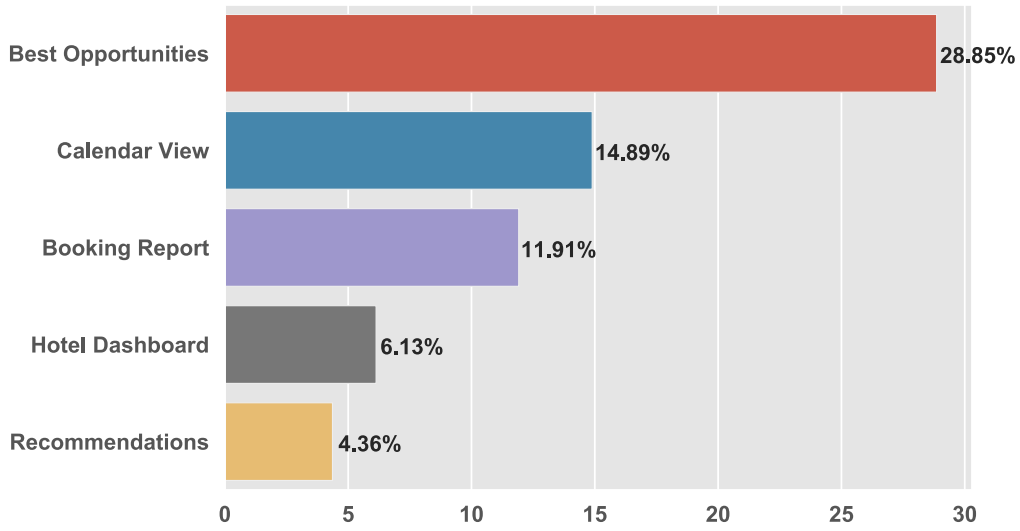

Figure 5.4: **Most visited pages**

The Best Opportunities page dominates the click rates as it alone represents **28.85%** of the total clicking events. The other 4 pages also have a decent share of the total number of clicks: Calendar View with **14.89%**, Booking Report with **11.91%**, Hotel Dashboard with **6.13%** and Recommendations with **4.36%**. As its name implies, the best opportunities feature provides the user with the best sales strategies in order to maximize profit and due to the importance of this feature it is understandable that it dominates the click count. However, we were suspicious that this is not the only reason and there is a better explanation for this behaviour. Therefore, we conducted further investigations and discovered an interesting trend within the sessions. Over all sessions, **24%** carry a repetitiveness pattern where users have visited the best opportunities page at least three times. Once again, this behaviour might be due to the importance of the page, however, it is not. After digging deeper into this repetitiveness mystery, we discovered that not only it greatly affects the best opportunities page but

also impacts the session itself (i.e. the same pages visited multiple times in the same session: [Page 1, Page 6, Page 3, Page 1, Page 3, Page 6]). Also, while conducting this investigation we discovered the existence of set of users that have some privileges: they can save files from EzRMS and even upload ones. After exploring these hidden patterns that we could not explain, we asked for the help of the domain experts (i.e. the business analysts). The answer for the repetitiveness in certain sessions was the fact that some users manage multiple hotels and they can switch between them in the same session. As for the privileged users, they are indeed super users. Actually, there are two types of users within EzRMS: Revenue Managers these are the super users that can modify the hotel pricing strategies and Access Users who can only visualize the interface without modifying anything (e.g. front desk officers, room managers, etc.)

5.5 Clustering

The exploratory data analysis revealed the existence of clusters within our sample. Users can be Revenue Managers, Simple Access Users or users managing single or multiple hotels like shown in the Figure 5.5. The Level 1 clustering has two types with a total of **4** groups: Revenue Managers (RM), Access Users (AU), Multiple Hotel Users (MHU) and Single Hotel Users (SHU):

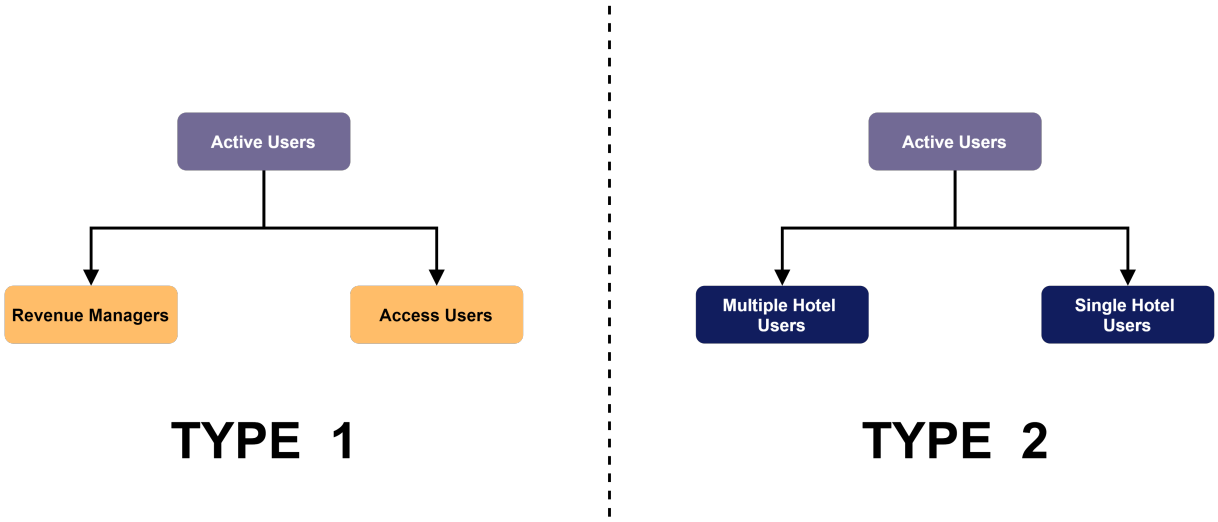

Figure 5.5: **Level 1 clustering diagram**

- **Revenue Managers**
  RM can edit, save and upload files that contain pricing strategies to the system.

- **Access Users**
  AU can access the interface but cannot issue any modifications.

- **Multiple Hotel Users**
  MHU manage multiple hotels at once with the same account.

- **Single Hotel Users**
  SHU manage one single hotel.

As you may have noticed, the two types of Level 1 clusters can be dependent (e.g. a RM can be a MHU or a SHU). Therefore, we decided to further split our users into what we call a Level 2 clustering in order to get rid of this dependency. The Figure 5.6 illustrates the Level 2 clusters: Multiple Hotel Revenue Managers (MHRM), Single Hotel Revenue Managers (SHRM), Multiple Hotel Access Users (MHAU) and Single Hotel Access Users (SHAU).

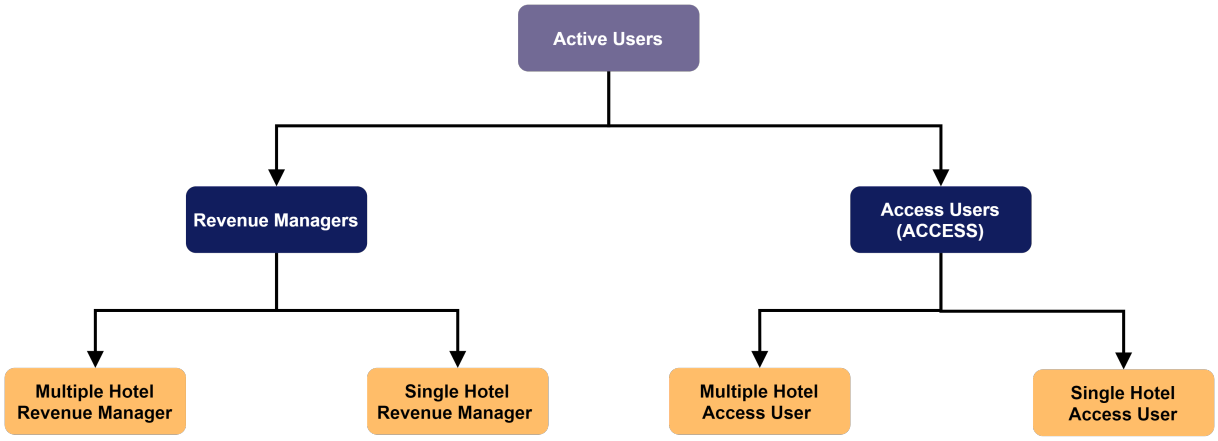

Figure 5.6: **Level 2 clustering diagram**

- **Multiple Hotel Revenue Managers**
  MHRM has the same privileges as RM and manages multiple hotels.

