UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Reccommender for restaurants

Author:
Pablo GRANATIERO

Supervisor:
Dr. Jerónimo HERNÁNDEZ GONZÁLEZ
and
David Martin SUAREZ
(app owner)

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

June 19, 2021

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Reccommender for restaurants

by Pablo GRANATIERO

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

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 day-by-day (think to *Netflix*, *Amazon*, *Youtube*, *ebay*) and, for this reason, one of the key element 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. A good personalized recommendations can add another dimension to the user experience.
So **Recommender Systems** is actually an important topic in Data Science and data scientist never stop developing new **Recommender Algorithms**, with a prolific production in scientific literature.

1.2 What is a Recommender

A *Recommender* 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 way 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 distinguish for making the same *recommendation* to everyone. The simplest Non Personalized Recommenders are the *K* Most Popular Items in the home page, for example top ten movies in Spain (the first ten movies ranked for number of views in Spain) that even if very simple and generalistic is still one of the most important Recommender in platform like *Netflix* or *Filmin*.
On the other way Personalized Recommender are Recommender that try to guess a specific user's taste inferring from his past behaviour. This kind of Reccommender has several advantages from the Non Personalized one, because it aims to give a more personalized suggestion, increasing the variety of recommended items and the user's serendipity. Again there are several way 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 and in the third and last section of the chapter we introduce a third kind of Recommender, derived from the Collaborative ones, called model based.

1.2.1 Content Based

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 use to go to japanese and chinese restaurants we will assign to him a high score to the food type feature *asian*).
Finally 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 filtering is the *Music Genome Project*, which is used for the Internet radio service *Pandora.com* using more then 400 features.

1.2.2 Collaborative

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 author *Collaborative Filtering*. These models don't 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 try 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 is investigated inside all the users preferences, where items that likes to more or less the same users, are more or less similar.
For example, consider a Japanese Restaurant. Its neighbors might include war other Japanese restaurants, Korean restaurants and other expansive 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 the *Model Based* mostly based in **Matrix Factorization** techniques. These model 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 embeds users and items in latent space which 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 Kitchen versus Italian Kitchen, amount of price, or neighborhood. But, depending from the number of dimensions, it can interprets more hard to find, sometimes uninterpretable, aspects.
In the recent past more sophisticated and complex model, like 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 Recommender for Restaurants, in particular for the brand new Spanish app **Velada**. In the next chapter we will describe the app main features, then we will describe the data we used to build the Recommenders and, finally, we will show the results we obtained. The last chapter is about future implementations regarding aspects related to time dependencies.

Chapter 2

Velada

2.1 The App

Released on March 2021, Velada is an app for *android* and *apple* devices. It borned 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*.

Actually 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 multiple, 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 owner *David Martin Suarez* 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 rule. Indeed, restaurants, contrary to a movie or a music track has a very different degree of recurrency. As we will see, maybe the app is in a too early stage to develop a time recommender, that needs clear users' behaviour paths through time, in a range of time longer than the actual one.

2.1.1 Mobile Flow

In this see 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 the 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 presetted set of filters (detecting a particular need like *With Michelin Stars, With delivery, ...*).

Pushing on one of the *home page* button, 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 became 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 is 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 type of food like *Italian* or *Japanese*) and the neighborhood

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 action 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 see in detail the data we used in order to try to build a Recommender system for Velada. In this section we anticipate that *David Martin Suarez* provided us, user 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.

