UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

A restaurant recommender system for a new-born app-based gastronomic guide

Author:
Pablo GRANATIERO

Supervisor:
Dr. Jerónimo HERNÁNDEZ GONZÁLEZ
and
David Martin SUAREZ
(Product Officer)

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

A restaurant recommender system for a new-born app-based gastronomic guide

by Pablo GRANATIERO

This is a project of applied data analysis made in collaboration with the *CPO* of the restaurants app Velada. We analyze the data collected by the app with the objective of building a recommender system for its restaurants. The initial objective was to give particular attention to the parameter time, building a model able to make the right recommendation in the right moment for each user of the app. Different attempts have been performed, using both collaborative filtering and content based recommender systems, with different types of information as data input. We show that the best approach is a binary collaborative filter, although the results are preliminary due to the lack of enough data. We also show that the performance will be easily improved as new data becomes available. Finally, we provide some insights on how the problem of making a recommendation for a restaurant just in time could be solved in the future.

The notebooks and the codes of this project can be found in
https://github.com/pablogrr/TFM_granatiero

Acknowledgements

I would first like to thank my supervisor, Dr. Jerónimo Hernández González, for his insightful feedback, his great kindness and patience.

A special thanks to David Martin Suarez for his willingness to cooperate.

Thanks to my wonderful classmates Mattia, Alex, Dave, Alejandro, Petar and Marya.

To the girl I love and to my father who would be proud of me.

Chapter 1

Intro

1.1 Introduction

Nowadays a consumer who is about to buy an item has to swim in a ocean, is inundated of, different possibilities. **Choosing** is becoming one of the most common activities of a consumer everyday life (think to *Netflix*, *Amazon*, *Youtube*, *ebay*) and, for this reason, one of the key elements of every online business. As choosing something is becoming more and more important, recommending the right thing is becoming crucial.
Therefore, more companies started using recommender systems, which analyze patterns of user behaviours about products to provide highly personalized and customized recommendations that suit a user's taste. Good personalized recommendations can add another dimension to the user experience.
So **Recommender Systems** is actually a hot topic in Data Science and data scientists never stop developing new **Recommender Algorithms**, with a prolific productions in the scientific literature.

1.2 What is a recommender system

A *Recommender* System is an algorithm that takes as input a user identification tag, a *user id*, and gives as output a set of one or more suggested items. There are several ways to do it, based on the information and the tools we are using to generate the recommendation. The first difference we enlight is the one between **Non Personalized** recommenders and **Personalized** ones.
Non personalized recommenders are very common and they stand out for making the same *recommendation* to everyone. The simplest Non Personalized Recommenders are the *K* Most Popular Items, for example top ten movies in Spain we found in the *Netflix* homepage (the first ten movies ranked for number of views in Spain), even if very simple, it is still one of the most important recommenders in many platforms.
On the other way Personalized Recommenders are the ones that try to guess a specific user's taste inferring from his past behavior. This kind of recommender has several advantages, because it aims to give a more personalized suggestion, increasing the variety of recommended items and the user's serendipity. There are several ways to build a Personalized Recommender, but the two most important and bigger branches are the Content Based and the Collaborative ones. In the next two sections we will briefly introduce these systems, in the third and last section we will introduce a third kind of recommender, part of the collaborative ones, called model based.

1.2.1 Content Based

Following (Pasquale Lops and Semeraro, 2011), the key concept behind a Content Based is to recover for users and items the same representation as to be able to evaluate distances between them. Broadly speaking, Content Based Recommender Systems are based on the approach to create a profile for each user and for each product to characterize its nature. For example, a restaurant profile could include attributes regarding its type of food, its price, the number of Michelin Stars, the Neighborhood and so forth. Then a user profile is built using exactly the same features, assign to each feature a score based on the restaurants actually visited by the user (for example if a user usually appreciates Japanese and Chinese restaurants we will assign to him a high score to the food type feature *Asian*).
In this way we can evaluate similarities between the user and all the restaurants. For example we can evaluate the 5 nearest neighbors in the set of the restaurants he hasn't tried yet. A known successful realization of Content Based Models is the *Music Genome Project*, which is used for the Internet radio service *Pandora.com* using more then 400 features.

1.2.2 Collaborative Filtering

Another way to make recommendations was introduced for the first time in (David Goldberg and Terry, 1992) and it is called precisely by the paper's author *Collaborative Filtering*. These models do not require to build a user vector features representation and it is based just on his past behaviour.
The system is centered on computing the relationships between items or, alternatively, between users (it can be indeed user oriented or item oriented). The item oriented approach tries to infer a user's preference for an item based on the ratings he gave in the past to other items, basing on the assumption that similar items will have similar score. Similarities between the items are investigated inside all the users preferences, where items appreciated more or less by the same users, are more or less similar.
For example, consider a Japanese restaurant. Its neighbors might include other Japanese restaurants, Korean restaurants and other Asian ones among others. To predict a particular user's rating for this restaurant, we would look for the restaurants's nearest neighbors that this user actually rated.
A major appeal of collaborative filtering is that it is domain free, yet it can address data aspects that are often elusive and difficult to profile using Content Based and it is generally more accurate than Content Based techniques.

1.2.3 Model based

The last kind of models we will introduce are a Collaborative Filtering called *Model Based* mostly using **matrix factorization** techniques. These models are properly defined *Collaborative* too, in the meaning that they are based on the same idea and tools. Thanks to matrix factorization these models aim to embed users and items in a latent space whose dimensionality is a parameter of the model.
Such latent factors are a machine alternative to the aforementioned human crafted items feature. For restaurants, the discovered features might measure obvious dimensions such as Asian cuisine versus Italian cuisine, amount of price, or neighborhood. But, depending from the number of dimensions, it can understand more hard to find, sometimes uninterpretable, aspects.
In the recent past more sophisticated and complex model, like in (Rendle, 2010) or

deep models, try to merge the information retrived from both the systems, Content and Collaborative, the so called hybrid models.

1.3 Restaurant Recommender

The objective of the present project is to build a recommender for restaurants, in particular for the new Spanish app **Velada**. In the next chapter we will describe the app main features but we can anticipate here that **Velada** is a digitized gastronomic guide, so a recommender system, suggesting the right restaurant to the right user, can obviously play a key role. Furthermore time can be also a crucial parameter, in particular the initial proposal suggests to pay attention to recurrency, i.e. trying to recommend restaurants considering, not just how much a user likes a given restaurant but also how often he enjoyed it.
In order to develop our models we followed these steps: the first one was to understand how the app works, and in the second chapter we will try to see in detail its main tools. Then we will see the data we used, describing how we collect them, all the analysis we perform in order to derive insights indispensable to take important decisions to model the recommenders. Finally, we will show the recommenders that we built, starting from very basic models proceeding to more elaborated ones. We will present the resluts that these recommenders reached, quite limited due to the lack of enough data. In the last chapter we tries to investigate the recommender's dynamic aspects, paying attention to recurrency, suggesting how we could include temporal dynamic, possible solutions and future implementations.

Chapter 2

Velada

2.1 The App

Released on September 2020, Velada is an app for *android* and *apple* devices. It was born with the aim of modernizing the world of gastronomic guides, updating the user experience of the most demanding *foodies*, loyal to the quality of guides such us the famous *Michelin*, to a more practical and digitized way.
They present themselves as the *Tinder* of restaurants finding the perfect match of users' needs and restaurants. The way Velada does it is adding to the usual features of a food application, like reviews, geo-localization or food-type, also filters linked with more insightful needs, like *First Date* ore *covered terrace*.

Currently is restricted to Madrid and Barcelona but in the future it will work in other cities like Milan, Lima, London, Paris.

The opportunities behind a recommender system for an app like *Velada* are several. Indeed it would allow a better experience for the user and a better way to promote restaurants and restaurant diversification inside the app.
In principle one of the main idea of the app developer was to build an algorithm tailor made for the restaurants' commodity sector needs, in particular an app able to consider time as a key parameter, not only able to suggest the right restaurant but also able to do it in the right moment. For a problem like this, considering the products characteristics plays a key role. Indeed, restaurants, contrary to a movie or a song have a very different degree of recurrency. As we will see, we would argue that the app is in a too early stage to develop a recommender that considers time as a parameter, that needs clear users' behaviour paths through time, in a range of time longer than the actual one (we have data of two months).

2.1.1 Flow

In this section we will see more in detail how the app works, trying to follow the user from switching on the app (figure 2.1, Step 1) to choosing a restaurant.
The *user journey* starts at the *home* page (figure 2.1, Step 2), where the user can choose between a pre selected set of restaurants, based on a pre setted set of filters (detecting a particular need like *With Michelin Stars, With delivery, ...*).

Pushing on one of the *home page* buttons, the user enters to a restaurant card of a restaurant under the given filters (figure 2.1, Step 3). Here we can understand why they call themselves the *Tinder* of restaurants. Indeed here the user can perform the *swipe* action that becames famous with the date app: swiping left (figure 2.1, Step 4a) he skips the restaurants, swiping down ((figure 2.1, Step 4b) he save the restaurant in his personal favourite restaurant list. In both cases, a new restaurant card is showed.

From the card section in any time the user can push the button on the upper right corner, opening the filters menu. Here (figure 2.1, Step 5) he can customize his search adding from three different kind of filters: Vibes (as we said insightful needs like *Good brunch*, *special occasions* or *first date*), Food-Type (precisely the types of food, like *Italian* or *Japanese*) and the neighborhoods.

Alternatively, in the restaurant card section a fourth action is allowed. Pushing the *view item* button the user enters into a restaurant dedicated section, where a series of actions and information about the specific restaurant are accessible (figure 2.1, Step 6). Here the user can: go to the restaurant website or restaurant position on *Google map*, go to the restaurant and/or chef *Instagram* page, save the restaurant into his favourite list (pushing the *favourite* button), reserve, call or request a delivery to the restaurant directly inside the app or being linked to dedicated page.

2.1.2 The Data

In the next section we will describe in detail the data we used in order to try to build a recommender system for Velada. In this section we anticipate that the app developers provided us, users and restaurants data, from two different sources: one directly stored by the app, with all the information of users and items collected at the sign-in, and one stored by *Google Analytics*, describing for each user all the actions he did in the app, from the registration to the present time.