- **Single Hotel Revenue Managers**
  SHRM is also a RM but he manages an only one hotel.

- **Multiple Hotel Access Users**
  MHAU has the same limited privileges of AU and manages multiple hotels.
- **Single Hotel Access Users**
  SHAU is an AU that manages one single hotel.

The Level 2 clusters are indeed independent. However, we had some doubts that splitting the community into this granular level would emphasize a balance issue between clusters. The Figure 5.7 proves that the split resulted in a somehow balanced clusters with **28.19%** and **24.35%** for SHRM and MHRM respectively. As for AU, **29.56%** and **17.89%** for MHAU and SHAU respectively which is more or less good enough to produce decent results. Thus, we decided that no over-sampling or under-sampling is required and proceeded with the data as it is.

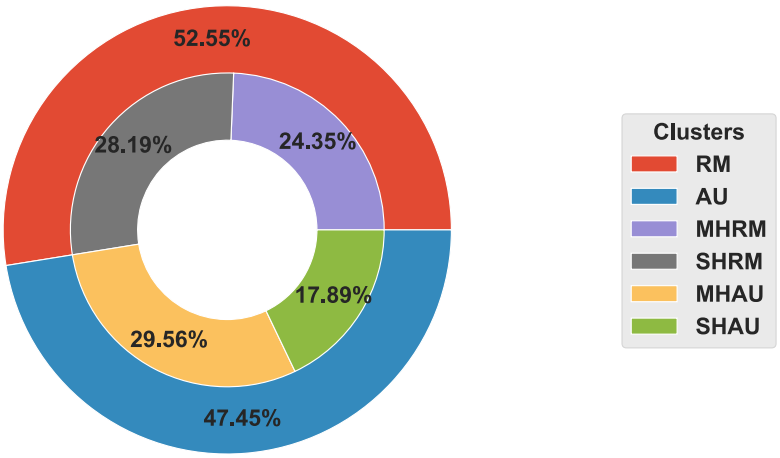

Figure 5.7: **Clusters proportions**

5.6 Conclusion

In this chapter we were able to unveil important trends and patterns within our data. These patters will orient our choices and the approach we need to take as explained in the following chapters.

Chapter 6

Achievement II: Recommendation

6.1 Introduction

With the exploratory data analysis' help, we were able to identify hidden patterns that might affect the performance of our models. In this chapter, we will present the approach we are taking and the models we implemented in order to solve the recommendation task.

6.2 Approach

In the second chapter, we discussed several approaches to solve the recommendation problem. In our use case, users sequentially visit pages on EzRMS' web interface and generally there is a meaning to the sequence (i.e. first the user checks the prices of his competitors then edits his prices based on that). Knowledge-Based RS involve users too much and it is well known that users hate to fill forms or answer questions. Also, KBRS do not take into consideration the meaning of the navigation sequence, therefore, KBRS are out question. As for Content-Based RS, EzRMS' items, or pages, are really close in terms of context: all of them involve financial reports, income, optimization, etc. So it will be really tricky and time consuming to hand-engineer the user and item profiles. CBRS are out of consideration due to the latter limitations and also to the fact that they do not take into consideration the sequential information as well. Collaborative Filtering requires ratings in order to build the item-user matrix. However, in our case, users cannot provide ratings, likes or any type of explicit feedback on the items. There

was a possibility to build the item-user matrix using only implicit feedback like the frequency of visit for each page and the time spent on each page. Unfortunately, the latter is not reliable because as we previously mentioned it is impossible to do a full tracking for the user's activity and the only thing we can for sure intercept is his clicks. Accordingly, in order to compute the time on screen we had to subtract two consecutive clicks' timestamps which is an issue if the user leaves the screen active, grabs a coffee and then resumes his navigation. So, we are left off with only the frequency of visits, and thus we gave up on CFRS. Our last option was Session-Based RS. These systems only require navigation history, so there is no need for the user to fill some kind of form or for us to hand-engineer features. SBRS are compatible with our use case since they capture the sequential information and exploit it in order to provide recommendations. Our goal is to build a next-item RS that predicts the intent of the user and provides him with the next pages to visit. Unlike classic systems that recommend the top-n elements of all times, next-item RS are trained in order to predict one step ahead based on the current session and the short-term intent. The Table 6.1 describes the notations used in the following sections.

Notation	Meaning
i	an item
s	a session
n	number of items
m	number of sessions
$\mathbf{v}$	a vector (represented by a bold lowercase letter)
$\mathbf{M}$	a matrix (represented by a bold uppercase letter)

Table 6.1: **Notation list**

6.3 Models

The literature introduces a variety of algorithms for SBRS. In this work we were able to only focus on five of them that we chose specifically because each models the recommendation problem from a completely different perspective.

6.3.1 Session Popularity

SPOP is a simple algorithm that only takes into consideration the popularity of the items. First, it computes the normalized global popularity of all items via the Equation 6.1 and then stores the computed values in the tensor $\mathbf{g}_{[n\times 1]}$.

$$\mathbf{g}(i) = \frac{popularity_i}{1 + popularity_i} \tag{6.1}$$

The second step would be to compute the local popularity : the number of visits in the current session on a certain item (i.e. for the session [i_1, i_1, i_1, i_8] the local popularity of i_1 is 3 and 1 for i_8). The local popularity varies with the session's variation so the recommendation list changes during the session as items gain more click events. The final phase would be to compute the sum between the global and local popularity in order to extract the session popularity ranking score. We are computing the global popularity in order to avoid ties (i.e. for the session [i_1, i_8] the local popularity for both items equals to one). SPOP might seem quite simplistic, however, it is extremely efficient in domains with high repetitiveness.

6.3.2 Item K-Nearest Neighbours

This model focuses only on the actual item in the session. Items similar to the current one are recommended after a similarity computation between the vector of their sessions. Technically, this is done by creating the item-session matrix $\mathbf{V}_{[n\times m]}$. The values in this matrix obey the Equation 6.2:

$$\mathbf{V}(i,s) = \begin{cases} 1 & \text{if } i \text{ in } s \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

After building the item-session matrix, the next step would be to build the item-item similarity matrix $\mathbf{I}_{[n\times n]}$. This is done by computing cosine similarity between all pairs of items via the regularized Equation 6.3:

$$\text{similarity}(i_p, i_q) = \frac{\text{\# of co-occurrences}(i_p, i_q)}{\sqrt{(\text{\# of session } (i_p) + \lambda) \times (\text{\# of session } (i_p) + \lambda)}} \tag{6.3}$$

The λ is a regularization term that penalizes the rare items in order to avoid coincidental high similarities (i.e. incidental co-occurrences). The matrix $\mathbf{I}$ is symmetric

which is quite convenient for storage purposes since we can store half of it. Each time a user visits a page, the system uses the matrix **I** in order to provide the most relevant recommendations. This model is one of the most common item-to-item solutions, that provides recommendations in the "*others who viewed this item also viewed these ones*" setting that we previously discussed for the CFRS. So IKNN is a CFRS adapted to work in a SBRS setting. Despite of its simplicity it is usually a strong baseline as shown in [Linden et al., 2003] and [Davidson et al., 2010].

6.3.3 Product to Vector

PROD2VEC was first introduced in [Grbovic et al., 2015]. Originally, it was used for product recommendation using e-mail receipts as purchase history logs. The model involves learning a representation of products in low-dimensional space using neural language models. The paper proposed several variants (namely *bagged-PROD2VEC* and *PROD2VEC-cluster*) but for this work we used the simple version of PROD2VEC. The latter models the purchase sequence as a "sentence" and the items as "words", borrowing the terminology from the Natural Language Processing (NLP) domain.