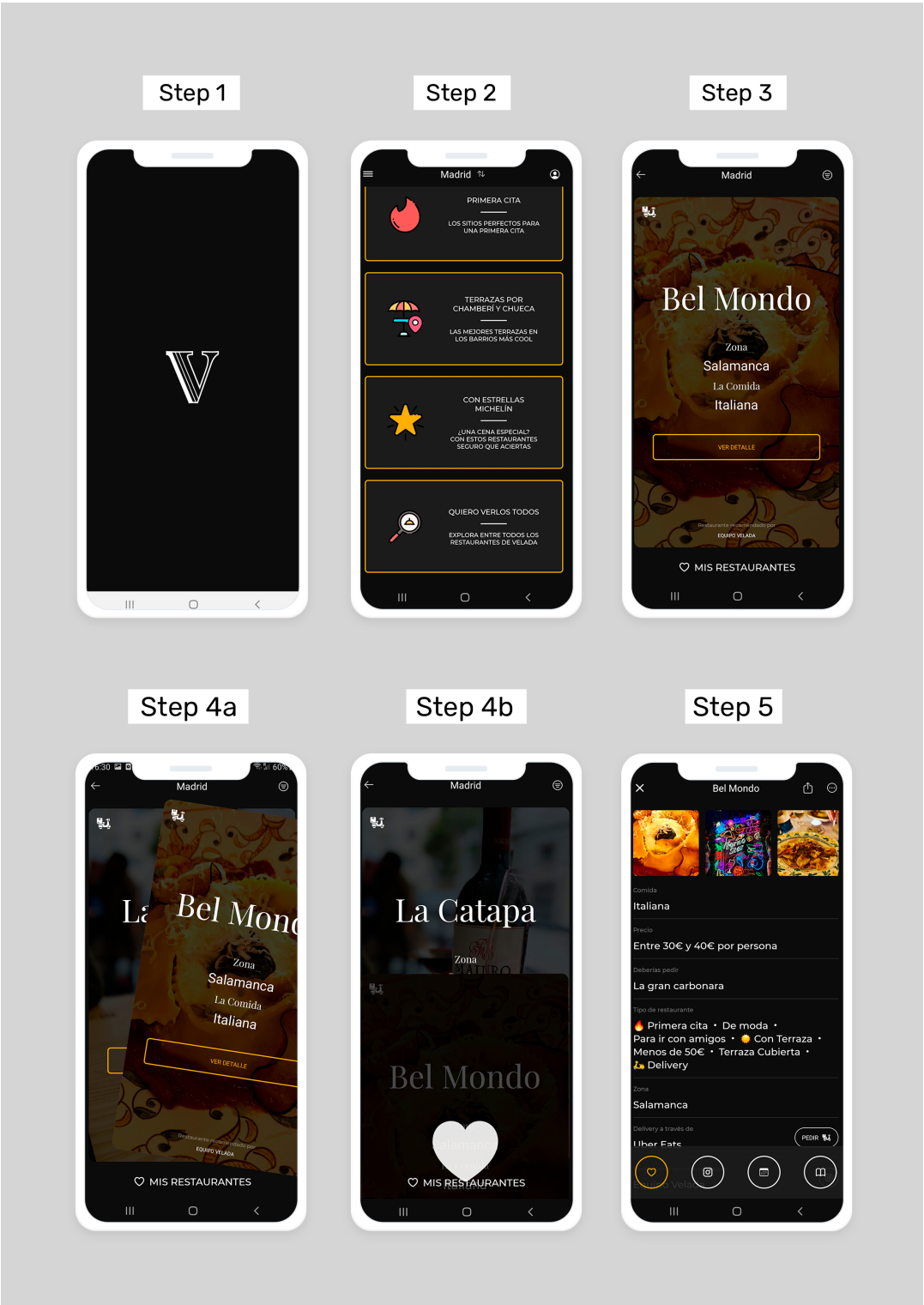


FIGURE 2.1: The basic user journey inside Velada app

.

Chapter 3

The Data

3.1 The data sources

As we said in the previous section 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* are 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 oriented Recommender, like *price*, *food-type*, or the aforementioned *Vibes* filters. We will see it more in detail in the *Data Analysis* paragraph.

The data downloaded from the app *Google* account, are 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 users 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* paragraph. These kind of information are by the way useful for a content based Recommender but, with some feature engineering identifying a target variable (i.e. a variable thanks to which we can establish if and how much a user approve an item) are also relevant to build a collaborative one.

In the next two section 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*, and provided by the app owner, are in `http://www.github.fake_link_data_app_1.com` and in `http://www.github.other_fake_link_data_app_1.com`.

3.2 Initializing raw data

In `http://www.github.fake_link_data_app_1.com` there is the `TFM_Granatiero_Init_data.py` file. A program that takes as input the folders of the

two different data (the ones from the app and the ones from *Google Analytics* account) and gives as output several *pandas* dataframe.

Preprocessing in real life is a background, hard to see, job that takes a lot of time and effort. This is what exactly happened in this case too. The more demanding readers can find in the Github repository the program to convert raw data in ready to use data frame.

As we said we have two source of information, the ones from the app and the ones from *Google*.