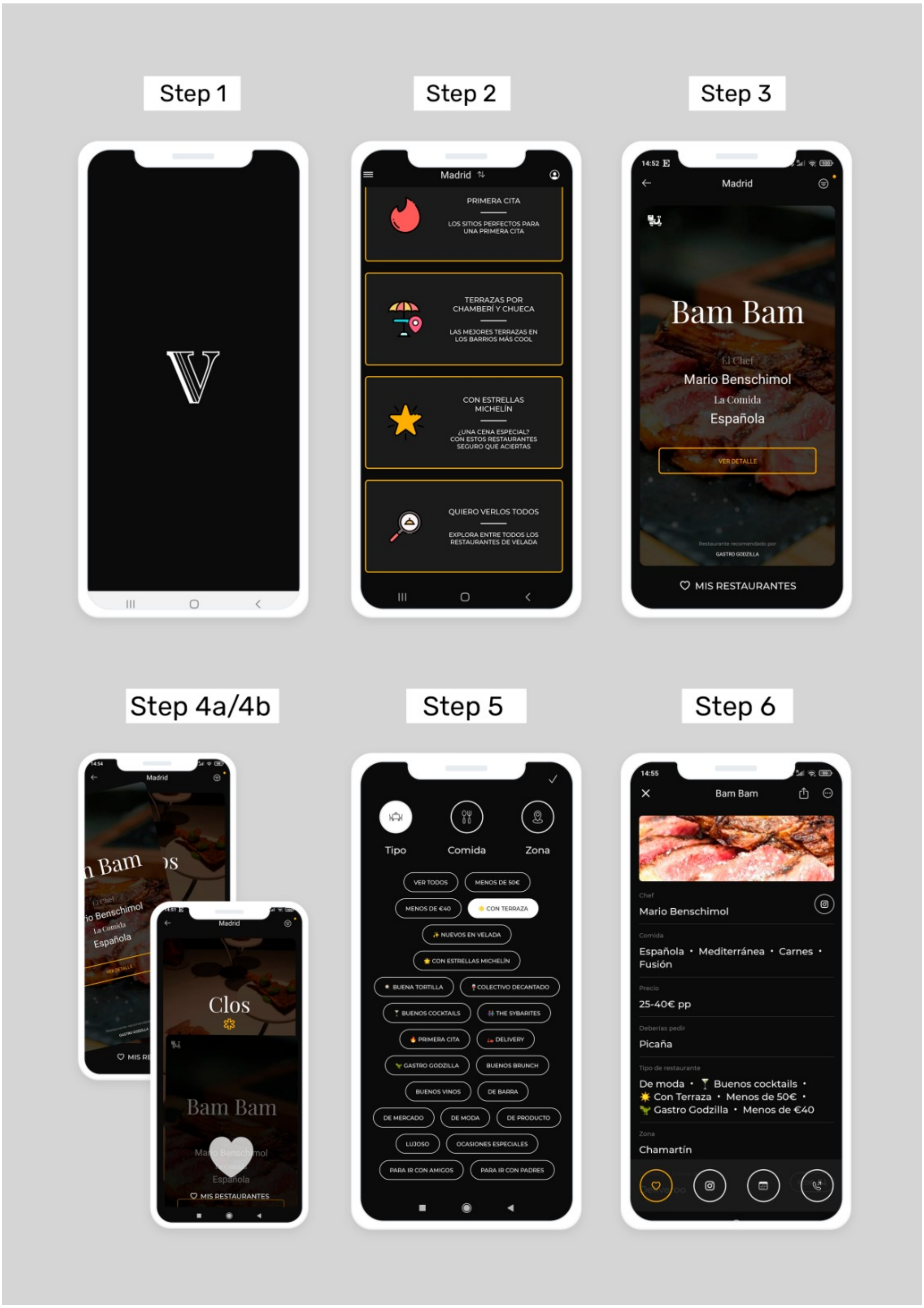


FIGURE 2.1: The basic user journey inside Velada app

.

Chapter 3

The Data

3.1 The data sources

As we said in the previous chapter, raw data files come from two different sources: one comes directly from the app and the other is derived by a query from the Google Analytics account of the app.

The data coming from *Velada* is composed of several tables with all the information about users and restaurants, collected by the app at the moment of the user and restaurant registration. They are static, without time dependency, related only to the instant of the *sign-in*.
Each table is a different dataframe describing a different restaurant or user feature, with information, mostly about restaurants, useful for a Content Based Recommender, like *price*, *food-type*, or the aforementioned *Vibes* filters. We will see it more in detail in the *Data Analysis* section.

The data downloaded from the app *Google* account is composed of several files with all the information about users and restaurants, collected by *Google Analytics*. These are dynamic data and each file is related to the information collected in one day user activity.
Each file is a different dataframe describing the different actions the users do interacting with the app, following the user experience saw in the previous chapter. We will see this too, more in detail in *Data Analysis* section. This kind of information are useful for a Content Based Recommender but, with some feature engineering, such as identifying a target variable (i.e. a variable thanks to which we can establish if and how much a user approves an item) is also relevant to build a collaborative one.

In the next two sections we will first describe the data frame we obtained initializing the raw data and then we will see more in detail the main information we retrieve analyzing them.

For the interested reader, an example of the data coming from *Velada*, is available in `https://github.com/pablogrr/TFM_granatiero`.

3.2 Initializing raw data

Preprocessing in real life is a background, hard to see, job that takes a lot of time and effort. It happened in our case too: indeed we spent a good percentage of our total time cleaning and preprocessing data. The more demanding readers can find in the Github repository the program to convert raw data in an exactly ready-to-use dataframe.

3.2.1 Data from the app

Let's see more in detail the data coming directly from the app. It includes: the info at the registration, stored in several tables, in particular, we will use the data contained in the following dataframes:

- `restaurants`: in this dataframe all the main information about restaurants are collected, for instance:
  - `name`: the name of the restaurant.
  - `restaurant_id`: the identification number of the restaurant.
  - `neighborhoods`: the city area of the given restaurant expressed with an identification number.
  - `price range`: the price range, from "less than 20" to "more than 100" proceeding with steps of "20€".
  - `Michelin stars`: the number of stars, from zero to three.
  - `city`: the city expressed with an identification number. At present Madrid and Barcelona are the only active cities, but there are already registered restaurants from other cities, like Lima, Paris, Milan, London.
- `restaurants_vibes`: as we saw, filtering, is one of the main Velada's tool and *Vibes* filters are the more distinctive. In this dataframe each `restaurant_id` is related to one or more of the 30 different vibes filters (with many different insights from *with delivery* to *first date* passing through *with terrace*.)
- `restaurants_food_types`: this is a more typical way to filter queries in food apps. In this data frame each `restaurant_id` is related with one or more `food-types`. For a total of 24 different food-types (*Asian*, *Galician*, *Italian*, ...)
- `cities_neighborhoods`: the same as before but with cities' areas. This data frame defines a mapping between a neighborhood and a restaurant.

This kind of categorical information is useful to build recommenders belonging to the category described in section 1.2.1, the content based. We built this recommender and we will describe it in the next chapter.

3.2.2 Data from Google Analyitics

We will describe the data from the *Google Analytics* app account. After some pre-processings we obtained a dataframe storing for each user all the actions of the *user journey* described in chapter 2.1.1, each action related to a specific instant described by a timestamp. So each row is a user-item interaction, each column a specific feature describing the interaction, for instance:

> `event_name`: it takes different values depending the action the user did.
>
> > `card_swipe`: one of the most iconic user interactions of Velada is to swipe. This feature says the user swiped a restaurant (see figure 2.1, step 4*a* and 4*b*).

  - `view_item`: the user enters in a restaurant page (see figure 2.1, step 5).
  - `restaurant_favourite`: the user saves a restaurant into his favourite list. One of the actions that the user can perform in a restaurant page (see figure 2.1, step 5).
  - `restaurant_action`: the user does one of the action_string_value actions. Again actions described above, allowed in a specific restaurant page (see figure 2.1, step 5).
  - `home_filters`: there are several ways to add filters, one of these are in the homepage. This feature shows the user added a filter in the homepage.
  - `filter_added`: the user added a filter outside the homepage.
  - `my_favs_remove_restaurant`: the user can remove a restaurant from his favourite restaurant list.
  - `my_favs_filter_added`: as restaurants, it exists also a favourite list of filters. The user saved a filter in his favourite filter list.
  - `restaurant_blacklisted`: the user moved a restaurant into a blacklist.

- `event_timestamp`: it relates each user action with a specific instant t.
- `user_pseudo_id`: a univocal map between a user and an *id* provided by *Google*.
- `type_string_value`: the food-type the user selected adding a filter.
- `action_string_value`: as we said the actions inside a restaurant page.
  - `menu`: the user saw the menu.
  - `favourite_press`: the user saved the restaurant into his favourite list.
  - `instagram`: the user saw the restaurant *IG*.
  - `website`: the user saw the restaurant website.
  - `book_url`: the user reserved the given restaurant.
  - `maps`: the user clicked the link to the map.
  - `curated_by`: the user pushed the *curated by* button.
  - `chefInstagram`: the user pushed the link to the chef *IG*.
  - `delivery`: the user pushed the link to ask delivery.
  - `call`: the user pushed the link to call a restaurant.
- `vibes_string_value`: the vibes that the user selected adding a filter.
- `dir_string_value`: how the user swiped a restaurant.
  - `down`: the user saved the restaurant in his favourite list.
  - `left`: the user skipped the restaurant.

The first thing that comes to our mind, when looking at these data with the aim of building a recommender system, is the absence of explicit features, like *thumbs up*, *thumbs down* or a classic *5 stars rate*. We will need this kind of information in order to build a recommender system of the Collaborative family.
By the way, we used the information contained in this dataframe also to build a Content Based Recommender, indeed they are indispensable to create a user representation vector.

3.3 Data analysis

With the data described in the previous chapter we can perform a lot of different interesting and insightful data analysis. For the sake of clarity in this chapter we will pay attention to all those aspects that aim to better understand how the recommenders we used work and why we took some decisions that we made throughout the development of this project.

3.3.1 The cities

One of the first decisions we agreed with the app developers, was to limit our model to the restaurants located in Madrid and the interactions with them. Indeed if we see at figure 3.2a, at the date of 30/04/2021 **the total number of restaurants is 289**, where the **67.82**% of them are from Madrid and the rest are from Barcelona or from other cities. Of these **196** restaurants from Madrid, **189** are active, in the meaning that users made with them one interaction or more.
The fact that the majority of the activity is from Madrid is even more clear if we look at figure 3.2b. We can see that **the total number of users is 7503**, of those users $\sim$ **91**% are from Madrid, where by this we mean users that interacted only with Madrid restaurants.
At this early stage of the app, Barcelona data is still very poor and this is why we restrict all our work to Madrid, users and restaurants.

3.3.2 Overall Interactions

In this section we will analyze more in detail how and how much the users interact with the app. First of all in figure 3.1 we can see the box plot of the users interactions, counting the statistics of the total number of interactions for all the users. As we will see, the most common action, in an unbalanced way, is *swipe_left*, for this reason we show the data with and without this action: at the top the box plots considering *swipe left* action in normal and logarithmic scale. The mean number of actions per user is $\sim$ **61.13** with the **75**% of all users between **1** and **71** actions. If we do not consider *swipe left*, the mean reduces to $\sim$ **24.18** with the **75**% of all the users between **1** and **26** actions, highlighting that most of the users are at the very first experiences with the app. We can also observe the presence of outliers, probably due to someone testing the app.