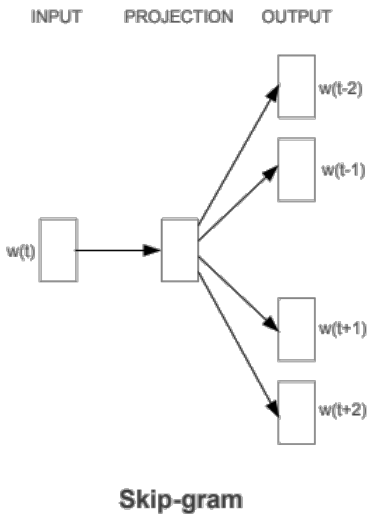

Figure 6.1: **Skip-gram architecture [Mikolov et al., 2013]**

More specifically, this algorithm is using the Skip-gram model [Mikolov et al., 2013] in order to compute item embeddings. Then, recommendations are generated by returning the k-nearest neighbours of the last purchased item, or visited item in our case.

The Skip-gram model is used to predict context words given a target word. As illustrated in the Figure 6.1, the model takes as input a target word w(t) that runs through a hidden layer and ends up with an output layer with the probability of each word appearing to be in the same context of w(t) at a given location limited by a window size.

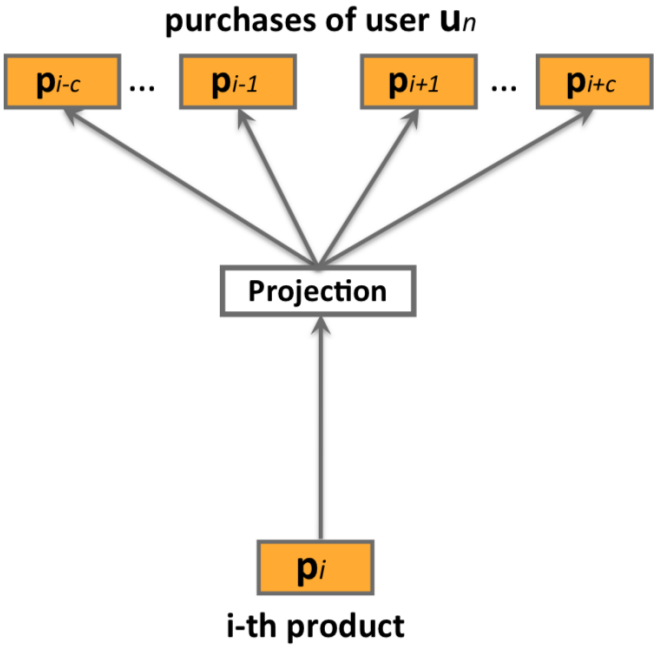

Figure 6.2: **PROD2VEC architecture [Grbovic et al., 2015]**

The same logic is applied by [Grbovic et al., 2015] as shown in the Figure 6.2. After generating the probabilities of the context items, the model performs a nearest-neighbour search in order to provide the most relevant recommendations. The main issue with this approach is that we are splitting the the navigation session into small chunks. Therefore, we are losing the flow of the navigation (i.e. the order relationship between items). For a better understanding, we explained PROD2VEC's hyper-parameters as shown in the Table 6.2.

Hyper-parameter	Meaning
embeddings size	The size of the item representation vector
window size	The size of the context window
decay rate	The exponential decay factor used to discount the past interactions. Lower values mean higher discounting.

Table 6.2: **PROD2VEC hyper-parameters**

6.3.4 Gated Recurrent Units for Recommendation

First introduced in [Hidasi et al., 2016], GRU4REC is a novel algorithm that represents a revolution for the SBRS. It is among the first algorithms that successfully adapted deep learning for the purpose of Session-Based recommendations and it provided state-of-the-art results beating the strongest baselines. This model exploits an advanced form of Recurrent Neural Network's (RNN) called Gated Recurrent Units (GRU) [Chung et al., 2014]. These units are a simpler form of Long Short Term Memory (LSTM) units [Schmidhuber et al., 1997] but in several cases, they can provide nearly the same performance with much less computations and resources.

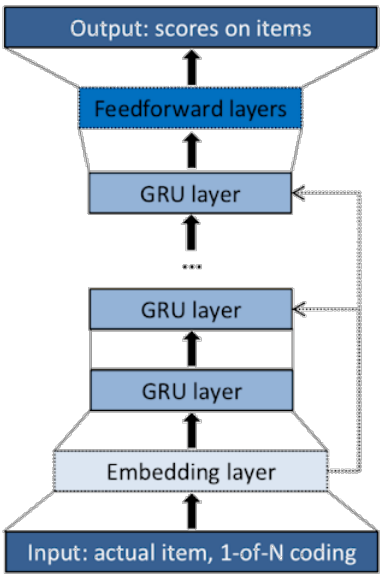

Figure 6.3: **General architecture of the network [Hidasi et al., 2016]**

The Figure 7.6 illustrates a generalized architecture of GRU4REC. The input of the model is the actual state of the navigation session and the expected output is the next-item within the same session. The navigation state can either be the actual item or all of the items visited so far. In the former state, the model takes in an encoded representation of the item (i.e. 1-of-N coding also known as One-Hot Encoding [URL2] or item Embeddings [URL5]). The latter setting adapts a weighted sum of the item representations, in order to discount visits that occurred earlier in the session. GRU are in the core of this model and the feedforward layers are at the edge just before the output layer . The expected output is the predicted preference of the items, (i.e. the likelihood of being the next in the session for each item). RNNs are known to excel in manipulating sequential data and more specifically in NLP tasks. Usually, solving those tasks requires the use of in-sequence mini-batches. For example, it is common to use a sliding window over the words of sentences and put these windowed chunks next to each other to form mini-batches exactly like we did in PROD2VEC. However, this does not fit our need because the length of sessions varies a lot and there are even sessions with only two click events while other may range over a hundred.

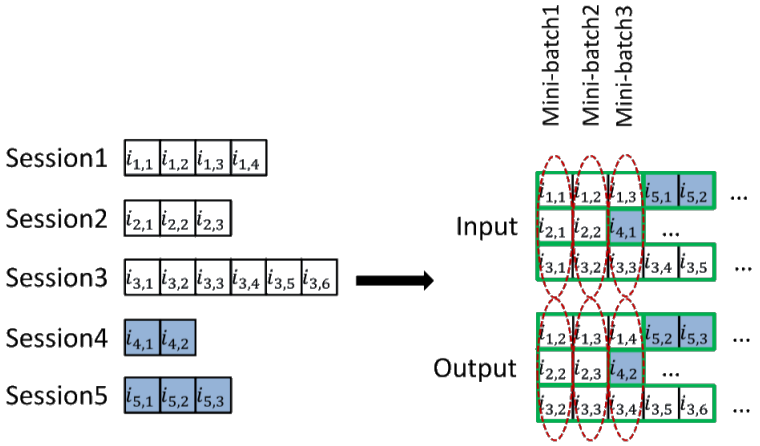

Figure 6.4: **Session-parallel mini-batch creation [Hidasi et al., 2016]**

Also, as we previously mentioned, when we split the session into small chunks, we lose the evolution of events. Therefore, the paper [Hidasi et al., 2016] proposed a creative way to tackle this issue: session-parallel mini-batches. Like shown in the Figure 6.4, we sort the sessions and then we use the first events from each as an input of the first mini-batch. The desired output is the second event of those sessions we selected.

The second mini-batch is formed from the second events and so on. In the case that one of the sessions ends, we replace it with the next available one. Since we assume that sessions are independent in this model, we reset the hidden state when we switch between sessions. Now that we have a global understanding of the model, let's jump into some specifics. So far, we exposed the general architecture of GRU4REC via the Figure 7.6 but we did not unveil the exact architecture we used.

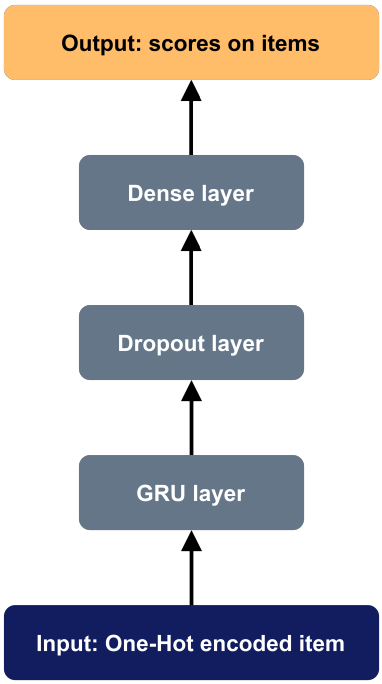

Figure 6.5: **GRU4REC adapted architecture**

The original paper [Hidasi et al., 2016] proved the efficiency of an architecture that's composed of one GRU layer that gets as input One-Hot encoded items directly without any embedding layers. The GRU layer is followed by a Dropout layer and a Feedforward layer, also known as Dense layer, exactly as shown in the Figure 6.5. Due to the complexity of tweaking the architecture and testing its efficiency, we decided to adapt the proposed architecture as it is. However, we performed an extensive Grid Search process in order to adapt those parameters to our data needs. The Table 6.3 lists the GRU4REC's hyper-parameters and explains their meanings.