3.2.1 Data from the app

Let's see more in detail the first one: the info at the registration time, are stored in several tables, in particular we will use the data contained in the following data frame:

- `restaurants_Datos_Init`: in this data frame are collected all the main information about the restaurants, for instance:
  - `name`: the name of the restaurant.
  - `restaurant_id`: the identification number of the restaurant.
  - `neighborhoods`: the city area where is located 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. Actually Madrid and Barcelona are the active cities, but, with a sight at the future, are already registered restaurants from other cities, like Lima, Paris, Milan, London.
- `restaurants_vibes_Datos_Init`: as we saw, filtering, is one of the main Velada's tool and *Vibes* filters are the more distinctive. In this data frame 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_Datos_Init`: 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-type`. For a total of 24 different food-type (*Asian*, *Galician*, *Italian*, ...)
- `cities_neighborhoods_Datos_Init`: the same as before but with cities, areas. This data frame defines a $1 - to - 1$ mapping between a neighborhood and a restaurant.

These kind of categorical information are useful to build Recommender belonging to the ones described in section 1.2.1, models content based. We built this Recommender and we will describe it in deepness in the next chapter.

3.2.2 Data from Google Analyitics

The app owner provided us the data from the *Google Analytics* app account. After some preprocessing we obtained a data frame 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`: as we said one of the more iconic user interaction of Velada is swipe. This feature said the user swiped. Figure 2.1 step 4*a* and 4*b*.
  - `view_item`: the user entered in a restaurant page. Figure 2.1 step 5.
  - `RESTAURANT_FAVOURITE`: the user saves a restaurant into his favourite. One of the action the user can perform in a restaurant page of figure 2.1 step 5.
  - `RESTAURANT_ACTION`: the user did one of the action_string_value actions. Again actions described above, allowed in a specific restaurant page (figure 2.1 step 5).
  - `HOME_FILTERS`: there are several ways to add filters, one of these are in the homepage. This feature said 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. This feature said he did.
  - `MY_FAVS_FILTER_ADDED`: as restaurants, exist 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 a 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 vibe the user selected adding a filter.
- `dir_string_value`: how the user swiped.

> `DOWN`: the user saved the restaurant in his favourite list.

> `LEFT`: the user skipped the restaurant.

The first thing that comes to mind looking at these data with the objective to build a Recommender is the absence of explicit features, like *thumbs up*, *thumbs down* or a classic *5 stars scores*. We will need this kind of information in order to build a Recommender of the Collaborative family. As we will see in the next chapter we tried to derive explicit information in some indirect way from these data.
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

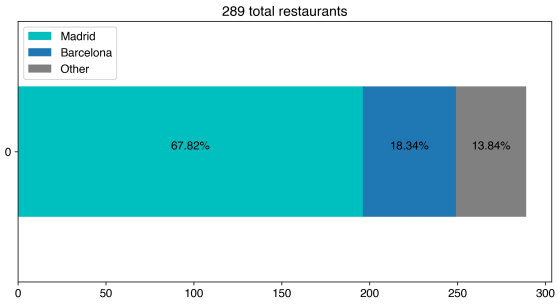
With data described in the previous chapter we can perform a lot of different interesting and insightful plots. For the sake of clarity in this chapter we will enlight all those aspect that aim to understand more in deepness how the Recommenders we code work and why we took some decision.

3.3.1 The cities

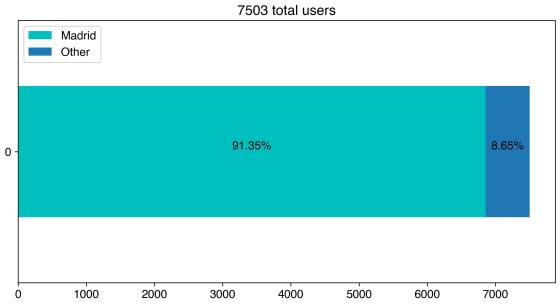
One of the first decision we agreed with the app owner, has been to limit our model to Madrid Restaurant. Indeed if we see at figure 3.1a, at the date of 30/04/2021 **the total number of restaurants is 289**, where the **67.82**% of them are from Madrid and the rest are from Barcelona or from other cities. Of these **196** restaurants from Madrid **189** are active, in the meaning that users made with them unless one interaction.
The fact that the majority of the activity is from Madrid is even more clear if we look at 3.1b, 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.
In this initial phase of the app the data from Barcelona are still poor, and we will restrict all our work on 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 app. First of all in figure 3.2 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 in a while 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 don't 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 see the presence of outlier, probably due to the owners testing the app.