3.3.3 Actions outside the restaurants pages

As we saw in sections 2.1.1 and 3.2.2, there are two main spaces of interactions, outside and inside the restaurants' pages. There are actions users can perform in both spaces and there are actions that they can perform just inside or outside restaurants' pages. Here we analyze all the actions they can do outside.
In figure 3.3a we display a bar plot with all the user actions. As we already said, the most common action is by far *card swipe*: swipe is the most entertaining action and mimicking *Tinder* with restaurant has been a good choice. Figure 3.3b shows that $\sim$ **91**% of all *swipe* are *swipe left*, suggesting that skipping restaurants is something equivalent to *zapping* in *Netflix*. Also in figure 3.3a we can also see that *view item* is, with almost an order of magnitude less than *card swipe*, the second most common action. And this is not surprising, as after some swipes, the most common thing to

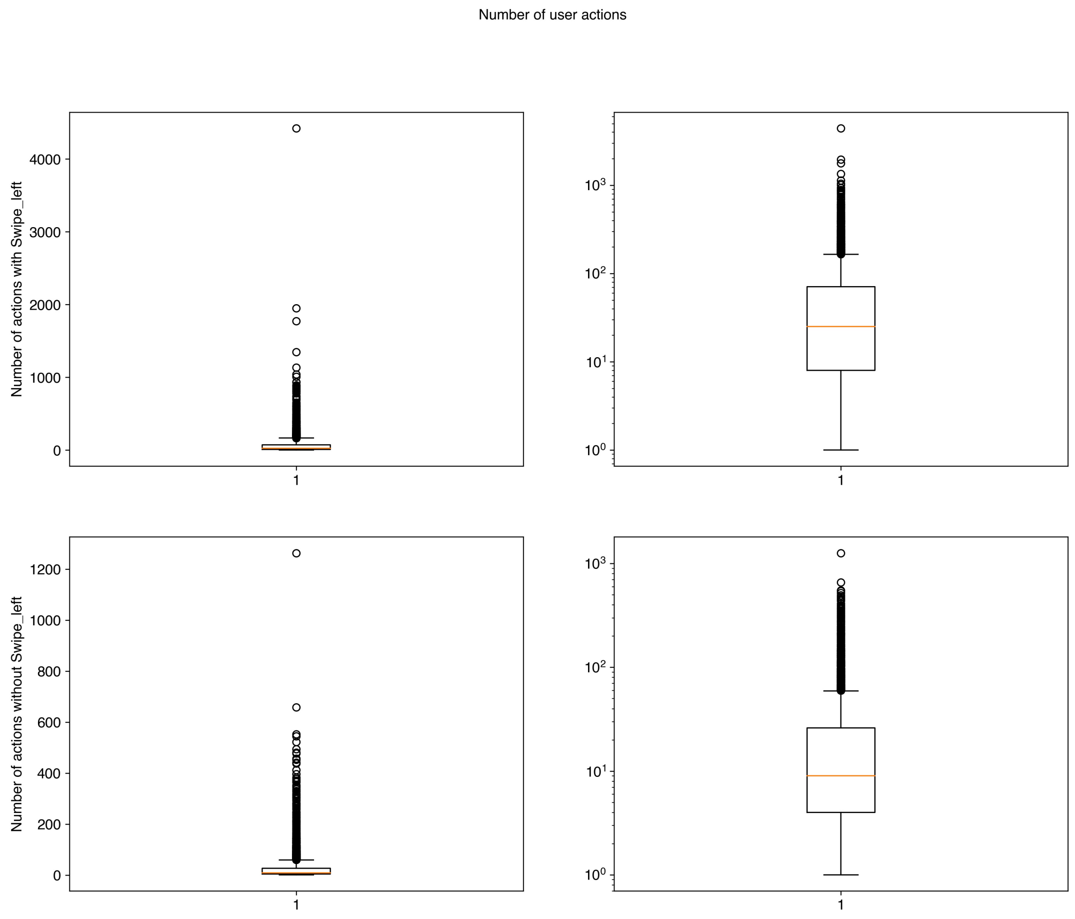
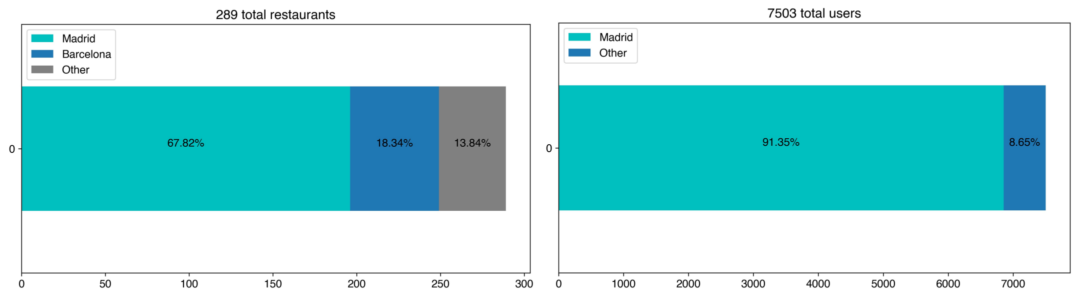


FIGURE 3.1: A box plot of the total number of users' interaction with the app. On the right in logarithmic scale and on the bottom without considering the Swipe Left action.



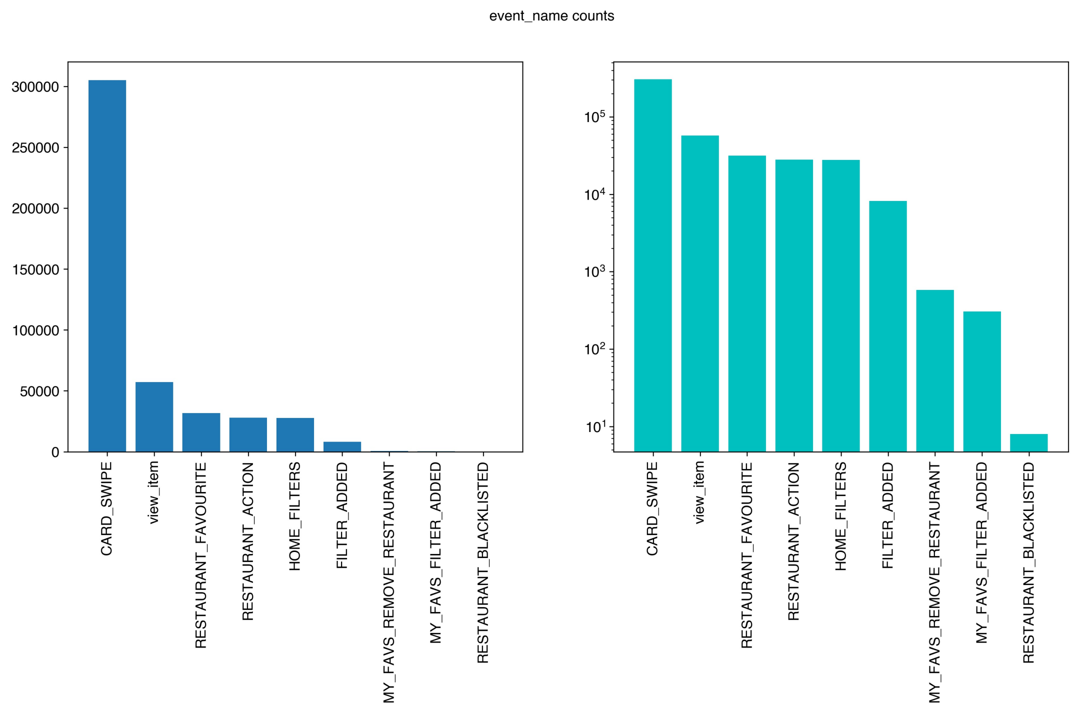
(A) All the restaurants in Velada. (B) All the restaurants in Velada.

FIGURE 3.2: Distribution of users and restaurants in Velada.

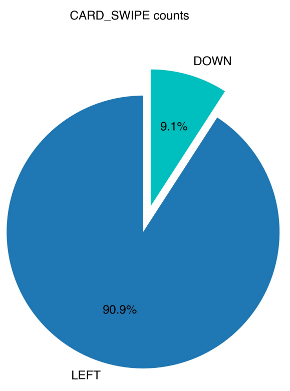
do is to get more information about a given restaurant, entering its page by pushing *view item* button. In the next page we see the most common actions inside a restaurant page.

3.3.4 Actions inside the restaurants pages

Inside a restaurant's page there are 10 possible actions, of which saving the restaurant in the favourites list is allowed in many different ways outside and inside the page. The remaining actions are exclusive of this area of the app.



(A) Total number of different *event_name* actions, on the right in logarithmic scale



(B) The total number of *card swipe* actions.

FIGURE 3.3: Actions outside the restaurants' pages.

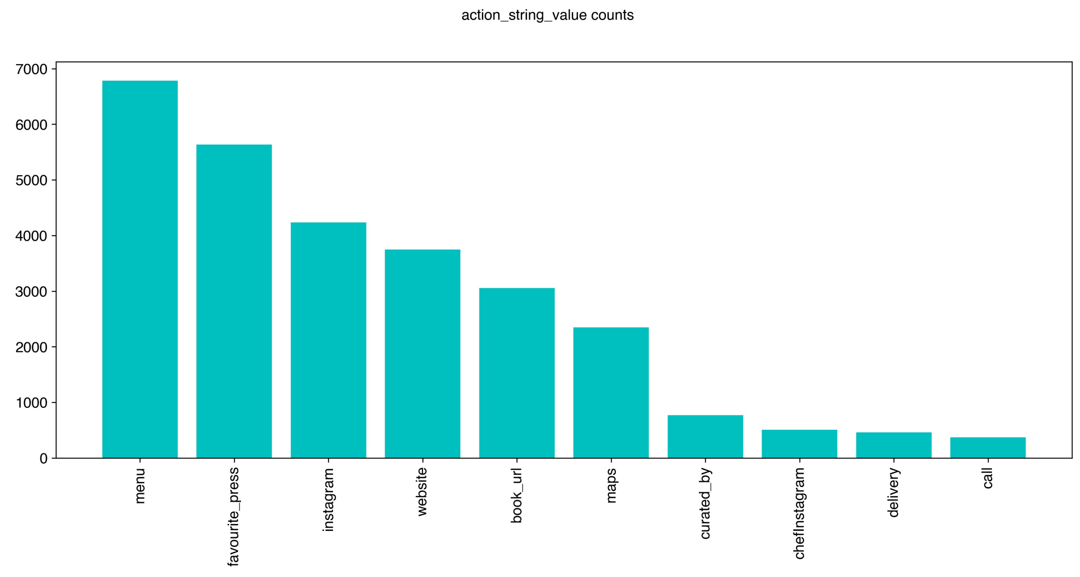


FIGURE 3.4: The total number of interactions the users can do inside a restaurant page.

