UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Classification of Travel Offers for Shared Mobility Applications

Author:
Alex MARTÍNEZ

Supervisors:
Dr. Oriol PUJOL
Dr. Cristian CONSONNI
Dr. Ludovico BORATTO

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

July 1, 2021

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc in Fundamental Principles of Data Science

Classification of Travel Offers for Shared Mobility Applications

by Alex MARTÍNEZ

Nowadays, there is an increasing amount of available digital data, and therefore there is a need to filter, prioritize and show personalized relevant information to the users. Recommender systems have completely changed the way we interact with many services by solving the problem of information overload. They are tools that search through large volumes of dynamically generated information to provide users with personalized content. Applications range from streaming services to online shopping websites. In this master's thesis, we build such a system for trip offers: given a mobility request from a user, we seek to provide a personalized ranking of trip alternatives which differ in aspects such as duration, distance, transportation modes, etc.

This work is organized in two parts. On the one hand, we focus on the characterization of the trip alternatives by defining 11 categories such as quick, reliable or environmentally friendly. We build a system which assigns a score to each one of these categories highlighting the particular compatibility of an alternative with that category. This first part is a direct contribution to Ride2Rail, a European project which we have used as the framework to develop this thesis. On the other hand, we use this trip representation to build a recommender system. In particular, we implement the Bayesian Personalized Ranking algorithm, which was specially designed to work with implicit feedback from the users. This algorithm relies on an underlying model class to make predictions. We take advantage of this fact to compare two approaches. Firstly, we use the Matrix Factorization, a classical collaborative filtering approach which does not take into account the previous categorization. Secondly, we use the Factorization Machines, a model which combines collaborative and content approaches to include the categorization information. The results clearly suggest that using the content information of the trip categorization increases the performance of the recommender system, which proves its usefulness.

Contents

Acknowledgements

First of all, I would like to thank all my three supervisors, Cristian Consonni, Ludovico Boratto and Oriol Pujol, for their guidance and advice during the development of this work.

Also, I want to express my deeply gratitude to Cristian Consonni and Ludovico Boratto for giving me the chance to intern at Eurecat and be able to develop this work in the framework of one of the projects of the company. With this, I want to thank all my colleagues at Eurecat, especially Luca Piras, whose contributions on the development of my work were really valuable.

I would also like to thank all the other partners involved on the project. I learnt a lot working with such experienced researchers.

Finally, I wish to thank my family and friends, who have always been of great support.

Chapter 1

Intoduction

1.1 Motivation & Goals

Route planning applications such as Google Maps have become a vital piece of technology in almost everyone's life. They are used to find an optimal means of travelling between two or more given locations, and may be constrained, for example, to leave or arrive at a certain time, to avoid certain waypoints, etc. Simple engines cover only public transport data for a single mode, others are multimodal, covering public transport data for several modes. Advanced engines may also include road and footpath routing to cover the access legs to reach the public transport stops and also simultaneously compute routes for travel by private car so that the user may make a comparison between public and private modes.

In general, these searches may be optimized on different criteria established beforehand by the user, for example *fastest, shortest, fewest changes* or *cheapest*. However, very few of them include a system which can learn the preferences of each user by itself based on their previous actions, and then, automatically show a personalized ranking. In this present work, which has been developed in the framework of the European project called Ride2Rail during my internship at Eurecat, we aim to build such an advanced system able to provide personalized rankings for trip offers.

Having said that, we organize the work in two main analysis. Firstly, we intend to build a state-of-the-art system able to categorize multimodal trip offers. This will allow us to find a powerful representation of the offers based on different dimensions (e.g. quick, reliable, cheap, etc.). Secondly, we use this trip representation to train a recommender system able to generate the aforementioned personalized ranking.

1.2 Contributions

Based on the goals we mentioned previously, we can summarize the project's contributions as follows:

- Directly related to the Ride2Rail project, we build three components that are used to implement a state-of-the-art system which is able to classify trip offers along different categories.
- Distancing from the direct objectives of the Ride2Rail project, we use the categorization to implement a matching and ranking algorithm, that also learns the user's preferences over time based on their travel choices. Users will thus benefit from the enrichment of travel offers provided, and more personalized travel solutions over time

Therefore, part of this work is built from my direct contributions to the Ride2Rail project, while in the other part we use the tools implemented to train a recommender system.

1.3 Project Outline

This work is organized as follows:

- Chapter 2 is completely devoted to the Ride2Rail project. We describe the main objectives, the organization of the project and give an overview of the expected implementation work.
- Chapter 3 covers the basic theoretical background about recommender systems necessary to understand the implementation of our system.
- Chapter 4 presents the different software technologies used for the coordination and implementation of the Ride2Rail project. Besides, a thorough description of the datasets used to test the different components is provided.
- Chapter 5 describes the implementation of my contribution to the Ride2Rail project and the recommender system we intend to build.
- Chapter 6 presents the final results regarding the performance of the recommender system
- Chapter 7 concludes this work, comments the most relevant limitations we have encountered and outlines future lines of research.

All the code produced can be found in https://github.com/alexmartinezmiguel/TFM-Travels-Offers-Classification.

Chapter 2

Ride2Rail Project

2.1 Background

In nowadays society, the car still has a very significant importance in the way we move around, occupying large parts of our streets and cities. Even though public transport in large urban areas provides a suitable alternative to private car-use, in rural areas, on the other hand, the car is still often considered as the most preferred way of traveling due to lack or infrequency of public services. Most direct implications point to traffic congestion, which is usually considered as one of the most challenging issues in large urban agglomerations. Together with congestion, people are spending an increasing amount of time commuting on their way to their workplace, and this time wasted might be at the expense of other economic or social activities. Not less important are the environmental impacts and energy consumption.

According to the European Environment Agency (*Occupancy Rates* 2020), car occupancy rate for commuting trips in EU countries is about 1.2 persons per vehicle, and up to 2 people for travel and leisure. At the same time, implications on the aforementioned issues remain the same – whether there are one or four people in one car.

As commonly considered, ride sharing has the potential to reduce the number of single occupancy vehicles. It refers to a mode of transport in which travelers share a vehicle for a trip and split costs such as gas, toll, or parking fees with others that have similar itineraries and time schedules. But even with recent mobile technologies facilitating this way of traveling, ride sharing still has to overcome many challenges to be considered a solid option for travellers (Furuhata et al., 2013). The reason for this might be a set of barriers starting from the necessity of instantaneous coordination with respect to schedules and cost-sharing among participants. This is troublesome because the proper information might not be known or even available to travellers and decisions are dynamic and instantaneous. Besides, trust is a major concern for individuals (Chaube, Kavanaugh, and Perez-Quinones, 2010). People strongly hope the person they are sharing the ride with to behave as expected. Besides, users need to trust the driving skills of the driver or whether the driver license is valid. Trust might play different roles.

2.2 Objectives

In this context is where the Ride2Rail project (*Ride2Rail* 2021) intends to play a part. It is an European project whose vision is to develop an innovative framework for intelligent mobility, facilitating the efficient combination of flexible and crowd-sourced transport services (e.g. ride-sharing) with scheduled transport services (e.g. rail or bus), thus enhancing the performance of the overall mobility system.

The main objectives of the Ride2Rail project are:

- To develop an innovative framework for intelligent mobility, facilitating efficient combination of flexible and scheduled transport services, integrating real-time information about public transportation and ride sharing.
- To create a tool that facilitates the comparison and choice between multiple options classified by a set of categories, for example environmental, travel time, comfort or cost.
- To encourage ride sharing as complementary for public transport.
- To enhance the performance of the overall mobility system, reducing road congestion and environmental impact, reinforcing the mobility offer in rural and low-demand areas.

2.3 Methodology

The methodology for Ride2Rail covers four technical workpackages. Workpackage 2 sets out the travel behaviour and system requirements for the Ride2Rail project, with an emphasis on end-user requirements for shared travel services. Workpackage 3 covers the technical development of Ride2Rail, involving mainly development of an application. Now talking of future tasks, Workpackage 4 involves the deployment of Ride2Rail at four demonstration sites, each with diverse characteristics to ensure Ride2Rail is a flexible solution. Finally, Workpackage 5 is providing the methodology to ensure the benefits and impacts of the system are fully demonstrated.

This present work has been developed in the framework of Workpackage 3 of the project. It addresses Ride2Rail's technical needs while introducing a novel system to enrich multimodal travel solutions, matching them to the user according to their preferences. In particular, Workpackage 3 is developing a state-of-the-art Offer Categorizer that enables the description of offers along 11 different categories, evaluating each trip offer along several dimensions such as comfort, environmental friendliness, and health impact.

2.4 Organization

There are several partners involved in the implementation of the Ride2Rail project and therefore the tasks were split among all of them. First of all, Eurecat is project leader of Workpackage 3, which means that we took responsibility of some of the most challenging parts. The other partners involved in the project are: University of Žilina (Slovakia), Polytechnic University of Milan (Italy), the Centre of Research & Technology (Greece) and Cefriel (Italy).

2.5 Offer Categories

Before describing the Offer Categorizer, the key element and our main contribution on this project, some concepts need to be defined. First of all, a **mobility or trip request** (identified with a *request_id*) is a petition from a user who intends to travel from an origin to a final destination at a given time (or desired arrival time). From this mobility request, a set of **offers/solutions/alternatives** (each one identified with a *offer_id*) will be generated. Essentially, they are different alternatives which can involve different routes, means of transport, schedules etc. At the same time, these offers can be formed by several **legs** (identified with a *leg_id*). They are parts of the trip involving different modes of transportation.

The offers are described by a set of objective variables (such as transportation mode, level of CO_2 emission, cost, etc.) which we call **determinant factors**. Then, the aggregation of certain determinant factors defines a score for a set of **offer categories**. As a result of this process, we will be able to attach a variable number of category labels to each offer, highlighting the particular compatibility of an offer with that category. In the list below you can find a description of all the categories considered and the determinant factors used to measure them:

- **Quick**: The Quick category measures how convenient and efficient the solution is in terms of time-related issues, considering the total travel time, the waiting time between legs, the number of stops required and the road-network distance covered. If the solution (data from the offers) includes a segment on-road (e.g., bus/car) and real-time data on traffic congestion are available, also these data can be taken into account.

- **Reliable**: The Reliable category concerns the likelihood of delays, traffic congestion, breakdowns or last-minute changes that could affect the travel time and comfort of the trip. Some solutions are inherently variable (e.g. traffic delays when crossing a city at rush hour), while other solutions might offer a small window to change the mode of transport that could cause massive idle times. For this reason, also the frequency of the service for involved solutions should be taken into account. Lastly, the influence of the weather on the trip is considered.