(A) All the restaurants in Velada



(B) All the users in Velada

FIGURE 3.1

3.3.3 Actions outside the restaurants pages

As we saw in chapter 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 can perform just inside or outside restaurants' pages. Here we analyze all the actions they can do outside.
In 3.3a we can see a bar plot of all of them. As we already said the most common action is by far *card swipe*: swipe is the most entertaining action and mimic Tinder with restaurant has been a good choice. Even if, as we will see, translate this action in information useful for a Recommender is not easy. From 3.3b it's shown that $\sim$ **91**% of all *swipe* are *swipe left*, suggesting that skipping restaurant is something equivalent to *zapping* in *Netflix*. Always in figure 3.3a we can see that *view item* is, with almost an order of magnitude less than *card swipe* the second common action. And is not surprising, as after some swipe, the most common thing to do is to take more information about a given restaurant, entering its page by pushing *view item* button. In the next page we see the most common action inside a restaurant page.

3.3.4 Actions inside the restaurants pages

Inside of a restaurant's page there are 10 different actions, of those, save the restaurant in the favourite list is allowed in many different ways outside and inside the page. The remaining actions are exclusive of this area of the app.
In 3.4 we can see that, the most common actions are the *menu* button and the *favourite* button.
The fact that the favourite action is popular is a good new for a Recommender, as we tried to translate it as an explicit positive feedback. As we can see, even 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 go to the restaurant by the user.

3.3.5 Likes

As anticipated before one of the key element of a *Collaborative Filtering* Model is the presence of explicit score, 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*.
In figure 3.5a and 3.5b are shown that total number of *likes* for any user and for any restaurant: the two curve show the typical long tail shape, where some restaurants are more popular then other and some user is more *generous* in *likes* then others.

3.3.6 Time dependencies

One of the main objective of the app owner was to build a Recommender with a time aspect. For this reason we analyze how long and how often the users interact with the app, indeed considering time require some path during time in the users' behaviour. So 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 3.6a we analyze how long the users use the app, for this reason we took 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 big majority of the users interacts for just one day and, by one order of magnitude down we arrive at 5 days ($O(10^2)$). In 3.6b we try to understand how often they use the app, for this reason we plot for each user the cardinality of the different day of activity set how to understand how many different days they use the app. 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.

These numbers suggest us that maybe is too early to build a Recommender with a time dependency, indeed our Recommender more than learn 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 in the next chapter, with these premises we obtained some result which leaves us hopeful for the future.