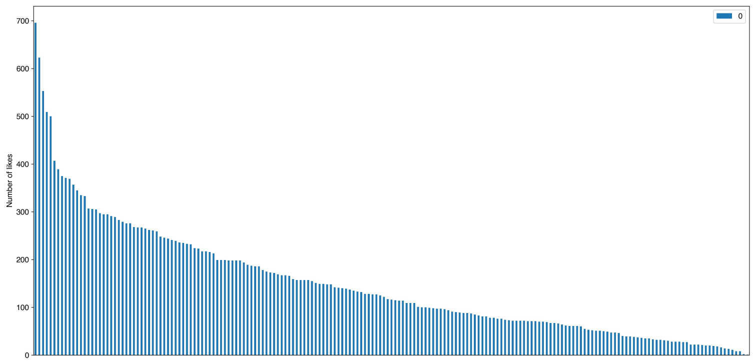
In figure 3.4 we can observe that, the most common actions are the *menu* button and the *favourite* button.
The fact that the *favourite* actions are popular is a good news for a recommender system builder, indeed we can consider it as an explicit positive feedback. Even if less popular, there are other actions that can be translate unequivocally as a *thumb up*: *book*, *delivery* and *call*. Indeed they express an active intention to enjoy the restaurant by the user.

3.3.5 Likes

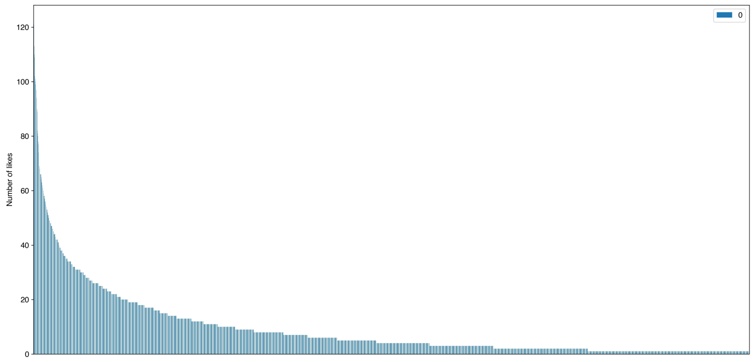
One of the key elements of a *Collaborative Filtering* Model is the presence of an explicit rate, but as we will see in the next chapter even for a Content Based, we will need the information if a user actually has positive opinion of a given restaurant. In absence of it, we built a target variable in a qualitative way, i.e. grouping together all the interactions that are, as we said in the previous chapter, unequivocally positive and we will simply call them *like*. We detect as positive features: *favourite* (in any way it happens), *book*, *delivery* and *call*.
Figure 3.5a and 3.5b shows the total number of *likes* for any user and for any restaurant: the two curves show the typical long tail shape, where some restaurants are more popular than others and some users are more *generous* in *likes* than others.

3.3.6 Time dependencies

One of the main objectives of the app project was to build a recommender able to consider time as a parameter: able to suggest not just the right restaurant but to do it also in the right moment. For this reason we analyze how long and how often the users interact with the app, trying to detect some pattern in the users' behaviour. In this section we analyze some dynamic aspects, passing from the first instant, $\mathbf{2021-03-07,\ 13:01}$, to the last instant, $\mathbf{2021-04-30,\ 21:59}$.
In figure 3.6a we analyze how long the users used the app. For this reason, we took



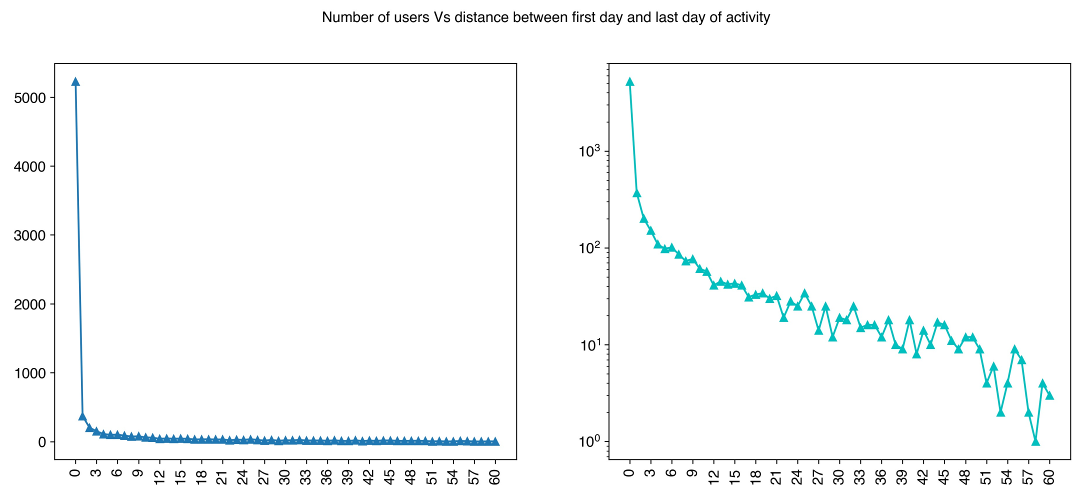
(A) The number of like for each restaurant.



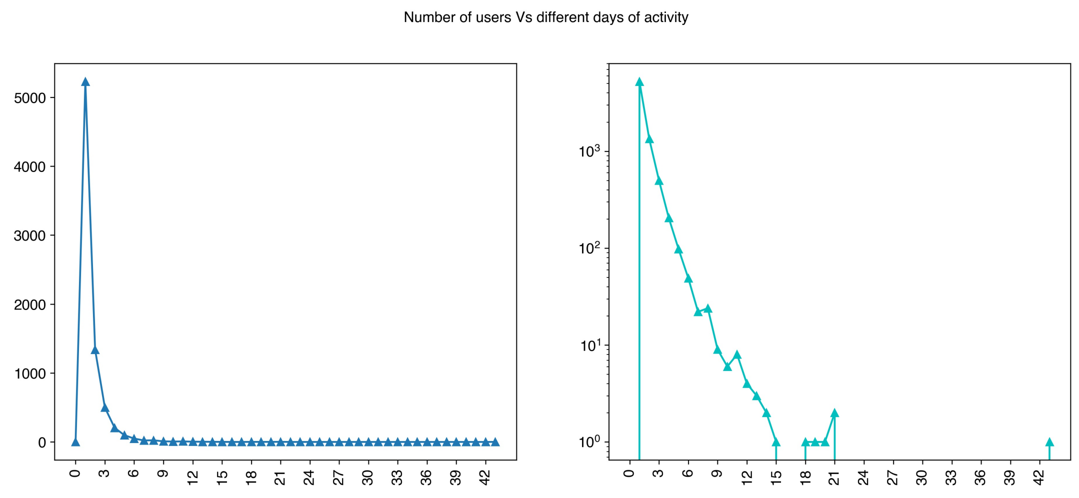
(B) The number of like for each user.

FIGURE 3.5: The long tail shape of the curves counting likes.

for each user the distance in days $t_{last} - t_0$, where t_{last} is the instant of the last interaction and t_0 is the instant of the first one: we can see that actually the vast majority of the users interact for just one day and, by one order of magnitude less, we observe a recurrency of 5 days ($O(10^2)$). In figure 3.6b we try to understand how often they use the app, for this reason we plot for each user the number of different days of activity, the cardinality of the set $d_0, \ldots, d_n$ where d_i is a given day where he did at least one action. In this case too, it falls down by one order of magnitude (from $O(10^3)$ to $O(10^2)$) passing from 1 single day to 5 different days of interaction.
These numbers suggest us that it is too early to build a recommender with a time dependency. Indeed our recommender, more than learning a clear users' opinion about the restaurants, will learn their very first opinion about them due to a first interaction with the app. Nevertheless, as we will discuss in the next chapter, with these premises we obtained some result which leaves us hopeful for the future.



(A) On the y axis the number of users, on the x axis the distance between the first and the last day of usage.



(B) On the y axis the number of users, on the x axis the number of different days of activity.

FIGURE 3.6: Time behaviour of users' activity. On the right in logarithmic scale.

Chapter 4

The Recommenders

In this section we will describe in detail the recommenders we built. In the first section we will describe the preprocessing and the decisions we took as to build a common framework for all the models. Then we will describe two base models, simple models built to be a reference to compare performances with each others. Finally we will show the recommenders we built and results we obtained with them.

4.1 Preprocessing

As we introduced in the previous chapter, the first decision we took is to filter our data to the restaurants (and interactions) located in Madrid. Indeed we saw in figure 3.2a, that they represent the majority of the total restaurants taking by now almost all the app interactions.
With an eye at the figure 3.1 we choose to filter the data as to take into account the users that have a number of interactions $N : 10 \leq N \leq 1000$, to have a minimum quantity of information and at the same time to remove the outliers.
Actually our data does not have an explicit feature such as score, rate or *bought*. As aforementioned, we derive it from our existing features, for instance: we create a boolean feature called *Is_Positive*, it takes **True** as a value if the restaurant is in the user's favourites list (in all the different ways it is possible) or if the users *consumed* the item. Using the variable defined in section 3.2.2, the following first three actions say the user is saving the restaurant in his favourite list, the remaining three say he is booking, asking a delivery or calling a restaurant:

- **the event_name has value restaurant_favourite**;
- **the action_string_value is favourite_press**;
- **the dir_string_value is down**
- **the action_string_value is book_url**
- **the action_string_value is delivery**
- **the action_string_value is call**

And takes **False** in all the other cases or if the restaurant has been removed from favourites or blacklisted. From here on out, if a user has *Is_Positive*==**True**, we will simply say that he *likes* the restaurant.
As starting point, we selected users and restaurants with at least one *like*. Aggregating all the *likes*, we obtained a matrix where each row corresponds to a user, each column to a restaurant and it takes boolean values depending the user actually likes

the given restaurant. The Users-Items matrix looks like:

$$UI = \begin{pmatrix} 0 & \dots & 1 & \dots & 0 \\ \vdots & 0 & \dots & 0 & \vdots \\ 1 & 0 & \dots & 1 & 0 \\ \vdots & 0 & \dots & 0 & \vdots \\ 0 & \dots & 1 & \dots & 0 \end{pmatrix} \tag{4.1}$$

With our data UI matrix has a shape (3027×189), with a sparsity, evaluated as the ratio of number of total ones over number of total elements, of **0.048**.

4.2 The Metrics

Actually, with the explicit variable *like* we built, our Recommender has to solve a (positive/negative) binary classification problem. So, in order to evaluate the models, we choose three different metrics: **Accuracy**, **Precision@K** and **Recall@K**. To define them, we let $Rec_K(u) = \{r_{i_1}, .. , r_{j_K}\}$ be the set of the K recommended restaurants and $Real_N(u) = \{r_{i_1}, .. , r_{j_N}\}$ the restaurants that are in the test set and the user u actually likes:

$$Acc(u) = \frac{|Rec_N(u) \bigcap Real_N(u)|}{N} \tag{4.2}$$

$$P@K(u) = \frac{|Rec_K(u) \bigcap Real_N(u)|}{K} \tag{4.3}$$

$$R@K(u) = \frac{|Rec_K(u) \bigcap Real_N(u)|}{N} \tag{4.4}$$

We choose these metrics because they are enough reliable, with the precision we need for our purpose, and moreover they are easy to interpret and to discuss with the app developers.
To better understand why we choose this metrics, we anticipate here that will use $K = 5$. For users with at least 1 likes, using a test size of 0.1, we obtain for a sample test set the statistics in table 4.1. As the mean number of likes is 1 and the third quartile is 3, we choose to use the accuracy, defined in equation 4.2, as to have a more solid and easy to interpret metric. Indeed the accuracy defined above, is independent of the number of likes in the test set.