- **Cheap**: The Cheap category captures the total price of a trip, the possibility of sharing part of it with others and the ease of payment, giving additional value to solutions that offer an integrated fare system and do not require the user to purchase different tickets from different platforms.

- **Comfortable**: The Comfortable category concerns objective factors such as the number of interchanges required or the possibility of having a comfortable seat. It also covers a set of other elements about the quality of the trip that has to be evaluated through users' feedback. Relevant factors are the cleanliness of the stations and vehicles used and the feeling of personal safety.

- **Door-to-door**: The Door-to-door category covers the distance of the user's start and endpoint (as defined in the mobility request) from the beginning and ending locations of the offers provided. It is measured by the amount of walking distance the user has to cover.

- **Environmentally friendly**: The Environmentally Friendly category is related to the green aspects of the trip, taking into account at least the amount of CO_2 emissions measured per kilometre/traveller for each mean of transport included in the offer. If available, additional determinant factors can be considered as the energy consumption, the NO_x emissions (nitrogen oxides) and the carbon footprint.

- **Short**: The Short category focuses on minimizing the distance covered. It also takes the number of stops needed to complete the trip.

- **Multitasking**: The Multitasking category measures the extent to which the user can perform other tasks while travelling. These activities can regard productivity (personal or work), fitness, or enjoyment. This category considers the amount of space available, the presence of silence or business area, as well as whether the internet connection and/or plugs are provided. Lastly, since the level of privacy might also influence the extent to which a person can work, it will also be considered as a determinant factor for this category.
- **Social**: The Social category concerns the maximization of the number of people the user will share the trip with.
- **Panoramic**: The Panoramic category promotes solutions passing through beautiful landscapes (like a particular village or a forest) or historical sites. This category also takes into account the usual sightseeing itineraries for tourists to promote solutions passing near monuments or other interesting spots.
- **Healthy**: The Healthy category concerns the involvement of walking and/or cycling in an offer.

All this information can have different origins, being the files containing the set of offers we need to categorize our main source. This data will be presented to us in an XML file format called **TRIAS**, and includes information such as the starting and ending times and locations for each one of the alternatives, modes of transport of each leg, price, etc. To test the implementation of the system, we were provided with a set of TRIAS examples. On the other hand, we also need to consider data from Transport Service Providers (TSP), which, unfortunately, is not present in the given TRIAS examples. The different TSP should provide information such as the likelihood of delay of each transport mode or the frequency of the service. Since we do not know whether we will have access to this information, the TSP-related factors will be considered as optional for the computation of the different categories. Finally, some of the features mentioned before need to be computed from external services. For example, for the weather categorization the OpenWeatherMap will be used, or OpenStreetMap for the panoramic category.

As already explained before, the main goal of the Workpackage 3 (current state of project) is the technical development of a system able to categorize the different offers given a mobility request.

2.6 System

In this section we describe the architecture and functioning of the system responsible for the aforementioned categorization, which we call the Offer Categorizer. After a general overview of this system, I will also point out the specific parts that were assigned to Eurecat, and more specifically, the ones that I developed.

As it can be seen in Fig. 2.1, the starting point or input of the Offer Categorizer are the set of **TRIAS Offers**. One of the goals of this project is to include additional data about travel offers that TSPs may provide to help a user choose among different options. However, the description of offers provided by TSPs is often limited and this can affect the number of determinant factors that can be computed for an offer. For this reason, we introduced in the architecture a component called **Offer Context Enricher** that may be configured to contact additional data sources and/or TSP services to enrich the offers with the information required to compute a higher number

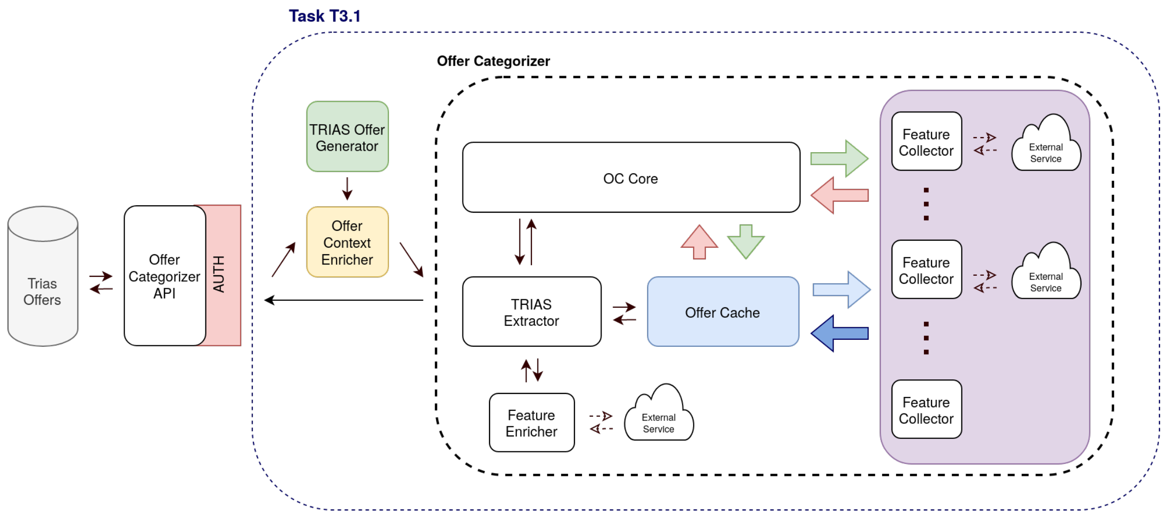


FIGURE 2.1: Architecture of the Offer Categorizer

of determinant factors. In the development phase, due to the lack of real data, we developed the **TRIAS Offer Generator** mimicking this functionality and generating synthetic data in the TRIAS format. In production, this component is not needed, and additional data would be either provided directly by the TSP in the offer, or gathered by the Offer Context Enricher.

After the enrichment of the data, requests are forwarded to the **Offer Categorizer Core** or OC core. This is the central component in charge of the management of the whole workflow of the Offer Categorizer. In particular, its responsibilities include setting up the connections between the other components or combining their input and output. When the system receives the data, the OC core will firstly call the **TRIAS extractor**. This is a module responsible for parsing the offers, converting them to an offer cache schema and finally storing all the required information into a cache. In this case, we use a Docker Redis instance and we name it the **Offer Cache**. Once this procedure is completed, the OC core will call different important modules named **Feature Collectors**. Each feature collector has the capacity to extract the data they need from the cache and with that compute a set of related offer determinant factors. For example, all the features related to time (e.g. duration of trip, waiting time) are assigned to the same module (Time-FC). We designed feature collectors as independent microservices to enable parallel computations of the processing required. Indeed, feature collectors may require retrieving information from external services (e.g., weather forecast), therefore, it is important to optimise the processing. In Table A.1, it can be observed how the determinant factors have been split into the different feature collectors. Once a feature collector has completed its corresponding computations, it will write the results back to the cache so that the OC Core can access them. Finally, with the results for each one of the determinant factors, the OC core will be able to aggregate them using the following weighting approach to compute the final score of an offer with respect each one of the categories. Let us assume the following notation:

- D - number of determinant factors with known values associated with an offer category.
- s_i - the value of the determinant factor $i = 1, \dots, D$.

- r_i - rank assigned to the determinant factor $i = 1, \ldots, D$ considering its importance within the set of determinant factors associated with the offer category.
- w_i - weight determining the importance of the determinant factor $i = 1, \ldots, D$ within the set of determinant factors associated with the offer category.

The value of the offer category, C, is calculated as:

$$C = \sum_{i=1}^{D} s_i w_i \tag{2.1}$$

The values of the weights w_i are determined based on the rank values r_i. In this case, we use the Rank Order Distribution (ROD) weights (Roberts and Goodwin, 2002), which are a sensible default for weights distribution. Essentially, the underlying assumption is that these weights come from a gaussian/normal distribution.

Finally, when splitting the work among the partners, Eurecat took the implementation of the OC core, a variant of the TRIAS Extractor to parse data from a source different than TRIAS (see Section 4.2) and three feature collectors. In this work, I will give a detailed explanation of modules that I implemented: two of the three feature collectors, the Weather-FC and the Panoramic-FC, and the offer extractor (detailed explanation in Section 5.1).

Chapter 3

Recommender Systems

Over the last years, we have been witnesses of the rise of important web services such as Netflix, Amazon or Spotify. With them, the so-called recommender systems have taken more and more importance in our day-to-day lives. From streaming services (suggesting to users movies or shows) to online shopping (suggesting to users new products, or similar items to what they have already bought), this piece of technology is arguably a vital element in our daily online journeys.

Recommender systems are a set of tools and algorithms which try to provide suggestions of items (e.g. movies, music, clothing or news) which will potentially be interesting for the users. To achieve this purpose, there are two main categories of methods: collaborative based and content based algorithms.

3.1 Collaborative Filtering

These kinds of methods collect and analyze large amount of information on users' behaviors, activities and preferences regarding their interaction with items. Using all this knowledge they try to predict what users will like based on their similarity to other users. Therefore, the hypothesis behind the collaborative methods is that these past interactions are sufficient to detect similar users and/or items and make predictions based on the estimated similarities. Early techniques belonging to this class were mainly based on neighborhood approaches (Resnick et al., 1994) while recently model-based techniques such as Matrix Factorization (Koren, Bell, and Volinsky, 2009) attracted more attention due to their superior performance and scalability.

The main advantage of collaborative approaches is that there is no need for information about the users or items. Moreover, the more users interact with the system, the more accurate will be the new recommendations. However, since they only consider past interactions, collaborative filtering heavily suffer from the so-called *cold start problem*. Essentially, it is a very challenging task to recommend anything to a new user or to recommend a new item to users.

3.1.1 Matrix Factorization

Matrix factorization techniques generally learn a low-dimensional representation of users and items by mapping them into a joint latent space consisting of *latent factors*. Recommendations are then generated based on the similarity of user and item factors.

Suppose a set of users, U, and a set of items, I. With matrix factorization the target matrix Y (the matrix of ratings, with users as rows and items as columns) is approximated by the matrix product of two low-rank matrices $W : |U| \times k$ and

$H : |I| \times k$:

$$\hat{Y} := WH^T \tag{3.1}$$

where k is the dimensionality/rank of the approximation (hyperparameter to be chosen). Each row w_u in W can be seen as a feature vector describing a user u and similarly, each row h_i of H describes an item i. Thus, the prediction formula can also be written as:

$$\hat{y}_{ui} = \langle w_u, h_i \rangle = \sum_{f=1}^{k} w_{uf} h_{if} \tag{3.2}$$

Many optimization approaches (e.g gradient descent methods) require the gradient of the model. In this case:

$$\frac{\partial}{\partial \theta} \hat{y}_{ui} = \begin{cases} h_{if}, & \text{if } \theta \text{ is } w_{uf} \\ w_{uf}, & \text{if } \theta \text{ is } h_{if} \end{cases} \tag{3.3}$$

3.2 Content-Based

Opposite to collaborative methods, which only rely on user-item interactions, content-based approaches use additional information about users and/or items. Put it in simple words, these methods assume that a user will be interested in items that share features (e.g. brand of a clothing piece, cast or genre of a movie, etc.) with items they liked in the past. Information about the users might also be used (e.g. demographics).