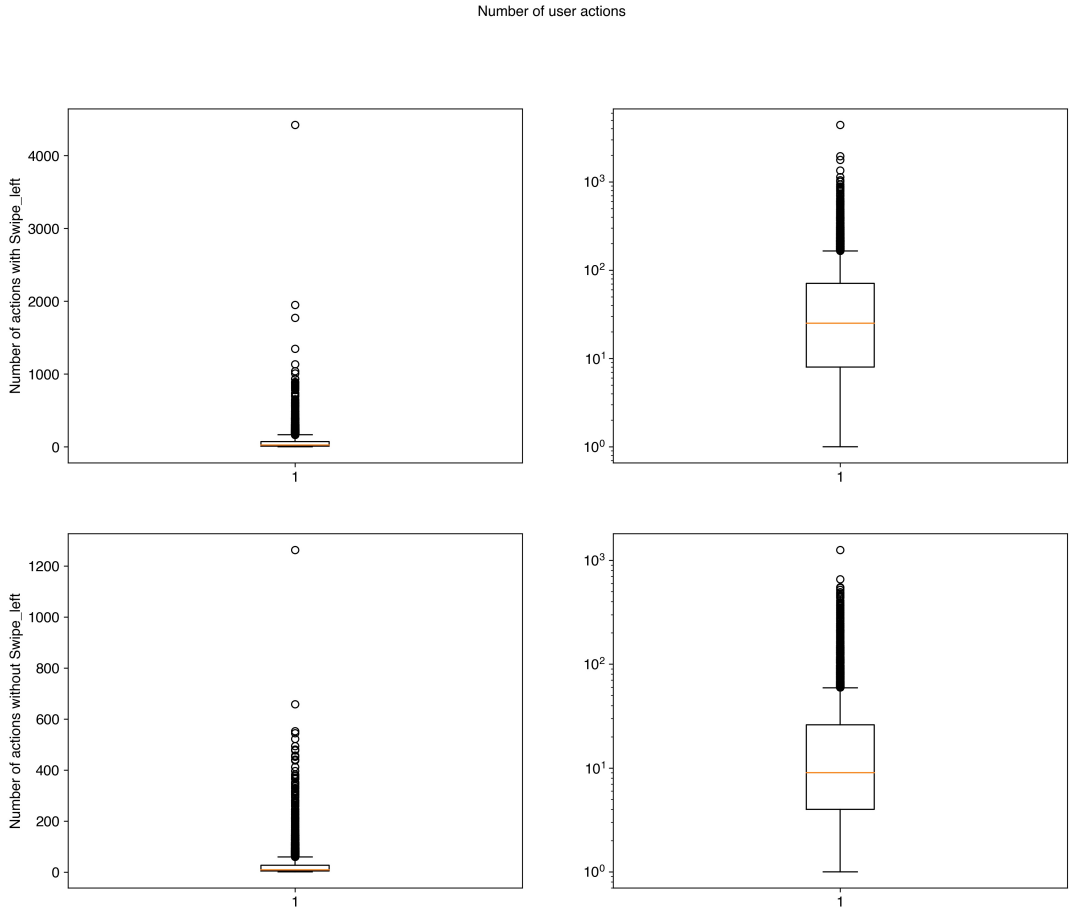
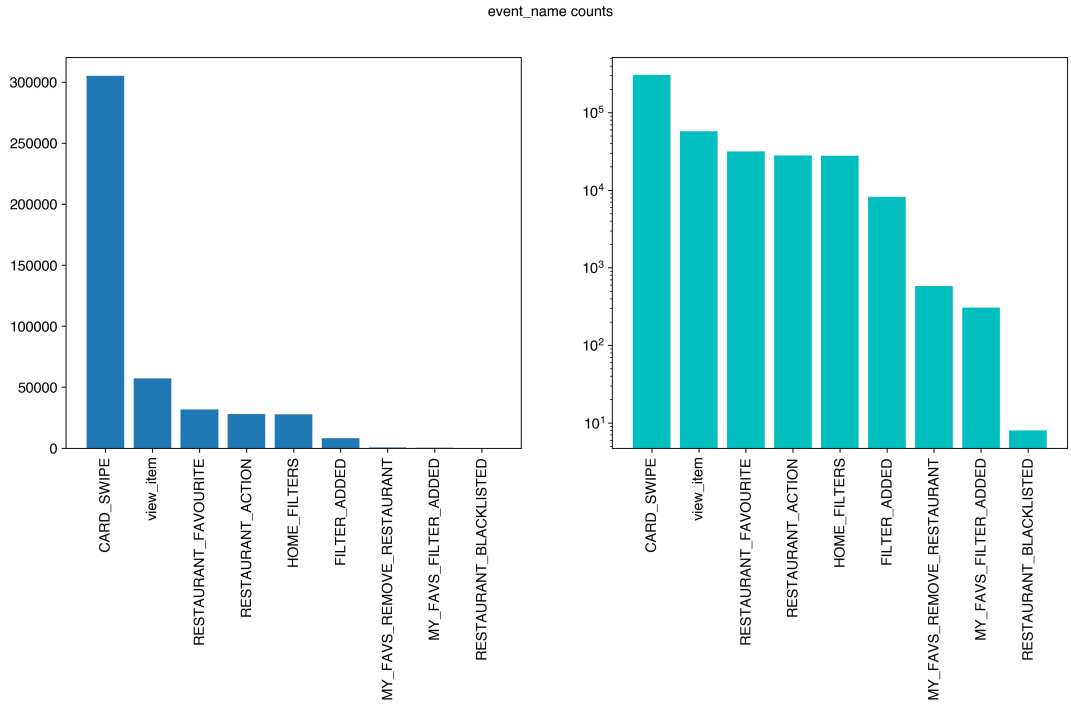
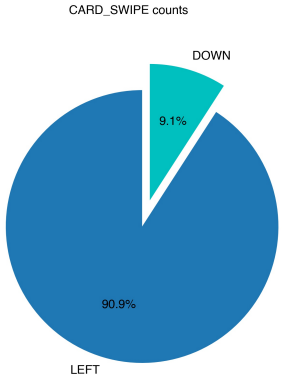


FIGURE 3.2: A box plot of the total number of users' interaction with the app. On the right in logarithmic scale and on the bottom without considering the Swipe Left action.