Hyper-parameter	Meaning
learning rate	The step size at each iteration while moving toward a minimum of a loss function
dropout rate	The regularization rate
batch size	The number of samples processed before the model is updated
epochs	The number times that the learning algorithm will work through the entire training set
hidden units	The number of units
loss function	A function that maps through one or multiple variables the cost of an event
optimizer	The function that updates the weight parameters to minimize the loss function.

Table 6.3: **GRU4REC hyper-parameters**

6.3.5 Factorizing Personalized Markov Chains

All of the models discussed so far are non-personalized models which means they have no idea who the user is and cannot establish a relationship between him and his personal navigation sessions. Those models only process relations between item co-occurrences with different contextual windows in order to create more or less generalized models that solve the next-item problem. FPMC is a hybrid model that combines Markov chains and CFRS Matrix Factorization models [Rendle et al., 2010] in order to create a personalized SBRS. Personalized SBRS, also known as Sequence-Aware Recommendation System (SARS), not only handle short-term preferences but also the long-term ones. As shown in the Figure 6.6, the model starts by building transition matrices for each and every user. These transition matrices are filled with maximum

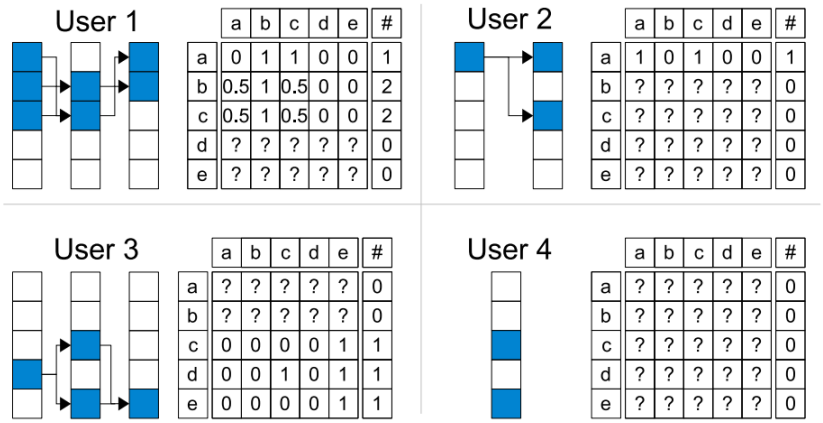

Figure 6.6: **Personalized Markov Chains transition matrices [Rendle et al., 2010]**

likelihood estimation probabilities exactly as detailed in the paper [Rendle et al., 2010]. As an oversimplification, you can think of those probabilities as frequency of visits for a certain item i_p given the fact that a user visited another item i_q prior to his visit to i_p. Entries with "?" are missing values as there is no data to estimate the probabilities.

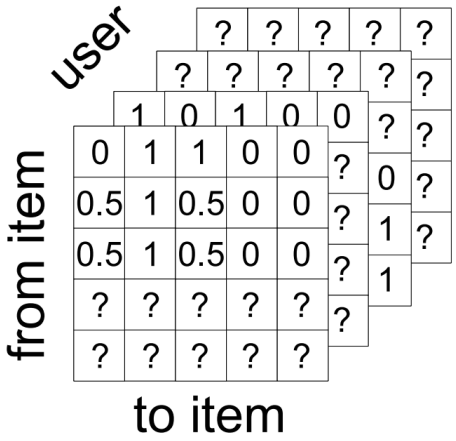

Figure 6.7: **FPMC transition cube [Rendle et al., 2010]**

This problem is solved by factorizing the transitions. Thanks to the transition cube that we were able to create after stacking all of the users' transition matrices, like show in the Figure 6.7, we can estimate those unknown entries efficiently. For an in-depth understanding the Table 6.4 lists FPMC's hyper-parameters and their meanings.

Hyper-parameter	Meaning
learning rate	The step size at each iteration while moving toward a minimum of a loss function
factor	the number of latent factors
regularization rate	The regularization rate
loss function	The function that updates the weight parameters to minimize the loss function.
negative samples	The number of negative samples used in learning.

Table 6.4: **FPMC hyper-parameters**

6.4 Conclusion

So far, we discussed the approach we are taking and the models we adapted in order to build an efficient next-item RS. In the next chapter, we will address the evaluation strategy and its results on each cluster for each model.

Chapter 7

Achievement III: Evaluation

7.1 Introduction

The exploratory data analysis, previously discussed, allowed us to discover patterns within our data and cluster our sample users based on their navigational behaviour. The latter, oriented the approach choice and the models we adapted in order to build an efficient next-item Recommendation System. In this chapter, we will explain our evaluation strategy, present its results and try to interpret them.

7.2 Evaluation strategy

Models discussed in the previous chapter use different approaches and have different perspectives for the recommendation problem. Therefore, in order to assess these models, we need to build a suitable evaluation strategy and meticulously choose the evaluation metrics.

7.2.1 Clustering validation

In the insights chapter, we discovered the existence of clusters within our sample users and as we already mentioned, the majority of the used algorithms are not personalized. Therefore, using these clusters in order to train our models might enhance their performance. Even FPMC, the only personalized algorithm we adapted, can potentially benefit from the clustering because it uses Matrix Factorization in order to predict

unknown transitions.

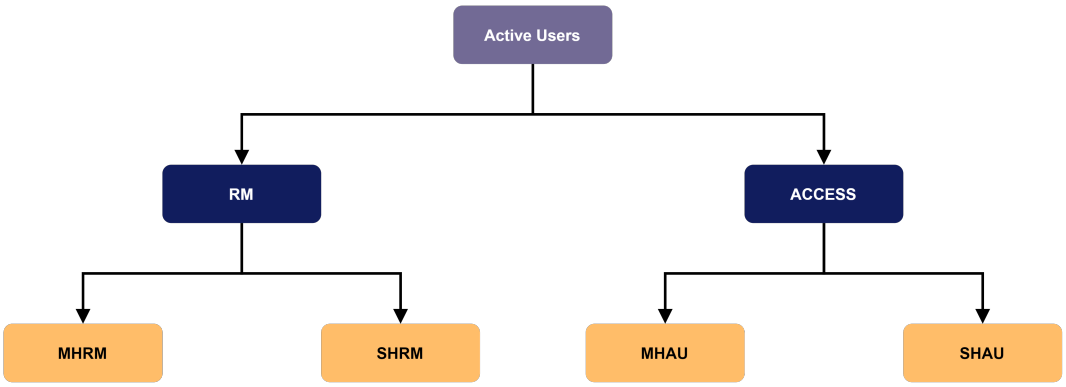

Figure 7.1: **Simplified Level 2 clustering diagram**

The Figure 7.1 represents the Level 2 clustering diagram in a simplified form. In order to evaluate the clustering impact on the models' accuracy, we adapted a three phased assessing method.

1. **Phase 1**

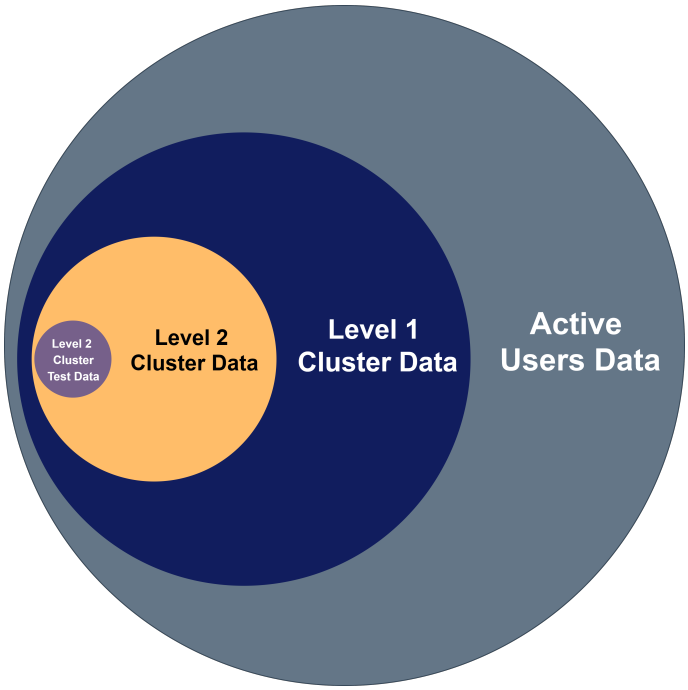

Figure 7.2: **Cluster validation strategy**

First we train a model on all of the active users' data. Then, we train another model on the adequate Level 1 cluster (e.g. if we want to evaluate the MHRM cluster, the adequate Level 1 clusters are RM and MHU). Obviously for both models we exclude the Level 2 cluster test set, the purple circle illustrated in the Figure 7.2, from each training set.