TABLE 4.1: Statistics for a sample test set with size 0.1

#like	2793.00
#users	1295.00
mean	2.16
std	1.84
min	1.00
max	15.00
50%	1.00
75%	3.00

4.3 The base models

The first two models we built are two models we will use as a baseline to compare recommenders with each other: a random recommender and a recommender based on ranking we called most popular recommender.

4.3.1 Random Recommender

The Random Recommender is the simplest recommender we can build. It takes as input a *user_id* u and it gives as output K restaurants randomly chosen from a set of $R(u) = \{r_1, \dots, r_n\} : UI(u, r_i) = 0$, i.e. r_i is a restaurant that the user u still does not like.

4.3.2 Most Popular K Recommender

The second base model we tested is the *Most Popular K*, another simple one, largely used during the cold start.
After having obtained the UI matrix (equation 4.1), for a user u we define $R(u) = \{r_1, \dots, r_n\} : UI(u, r_i) = 0$, as before the set of restaurants that the user u still does not like. We sort the restaurants in $R(u)$ for the number of likes, from the restaurants with more likes to the restaurants with less likes. In other words ordered by the quantity $\sum_u UI(u, r_i)$, obtaining the set $Sort\_R(u) = \{r_{1_i}, \dots, r_{n_j}\}$. In this way we define the set of the most popular K recommended to the user u, the first K restaurants of the set $Sort\_R(u)$: $Rec_K(u) = \{r_{1_i}, \dots, r_{K_j}\}$.

4.4 The Collaborative Filtering

In this section we will show in detail our Item based Collaborative Filtering Recommender, introduced in chapter one, following (Badrul Sarwar and Riedl, 2001). A Collaborative Filtering can be item based or user based depending the way we do the predictions. In the present work, we choose to use the item based model.
This kind of recommender is based on the sparse binary UI matrix of equation 4.1: each restaurant is described by a N dimensional sparse vector $\mathbf{r} = (0, \dots 0, 1, 0 \dots, 0)$, that actually is a column of the UI matrix. We try to suggest to u, K restaurants he still does not like, based on the choices of the other users. In order to do that we define a distance between two restaurants r_i and r_j, a function $d(r_i, r_j)$. In the binary case, the most used distance is the cosine distance:

$$d(r_i, r_j) = 1 - \frac{(r_i, r_j)}{|r_i||r_j|} \tag{4.5}$$

The smaller $d(r_i, r_j)$, the more similar should be the users' tastes on r_i and r_j. Alternatively we can define:

$$sim(r_i, r_j) = \frac{(r_i, r_j)}{|r_i||r_j|} \tag{4.6}$$

with $0 \leq sim(r_i, r_j) \leq 1$. $\frac{(r_i, r_j)}{|r_i||r_j|}$ is the cosine between the two vectors, a perfect similarity is reached when the two vectors are parallels.
Using the similarity of r_i with others restaurants we can assign a score to all the restaurants $r \in R(u)$ as $R(u)$ is defined in section 4.3.1:

$$score(u,r) = \frac{\sum_{r_i} sim(r_i,r) UI(u,r_i)}{\sum_{r_i} sim(r_i,r)} \tag{4.7}$$

We will then sort all the restaurants $r \in R(u)$ based on their scores $score(u,r)$, obtaining a ordered list of restaurants for u. The first K restaurants in the list are the restaurants we are recommending to u.

4.5 Content Based

We also tested a Content Based Recommender. With this recommender we are able to use the information contained in the dataframes described in chapter 3.2, the dataframe generated at the registration of the user and of the restaurants, with all the static features, like *price*, vibes, stars, etc. etc. In order to build it we will represent the restaurants in a different way than the one we used in the Collaborative Filtering. As we saw the restaurants in Velada are defined by a series of features, such as:

- `Price`: 6 different ranges of price;
- `Food Type`: 24 different food type categories;
- `Vibe`: 30 different kinds of vibe;
- `Stars`: from 0 to 3 stars Michelin;
- `Neighborhood`: 19 different urban areas.

The restaurant r is represented as a *One-Hot-Encode* of these five features, obtaining a vector $V_r = (0, \ldots, 1, \ldots, 0)$ where $V_r \in R^{N_{feat}}$ and N_{feat} is the total number of different values the five features can assume, in this case $N_{feat} = 83$. We then normalized each part of the vector corresponding to a particular feature to a ℓ_1 norm. For instance, if the indexes $[i : j]$ corresponds to the Vibe's values, then $\sum_{s=i}^{j} V_r^{(s)} = 1$. Finally the obtained vector is normalized in an ℓ_2 norm.
The main idea of a Content Based recommender is to represent users and items in the same space, as to evaluate distances between users and items. The way we did it is again using the UI matrix defined in equation 4.1.
We initialize an empty user vector for the user u in the same space of V_r, we then select the set of restaurants the user u liked, the set $Real_N(u) : r \in Real_N(u)$ if $UI(u,r) = 1$. We add one to each component $V_u^{(i)}$ each time a restaurant in $Real_N(u)$ has $V_r^{(i)} \neq 0$. Finally we normalize twice the vector V_u, in an ℓ_1 for each category slice, and in ℓ_2 the whole vector. In this way, we obtain a representation of the user comparable with items. That allows us to evaluate the distance of the user from all the other items: the nearest K restaurants are the restaurants we will recommend. Once again a *cosine* distance is used: $d(u,r) = 1 - \frac{(V_u, V_r)}{|V_u||V_r|}$.
Each user and each item is a sparse vector V, where each slice of the vector is a one-hot-encode of a given categorical variable. We can think to select features from the vectors, just slicing the part corresponding to the desired features. For example if the components from k to $k+6$ are the six components describing prices' ranges, the vector $V = (V_1, \ldots, V_{k-1}, V_{k+7}, \ldots)$ represents the given user or item without considering price as a feature.

4.6 Results

In this section we will describe and discuss all the experimental results we did, using the different recommenders we described under different assumptions and setups. Unless specified, as to compare the models each others we will use the same framework: we used the filtered data as explained in section 4.1, with a number of actions N, s.t. $10 \leq N \leq 1000$. Furthermore we took the decision to just consider users and restaurants with at least 5 likes as to have enough information for each user and for each restaurant. We will set the number of recommended restaurants as $K = 5$.

4.6.1 Experiment with base models

Under these assumptions we tested the base models. In order to do it, we built a dataframe with all the likes, all the user/restaurants couples that have *Is_Positive*==**True**, and we splitted it into train and test set with a test size of **0.1**. As to have more consistence of results we repeated the experiment for 5 different splits.
Under these assumptions we obtained for the Random Recommender the following means.

TABLE 4.2: Score for the base models.

	Accuracy	P@5	R@5	Sigma Accuracy
Random	.014976	.016159	.031632	.002089
Popular	.098884	.082661	.161096	.007132

The results in 4.2 are not surprising. Indeed if we see table 4.1, we can see that in general $K < 5$, so $P@5 < R@5$. As we will see in all the cases $P@5 < R@5$. Furthermore by measuring if $Accuracy \lessgtr P@5$, we can evaluate how much $|Rec_N(u) \bigcap Real_N(u)| \lessgtr |Rec_K(u) \bigcap Real_K(u)|$, i.e. how much count the order inside the list of the recommended K restaurants. As might be expected, the only case where it does not count is the random case.

4.6.2 Experiments with collaborative filtering

In this section we will describe all the experiments performed with the collaborative item based recommender under different experimental setups.

Rising the minimum number of likes

In this experiment we want to show that, rising the minimum amount of information we rise the performance of the model.
As a setup for the first experiment we used the filtered data saw in section 4.1, with a number of actions N, s.t. $10 \leq N \leq 1000$ and selecting users and restaurants with almost one *like*, for a total of 3027 Madrid users and 189 Madrid restaurants. As before, we split in train and test set, the set of all the couple (u, r) with `Is_Positive` $==$ **True**, with a test size of 0.1 and we repeated the split five times, obtaining the averaged quantities in table 4.3.

Over all the users in common between test set and train set, that are of the order of ~ 1230, we obtained almost the 10% of probability to recommend to a user a restaurant that he truly likes. In a way it is in his favourite list or he actually enjoyed the restaurant (he did a *call* or *delivery* or *book_url* action). The total number of like in

the test set, for the users that are also in the train set, is of the order of ~ 2700.
We then rise the minimum number of likes, considering just the users and restaurants with almost 5 likes (we are in the same set up of the base model experiment). We obtained the results in table 4.3.

TABLE 4.3: Score for the collaborative rising the minimum number of likes.

	Accuracy	P@5	R@5	Sigma Accuracy
Min_like 1	0.09634	0.0885	0.19127	0.00323
Min_like 5	0.11420	0.10683	0.21050	.004132

Rising the minimum number of likes, the accuracy and other metrics are improved considerably. The price to pay is that with this setup we are able to make predictions for less users and restaurants, for instance: the number of Madrid users with at least five like is 1450 and the number of Madrid restaurants liked by at least five users is is 187.