These methods are far less affected by the cold start problem than collaborative approaches: new users or items can be described by their features (content) and therefore new suggestions can be done for these new entities. Only new users or items with previously unseen features will suffer from this drawback, but once the system is old enough, this has few to no chance to happen.

3.3 Factorization Machines

Most recommendation approaches assume a rating dataset formed by a collection of user-item-rating tuples. This is the starting point for most variations of the previously explained collaborative filtering algorithms. However, in many situations, we have plenty of item metadata (tags, categories, genres) that can be used to make better predictions. This gives birth to the hybrid methods, which combine collaborative and content based approaches. Such examples are the Factorization Machines (Rendle, 2010), which are general factorization models that not only learn user and item latent factors, but also the relation between users and items with any auxiliary features. This is done by also factorizing these features to the same joint latent space. In contrast to the conventional factorization techniques where the training data is represented by a matrix, the input data for factorization machines are feature vectors. This creates a great flexibility by allowing the algorithm to incorporate any additional information in terms of auxiliary features. In more detail, this model encodes each user-item interaction with a feature vector x with its corresponding output y. Usually, they are really sparse binary vectors with non-zero values corresponding to the user and item (besides the auxiliary features). When dealing with explicit feedback, the output y would be the rating given by the user. In Table 3.1 it can be observed how the input data can be modelled to generate these feature vectors.

	User				Items				Features				Output	
x^1	1	0	0	...	1	0	0	...	1	0	1	...	3	y^1
x^2	0	1	0	...	0	1	0	...	0.5	0.3	-0.1	...	4	y^2
x^3	0	0	1	...	0	1	0	...	0.5	0.3	-0.1	...	1	y^3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x^n	0	0	1	...	0	0	1	...	0	0	1		5	y^n

TABLE 3.1: Example of sparse vector representations. Every row represents a feature vector x^i with its corresponding output y^i (in this example it is assumed a 1 to 5 scale). The first 4 columns represent indicator variables for the active user; the next 4 indicator variables for the active item. The next columns contain the auxiliary features and the right most column is the output.

Let us assume that the input data is represented by tuples $(\mathbf{x}, y)$ where $\mathbf{x} = (x_1, ..., x_N) \in \mathbb{R}^N$ is a N-dimensional feature vector and y is its corresponding output value. Then, the model equation for a factorization machine of degree $d = 2$ is defined as:

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{N} w_i x_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} \langle \mathbf{v}_i \mathbf{v}_j \rangle x_i x_j \tag{3.4}$$

where the model parameters that have to be estimated are:

$$w_0 \in \mathbb{R}, \;\; \mathbf{w} \in \mathbb{R}^N, \;\; \mathbf{V} \in \mathbb{R}^{N \times k} \tag{3.5}$$

And $\langle \cdot, \cdot \rangle$ is the inner product of two vectors of size k:

$$\langle \mathbf{v}_i \mathbf{v}_j \rangle := \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f} \tag{3.6}$$

A row $\mathbf{v}_i$ within $\mathbf{V}$ describes the i-th variable with $k \in \mathbb{N}_0^+$ factors, which is a hyperparameter that defines the dimensionality of the factorization.

This factorization machine model of $d = 2$ captures all single and pairwise interactions between variables:

- w_0 is the global bias
- w_i models the strength of the i-th variable
- $\langle \mathbf{v}_i \mathbf{v}_j \rangle$ models the interactions between the i-th and j-th variable.

There are actually different ways of learning this model. One of the most common approaches is by using gradient descent methods (e.g. stochastic gradient descent) for a variety of losses, among them are the square or hinge loss. The gradient of the model is:

$$\frac{\partial}{\partial \theta} \hat{y}(\mathbf{x}) = \begin{cases} 1, & \text{if } \theta \text{ is } w_0 \\ x_i, & \text{if } \theta \text{ is } w_i \\ x_i \sum_{j=1}^{N} v_{j,f} x_j - v_{i,f} x_i^2, & \text{if } \theta \text{ is } v_{i,f} \end{cases} \tag{3.7}$$

3.4 Learning to Rank

Most recommender systems approaches try to predict a rating for each user-item pair by optimizing an accuracy (e.g Root Mean Squared Error). Since the ultimate goal of several applications is to provide a personalized ranking (without minding the actual predicted rating), these ratings are used to rank the different items. However, there have been studies (McNee, Riedl, and Konstan, 2006) discussing that recommendations that are most accurate according to the standard metrics are sometimes not the recommendations that are most useful to users. This is why there are other approaches that, instead of ratings, they try to directly learn the ranking. They are the so-called learning to rank algorithms.

The three main approaches to learning to rank are known as pointwise, pairwise and listwise. In a very general way:

- **Pointwise approaches** look at a single item at a time in the loss function. They essentially take a single item and train a machine learning model on it to predict how relevant (e.g. position in the ranking) it is for the current query/user.
- **Pairwise approaches** look at a pair of samples at a time in the loss function. Given a pair of samples, they try and come up with the optimal ordering for that pair and compare it to the ground truth. Put it in simple words, these approaches do not care much about the exact score of each item. Instead, they focus on the relative ordering among all items.
- **Listwise approaches** directly look at the entire list of items and try to come up with the optimal ordering for it.

3.5 Bayesian Personalized Ranking

Bayesian Personalized Ranking (BPR) (Rendle et al., 2014) is a pairwise recommendation algorithm designed to be used with implicit feedback (e.g. clicks, purchases). As we will see, the fact that we deal with implicit data is a central aspect of the algorithm. The method is based on a generic optimization criterion called BPR-OPT for personalized ranking that is the maximum posterior estimator derived from a Bayesian analysis of the problem. We will also see a generic learning algorithm for optimizing models with respect to BPR-OPT.

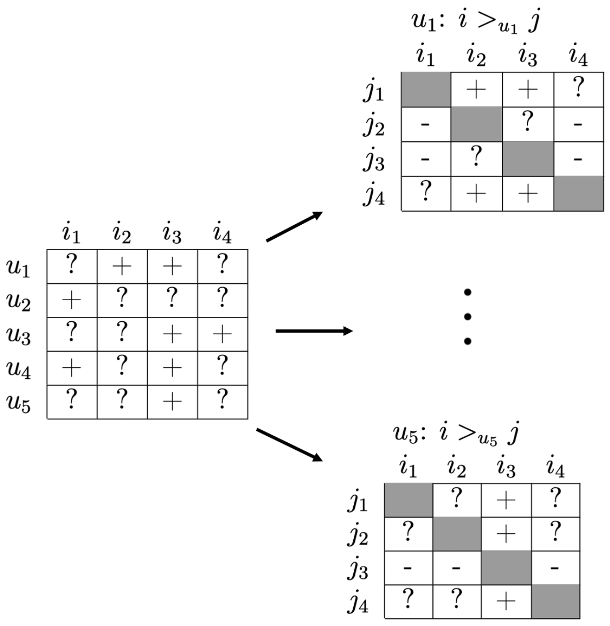
Starting with the formalization of the problem, let U be the set of all users and I the set of all items. In this scenario implicit feedback $S \subseteq U \times I$ is available, S meaning the positive observations (e.g. items purchased). The task of the recommender system is to provide with a personalized ranking $>_u \in I^2$ of all items, where $>_u$ refers to the specific ranking of a user. For convenience, it is also important to define:

$$I_u^+ := \{i \in I : (u,i) \in S\} \tag{3.8}$$

$$U_i^+ := \{u \in U : (u,i) \in S\} \tag{3.9}$$

In simple words, I_u^+ is the set of positive items of user u, and U_i^+ is the set of users that have a positive outcome on item i.

As we have already mentioned, we suppose a situation in which we deal with implicit feedback. In this scenario, only the positive classes are observed, while the remaining data is a mixture of actually negative and missing values. There are different approaches to cope with this challenge. The idea proposed by the authors of this algorithm is to use item pairs as training data and optimize for correctly ranking items pairs. From S, the goal is to reconstruct for each user parts of $>_u$. If an item i has been viewed by user u - i.e $(u,i) \in S$ - then we assume that the user prefers this item over all non-observed items. For example, in Figure 3.1 user u_1 outputted



	i_1	i_2	i_3	i_4
u_1	?	+	+	?
u_2	+	?	?	?
u_3	?	?	+	+
u_4	+	?	+	?
u_5	?	?	+	?

u_1: $i >_{u_1} j$

	i_1	i_2	i_3	i_4
j_1	▨	+	+	?
j_2	-	▨	?	-
j_3	-	?	▨	-
j_4	?	+	+	▨

u_5: $i >_{u_5} j$

	i_1	i_2	i_3	i_4
j_1	▨	?	+	?
j_2	?	▨	+	?
j_3	-	-	▨	-
j_4	?	?	+	▨

FIGURE 3.1: On the left side, the observed data is shown. The approach is to create user specific pairwise preferences $i >_u j$ between a pair of items. On the right side, plus (+) indicates that a user prefers item i over item j; minus (-) indicates that they prefer j over i

the positive class on item i_2 but not item i_1, so we assume that this user prefers item i_2 over i_1: $i_2 >_{u_1} i_1$. For items that have both the positive or negative/unknown class, we cannot infer any preference. To formalize this, it is convenient to create the training data $D_S : U \times I \times I$ by:

$$D_S := \{(u, i, j) | i \in I_u^+ \cap j \in I \backslash I_u^+\}, \tag{3.10}$$

where $(u, i, j) \in D_S$ means that user u prefers item i over item j.

Once this formulation has been set, it is time to explain this general optimization criterion for personalized ranking called BPR-OPT. The Bayesian approach of finding the correct personalized ranking for all items is to maximize the following posterior probability:

$$p(\Theta| >_u) \propto p(>_u |\Theta) p(\Theta), \tag{3.11}$$

where Θ is the parameter vector of the model. Assuming that all users act independently and that the ordering of each (i, j) pair for a specific user is independent of the ordering of every other pair, the above user-specific likelihood can first be rewritten as a product of single densities and second be combined for all users:

$$\prod_{u \in U} p(>_u |\Theta) = \prod_{(u,i,j) \in D_S} p(i >_u j|\Theta) \tag{3.12}$$

Now, we define the specific individual probability that a user prefer i over j:

$$p(i >_u j|\Theta) := \sigma(\hat{f}_{uij}(\Theta)) \tag{3.13}$$

where σ is the sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.14}$$

Here $\hat{f}_{uij}$ is an arbitrary function of the model parameter vector Θ which is supposed to capture the relationship between user u, item i and item j. In other words, the task of modelling this relationship will depend on an underlying model class (e.g. Matrix Factorization).