2. **Phase 2**
   The second phase consists in evaluating the models we trained in phase 1 as summarized in the Table 7.1 :

Train on	**Test on**
All of the active users' data Type 1 Level 1 cluster data Type 2 Level 1 cluster data Level 2 cluster data.	Level 2 cluster test data.

Table 7.1: Cluster validation method

3. **Phase 3**
   The final phase is quite simple. All we have to do is to train the model on the Level 2 cluster (MHRM, SHRM, MHAU, or SHAU) training set and then assess this model on the test set corresponding to the same cluster. Finally, we compare the results of the four models and pick the best performing one.

7.2.2 Train-Test split

As mentioned in the previous chapter, we are working on **2019** history logs expanding from January to December. In order to make sure that our model generalizes over the whole year's behaviour, we built a custom Train-Test splitting strategy as illustrated in the Figure 7.3.

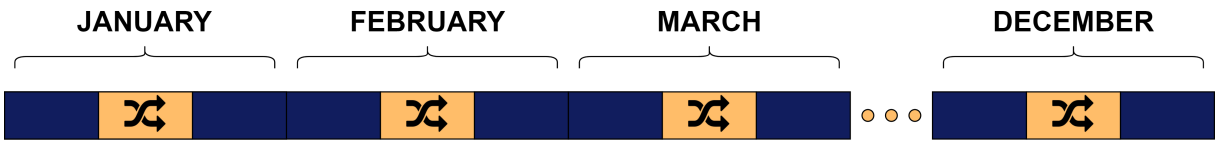

Figure 7.3: **Custom-built Train-Test split strategy**

From each month we randomly select one session from each user's monthly sessions. We assign those sessions to the test set and the rest goes to the training set. The test set will always include $\mathbf{12}\boldsymbol{x}$ session where $\boldsymbol{x}$ represents the number of users. With this method, not we only cover the whole year but we shuffle the set so that the model's results will not be biased on a certain period of time.

7.2.3 Metrics

Choosing the adequate evaluation metrics is of extreme importance to correctly assess the models. Therefore, we did not deviate from the metrics choice adapted in the original papers of the implemented models : Recall and Mean Reciprocal Rank (MRR).

1. **Recall**
   Recall is a measure of completeness that determines the fraction of relevant items retrieved out of all relevant items. Recall@k is the proportion of relevant items found in the top-k recommendations.

$$Recall@k = \frac{\text{\# of recommended items@k that are relevant}}{\text{total \# of relevant items}}$$

2. **MRR**
   Mean Reciprocal Rank is a statistic measure for assessing list of possible outcomes ordered by probability of correctness. MRR@k is the average of the reciprocal ranks of results for a sample of queries Q.

$$MRR@k = \frac{1}{|Q|}\sum_{p=1}^{|Q|}\frac{1}{rank_p}$$

   where $rank_p$ refers to the rank position of the first k elements in the p-th query.

7.2.4 Sequential revealing

The quality of the recommendations is evaluated via a sequential revealing process. We adapted this particular process in order to evaluate our models based on the next-item predictions he's able to provide. However, even though we are trying to predict the next-item, we are assessing the recommendations on the top-5 elements. The metrics

formulas discussed above are generalized formulas. In order to get a better intuition on the sequential revealing process, let's express those formulas in basic forms:

$$Recall@5 = \frac{\text{\# of predicted items in the ground truth}}{\text{\# of ground truth items}}$$

$$MRR@5 = \frac{1}{\text{rank in predictions}}$$

And for the sake of explanation, let's hand compute Recall and MRR in a setting where we sequentially reveal a session: Let's suppose that our test session is [i_1, i_5, i_8].
P.S : *The business analysts advised us to use k=5 because displaying more than 5 elements in the interface might affect the user experience.*

- **Step 1**
  Reveal the first item to the model, get its predictions based on the ground truth (i.e. desired next-item) and compute the $Recall@5_{step}$ and $MRR@5_{step}$ for the current step.

  Current sequence = [i_1], Ground truth = [i_5], Predictions = [i_3, i_5, i_4, i_6, i_8]

  $$MRR@5_1 = \frac{1}{2} \qquad\qquad Recall@5_1 = \frac{1}{2}$$

- **Step 2**
  Reveal the second item to the model, get its predictions based on the ground truth and compute the $Recall_{step}$ and MRR_{step} for the current step.

  Current sequence = [i_1, i_5], Ground truth = [i_8], Predictions = [$i_2, i_6, i_{14}, i_{13}, i_4$]

  $$MRR@5_2 = 0 \qquad\qquad Recall@5_2 = 0$$

- **Step 3**
  Now we compute the average Recall and MRR of the whole navigation session

  $$SessionRecall = \frac{Recall@5_1 + Recall@5_2}{2} = \frac{1}{2}$$

  $$SessionMRR = \frac{MRR@5_1 + MRR@5_2}{2} = \frac{1}{4}$$

- **Step 4**
  So out of the scope of this simple example, the final step to execute is to compute the Session Recall and MRR for all of the test sessions in our test set and then compute the overall average which leaves us with the following formulas:

$$Recall@5 = \frac{1}{m} \times \sum_{q=1}^{m} SessionRecall_q = \frac{1}{m \times l} \times \sum_{q=1}^{m} \sum_{p=1}^{l} Recall@5_{q,p}$$

with l as the number of steps required to compute $SessionRecall_q$.

$$MRR@5 = \frac{1}{m} \times \sum_{q=1}^{m} SessionMRR_q = \frac{1}{m \times l} \times \sum_{q=1}^{m} \sum_{p=1}^{l} MRR@5_{q,p}$$

with l as the number of steps required to compute $SessionMRR_q$.

7.3 Results

7.3.1 Session Popularity

SPOP uses a global and local popularity ranking score in order to provide recommendations. It computes the global normalized popularity of items and during the session it computes the current item's local popularity, combines both scores and ranks the items. So, clustering will only affect SPOP's global popularity computation since the scope is narrower in terms of sessions within sub-groups. The Table 7.2 represents the results of the evaluation on each and every cluster. For the MHRM we are not observing much of a difference between whether we cluster users or not. This might be due to the fact that items that are popular for the MHRM are also popular for all of the other clusters due to the fact that MHRM inherits the behaviour of multiple SHU at the same time. For the opposite reasons, popular items for SHRM might not be popular in other clusters: for example, in the RM cluster we are observing lower performance because items' popularity are getting biased by the MHRM behaviour which is quite repetitive. AU can be front desk officers, room managers, etc. Therefore, the pages they're interested in are not strongly related to price optimization rather than current situation management. This explains the lead MHAU and AU cluster are taking while testing on MHAU data. As for SHAU, we observe the same behaviour of SHRM

Mutiple Hotel Revenue Manager				
Train on	**ALL**	**RM**	**MHU**	**MHRM**
Recall@5	**0.7939***	0.79144	0.7896	0.79032
MRR@5	**0.4268***	0.4214	0.4138	0.4201
Single Hotel Revenue Manager				
Train on	**ALL**	**RM**	**SHU**	**SHRM**
Recall@5	0.7039	0.7038	0.7332	**0.7407***
MRR@5	0.3223	0.3222	0.3335	**0.3359***
Multiple Hotel Access User				
Train on	**ALL**	**AU**	**MHU**	**MHAU**
Recall@5	0.7407	0.7846	0.7477	**0.7895***
MRR@5	0.3897	0.4067	0.3998	**0.4149***
Single Hotel Access User				
Train on	**ALL**	**AU**	**SHU**	**SHAU**
Recall@5	0.6651	0.7238	0.7465	**0.7537***
MRR@5	0.3055	0.3121	0.3310	**0.3315***

Table 7.2: **SPOP evaluation resuts**

where the genuinely popular items within this cluster are being biased in the other more general clusters which is why performance is better if we only use the Level 2 cluster's training data. Also, SPOP favours repetitiveness and this is quite clear in the MRR results since MHRM and MHAU easily reached **40%** while SHRM and SHAU are struggling and could not go above **33%**.