Adding the "neighbors" parameter

We obtained another improvement in the model by introducing a new parameter we called N_{neigh}. N_{neigh} is the number of neighbors (restaurants) we will consider, evaluating the score with the equation 4.7. For instance, evaluating $score(u, r)$, in the summation instead of summing over all the restaurants, we will sum just over the N_{neigh} restaurants more similar to r, basing the similarity on cosine distance (equation 4.5).
With the setup of the last experiment with $min\_like = 5$, adding the parameter $N\_neigh = 10$, we obtained another big improvement in accuracy, this time for free, without *sacrificing* users:

$$\overline{Acc} = 0.12907 \; ; \; \overline{P@K} = 0.11547 \; ; \; \overline{R@K} = 0.22402 \; ; \; \overline{\sigma_{acc}} = 0.004642$$

Time evolution

With two months of data, for our collaborative recommender, we reach an accuracy of $\sim 13\%$, much better of the base models, but not too encouraging. Our hypothesis is that it is due to a too small data set. To prove it we tested if rising the amount of data that we use for learning the recommenders, the results are improved.
Therefore we further investigated this model analyzing the change in accuracy with time, we evaluate the performance under the same conditions of the previous experiment, but taking the different data set through time, including new data every two weeks:

- **Week two**: The number of Madrid users after 2 weeks with at least five likes is 205. The number of likes is 3243;
- **Week four**: The number of Madrid users after 4 weeks with at least five likes is 532. The number of likes is 8496;
- **Week six**: The number of Madrid users after 6 weeks with at least five likes is 858. The number of likes is 13808;

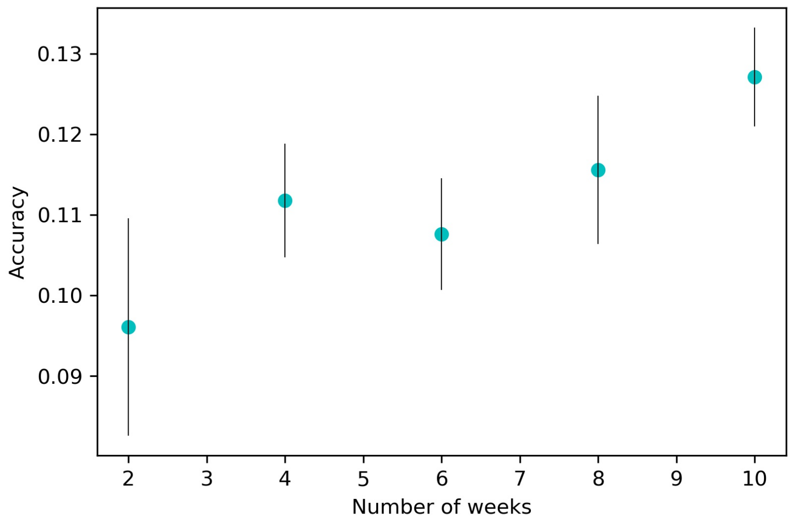


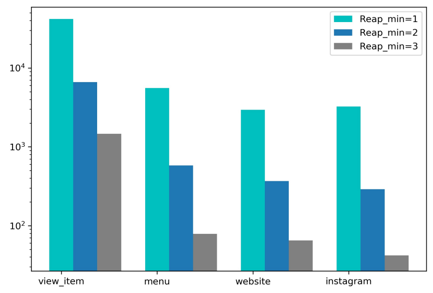
FIGURE 4.1: Change in accuracy adding data every two weeks.

- **Week eight**: The number of Madrid users after 8 weeks with at least five likes is 1347. The number of likes is 22855;
- **Week ten**: The number of Madrid users after 10 weeks with at least five likes is 1450. The number of likes is 24740;

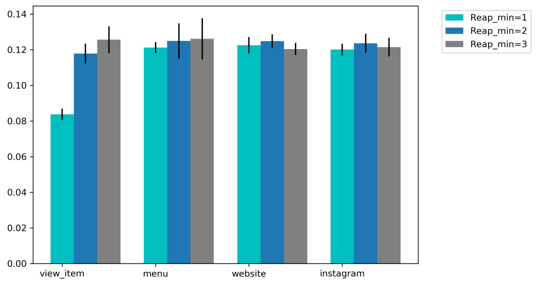
Figure 4.1 shows how, adding users and interactions to the model, the Collaborative Filtering is able to learn more about users' tastes, improving with time, with more data, the quality of our recommendations. This result supports our hypothesis: the currently available amount of data is too limited, and the results of this type of recommender are expected to improve as the app gets more interactions within time.

Trying to add features

We then tried to add to this model more information coming from the other actions of the users. In order to do that we perform the following experiment: we added other information to the train set, for instance a 1 in the UI train matrix, for every u that did a specific action on restaurant r. We did it for four different actions: `view_item`, `menu`, `website`, `instagram`. We added the parameter $Reap\_min$, i.e. the minimum quantity of time the user has to do the specific action to be counted (for example, with $Reap\_min = 3$ and `view_item`, we will count as 1 all the u that pushed the `view_item` button of restaurant r, 3 times or more). We performed a grid search with all the combinations of these four actions and $1 \leq Rep\_min \leq 3$, obtaining 45 different settings. Looking at figure 4.2b we can see that, with this data and this model, there is no evidence that adding other actions actually improves the accuracy. In all the setups the value of the accuracy is comparable with the one without extra features. It seems also that the `view_item` actions, at the present moment, acts more as a noise, becoming bigger the accuracy rising the $Reap_{min}$ parameter. Looking at 4.2a, it has to be noted that actually the number of users that did an action more than three times is very small (an order of magnitude $\leq O(10^2)$). All this means that this action not adds relevant information to the model; it is still not able to improve the normal binary case.



(A) The number of view_item, menu, website and instagram actions, for different values of Reap_min.



(B) The mean accuracy adding view_item, menu, website and instagram actions, for different values of Reap_min.

FIGURE 4.2: Exploring Collaborative Filtering Model adding features.

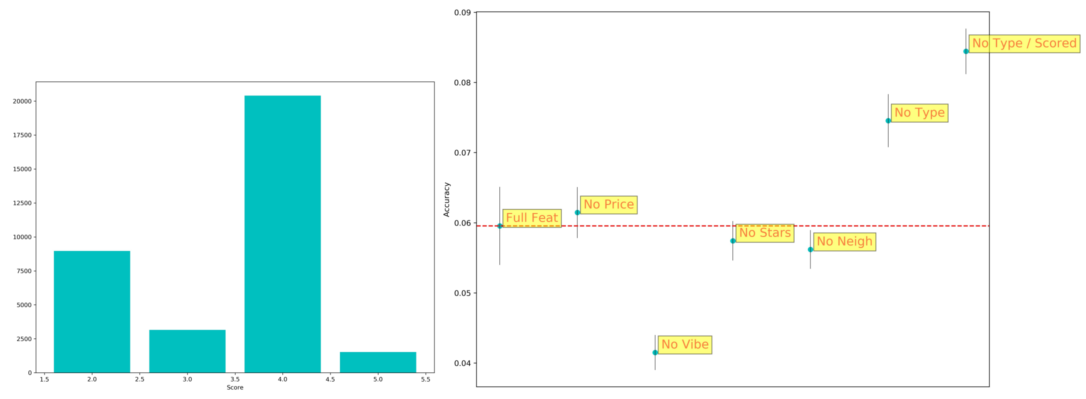
4.6.3 Experiments with Content Based models

As experiment with the Content Based Model, to take advantage of all the information we have, we tried to add the users' actions. In order to do this we used a different strategy to build the User-Item matrix.
First of all, we assigned a score to each action in a qualitative way:

- **One Point**: Card-Swipe-Left;
- **Two Points**: View-Items;
- **Three Points**: All the actions inside the restaurant page excepted the ones creating favourites or making an order/reservation;
- **Four Points**: All the actions putting the restaurant in the favourite list;
- **Five Points**: All the actions generating an order or a reservation.

It can be seen that we choose this kind of score basing on a qualitative reason; for instance, how deep goes the interest of the user for the restaurant (Skip it $\rightarrow$ Enter his page $\rightarrow$ More Actions on it $\rightarrow$ Favourite $\rightarrow$ Reserve). We then consider for each couple (u, r) all the set of actions user u did on restaurant r, we assign the score on each action, obtaining for each (u, r) a set of scores. We finally assign to (u, r) the max of the set.
We obtained in this way a different User-Item matrix, no more binary but with values included between zero and five (filling with zeros the $UI(u, r)$ that does not correspond to any actions). Using this UI matrix, when we build the user vector V_u, we will weight the user features based on the restaurants' scores and on quite different information.
To generate the same framework of the experiments in section 4.6.2, we drop a portion of $testsize = 0.1$ likes form the actions data frame, generating a test set of $Is\_Positive == True$ restaurants and a train set of actions. We then generate the UI with the scores, and we evaluated the K recommended as we did in the binary case. Figure 4.3b shows the accuracy obtained. We can see that it performs worse than the Collaborative Filtering. We can also see how it behaves if we drop one feature from the model. We can see that *Price*, *Stars* and *Neighborhood* do not affect considerably the result, otherwise *vibe* improve it and *food type* worsens it. In particular we can see that, adding actions information on the model improves the accuracy, indeed the best set up is the one without Food Type features and considering the scores in the

User-Item Matrix. This fact is a bit wondering and as to be further investigated. We can make the hypothesis that it is due to the lack of data or maybe, users in Velada, use to differentiate a lot the type of food, so much that it eventually becomes a noise.



(A) The number of times each score occurs for the whole data set of actions, neglecting the swipe left actions and removing the outliers.

(B) Change in accuracy dropping one feature for Content Based.

FIGURE 4.3: Content based accuracy and rates distribution.

4.7 Failed experiment: non-binary rates

Trials and errors took the major part of the effort of the present project. In this section we will briefly describe a failed line of improvement (no relevant results were reached) in which a lot of efforts were put.
We tried to build a standard Collaborative Filter Recommender as it is described in (David Goldberg and Terry, 1992), based on items' rates. As we do not have rates, we assigned to each (u, r) couple a 1 to 5 score based on the interactions they had. We did it in two different ways: first of all, we created an assignation rule rating each action, in a similar way we did for the Content Based model. Then we evaluated the set of actions for each couple (u, r) assigning as value NaN to all the couples without any interactions. The first approach was to sum all the values and then scale them to a 1 to 5 interval. The second approach was the same we used for the Content Based Model: just to take the maximum value.
After that, we were able to build a sparse matrix UI with all the rates and, rather than zeros, filled with NaN. With this matrix, we are able to make predictions using the same ideas we used for the Binary Collaborative Filtering, for instance:

$$pred(u,r) = \frac{\sum_{s \in N} sim(r,s) * (UI_{r,s})}{\sum_{s \in N} sim(s,r)}$$

Here N is the set of restaurants already rated by u and $sim(r,s)$ is a similarity function that has to be defined. We tried to use the most used similarities: one based on the euclidean distance:

$$sim(a,b) = 1 / \left(1 + \sqrt{\sum_{p \in P} (r_{a,p} - r_{b,p})^2} \right)$$

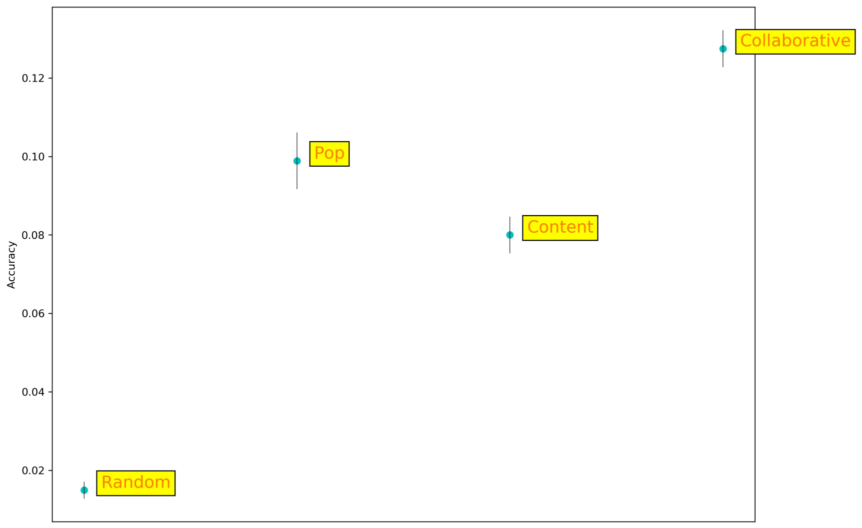


FIGURE 4.4: Comparing the accuracy of the four model: the random, the most popular, the content based evaluated with the scored UI and the Collaborative with binary UI.

and one based on Pearson Correlation:

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \bar{r_a})(r_{b,p} - \bar{r_b})}{\sqrt{\sum_{p \in P}(r_{a,p} - \bar{r_a})}\sqrt{\sum_{p \in P}(r_{b,p} - \bar{r_b})}}$$

We then create a grid of many different assignation rules with both the aforementioned ways (the one with the maximum and the one scaling the sums), running the model with the predictions using both similarities.
In all the cases, we reached much worse results in accuracy than the ones we obtained with the binary models. It seems that actually this way to rate the restaurants, works as a noise not revealing the real taste of the users. It is really difficult to tell if the failure to create a rating from different implicit actions is due to the lack of enough data or a wrong conceptualization.