In order to complete the Bayesian model we need to introduce a prior density $p(\Theta)$. We will use a normal distribution with zero mean and variance-covariance matrix Σ_Θ, i.e. $p(\Theta) \sim \mathcal{N}(0, \Sigma_\Theta)$. Also, in order to reduce the number of unknown hyperparameters it is convenient to set $\Sigma_\Theta = \lambda_\Theta I$, where I is the Identity matrix. To sum it all up, the full form of the BPR-OPT is:

$$\begin{aligned}
\text{BPR-OPT} &:= \ln p(\Theta| >_u) \\
&= \ln p(>_u |\Theta) p(\Theta) \\
&= \ln \prod_{(u,i,j)\in D_S} \sigma(\hat{f}_{uij}) p(\Theta) \\
&= \sum_{(u,i,j)\in D_S} \ln \sigma(\hat{f}_{uij}) + \ln p(\Theta) \\
&= \sum_{(u,i,j)\in D_S} \ln \sigma(\hat{f}_{uij}) - \lambda_\Theta ||\Theta||^2
\end{aligned} \tag{3.15}$$

where:

- Since the natural logarithm is a monotonic transformation, we can apply it to the posterior distribution of Eq. 3.11 to make computations easier without affecting the optimization process.

- Regarding $p(\Theta)$, we have already mentioned that we use a normal distribution with zero mean ($\mu = 0$) and unit variance ($\Sigma = I$, let us ignore λ_Θ for now). The expression for a multivariate normal distribution of k-dimension random vector $\Theta = (\theta_1, \ldots, \theta_k)$ is:

$$\begin{aligned}
\mathcal{N}(\Theta|\mu, \Sigma) &= \frac{1}{(2\pi)^{k/2}\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(\Theta-\mu)^T\Sigma^{-1}(\Theta-\mu)\right) \\
&= \frac{1}{(2\pi)^{k/2}} \exp\left(-\frac{1}{2}\Theta^T\Theta\right)
\end{aligned} \tag{3.16}$$

  In the expression above, the only thing that depends on Θ is $\exp\left(-\frac{1}{2}\Theta^T\Theta\right)$, the rest is just a multiplicative constant that we don't need to worry about. Hence if we take the natural logarithm of that formula, the exponential goes away and our $p(\Theta)$ can be written as $-||\Theta||^2$. Lastly, we simply multiply the λ_Θ back, which can be seen as the model specific regularization parameter.

With this, the derivation of the optimization criterion for the personalized ranking is concluded. Now, we shall adress the learning method to optimize with respect the BPR-OPT. Since this one is differentiable, gradient descent algorithms are a suitable option for maximization. The authors propose what they call LEARNBPR, a stochastic gradient-descent algorithm based on bootstrap sampling of training triplets. A summary of the learning method can be seen below:

```
1: procedure LEARNBPR
2:     initialize $\Theta$
3:     repeat
4:         sample $(u, i, j)$ from $D_S$
5:         compute $\hat{f}_{uij}$
6:         update parameters according to Eq. 3.18
7:     until convergence
8:     return $\Theta$
9: end procedure
```

Since we will use a gradient descent method, we need to compute the gradient of the BPR-OPT with respect to the model parameters:

$$\begin{aligned}\frac{\partial \text{BPR-OPT}}{\partial \Theta} &= \sum_{(u,i,j)\in D_S} \frac{\partial}{\partial \Theta} \ln \sigma(\hat{f}_{uij}) - \lambda_\Theta \frac{\partial}{\partial \Theta} ||\Theta||^2 \\ &\propto \sum_{(u,i,j)\in D_S} \frac{-e^{-\hat{f}_{uij}}}{1+e^{-\hat{f}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{f}_{uij} - \lambda_\Theta \Theta\end{aligned} \tag{3.17}$$

Using stochastic gradient descent to update the parameters, for each $(u, i, j) \in D_S$ an update is performed according to:

$$\Theta \leftarrow \Theta + \alpha \left(\frac{e^{-\hat{f}_{uij}}}{1+e^{-\hat{f}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{f}_{uij} + \lambda_\Theta \Theta \right) \tag{3.18}$$

The authors have also shown that following an strategy that traverses the data item-wise or user-wise will lead to poor convergence since there are too many consecutive updates on the same user-item pair – i.e. for one user-item pair (u, i) there are too many j such that $(u, i, j) \in D_S$. Instead, they show that using a bootstrap sampling approach with replacement leads to a faster convergence. With this approach the chances of picking the same user-item combination in consecutive update steps is small.

Finally, in order to learn models with BPR we need the define the function $\hat{f}_{uij}$:

$$\hat{f}_{uij} := \hat{f}_{ui} - \hat{f}_{uj} \tag{3.19}$$

where the individual predictions $\hat{f}_{ui}$ are real numbers given by the underlying model class. To compute them we could apply standard models such as Matrix Factorization or Factorization Machines.

Regarding the evaluation, we need to create a test set. For each user, we randomly select a small percentage (e.g. 10%) of its actions (user-items pairs) from their history, i.e. we remove some entries from I_u^+ per user u. This results in a disjoint train set S_{train} and test set S_{test}. The model is then learned on the training and the evaluation is performed on the test set.

Chapter 4

Methodology

4.1 Software Technologies

Building a system such as the Offer Categorizer is challenging in many ways, both in terms of organization and implementation. Selecting the correct tools for the development of the different components is vital for a proper and fluid workflow. In this section, we list the most important software technologies we use to build the system.

4.1.1 Github

Github (*Github*) is an online hosting platform for software development and version control using the source-code management tool git. It provides several collaboration features such as issue tracking, feature requests, task management, continuous integration and wikis, which are especially useful in projects like this with a considerable number of partners.

In our case, we created an Organization containing several repositories. More specifically, there is a central repository containing the code for the OC Core, single repository for each feature collector and some other ones such as the **trias-extractor**, which extracts the relevant data from the TRIAS files, or the **r2r-offer-utils**, which contains several utilities shared across different components.

4.1.2 Docker

Docker (*Docker*) is an open platform for developing, shipping, and running applications by means of the so-called containers, which are loosely isolated environments. Putting it in simple words, these containers can be understood as a lightweight equivalent of a virtual machine, allowing the users to package an application with all its dependencies.

The most important Docker objects to understand how it works are:

- **Images**: images are the template for building the containers. Often, an image is based on another image, with some additional customization. For example, in our case, the images are based on the Python image, as well as the configuration details needed to make the applications run.

- **Containers**: containers are the organizational units of Docker. A container is a runnable instance of an image, which essentially means that when an image is built, it runs on a container. The container analogy is used because of the portability of the software stored in them.

In the project, we dockerized all the different components - feature collectors, extractors etc. - and we set up a network so that each components can communicate between them.

4.1.3 Flask

Flask (*Flask*) is a micro web framework written in Python. It provides you with tools, libraries and technologies to build web applications. It is a framework with minimal or no dependencies on external libraries.

In our implementation, we use Flask to send the data from the Offer Categorizer Core to the different feature collectors. More specifically, the OC core sends the identification associated with the request of user, *request_id*, and with that information, each feature collector can read all the data they need from a cache (see next section). So, simply put, Flask allows us to create an endpoint for the OC core and the feature collectors to communicate.

4.1.4 Redis

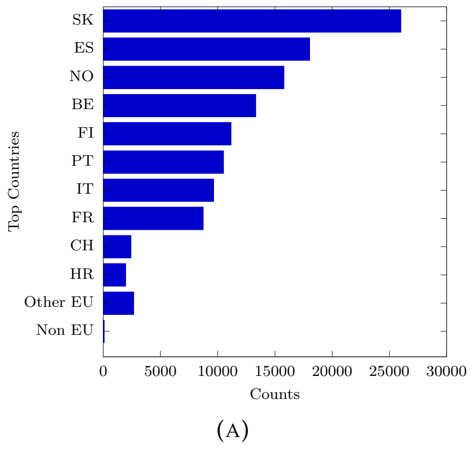
Redis (*Redis*), which stands for Remote Dictionary Server, is an open source, inmemory key-value data structure store, typically used as a database, cache, and message broker. It provides a wide variety of data structures such as strings, hashes, lists or sets. In this project, we use the Redis Docker image to run an instance and use it as a cache to decrease data access latency, increase throughput, and ease the load off our application. In this way, we are able to efficiently store the relevant information from the different sources and get specific data using predefined keys.

4.2 Datasets

In this section, I will describe the different datasets used to test the implementation of the different components.
As already mentioned in Chapter 2, the set of offers to categorize will be send to the Offer Categorizer in an XML format called TRIAS. The main problem we faced during the development of the different components is the lack of data, which would not allow us to properly test the system. More precisely, we were provided with just a few tens of samples. On the other hand, we do not have the actual user choice for this set of TRIAS offers and thus we would not be able to build a recommender system with this data. Hence, we decided to use other trips datasets which we describe in the following section.

4.2.1 MoTiV & routeRANK

Mobility and Time Vale (*MoTiV*) is an European project which addresses the emerging perspectives on changing the value of travel time. Accordingly, it explores the dynamics of individual preferences, behaviors and lifestyles that influence travel and mobility choices. In this project, a set of real trips was collected, through a dedicated app (Consonni et al., 2021), from a set of users over a period of several months in 2019 around mainly European countries (see Fig 4.1). This is an open dataset available at https://zenodo.org/record/4027465.YL0eZS0lNUM. For the purposes of our work, what we need to know is that there are around 67000 trips registered and that, for all of them, we know the starting and ending coordinates, the departure and arrival times and the mode of transport of each leg.