7.3.2 Item K-Nearest Neighbor

IKNN's approach is to pre-compute an item-item similarity matrix and during the session use that matrix in order provide the most similar items to the current one the user is visiting. Similarity is computed based on co-occurrence of items in the same sessions. IKNN discounts rare items with a regularization term λ. We tried several values of λ in a small Grid Search process covering values from 10 to 50 with a step of 5 and the values between 20 and 30 seemed to provide the best outcomes for each model with minor differences. So, we settled on $\lambda = 20$ which is the same value

proposed in the paper [Quadrana et al., 2018]. Clustering will only affect IKNN while computing the similarity because it heavily depends on co-occurrence of items within the same session. The Table 7.3 shows the IKKN's evaluation results.

Mutiple Hotel Revenue Manager				
Train on	**ALL**	**RM**	**MHU**	**MHRM**
Recall@5	**0.7256***	0.7133	0.6995	0.6997
MRR@5	**0.4613***	0.3850	0.3250	0.3239
Single Hotel Revenue Manager				
Train on	**ALL**	**RM**	**SHU**	**SHRM**
Recall@5	0.6544	0.7353	0.7014	**0.7528***
MRR@5	0.4259	0.4943	0.4129	**0.5082***
Multiple Hotel Access User				
Train on	**ALL**	**AU**	**MHU**	**MHAU**
Recall@5	0.6662	**0.7489***	0.6520	0.7362
MRR@5	0.4504	**0.5095***	0.3158	0.4301
Single Hotel Access User				
Train on	**ALL**	**AU**	**SHU**	**SHAU**
Recall@5	0.6663	0.7456	0.79433	**0.8011***
MRR@5	0.4607	0.5036	0.5329	**0.5479***

Table 7.3: **IKNN evaluation results**

It is quite clear that repetitiveness is impacting the performance because co-occurrence of items within sessions becomes extremely high. If we observe the results for MHRM, training on all of the active user's data provides the best performance and this is for exact same reasons we mentioned before. However, for the MHAU case, training on AU is better than training on all active user's data and this is due to the navigational behaviour that we already discussed. As for SHRM and SHAU, training on their proper data is best because the co-occurrence similarity is not poisoned by the repetitiveness. IKNN is providing better MRR@5 scores than SPOP since for certain cases, the values have reached **50%** and above but slightly worse in terms of Recall@5 especially for the MHU clusters.

7.3.3 Product to Vector

PROD2VEC uses the same logic as WORD2VEC and more specifically its Skip-gram model that extracts item embeddings using a shallow neural network and provides recommendations by a simple nearest neighbour search on the output. Embeddings are deduced after scanning sessions and understanding context through a fixed size window.

Mutiple Hotel Revenue Manager				
Train on	**ALL**	**RM**	**MHU**	**MHRM**
Recall@5	0.6345	0.6289	0.6411	**0.6610***
MRR@5	0.3882	0.3657	0.3897	**0.4132***
Single Hotel Revenue Manager				
Train on	**ALL**	**RM**	**SHU**	**SHRM**
Recall@5	0.6079	0.6252	0.6488	**0.6751***
MRR@5	0.3095	0.3046	0.3108	**0.3274***
Multiple Hotel Access User				
Train on	**ALL**	**AU**	**MHU**	**MHAU**
Recall@5	0.6058	0.5244	**0.6568***	0.4774
MRR@5	0.4107	0.3492	**0.3865***	0.2415
Single Hotel Access User				
Train on	**ALL**	**AU**	**SHU**	**SHAU**
Recall@5	0.6245	0.5576	**0.6725***	0.5800
MRR@5	0.3763	0.3220	**0.3897***	0.2977

Table 7.4: **PROD2VEC evaluation results**

The sliding window is breaking the sessions into small chunks which is a huge disadvantage because we are losing precious information about the navigation's evolution through time. The Table 7.4 shows the mediocre results PROD2VEC provided. At a first glance, we thought that this might be a hyper-parameters tuning issue. Therefore, we conducted a Grid Search on the *embeddings size*, *decay rate* and the *window* size that have as default values 64, 0.05 and 5 respectively. For each model and in each time we tweak any parameter, the performance gets worse causing an unsuccessful Grid

Search. This approach performed poorly because of the sliding window that ignored the navigational order. This fundamental flaw has made any other interpretation or explanation of the observed results irrelevant.

7.3.4 Gated Recurrent Units for Recommendation

GRU4REC takes the SBRS to a whole new level thanks to its revolutionary session-parallel approach that preserves the evolution over time of the sessions. However, GRU are complicated beasts that require an extensive hyper-parameter optimization in order to perform well. The Table 7.5 shows the results of the conducted Grid Search operation that optimized the 7 parameters for all four clusters. We tried multiple optimizers like *Stochastic Gradient Descent*, *RMSprop* and *Adam*. As for the loss function, we only used the Categorical Cross Entropy. Concerning the batch size, we tried values within the following set {16, 32, 50, 64, 128}. For the learning rate and the dropout rate, we tried {0.001, 0.005, 0.01, 0.05, 0.1} and {0.2, 0.25, 0.5} respectively. And finally, for the number of epochs and hidden units, we tried {5, 10, 15, 20, 30} and {100, 200, 500} respectively.

Hyper-parameter	**MHRM**	**SHRM**	**MHAU**	**SHAU**
learning rate	0.001	0.001	0.001	0.005
dropout rate	0.25	0.2	0.25	0.2
batch size	50	50	32	32
epochs	10	5	5	5
hidden units	200	100	100	100
loss function	Categorical Cross Entropy			
optimizer	Adam			

Table 7.5: **GRU4REC optimized hyper-parameters**

This large scale Grid Search operation was fruitful since we obtained astonishing results. As illustrated in the Table 7.6, training with those optimized hyper-parameters offered a high-quality performance RS that easily beats all of the previous ones. GRU4REC is a deep learning algorithm. The latter perform exceptionally well with large amounts of data. It easily detects and exploits complex relationships and hidden patterns within

the data and provide accurate results. This is exactly the case for MHRM an MHAU as training on all active users' data provides more training samples for the algorithm, and thus a better exposure on the complex and repetitive behaviour they have. For the same reasons, SHU represents the cluster with the most data samples that best describes the behaviours for SHRM and SHAU.

Mutiple Hotel Revenue Manager				
Train on	**ALL**	**RM**	**MHU**	**MHRM**
Recall@5	**0.8696***	0.8471	0.8486	0.8445
MRR@5	**0.6531***	0.6522	0.6510	0.6495
Single Hotel Revenue Manager				
Train on	**ALL**	**RM**	**SHU**	**SHRM**
Recall@5	0.7737	0.7752	**0.8486***	0.7923
MRR@5	0.5551	0.5692	**0.5756***	0.5707
Multiple Hotel Access User				
Train on	**ALL**	**AU**	**MHU**	**MHAU**
Recall@5	**0.8216***	0.8116	0.8177	0.8129
MRR@5	**0.6554***	0.6391	0.6397	0.6235
Single Hotel Access User				
Train on	**ALL**	**AU**	**SHU**	**SHAU**
Recall@5	0.7731	0.7878	**0.8032***	0.7789
MRR@5	0.5865	0.5802	**0.5917***	0.5772