4.8 Conclusion

Finally in figure 4.4 and in table 4.4 (here the traditional f1 score is the harmonic mean of precision and recall $f1 = \frac{2}{r@k^{-1}+p@k^{-1}}$) we compare all the models that we have presented in this section. As before we evaluated the model splitting a test set of size 0.1 from all the user-items couples that have $Is\_Positive == True$, filtering from the UI matrix only the users and restaurants that respectively gave at least 5 *likes* and received at least 5 *likes*. The Content Based Recommender is the best one between the ones in figure 4.3b, i.e. the one that uses the UI matrix with rates without considering the *food type* feature.
As we can see all the models, at the date of May2021, after just two months collecting data, perform much better than the "Random" one, with the "Popular" and the "Content Based" having similar accuracy and the Collaborative that actually, with $acc \sim \mathbf{0.13}$, is our best model.
The Collaborative, being more performing, it aims to suggest the right restaurant with a bigger probability than the Most Popular one. Furthermore is relevant to

note that the number of different restaurants suggested by the Most Popular model, applied to the whole data set is **22**, that is very small compared to the number of different restaurants suggested by the collaborative, that is **189**. This diversification guarantees to the app a better rotation of the suggested restaurants and a better experience to the user, suggesting him more unexpected restaurants, the so called serendipity.
Furthermore we tried to build a collaborative filtering recommender, based to a 1 to 5 rate, assigning scores to the actions. At present this strategy failed. The reason it happened has to be investigated more in detail. We guess it is due to the lack of data, in particular to the lack of data during time. Indeed the users' actions are concentrated in one single day and, rather than express users taste, they are the way they are exploring the app. It may add inconsistent information to our model.

TABLE 4.4: Final score for all the models.

	Accuracy	f1 Score	P@5	R@5	Sigma Accuracy
Random	.014976	.018997	.016159	.031632	.002089
Popular	.098884	.097291	.082661	.161096	.007132
Content	.080005	.079054	.065315	.138642	.004605
Collaborative	.129067	.138785	.115473	.224024	.004642

Chapter 5

Future implementations

5.1 Other models for the recommender system

We reached the best result with a binary collaborative filtering. In the absence of an explicit feature indicating user-item match, we generate it basing on qualitative arguments. At an app level, maybe an implementation that would facilitate future improvement in the developing of a recommender algorithm is the introduction in the app of an explicit score in the meaning of a five star scoring like the one existing in Netflix, Amazon or something similar.
At a model level, two main models should be implemented in the future, once we will have more solid users' behaviour and more solid explicit score: Matrix Factorization Models and Factorization Machine.

5.1.1 Matrix Factorization-based algorithms

A similar but slightly different approach to the ones used for the Collaborative Filtering is the SVD model. Following (Badrul M. Sarwar, 2000), we briefly introduce SVD model. The main idea is to embed users and items in a latent feature space, approximating the UI matrix by a lower rank matrix $\overline{UI}$ using SVD decomposition. The rank of the approximation is a parameter of the model and it represents the dimensionality of the latent space. Let's say it is K, with N users and M items:

$$\overline{UI} = PQ \text{ where } P \in R^{N\times K} \text{ and } Q \in R^{K\times M} \tag{5.1}$$

According to equation 5.1 each row of the P matrix is an embedding of an user and each column in Q is an embedding of an item in the same $K-dim$ space. Inferring the missing value in UI and obtaining its decomposition $\overline{UI}$, we can assign a score to each (u,i) couple:

$$\hat{r}_{u,i} = P_u Q_i^T \tag{5.2}$$

The hardest challenge of a matrix decomposition based model, is to impute the missing value to generate a dense matrix. The more recent works suggests modeling directly the observed ratings only, while avoiding overfitting through a regularized model. To learn the factor vectors P_u and Q_i, we can solve the minimization problem related to the regularized squared error on the set of known ratings:

$$min[\sum_{(u,i)=0}^{K}(r_{u,i}-\hat{r}_{u,i})^2 - \lambda(||P_u||_2 + ||Q_i||_2)] \tag{5.3}$$

The computation of equation 5.3 is possible using gradient descent algorithm such as the ones of the stocastic gradient descent family.

A benefit of the matrix factorization approach compared to the collaborative filtering is its flexibility in dealing with various data aspects and other application-specific requirements. For instance we can model equation 5.1.1 splitting it in four parts:

$$\hat{r}_{u,i} = \mu + b_u + b_i + P_u Q_i^T \tag{5.4}$$

Here μ is the overall average rating. The parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average. $P_u Q_i^T$ is the user-item interaction.
Another advantage of the SVD algorithm of particular interest for Velada is the possibility to include in additional sources of information about the users and the items.

5.1.2 Factorization Machine

Following the original paper (Rendle, 2010), we introduce here a model that we think fits very well with Velada data structure.
The starting point of the model is a different user-item representation. We introduce the sparse vector:

$$x = (0, \dots, 1, \dots, 0) \text{ where } x_i \in \mathrm{R}^{N+M+F} \tag{5.5}$$

where x represents a user-item interaction, including implicit information regarding the user and the item. Each slice of the vector carries a different kind of information: the first N components are a one-hot-encode of the users, the next M are a one-hot-encode of the items, the next F comes from a particular implicit feature. For example, if the feature is the number of Michelin Stars of the given restaurant, then it adds to the x vector $F = 4$ dimensions more, derived from a one-hot-encode of Michelin Star features (from zero to 3 stars). In line with this thinking we can add as many dimensions (features) as we want to the model.
The factorization machine model tries to infer the real score $r_{u,i}$ with a $\hat{r}_{u,i}$ that takes into account the second order interactions:

$$\hat{r}_{u,i} = f(x) = w_0 + \sum_{i}^{M+N+K} w_i x_i + \sum_{i,j} x_i x_j V_{ij} \tag{5.6}$$

where the weights w are to be learned training the model. If $Z = N + N + F$, are the aforementioned dimensions of x, $\hat{r}_{u,i}$ is given by a linear part and a non linear part. The linear part is made by $Z + 1$ parameters (the bias w_0 plus the Z components vector w). The non linear interactions are characterized by a $Z \times Z$ matrix $V_{i,j}$, giving a huge amount of parameters.
The core of the Factorization Machine is to reduce the dimensionality thanks to a factorization of the matrix V similar to the one performed for SVD. V is approximated by $V_{i,j} = (V_i, V_j) = \sum_{p=0}^{K} V_{ip} V_{jp}$, passing from $(Z \times Z)$ parameters to $K \times K$, where K is a parameter of the model.
Further simplifications can be made in order to improve the optimization problem related with the training of the weights. For instance, for the prediction (Eq. 5.6) we take advantage of the reformulation of the interaction part, that has only linear complexity $O(KZ)$, where K is the dimension of the latent space, and Z of the vector x:

$$\frac{1}{2} \sum_{f=1}^{K} [(\sum_{i=1}^{Z} V_{i,f} x_i)^2 - \sum_{i=1}^{Z} V_{i,f}^2 x_i^2)]$$

The components of the gradient in the Stocastic Gradient Descent, related to the quadratic loss function $L(y, \hat{y}) = [y - \hat{y}(\theta)]^2$ are $\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta}$.
Where:

$$\begin{cases} \frac{\partial \hat{y}}{\partial \theta} = 1 & \text{if } \theta = w_0 \\ \frac{\partial \hat{y}}{\partial \theta} = x_i & \text{if } \theta = w_i \\ \frac{\partial \hat{y}}{\partial \theta} = x_i [\sum_1^N V_{j,f} x_j - V_{i,f} x_i] & \text{if } \theta = V_{i,f} \end{cases} \tag{5.7}$$

Having a lot different possibilities, we think that in the future this model will permit a compact way to test all different scenarios of features engineering that Velada's data offer.

5.2 Adding the time component to the recommender system

Most recommendation algorithms do not explicitly take into account the temporal behavior and the recurrent activities of users. Two central but less explored questions are how to recommend the most desirable item at the right moment, and how to predict the next returning time of a user to a service.

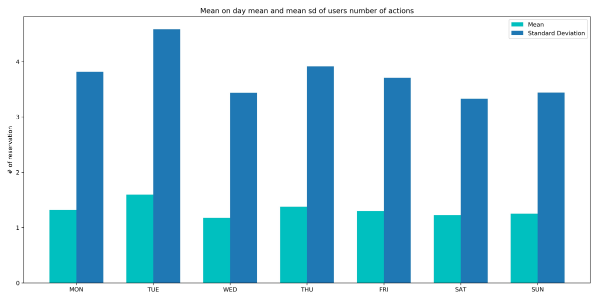
As we said, one of the initial purposes of this experimental project, was to build a recommender system able, not only to recommend the right restaurant but also, considering the behavior of users in time, to make the recommendation at the right moment. For this reason in this section we will give our proposal, trying to give an answer to this need. We anticipate here that after having analyzed the data, we decided to focus our work in the static part of the recommender, due to the lack of data and information useful to explore recurrency. Nevertheless we implemented a very simple model that, currently, has no way to be tested.

5.2.1 Introduction

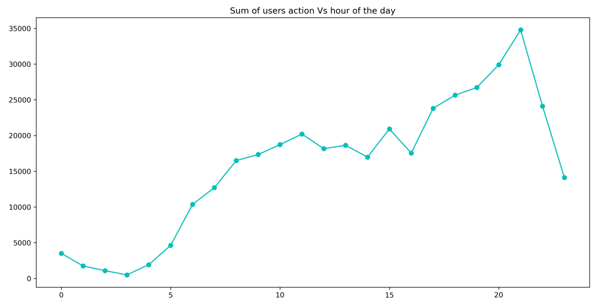
Periodic behavior is a kind of life behavior. The user arriving in a certain area periodically or consuming a certain product, is called a periodic behavior. For example, a user goes shopping every weekend or a user enjoys his favourite restaurant every two Fridays. Detecting these frequencies of periodic behaviour has a particular relevance for an app like Velada, as it aims to recommend the right restaurant at the right time. To do so, we would require user-related time series (more in general, time related information) of a certain length, that Velada, with its months of life, still does not have.
Nevertheless in this section we try to introduce the time dependencies and how to manage the recurrent visits to restaurants for our specific case.