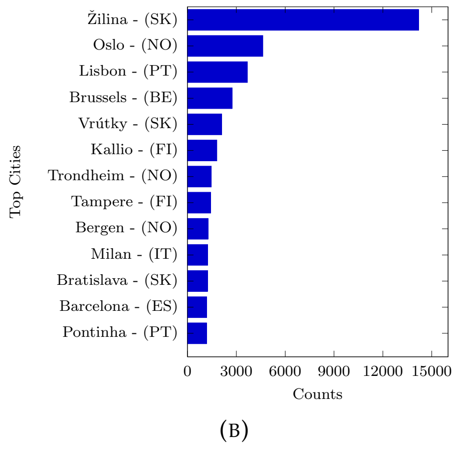
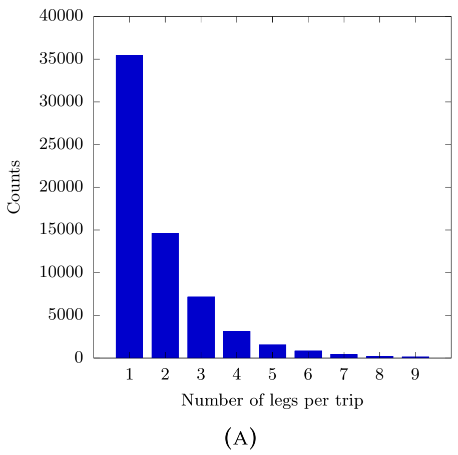


FIGURE 4.1: Distribution of locations of trips registered in the MoTiV dataset

As it can be observed in Figure 4.2a, most of the trips are composed by a small number of legs, being just one leg the most common trip. The counts show a decreasing behaviour as the number of legs per trip increases. For the sake of the visualization, in this plot we only show the trips formed up to nine legs, but in the dataset there is a residual number of trips (around 0.4%) which have a larger number (up to 60 legs). On the other hand, the trips were registered by a total number of 3330 different users. In Figure 4.2b, you can see the distribution of trips registered by each user. The behaviour is really similar to the previous one: users mainly registered between one and five trips, and the number of counts decreases as we increase the number of trips. Again, this plot only shows up to seventy trips registered, but around 5% of the users registered more trips than this number (up to 945 trips). Regarding the modes of transport used by the users, in Fig. 4.3 it can be seen the distribution of the ones that were most used, being specially relevant the trips performed on foot. Other modes of transport not shown in this plot include taxi, ferry or plane.



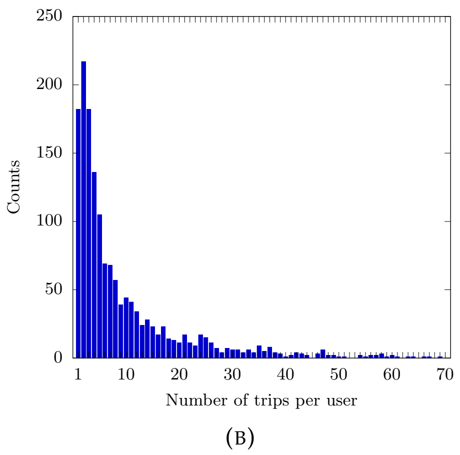


FIGURE 4.2: (A) Distribution of number of legs per trip on the MoTiV dataset. (B) Distribution of trips performed by each user on the MoTiV dataset

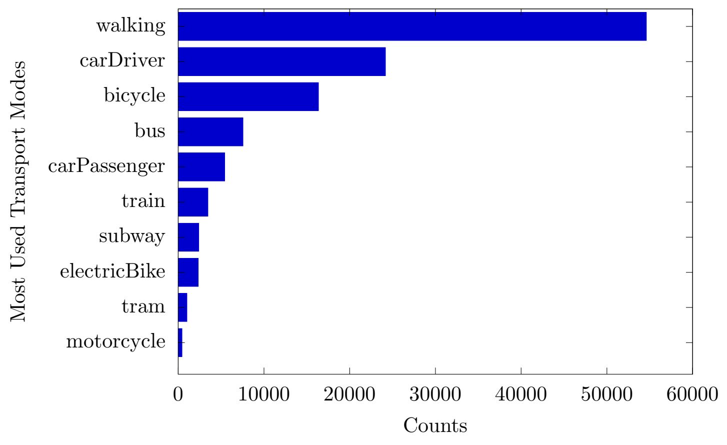


FIGURE 4.3: Distribution of the modes of transport used on the trips registered in the MoTiV dataset

The MoTiV dataset only contains real trips, without alternatives. Just as in Ride2Rail, the presence of alternatives was also necessary for the development of the MoTiV project. To generate them, around 23000 of the 67000 trips from MoTiV were sent to a search engine for trips called routeRANK (*routeRANK*). More precisely, what was send to the offer generator was the starting time together with the starting and ending coordinates for each one of the legs. This means that search engine did not generate the alternatives for the trips, instead it generated alternatives for the legs. It was done in such a way because many trips on the MoTiV dataset have very close departure and destination, and the researchers working on this project considered that this was a better option for their objectives. However, we understand that the user choices on the offers are made on the whole trip, not on the single legs. Hence, we consider it to be more interesting to have alternatives for the whole trips. In order to generate them, we decided to combine (accordingly to the original leg ordering) the alternatives of all the legs. For example, let us suppose that a given trip is composed by two legs. The first leg presents three alternatives (*leg_1.1*, *leg_1.2*, *leg_1.3*) and the second one presents two alternatives (*leg_2.1*, *leg_2.2*). Then, the combination would be:

$$\begin{array}{c} leg\_1.1 \\ leg\_1.2 \\ leg\_1.3 \end{array} \times \begin{array}{c} leg\_2.1 \\ leg\_2.2 \end{array} \longrightarrow \begin{array}{c} leg\_1.1 - leg\_2.1 \\ leg\_1.1 - leg\_2.2 \\ leg\_1.2 - leg\_2.1 \\ leg\_1.2 - leg\_2.2 \\ leg\_1.3 - leg\_2.1 \\ leg\_1.3 - leg\_2.2 \end{array}$$

As it can be seen in Fig 4.4, following this strategy we might generate really large number of alternatives for the trips. In this plot, for the sake of the visualization, we only show the counts up to 27 number of alternatives per trip. However, around a 10% of the total number of trips present even more alternatives.

Regarding the transport modes of the alternatives offered by the routeRANK generator, the most common solutions include walking, followed by bus, car, bike, train and taxi.

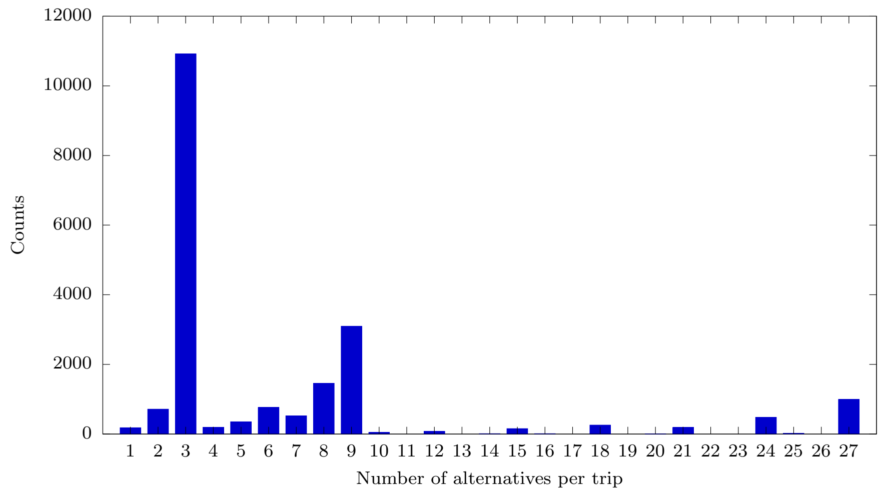


FIGURE 4.4: Distribution of offers/alternatives per trip requests in the routeRANK dataset

Finally, in Appendix B, you can see the output schema of the JSON files which are generated by the routeRANK search engine.

Chapter 5

Implementation

In previous chapters we have described the main concepts to understand the different modules we need to build for the Offer Categorizer, we have introduced the theoretical framework for recommender systems together with some model examples, and we have described the datasets we use to test all the aforementioned implementations. Now, we are ready to explain our approach to these implementations.

5.1 Components

In this section, I will cover the different components I implemented for the development of the Offer Categorizer. As previously mentioned, these modules are the data extractor, the Weather-FC and the Panoramic-FC.

5.1.1 Offer Cache & routeRANK Extractor

The routeRANK Extractor is a module responsible for parsing offers from the routeRANK dataset and for converting them to the offer cache schema. The reason this additional extractor is needed (besides from the original TRIAS extractor) is because the routeRANK dataset is presented in a JSON format, different from the XML files of the TRIAS offers. Also, there are some inconsistencies between the TRIAS and the routeRANK offers, e.g. the name of the transportation modes. Therefore, some additional work to solve these discrepancies between the two datasets is needed.

The Offer Cache component represents a shared data layer accessed by the different modules composing the Offer Categorizer. As discussed in Section 4.1.4, the Redis key-value store was chosen to implement it. To facilitate the integration and development of the different components, an offer cache schema was defined specifying the set of keys and value datatype used. We divide its internal structure into different levels depending on whether the data is related to the request, the offers or the legs. Each one of the levels can be accessed using a predefined key. For example, to get the data from a specific leg of an offer, the key to be used is:

<request_id>:<offer_id>:<leg_id>:<data_key>

where the different identifications come from the data sources, and the *<data_key>* varies depending on the data we want to get, e.g. *transportation_mode* to get the mode of transport. The offer cache schema specifies:

- How the offers should be represented to enable the categorization process, i.e., the data format that should be adopted by an offer parser to model the offers received in a specific input data format.
- How the offer features computed by each feature collector component are stored.

- How the final categorization of the offers computed by the OC core is saved.

In Appendix C, you can find a complete description of keys and values to be used for each one of the determinant factors.

5.1.2 Weather-FC

The Weather Feature Collector is a module of the Offer Categorizer responsible for the computation of the weather forecast for the day of the trip. This feature is then used in the Reliable and Comfortable categories. The work of this module can be split into two minor sub-tasks: first of all, we intend to classify the weather forecast of the specific day of the trip request into different scenarios. After that, we convert this classification into a score to estimate the probability of delay.

To obtain the information needed to perform the necessary computations, an API from OpenWeatherMap (*Weather API*) is used. This website offers several APIs (for both free and paid subscriptions) which allow current and forecast weather data collection. The one we use is called *One Call API* and allows to get minutely forecast for 1h, hourly forecast for 48h and daily forecast for 7 days. After investigating the official documentation to see what kind of information we could get, we decided to define 7 scenarios (see Table 5.1) depending on the values of the clouds, precipitation, wind and temperature. Notice that an offer can fall in more than one scenario.

Weather Scenario	Clouds	Precicpitation	Wind	Temperature
Neutral/Good	No clouds or Clear sky or Few clouds	No rain/snow or Light rain/snow	Light breeze	Comfortable
Cold	Any	Any	Any	Cool
Warm	Any	Any	Any	Warm
Uncomfortable Temperature	Any	Any	Any	Uncomfortably hot or cold
Rainy/Snowy	Any	Moderate or Heavy	Any	Any
Cloudy	Partially or cloudy Completely	Any	Any	Any
Windy	Any	Any	Strong breeze or Gale	Any

TABLE 5.1: Different possible weather scenarios depending on the status of the clouds, precipitation, wind and temperature. In Appendix D you can see the details of how to map the values obtained from OpenWeatherMap to the labels shown in this table.