Table 7.6: **GRU4REC evaluation results**

7.3.5 Factorizing Personalized Markov Chains

Mutiple Hotel Revenue Manager				
Train on	**ALL**	**RM**	**MHU**	**MHRM**
Recall@5	0.7808	0.7774	**0.7943***	0.7893
MRR@5	0.4840	0.4897	**0.5109***	0.4827
Single Hotel Revenue Manager				
Train on	**ALL**	**RM**	**SHU**	**SHRM**
Recall@5	0.8483	0.8533	**0.8600***	0.8574
MRR@5	0.4918	0.5052	**0.5943**	0.5713
Multiple Hotel Access User				
Train on	**ALL**	**AU**	**MHU**	**MHAU**
Recall@5	0.8339	0.8399	**0.8452***	0.8371
MRR@5	0.5582	0.5813	**0.5943***	0.5491
Single Hotel Access User				
Train on	**ALL**	**AU**	**SHU**	**SHAU**
Recall@5	0.9253	0.9187	**0.9289***	0.9276
MRR@5	0.6331	0.6201	**0.6449***	0.63043

Table 7.7: **FPMC evaluation results**

FPMC is a hybrid model that combines Markov Chains with Matrix Factorization in order to provide accurate personalized recommendations. In order to optimize FPMC's performance, we conducted yet another extensive Grid Search process covering all 5 hyper-parameters for each cluster as shown in the Table 7.8. We tried the following values for the learning rate and the regularization rate, {0.001, 0.005, 0.01, 0.05, 0.1} and {0.001, 0.002, 0.005, 0.01, 0.02, 0.05} respectively. Concerning the latent factor size and the number of negative samples we tried {32, 64, 128, 256} and {5, 10, 15, 20, 25}. The Bayesian Ranking Process is used by FPMC in order to perform the Matrix Factorization process so there is no need to change it. As you may have noticed, FPMC seems to be somehow immune to the clustering as all Recall values are really close. The clustering process only affects FPMC while building the transition cube and trying to guess the unknown values and it seems that the Level 1 Type 2 clusters (i.e. MHU and

SHU) are providing the best performance in terms of not only Recall but MRR too.

Hyper-parameter	**MHRM**	**SHRM**	**MHAU**	**SHAU**
learning rate	0.01	0.01	0.01	0.01
regularization rate	0.005	0.001	0.001	0.001
factor	128	32	64	32
loss function	Bayesian Ranking Process			
negative samples	15	10	10	15

Table 7.8: **FPMC optimized hyper-parameters**

7.4 Comparison

So far, the clustering strategy was a huge success since we observed a better performance while training on clustered data rather than training on the whole sample. In this section, we will summarize the evaluation results by illustrating the best performing models for each Level 2 cluster.

7.4.1 Multiple Hotel Revenue Managers

As shown in the Figure 7.6, GRU4REC is the top performing model leaving behind the personalized FPMC with an **8%** and **14%** gap in terms of Recall and MRR respectively. The reason behind this excellent performance is the fact that GRU4REC can dynamically extract relationships hidden within data and exploit them to enhance the accuracy which is the case for MHRM since their behaviour is complex and repetitive. As for SPOP, it is performing really well since it is beating PROD2VEC and strongly competing with IKNN which is a much more advanced approach. This unexpected high performance is due to the fact that SPOP handles really well the repetitiveness as explained previously.

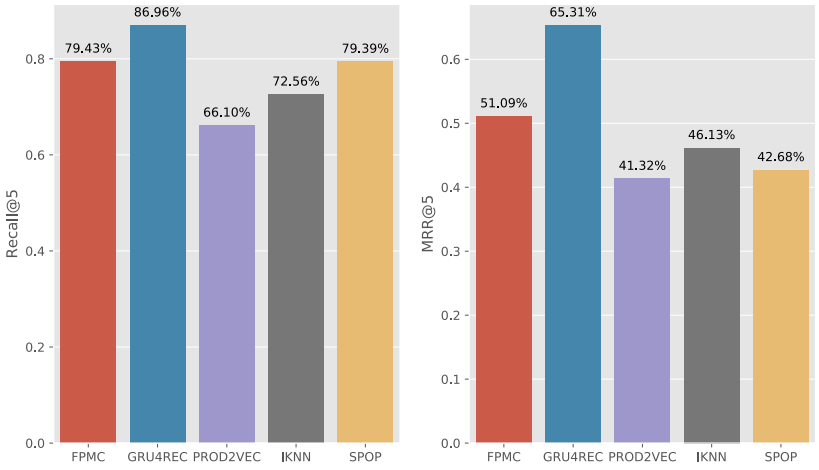

Figure 7.4: **MHRM best performing models**

7.4.2 Single Hotel Revenue Managers

The Figure 7.5 shows that GRU4REC and FPMC are closely competing over the first place with the latter beating the former with just **2%** for both Recall and MRR.

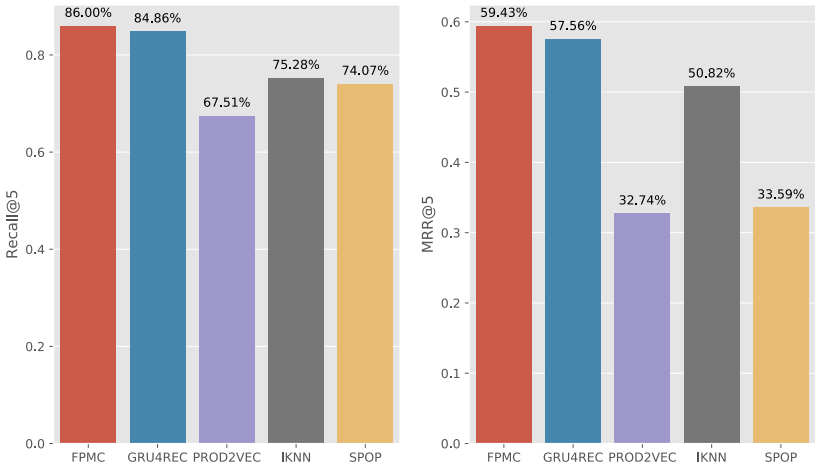

Figure 7.5: **SHRM best performing models**

Also, IKNN is providing a quite decent performance with **75.28%** of Recall and **50.82%** of MRR. SPOP is still in competing as it is providing a Recall score close to the score IKNN provided. However, if we take a closer look at the MRR related to that same Recall score, it is lower than IKNN's score and much lower compared to the best performing model, with nearly **25%** difference, but still beating the more complex PROD2VEC model.

7.4.3 Multiple Hotel Access Users

Once again, as shown in the Figure 7.6, FPMC is beating GRU4REC for the highest Recall score with a slight difference of **2%**. However, in terms of MRR, there is a margin of **6%** that favours GRU4REC over FPMC.

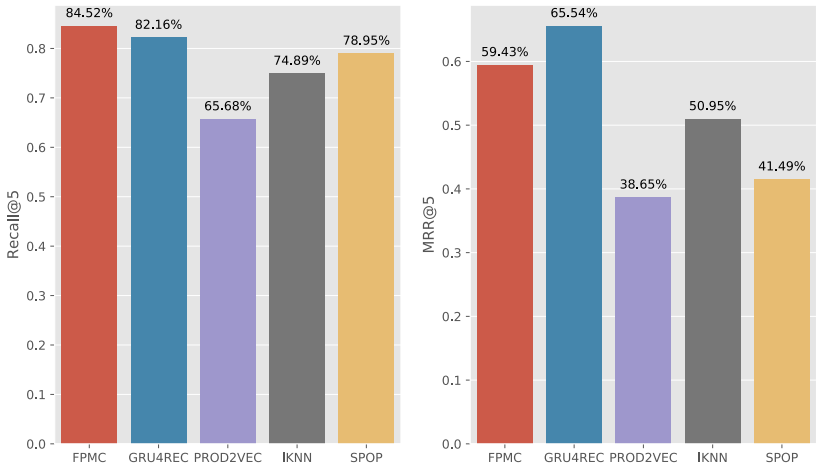

Figure 7.6: **MHAU best performing models**

As we already mentioned, deep learning approaches excel in modelling complex behaviour. In our case, the complex behaviour is expressed via MHAU's and MHRM's repetitiveness which explains why GRU4REC is performing well in those clusters. SPOP is back on the race with decent scores of **78.95%** and **41.49%** for Recall and MRR respectively beating IKNN in terms of the former but not the latter since IKNN is taking a lead of **9%**. PROD2VEC is still performing poorly with **65.68%** of Recall and **38.65%** of MRR.

7.4.4 Single Hotel Access Users

The Figure 7.7 shows a huge gap between GRU4REC and FPMC. The latter is dominating the race with a large margin of **12%** in terms of Recall and a lead of **5%** in terms of MRR. The other models are not doing extremely bad since IKNN has reached **80%** of Recall for its first time and even PROD2VEC has finally beaten the simplistic SPOP baseline in terms of MRR. However, FPMC is modelling SHAU the best and providing astronomical scores.