5.2.2 Time Dependencies in Velada

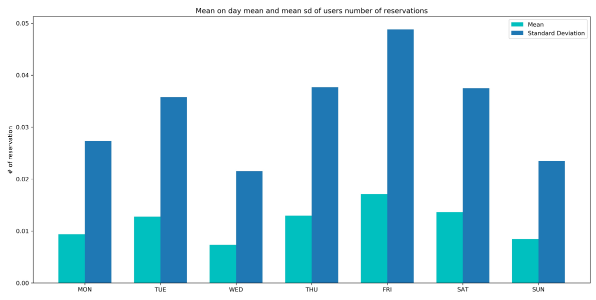
In order to try start implementing the *time* parameter inside our recommender system, as a first step we carried out some analysis, trying to display the time behaviour of users' actions and restaurants' reservations.
Figures 5.1 show the mean number of total actions for each day of the week, for each hour of the day and the mean number of total *book*, *call* and *delivery* for each day of



(A) Mean user actions each day of the week and standard deviation averaged over all users.



(B) Sum of users' actions each hour of the day



(C) Mean user reservations each day of the week and standard deviation averaged over all users.

FIGURE 5.1: Inspecting average users' activity during the days of the week and during the hours of the day.

the week. We can see that the numbers are still very smalls (and, surprisingly, Tuesday is the day with the bigger activity). The big variance reveals what we saw in 3.6a and 3.6b: the user activity in this phase of the app is predominantly concentrated in a single day of usage. For example, the day with more reservations is Friday, with a total number of reservations equal to 243. Of those 18 users made a reservation two different Fridays and 3 users did it three different Fridays. It is also to be noted that the event that the same restaurant has been reserved by the same user is still very rare (it happened just 99 times). Under these premises it is maybe too early to build some implementations for both the problems we mentioned above: is very hard to detect the best moment of the recommendation in terms of days, indeed is impossible to detect the day of major activity of a given user. Is it also impossible to test any kinds of implementation approaching the recurrency problem. Indeed there are no information about recurrent choices of restaurants.
Anyway we suggest an implementation we coded about the recurrency problem that can be easly tested in the future.
Given a user u, rather than recommend him five restaurants he still does not like, we also consider the last restaurants that he reserved. Let's say we built a recommender that suggests K restaurants $Rec_K = r_1, \ldots, r_K$, without dropping from Rec_K restaurants u already liked. In order to suggest a single restaurant to u, we initialize a probability vector $p = (1/K, \ldots, 1/K)$ for a multinomial distribution $Mult(p)$. Let $\mu = \overline{\Delta t}$, the mean distance in time between two consecutive reservations of the same restaurant r_i. We then modeled the probability p_i in a linear way s.t., if t_0 is the last reservation day $p = \lambda(t)\frac{1}{K}$:

$$\lambda(t) = \begin{cases} 1 & \text{if } t > t_0 + \mu \\ \frac{1}{\mu}(t - t_0) & \text{if } t_0 \leq t \leq (\mu + t_0) \end{cases} \tag{5.8}$$

In this way, if we are near the day user u reserved the restaurant r_i, this restaurant has small probability to be suggested. If we are at a distance near or bigger to μ from the last reservation, the probability returns to be the same of the other, unseen, restaurants.

5.2.3 Related literature

Looking for some reference article, the most hard thing, was to find the kind of item that can be compared to restaurants. Most of the models we found relates to short term recurrency with long length time series, like songs or page clicks.
We would suggest to follow the ideas of (Bing Xu and Chen, 2017) proposed a simple recommender for *LBSN* (Location-Based Social Network), a new kind of social network which combines the user's friendship and the user's position. This model wants to make recommendations based on geo-localization, finding recurrency in users' displacements. In real life, some users go to a certain area, within multiple periods. For example, a user goes to an area not only every Friday but also every two weeks on Monday. This paper, proposed a period acquisition algorithm which can mine multiple periods in time sequence correctly and efficiently. This could be applied to Velada, in a way that, as we saw in equation 5.8, rather than finding a single μ, we should find a set $(\mu_1, \ldots, \mu_n)$ of periods. What we are looking for are the dominant frequencies of a Fourier Transformation of the time series of the users' actions, the periods related to the bigger coefficients of the Fourier Series.

We think that for its semplicity and immediacy, these ideas could be truly implemented in a not too far future (two or three years) for the most loyal users. Following (Bing Xu and Chen, 2017): let's say $S_u(t_i)$ a binary series of user u related to a target event (in our case for example *Is_Positive*) and t_i a generic timestamp event. Let's say $L = t$ the set of instants where $S_u(t_i) = 1$. For the sake of clarity, if time sequence is $S(t_i) = 0001100101010011$, the target state is 1, the time stamps at which the target state happened are $\{3, 4, 7, 9, 11, 14, 15\}$. With these instants indexes we can create a list of candidates periods, i.e. all the differences between indexes:

$$T = \{sp|sp = (t_i - t_j) \text{ if } \quad t_i < t_j\} \tag{5.9}$$

in our case of $S(t_i)$, $T = \{3, 4, 7, 9, 11, 15\}$ and each $sp \in T$ has different cardinality, i.e. there are different $(t_i - t_j)$ that give the same sp (in oure case $sp = 4$ is the most common and happens three times). Then, once we fixed a threshold l, we will find the true periods $(\mu_1, \ldots, \mu_n)$, using the support:

$$SUP_i(t) = h_i \times \frac{sp_i}{\Delta t_0} \tag{5.10}$$

Here h_i is the cardinality of sp_i, Δt_0 is the distance in time between t and the first t_i in which the target event happened. So fixed a threshold l, if $l \leq SUP_i(t)$ we can finally store $sp_i = \mu_i$ in our list of periods $(\mu_1, \ldots, \mu_n)$, having multiple learned time interval to recommend restaurant r which the series $S_u(t_i)$ is related to.
This model needs even more data than the linear one but, rather than learn one frequency it learns multiple periods, giving a more sophisticated solution of the recurrency problem. The importance of the learned frequencies is evident and it can be used in several ways. We might even learn the different frequencies, not just for a single user but for a user-item couple. In this way we should easly implement the recommendation in a simple and deterministic way: suggesting the recurrent restaurant when is its period and an unseen restaurant in any other moment.

Chapter 6

Conclusion

6.1 Conclusion

In the present project we described in detail how we built a recommender for the new restaurant app Velada. We tried to give attention not just on the restaurant suggested, but also on the moment of the suggestion, a recommender able to suggest the right restaurant at the right moment. After having analyzed the data, we understood it is too early to investigate this second aspect, and we focused our effort on the static part of the recommender.
The data provided by the App developers, that come from two different sources: one coming from the app itself, with implicit features describing static qualities like *price ranges* or *neighborhoods*, and the other one from *Google Analytics* describing all the actions of the users inside the app during two months. Initializing, preprocessing and analyzing them took a big part, in terms of time, of the whole project.
In the data provided there is no explicit information about the user-item preferences. We tried to create it, mainly in two different ways, generating a one-to-five score or generating a binary (*thumb up*) rate.
We built two main recommenders: a *Content Based* model and a *Collaborative Filter* model. We tested them in many different setups, comparing them with two base models: a *Random Recommender* and a ranking-based one we called *Most Popular*. In all the settings our models outperform the Random Recommender, in particular the *Content Based* is a bit less accurate than the *Most Popular* and the *Collaborative Filter* using a binary rate is actually our best model with a 13% of accuracy score. Nevertheless, we checked that rising the volume of the data, we improved the accuracy of our model, suggesting that one of the bigger problems of the model is the lack of data. In particular we detected that most of the users interacted with the app just the first day. It means that, more than learning users' tastes, the model is learning how the users are making their first exploration of the app.
How to use all the implicit information, was one of the biggest challenges: we tried to create an assignation rule, giving a rate to each couple user-restaurant based on their interactions. It is not clear at all if this strategy works for the Collaborative Filter. Instead, we obtained good improvements for the Content Based.
Although one of the initial goals was to include the temporal behavior and the recurrent activities of users in the recommender system, the short amount of data available made it unfeasible. We propose several basic ideas that could be tested as a first step in this direction.
As future work, there are many questions open after this project. First of all, we would suggest Velada's developers to include a more clear, explicit, feature about user-item preferences (such as one-to-five stars or a simple like) and/or a more clear way to verify if a user actually went to a reserved restaurant. Regarding the model,

in Chapter 5 we propose different future implementations that could help to improve the recommender system, both regarding the recommender type (with models that particularly fit we the Velada's data) and the incorporation of the time component. It would be also interesting to repeat our experiments with a much larger dataset to test if some of our failed approaches were in fact promising ideas tested without enough data.

Bibliography

Badrul M. Sarwar George Karypis, Joseph A. Konstan John T. Riedl (2000). "Application of Dimensionality Reduction in Recommender System – A Case Study". In: *ACM Press* Proc. KDD Workshop on Web Mining for e-Commerce: Challenges and Opportunities (WebKDD). DOI: http://robotics.stanford.edu/~ronnyk/WEBKDD2000/papers/sarwar.pdf.

Badrul Sarwar George Karypis, Joseph Konstan and John Riedl (2001). "Item-Based Collaborative Filtering Recommendation Algorithms". In: *Accepted for publication at the WWW10 Conference.*

Bing Xu, Zhijun Ding and Hongzhong Chen (2017). "Recommending Locations Based on Users' Periodic Behaviors". In: *ieeexplore.ieee.org* Volume 2017.Article ID 7871502. DOI: https://doi.org/10.1155/2017/7871502.

David Goldberg David Nichols, Brian Oki and Douglas Terry (1992). "Using Collaborative Filtering to Weave an information tapestry". In: *Communications of the ACM* Volume 35.Issue 12. DOI: http://www.bitsavers.org/pdf/xerox/parc/techReports/CSL-92-10_Using_Collaborative_Filtering_to_Weave_an_Information_Tapestry.pdf.

Nicolas Jones, Pearl Pu (2007). "User Technology Adoption Issues in Recommender Systems". In: pp. –.

Pasquale Lops, Marco de Gemmis and Giovanni Semeraro (2011). *Recommender systems handbook*. Springer Science.

Rendle, Steffen (2010). "Factorization Machines". In: *ieeexplore.ieee.org* IEEE International Conference on Data Mining. DOI: https://www.csie.ntu.edu.tw/~b97053/paper/Rendle2010FM.pdf.

Solache, Sara. *El nuevo Tinder, en versión gastronómica, se llama Velada*. URL: https://www.lavanguardia.com/comer/al-dia/20210421/6988159/nuevo-tinder-version-gastronomica-app-restaurantes-velada.html.

Yehuda Koren Robert Bell, Chris Volinsky. *Matrix factorization techniques for recommender systems*. URL: https://www.data-science-repo/Recommender-Systems-[Netflix].pdf.