The input needed for the API is just the time of the trip (relative to the current time) and the location coordinates, which is convenient because this is information that we already have available in the set of offers we get (either TRIAS or routeRANK). In more detail, we get this data for all the legs of each one of the offers. However, it is expected for the weather forecast not to change too much (or at all) from one leg to another. Therefore, to avoid unnecessary computations, we only query the API for different cities and days. More specifically, for each one of the legs, we store the date and the city from the starting moment. Then, we compute the weather labelling for each one of the different combinations of city and date.

Once this procedure is finished, all the legs of each one of the offers are classified with some of the weather scenarios. The next step is to estimate the probability of delay using these labels. Due to the lack of data, we decided to take a simple approach. We consider that Rainy/Snowy, Windy and Uncomfortable Temperature are the scenarios which can potentially cause drawbacks in the normal functioning and schedule of the transport modes. Hence, for each leg, we count how many of these *extreme conditions* are present in the classification, and according to this number we assign a predefined probability of delay. For example, for a leg whose weather categorization contains the three *extreme conditions*, the probability of delay will be higher than for another leg whose categorization only contains one of these conditions. For the predefined values we use again the ROD weights.

Finally, to aggregate the results over all the legs and get the final score of each offer, we just keep the higher probability of delay (worst scenario) among all legs. With that, we have a score for each one of the offers representing the probability of delay due to the weather forecast. As a last step, we normalize these results. The goal with that is to have a relative score with respect to all the offers associated to that trip request rather than an absolute score. We use the z-score:

$$x'_i = \frac{x_i - \mu}{\sigma} \tag{5.1}$$

where μ is the mean values and σ is the standard deviation.

5.1.3 Panoramic-FC

The Panoramic Feature Collector is a module of the Offer Categorizer responsible for the computation of the number of monuments, historical sites and landscapes that the users could observe taking each one of the offers. All these factors are only used in the Panoramic category. The approach is to get the number of these relevant spots for an area of 100 *m* radius around the starting and end ending point of each one of legs. Then, the results are added up over all the legs and finally normalized to obtain a total score for this particular offer category.

To complete these computations we need to use an external service again. In this case, we query the OpenStreetMap (*OSM*), a collaborative project to create a free editable map of the world. The OSM data model is structured in three basic components: nodes, ways and relations. In simple terms, nodes are points on the maps (in latitude and longitude), ways are ordered lists of nodes, which could correspond to a street or the outline of a park, and finally relations which are also ordered lists containing either nodes, ways or even other relations. For the purpose of this module, nodes are enough. On the other hand, OSM represents physical features on the ground (e.g., monuments or buildings) using tags attached to its basic data structures. Each tag describes a geographic attribute of the feature being shown. For example, in our case, to query the number of monuments, we would use the tag *historic* with the value *monument*. In Table 5.2 you can see which specific tags and values are used in this case.

	tag: value
Monuments	historic: monument, toursim: attraction, tourism: artwork
Historical Sites	historic: archaelogical_site, historic: memorial historic: wayside_shrine, historic: wayside_cross, historic: ruins, historic: yes
Landscapes	natural: coastline, natural: water, natural: peak, natural: cliff, natural: viewpoint

TABLE 5.2: Tags and values used to retrieve the number of monuments, historical sites and landscapes.

It is worth mentioning that we encountered a huge drawback when using the OSM. If too many requests are sent from the same IP address, the server blocks some requests to avoid that a user uses up all resources. Even in the testing phase we were getting this problem, so this approach would not be enough to meet the requirements of the system in production. This is why Eurecat's *Unitat de Desenvolupament de Software* (UDS) set up an internal mirror of the server (using Docker containers) containing the data of the most common Europen countries present in the routeRANK dataset (see Fig 4.1). In this way, instead of querying the OSM, we query an internal server and we avoid the previous problem.

5.2 Simplified Offer Categorizer

One of the main objectives of the categorization is to obtain a powerful representation of the offers. With it, we should be able to implement a ranking algorithm to learn the user's preferences while choosing the alternatives of a given trip. However, the Offer Categorizer was not ready and did not meet the deadline of this work. This is why we decided to build a simplified version of the system.

For each one of the categories, we took at least one determinant factor (see Table 5.3). We tried to pick determinant factors whose computation could be done directly from the routeRANK dataset (so that we do not have to rely on any external service). For some of the categories this was not possible, since most of the determinant factors come from the TSP's. Such cases are the seating quality (from the comfortable category) and the privacy level (from the multitasking and comfortable categories). For them, we used predefined values (in a scale from 1 to 5) depending on the transportation mode. For example, for the privacy level, this value is larger for car than for bus. On the other hand, we could have used the Panoramic-FC for the number of monuments. However, passing all the offers through this feature collector would have taken an unfeasible amount of time. This is why we decided to generate these numbers randomly.

Category	Determinant Factor	Category	Determinant Factor
Quick	Trip duration	Environmentally friendly	CO2 emissions/km
Reliable	Weather forecast	Short	Distance covered
Cheap	Total price		Number of stops
	Seating quality	Multitasking	Privacy level
Comfortable	Privacy level	Social	People sharing the trip
	Weather forecast	Panoramic	Number of monuments
Door-to-Door	Walking distance	Healthy	Fraction of trip by bike/walk
			Length of fraction by bike/walk

TABLE 5.3: Determinant factors used for the simplified version of the Offer Categorizer.

5.3 Recommender System

Using the simplified Offer Categorizer, we can compute the categorization of the offers from the routeRANK dataset. The goal is to use these scores in a ranking algorithm to learn the user's preferences and be able to build a recommender system. Notice that, as explained in Section 4.2, the way the data was obtained is not ideal for our purposes. In an ideal situation, the user should have been presented beforehand with the set offers and then choose one among them. Our case is different; a set of users registered real trips (MoTiV dataset). Then, this data was sent to a search engine for trips, generating in this way the routeRANK dataset. Therefore, the user did not explicitly choose one among many offers, which can be problematic for the correct behavior of the learning algorithm. It also adds some challenges in identifying the real trips among the offers. Doing a matching based only on the transportation mode of each leg, in around 30% of trips we do not find the actual trip among the alternatives.

This can be treated as an implicit feedback recommender system problem; the offer corresponding to the real trip is the positive class, while the other ones are the negative/unknown class. The BPR method explained in Section 3.5 is a suitable approach to treat this problem. Besides, as the underlying class to make the predictions for each user-offer pair, we will use two different approaches:

- Matrix Factorization: As explained before, this model tries to learn the relation between users and items and make predictions based on similarities between them. In our problem, we have a clear issue: all the items, i.e. the offers, are in principle unique and therefore it is not possible to learn these user-item relations. In order to solve this problem, we propose to identify two offers as equal if the different transport modes cover the same fraction of the trip. To simplify the problem, we only consider five different transportation modes: walk, bike, car, public transport and long public transport. Hence, a trip composed by *car-walk-walk* would be the same as another trip composed by *walk-walk-car*.

  This classical simplified collaborative-based approach will allow us to have a baseline model to compare with another one in which we will use the content information.

- Factorization Machines: It will allow us to introduce the scores for each one of the categories (content) as features and find the possible relations between them and the users. Hence, an example of an input vector of the algorithm is as shown below:

	User				Offer Categories			
x^i	1	0	0	...	-0.5	1.4	...	0.6

  where after the sparse user representation we have the 11 scores corresponding to each one of the categories. Compared to the input vectors shown in Table 3.1, now we do not include the sparse representation of the item identifiers. The reason is that in our problem the items (offers) are unique and therefore it does not make sense to look for relationships between the users and features with the items identifiers (they are all different).

Regarding the creation of the training data $(u, i, j) \in D_S$, we need to take into account that i and j must be offers belonging to the same trip request.

One the other hand, with this approach, given a pair of offers, we can say which one is more relevant than the other for an specific user using the probability given by Eq. 3.13. Since our intention is to create a ranking, we propose the following strategy:

- For each trip request, generate all possible pairs of offers and compute the probability of one of them being more relevant than the other.

- For each offer, keep count of the times that it was considered more relevant in front of another one.
- Use these counts to rank the offers for that specific trip request.

5.3.1 Evaluation Metrics

Recall is a classical evaluation metric in binary classification algorithms which has been "translated" to help evaluate recommender systems. For a given ranking of k items, the *Recall@k* is defined as:

$$R@k = \frac{\text{number of positive items among the top } k}{\text{total number of positive items}} \tag{5.2}$$

Translated to our problem, for each mobility request there is only one positive item, which is the offer that the user took. Therefore, in our specific problem, this number is either 0 or 1. Then, to obtain the average *Recall@k* for a given user, we can average this metric over all their requests. Finally, we can average over all the users to obtain this metric for the whole set.

Precision is another classical metric for binary classification problems which has been adapted to the recommender systems world. For a given ranking of k items, the *Precision@k* is defined as:

$$P@k = \frac{\text{number of positive items among the top } k}{k} \tag{5.3}$$

The Average Precision (AP) calculates the precision at the position of every corrected item in the ranking. For a given ranking of N items, the AP is defined as:

$$\text{AP} = \frac{\sum_{k=1}^{N} P@k \cdot \text{rel}(k)}{\text{total number of positive items}} \tag{5.4}$$

where $\text{rel}(k)$ is just an indicator that says whether that k-th element in the ranking was relevant ($\text{rel}(k) = 1$) or not ($\text{rel}(k) = 0$). Again, in our problem the total number of positive items is just 1. Also, to obtain a more general metric, we can average over mobility requests from an specific user, and finally over all the users. This is called the Mean Average Precision (MAP).

Chapter 6

Results

In this chapter I will give a detailed explanation of the experiments performed to test our recommender system together with the corresponding results. We design several experiments especially focusing on the behavior of the system depending on the number of trips registered by the users.

First of all, as previously explained, we have generated the trips alternatives by combining the legs alternatives. Because some of the trips have several legs, we can find a reduced number of trips with thousands of offers in our new dataset. On the other hand, the computational complexity of our evaluation strategy is of the order of N^2 (for a given trip request with N offers). For those trips with a lot of offers, generating the ranking is a really hard task, and this is the reason why we filter the trips we use for training based on the number of offers.

We study two different situations: trips with less than 100 offers and trips with less than 10 offers. This last scenario is closer to a more realistic situation where the number of alternatives will not be so large. In both cases, we train the system using three different subsets of the dataset based on the number of trips registered by the users: all users, users with more than 20 trips registered and users with more than 40 trips registered.