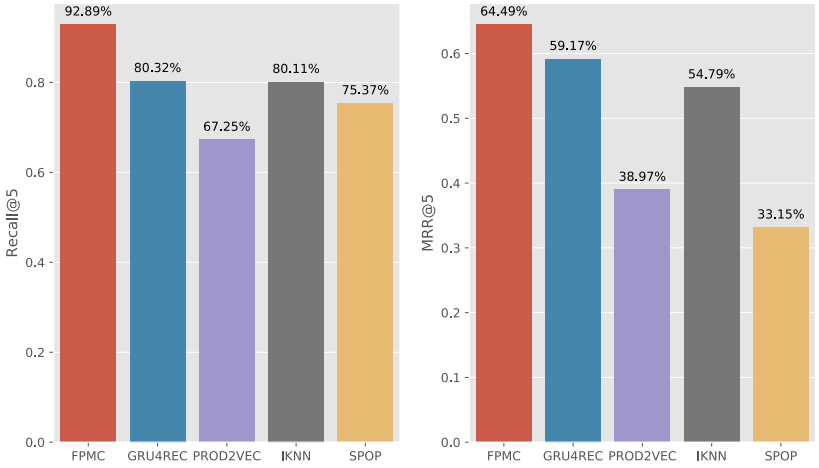

Figure 7.7: **SHAU best performing models**

7.5 Conclusion

In this chapter we explained our custom-built evaluation strategy that provided common grounds for all models, with their different perspectives and different approaches, to get evaluated. Furthermore, we applied that strategy and exposed the results of each model and each cluster. Also, we compared the models and provided concise interpretations in order to have better intuition on why we have such results.

Conclusion & Future work

Data overflow is a serious issue that shackles the user experience making consumers spend too much time scrolling in piles of information and filtering out unwanted content. This was the driving force for the rise of Recommendation Systems. Thanks to their flexibility, such systems can be embedded within any application and skyrocket its market value by not only enhancing the user experience but also by playing an important role in optimizing revenue.

EzRMS, the revenue management system developed by Infor, suffers from this overflow issue and has started losing its market dominance with new products emerging in the business. Thus, in order to keep up with the fierce competition, enhancements for the interface took place with the addition of a Recommendation System feature. This is where our contribution took place as we successfully conceived, built and evaluated a highly accurate next-item Recommendation System that tracks the user behaviour, learns from it and predicts his intents.

This document describes the phases this project went through in order to build a Recommendation System that will contribute in EzRMS' race for market dominance.

The first chapter concerns the general context of this work and addresses the problem formulation and the methodology adapted in order to reach the desired results. Next, we exposed the requirements for this system, its software attributes and its development life cycle. Furthermore, we unveiled the blueprint of this project and detailed its design methodologically from a simple intuition to a crystal-clear perspective. After carefully laying out the development strategy, we started exploiting our resources and told stories from piles of what appeared to be meaningless data. Those stories en-

lightened the path by exposing hidden patterns and trends that oriented the approach we took and the models we adapted as explained in the sixth chapter. This work was concluded with the evaluation and interpretation of the observed results.

Even though we faced a lot of hardships while working on this project (i.e. the pandemic situation), we were able to deliver a high-quality system that satisfied the objectives and even exceeded them. Still, there are some enhancements that can be made. In this project, we adapted an offline evaluation strategy instead of an online one because deploying the solution into the production level product requires a lot of preparations which is out of scope for this internship. However, an online A/B testing is much more reliable than offline one because you will have the ability to assess how users will actually react and use the new feature in real-time. Also, while working on this project we invested a lot of effort in non-personalized systems instead of personalized ones. This is due to the fact that the literature supports better those systems and to the fact that we needed strong baselines to work with. Luckily, these prospects can be easily achieved thanks to the rigorous design of the Recommendation System and to the fact that it is deployment-ready.

Bibliography

[Aggarwal, 2016] Aggarwal, C. C. (2016). *An Introduction to Recommender Systems*. Springer, Cham.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. *arXiv:1412.3555 [cs.NE]*.

[Davidson et al., 2010] Davidson, James, Liebald, Benjamin, Liu, and Junning (2010). *The YouTube video recommendation*. *Recsys'10: ACM Conf.*

[Grbovic et al., 2015] Grbovic, Mihajlo, Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V., , and Sharp, D. (2015). *E-commerce in your inbox: Product recommendations at scale*. *ACM pp. 1809-1818.*

[Hidasi et al., 2016] Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2016). *A Survey on Session-based Recommender Systems*. *arXiv:1511.06939 [cs.LG]*.

[Jannach and Jugovac, 2019] Jannach, D. and Jugovac, M. (2019). Measuring the business value of recommender systems. *arXiv:1908.08328 [cs.IR]*.

[Koren et al., 2009] Koren, Y., Bell, R., and Volinsky., C. (2009). *Matrix Factorization Techniques for Recommender Systems*. *IEEE*.

[Linden et al., 2003] Linden, Smith, and York (2003). *Amazon.com recommendations: Item-to-item collaborative filtering. Internet Computing*. *IEEE, 7(1):76–80.*

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean., J. (2013). *Distributed representations of words and phrases and their compositionality*. *NIPS, pages 3111–3119*.

[Quadrana et al., 2018] Quadrana, M., Cremonesi, P., and Jannach, D. (2018). *Sequence-Aware Recommender Systems*. *ACM Computing Surveys, Vol. 51, No. 4, Article 66.*

[Rendle et al., 2010] Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. (2010). *Factorizing personalized Markov chains for next-basket recommendation*. *WWW'10. 811–820.*

[Schmidhuber et al., 1997] Schmidhuber, J., Hochreiter, S., and Hochreiter, S. (1997). *LONG SHORT-TERM MEMORY*. *Neural Computation 9(8):1735-1780*.

[Çano, 2017] Çano, E. (2017). Hybrid recommender systems: A systematic literature review. *Intelligent Data Analysis*.

Netography

[URL1] Google. Content-based filtering. developers.google.com/machine-learning/recommendation/content-based/basics, Last visit 02/26/2020.

[URL2] One hot encoding. Vasudev. https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f, Last visit 04/12/2020.

[URL3] Netflix. Netflix prize dirichlet allocation. netflixprize.com, Last visit 02/21/2020.

[URL4] Baptiste Rocca. Introduction to recommender systems. towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada, Last visit 03/03/2020.

[URL5] Wikipedia. Embeddings. https://en.wikipedia.org/wiki/Embeddings, Last visit 04/12/2020.

ملخص

يهدف هذا المشروع إلى بناء نظام ترشيحات قادر على أن يتنبأ بالصفحات التي سينقر عليها المستخدم. سوف يتعلم هذا النظام من سلوك المستعمل من أجل استخراج توجهاته والصفحات المفضلة لديه بهدف الحصول على أفضل البيانات و تقديم ترشيحات ذات دقة عالية. سيتم دمج هذا المحرك في واجهة إنفور إزي آر آم آس وسيعمل كأداة تبسط التصفح وتعزز تجربة المستخدم.

كلمات مفاتيح: بايثون، تعلم الآلة، التعلم العميق ، نظام الترشيحات، تحضير البيانات

Résumé

Ce projet vise à construire un système de recommandation qui prédit les prochains éléments sur lesquels un utilisateur cliquera. Ce système exploitera le comportement de navigation afin d'extraire les tendances et les habitudes pour finalement fournir des recommandations pertinentes. Ce moteur sera intégré dans l'interface d'Infor EzRMS et servira d'outil pour simplifier la navigation et améliorer l'expérience de l'utilisateur.

Mots clés: *Python, Oracle, SQL, Apprentissage automatique, Apprentissage profond, Prétraitement des données, Systèmes de recommendation*

Abstract

This project aims to build a recommendation system that predicts the next-items a user will click on. This system will exploit navigational behavior in order to extract patterns and trends and best-fit the data in order to provide relevant recommendations. This engine will be embedded into Infor EzRMS' interface and act as tool that simplifies the navigation and enhances the user experience.

Keywords: *Python, Oracle, SQL, Machine learning, Deep learning, Data cleaning, Recommendation systems*