6.1 Trips with less than 100 offers

As mentioned in previous sections, in order to see whether or not including the categories as features is useful, we train using two different underlying models needed for the BPR algorithm: Factorization machines, using the content information of the categories, and Matrix Factorization. Results can be observed in Fig 6.1 and Fig 6.2, where we plot the *Recall@1*, i.e. we create a ranking of 1 item and check whether or not this is the positive offer, and the MAP for a ranking of 5 offers, which takes into account the position of the correct offer in the ranking.

First of all, comparing the results between the two underlying models, the ones obtained using the Factorization Machines are better in all situations. Hence, it is clear that the content information used for this model supposes a considerable improvement in the performance of the system. The bad performance of the Matrix Factorization might be due to the oversimplification we had to do to train the system. As previously explained, the offers are all unique, and in this situation the collaborative-based algorithms are not useful whatsoever. To solve this problem, we decided to identify trips as equal based on the fraction of the trip covered by each transport mode. This allowed us to train the system, but it seems that this approach is not allowing the model to properly learn user's preferences.

On the other hand, let us analyze the relation with the number of trips registered by each user. In this case, there are around 1500 different users, about 200 of them registered more than 20 trips and only 60 registered more than 40 trips.

We can observe that the more trips, the more information the system has to learn the interactions between the users and the categories, concluding in a better performance. In the best scenario (users with more than 40 trips), we achieve *R@1*=0.58 and MAP=0.73.

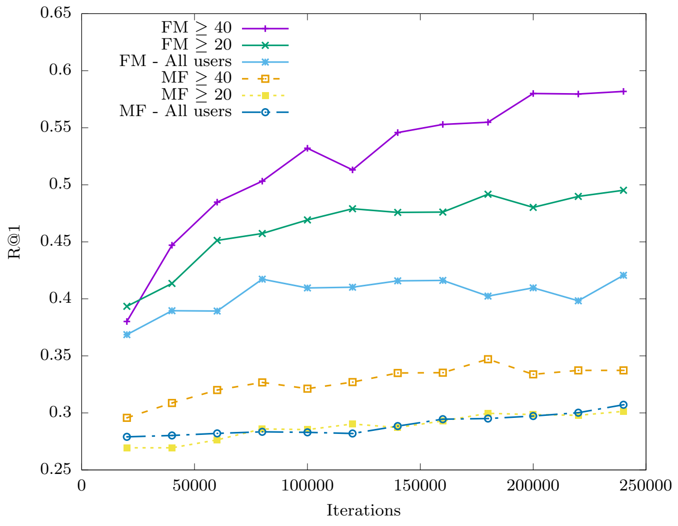


FIGURE 6.1: Average *Recall@1* evaluated on the test set as a function of the number of iterations during training. Solid lines are the results obtained with Factorization Machines, while dashed lines show the results obtained with Matrix Factorization.

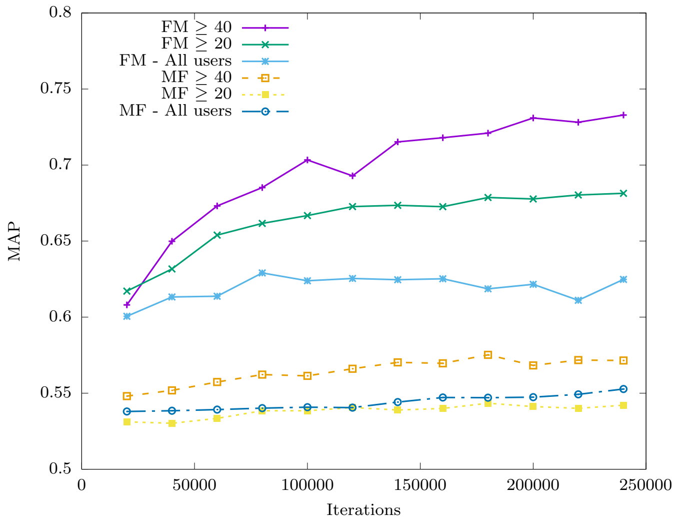


FIGURE 6.2: Mean Average Precision of a ranking of 5 items evaluated on the test set as a function of the number of iterations during training. Solid lines are the results obtained with Factorization Machines, while dashed lines show the results obtained with Matrix Factorization.

6.2 Trips with less than 10 offers

Let us consider now another situation. In a real scenario the number of offers we will need to classify and rank for a given trip request will not be as large as 100. This is why we decided to repeat the previous analysis but this time only using trips with 10 offers or less, which, on the other hand, were already representing most of the offers in the previous study (around 90%).

As it can be observed in Fig. 6.3 and Fig. 6.4, the behaviour does not change with respect to the previous analysis. However, in total numbers, we are able to increase the performance of the system in both metrics, achieving *R@1*=0.67 and MAP=0.81 for the case of users with more than 40 trips. This supposes an improvement of around 16% for the *Recall@1* and around 11% for the MAP. In an intuitive way, it is easier to make a ranking if we only have 10 offers than if we have more.

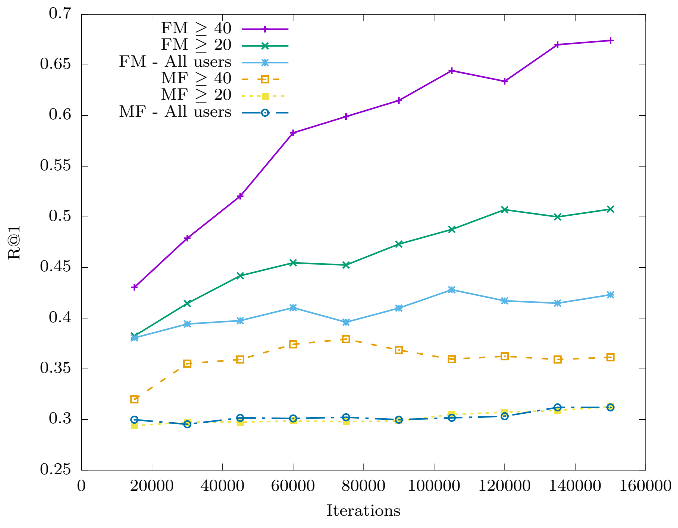


FIGURE 6.3: Average *Recall@1* evaluated on the test set as a function of the number of iterations during training. Solid lines are the results obtained with Factorization Machines, while dashed lines show the results obtained with Matrix Factorization.

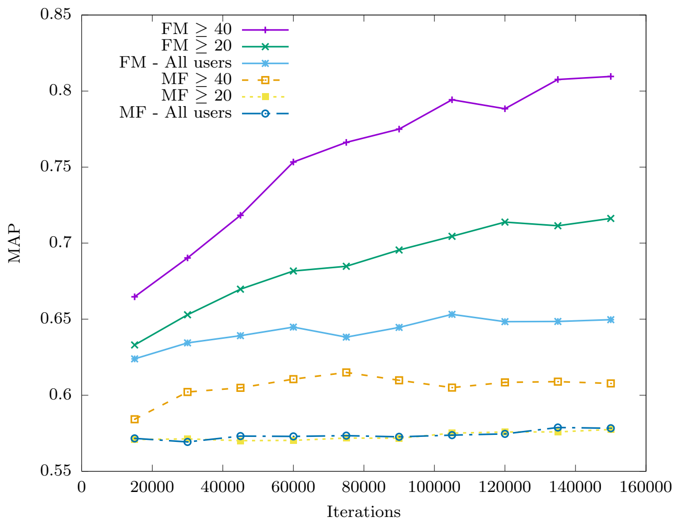


FIGURE 6.4: Mean Average Precision of a ranking of 5 items evaluated on test set as a function of the number of iterations during training. Solid lines are the results obtained with Factorization Machines, while dashed lines show the results obtained with Matrix Factorization.

Chapter 7

Conclusions

In this last chapter we detail the project's conclusions, we comment on the limitations we faced and propose future lines of work.

7.1 Conclusions

First of all, we recall that the main objectives of this work were to contribute in the development of the Offer Categorizer within the Ride2Rail project, and then use this tool to build a ranking algorithm which could learn user's behaviours and present them the best trip offers.

Regarding the direct contributions to the Ride2Rail project, three modules have been implemented: the Weather-FC and the Panoramic-FC, which compute the determinant factors related to the weather and the number of relevant spots, and the routeRANK extractor, which parses data from the routeRANK dataset and convert it to the Offer Cache specifications. The implementation of these modules within the whole system was still in the testing phase by the time this work was finished.

Then, we trained a recommender system using the implicit data from the routeRANK and MoTiV datasets. For that, using the BPR algorithm, we have benchmarked two underlying models: a classical Matrix Factorization and the Factorization Machines. For this last one we could include content information (in the form of the score categories) and the results are quite satisfactory, especially when the system is trained with users who have interacted many times with the system.

7.2 Limitations

In this section I will describe the two main limitations we faced while developing this work, especially relevant for the recommender system.

On the one hand, as it has already been explained, the lack of data supposed a big challenge because it did not allow us to properly test the different components of the Offer Categorizer. We tried to solve this problem by using the routeRANK dataset, which contains a set of alternatives from the real trip requests of the MoTiV project. The problems with this new dataset are:

- The alternatives were generated for the legs and not for the whole trip. In order to solve this issue we combined these leg alternatives. This caused the original dataset to increase considerably, and in some cases generating too many offers compared to a real scenario.
- The alternatives were generated after the real trip was registered. This means that the user did not explicitly choose among a set of offers, which would have

been the ideal situation to properly train the recommender system and learn the preferences of the users.

On the other hand, due to time limitations, the actual Offer Categorizer was not finished and therefore not available for the recommender system. This is why we decided to build a simplified version of the system. The main consequence is that some of the categories are not accurate. Particularly, the values of the Panoramic category were generated randomly. Also, the values of other few determinant factors (privacy level and seating quality) were chosen based on our criterion.

7.3 Future Lines of Work

With all that said, it is clear that there is plenty of room for further investigation. To conclude this work, I will point out the most immediate contributions that could be added to the present work. Related to the last point of the previous section, the next step is clearly to train again the recommender system but using the real Offer Categorizer to compute the offer categories. In this way, we should have a more powerful representation of the offers, allowing the system to better learn the preferences of the users.

Appendix A

Offer Categories

Offer Category	Determinant Factors	Source	Feature Collector
Quick	Trip Duration	TRIAS	Time-FC
	Waiting Time	TRIAS	Time-FC
	Time to Departure	TRIAS	Time-FC
	Number of stops	TRIAS	Position-FC
	Road-network distance/ Air-distance	TRIAS/ External	Position-FC
	Traffic	External	Traffic-FC
Reliable	Likelihood of delays	TSP	TSP-FC
	Traffic	External	Traffic-FC
	Last-minute changes	TSP	TSP-FC
	Time of trip (rush vs non-rush hour)	TRIAS	Time-FC
	Frequency of service	TSP	TSP-FC
	Weather forecast	External	Weather-FC
Cheap	Total price	TRIAS	Price-FC
	Can share cost	TSP	Price-FC
	Ticket coverage	TRIAS	Price-FC
Comfortable	Number of stops	TRIAS	Position-FC
	User feedback	TSP	TSP-FC
	Cleanliness (station, vehicle)	TSP	TSP-FC
	Seating quality	TSP	TSP-FC
	Amount of space available	TSP	TSP-FC
	Presence of silence area	TSP	TSP-FC
	Privacy level	TSP	TSP-FC
	Ticket coverage	TRIAS	Price-FC
	Ride smoothness	TSP	TSP-FC
	Weather forecast	External	Weather-FC
Door-to-door	Walking distance	TRIAS/ External	Active-FC
Environmentally friendly	CO2 emissions/km	External	Environment-FC
	Energy consumption	External	Environment-FC
	NOx emssions/km	External	Environment-FC
	Carbon footprint	External	Environment-FC
	Bike-on-board	TSP	TSP-FC
Short	Distance covered	TRIAS/ External	Position-FC
	Number of stops	TRIAS	Position-FC
Multitasking	Amount of space available	TSP	TSP-FC
	Presence of silence area	TSP	TSP-FC
	Internet connection	TSP	TSP-FC
	Plugs/charging points	TSP	TSP-FC
	Privacy level	TSP	TSP-FC
Panoramic	Number of monuments	External	Panoramic-FC
	Number of historical sites	External	Panoramic-FC
	Number of landscapes	External	Panoramic-FC
Healthy	Fraction of trip by bike/walk	TRIAS	Active-FC
	Length of fraction of trip by bike/walk	TRIAS/ External	Active-FC

TABLE A.1: Determinant factors, source of information and feature collector associated.

Appendix B

routeRANK dataset schema

```
{
    "tripId": "<tripId>",
    "from": {                           // trip origin coordinates
        "latitude": 11                  // number
        "longitude": 11                 // number
    },
    "to": {                             // trip destination coordinates
        "latitude": 22                  // number
        "longitude": 22                 // number
    },
    "date": "<trip_date>",              // trip's date
    "time": "<trip_time>",              // trips's time
    "alternatives": [{                  // list of all alternatives
        "id": "<altid>",                // alternative journey_id
        "segments": [{                  // list of all segments
            "id": "<segid>",            // segment id
            "from": {
                "id": "<id>",           // place id
                "name": "<name>"        // string
            },
            "to": {
                "id": "<id>",           // place id
                "name": "<name>"        // string
            },
            "co2": 1,                   // number, kg
            "price": 2,                 // number, EUR
            "duration": 3,              // number, seconds
            "distance": 4,              // number, km
            "transport": "<transport>", // string
            "arrivalTime": "<time>",    // string, hh:mm
            "arrivalDate": "<date>",    // string, yyyy-MM-dd
            "departureDate": "<time>",  // string, hh:mm
            "departureTime": "<date>",  // string, yyyy-MM-dd
            "legs": [{
                "id": "<id>",           // leg id
                "from": "<from>",
                "to": "<to>",
                "co2": 5,               // number, kg
                "price": 6,             // number, EUR
                "transport": 7,         // string
                "arrivalTime": "<time>",    // string, hh:mm
                "arrivalDate": "<date>",    // string, yyyy-MM-dd
                "departureTime": "<time>",  // string, hh:mm
                "departureDate": "<date>",  // string, yyyy-MM-dd
            }],
        }],
        "totals": {
            "co2": 3,                   // number, kg
            "price": 4,                 // number, EUR
            "duration": 5,              // number, seconds
        }
    }],
    "places": {
        "<id>": {
            "latitude": 11,             // number
            "longitude": 11,            // number
            "name": "<name>",           // string
            "countryCode": "<code>",    // string, country code
        }
    }
}
```

LISTING 1: routeRank output format

Appendix C

Offer Cache schema

Request-level information (Key: <*request_id*>:)
offers
Description: List of <*offer_id*> for the Mobility Request
Format: List of strings

Offer-level information (Key: <*request_id*>:<*offer_id*>:)
duration
Description: Duration of the trip associated to the offer
Format: String formatted as xsd:duration
start_time
Description: Starting time of the trip associated to the offer
Format: String formatted as xsd:datetime
end_time
Description: Ending time of the trip associated to the offer
Format: String formatted as xsd:datetime
num_interchanges
Description: Number of interchanges of the trip associated to the offer
Format: Integer
complete_total
Description: Total price of the offer
Format: HashMap containing two keys:

- value: Integer (last two digits are decimal, e.g., 6215 EUR represents 62,15 euro)
- currency: String formatted as ISO4217 currency code, e.g. EUR for Euro

legs
Description: Ordered list of <*leg_id*> composing the trip associated to the offer
Format: List of strings
categories
Description: Final scores computed from the offer categorization process for each offer category.
Format: Map containing the scores for each offer category. Keys are: *quick, reliable, cheap, comfortable, door_to_door, environmentally_friendly, short, multitasking, social, panoramic, healthy*

Leg-level information (Key: *<request_id>:<offer_id>:<leg_id>*)
leg_type
  Description: Type of a leg
  Format: String. Admissible values are: *timed, continuous, ridesharing*
duration
  Description: Duration of the leg associated to the offer
  Format: String formatted as xsd:duration
start_time
  Description: Starting time of the leg associated to the offer
  Format: String formatted as xsd:datetime
end_time
  Description: Ending time of the leg associated to the offer
  Format: String formatted as xsd:datetime
transportation_mode
  Description: Transportation mode for the leg
  Format: String
leg_stops
  Description: Coordinates of the stops performed during the leg
  Format: String formatted as GeoJson LineString. At least coordinates of starting and ending point

The following information will only be stored if available:
likelihood_of_delays
  Description: Number of times that previous users reported delays over the total number of times that users used the offer
  Format: Real value between 0 and 1
last_minute_changes
  Description: Measurement of likelihood of last-minute changes from the TSP
  Format: Real value between 0 and 1
frequency_of_service
  Description: Number of scheduled trips per hour (depending on time and day of the trip)
  Format: Real value
user_feedback
  Description: Previously reported average value about overall experience
  Format: Real value from 1 to 5 (five stars scale)
cleanliness
  Description: Previously reported average value about experienced cleanliness (of stops, stations and vehicles) used by a given TSP
  Format: Real value from 1 to 5 (five stars scale)
seating_quality
  Description: Previously reported average value about experienced seats comfort of vehicles used by a given TSP
  Format: Real value from 1 to 5 (five stars scale)
silence_area_presence
  Description: Whether there is a silence area or not
  Format: Binary value: 0 or 1
privacy_level
  Description: Previously reported average value about experienced privacy level

in vehicles used by a given TSP
Format: Real value from 1 to 5 (five stars scale)
business_area_presence
Description: Whether there is a business area or not
Format: Binary value: 0 or 1
internet_availability
Description: Whether there is WiFi connection or not
Format: Binary value: 0 or 1
plugs_or_charging_points
Description: Whether there is charging points or not
Format: Binary value: 0 or 1
safety_features
Description: Previously reported safety of the trip as perceived by the passenger
Format: Real value from 1 to 5 (five stars scale)
ride_smoothness
Description: Previously reported smoothness (e.g. presence and intensity of sudden changes in the speed and direction of the vehicle) of the trip as perceived by the passenger
Format: Real value from 1 to 5 (five stars scale)

Appendix D

Weather Feature Collector

In the categorization of the weather forecast we use information of the temperature, clouds, wind and precipitation. In the tables below, we present how we map the data obtained from OpenStreetMap to the labels used for the categorization. In the case of the precipitation, we use the data as we obtain it from the API.

Temperature (°C)	Labels
<0	Uncomfortably cold
$0 - 15$	Cool
$15 - 25$	Comfortable
$25 - 32$	Warm
>32	Uncomfortably hot

TABLE D.1: Conversion from degrees Celsius (as given by the *One Call API*) to the labels used for the categorization of the weather forecast.

Cloudiness (%)	Labels
0-11	No clouds/ Clear sky
11-84	Partially cloudy
84-100	Completely cloudy

TABLE D.2: Conversion from percentage of cloudiness (as given by the *One Call API*) to the labels used for the categorization of the weather forecast.

Speed (m/s)	Labels
$0 - 5.5$	Light breeze
$5.5 - 17.1$	Strong breeze
>17.1	Gale

TABLE D.3: Conversion from wind velocity (as given by the *One Call API*) to the labels used for the categorization of the weather forecast.

Bibliography

Chaube, Vineeta, Andrea L Kavanaugh, and Manuel A Perez-Quinones (2010). "Leveraging social networks to embed trust in rideshare programs". In: *2010 43rd Hawaii International Conference on System Sciences*. IEEE, pp. 1–8.

Consonni, Cristian et al. (2021). "What's Your Value of Travel Time? Collecting Traveler-Centered Mobility Data via Crowdsourcing". In: *arXiv preprint arXiv:2104.05809*.

Docker. URL: https://www.docker.com/.

Flask. URL: https://flask.palletsprojects.com/en/2.0.x/.

Furuhata, Masabumi et al. (2013). "Ridesharing: The state-of-the-art and future directions". In: *Transportation Research Part B: Methodological* 57, pp. 28–46.

Github. URL: https://github.com/.

Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). "Matrix factorization techniques for recommender systems". In: *Computer* 42.8, pp. 30–37.

McNee, Sean M, John Riedl, and Joseph A Konstan (2006). "Being accurate is not enough: how accuracy metrics have hurt recommender systems". In: *CHI'06 extended abstracts on Human factors in computing systems*, pp. 1097–1101.

MoTiV. URL: https://motivproject.eu/.

Occupancy Rates (2020). URL: https://www.eea.europa.eu/publications/ENVISSUENo12/page029.html.

OpenWeatherMap.org. *Weather API*. URL: https://openweathermap.org/api.

OSM. URL: https://www.openstreetmap.org/.

Redis. URL: https://redis.io/.

Rendle, Steffen (2010). "Factorization machines". In: *2010 IEEE International Conference on Data Mining*. IEEE, pp. 995–1000.

Rendle, Steffen et al. (2014). "Bayesian personalized ranking from implicit feedback". In: *Proc. of Uncertainty in Artificial Intelligence*, pp. 452–461.

Resnick, Paul et al. (1994). "Grouplens: An open architecture for collaborative filtering of netnews". In: *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pp. 175–186.

Ride2Rail (2021). URL: https://ride2rail.eu/.

Roberts, Ron and Paul Goodwin (2002). "Weight approximations in multi-attribute decision models". In: *Journal of Multi-Criteria Decision Analysis* 11.6, pp. 291–303.

routeRANK. URL: https://www.routerank.com/en/.