(A) 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.

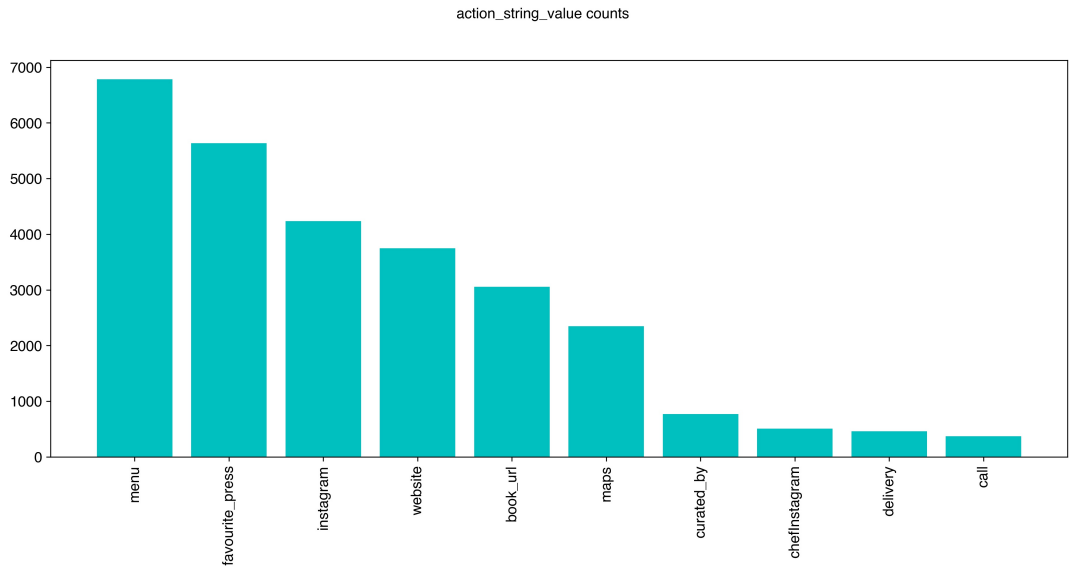
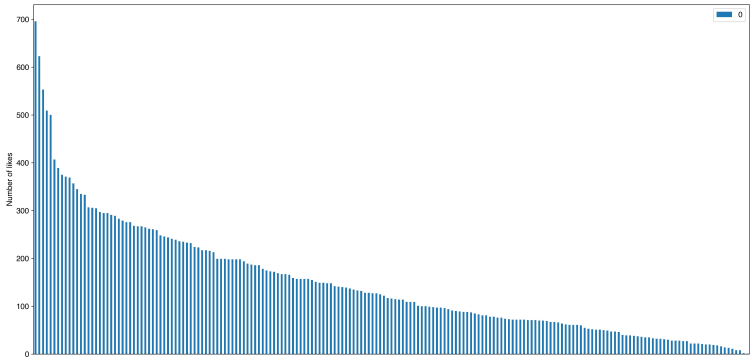
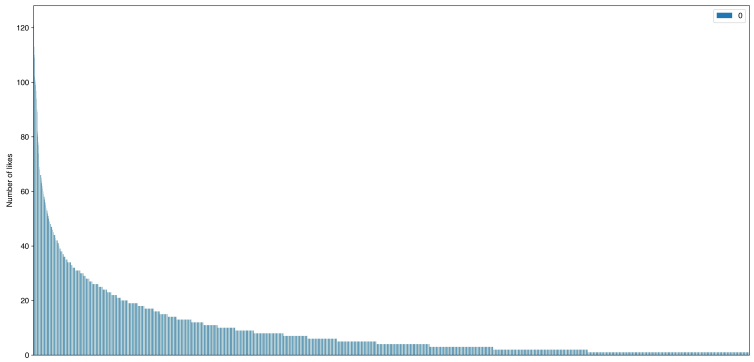


FIGURE 3.4: The total number of interactions the users can do inside a restaurant page.

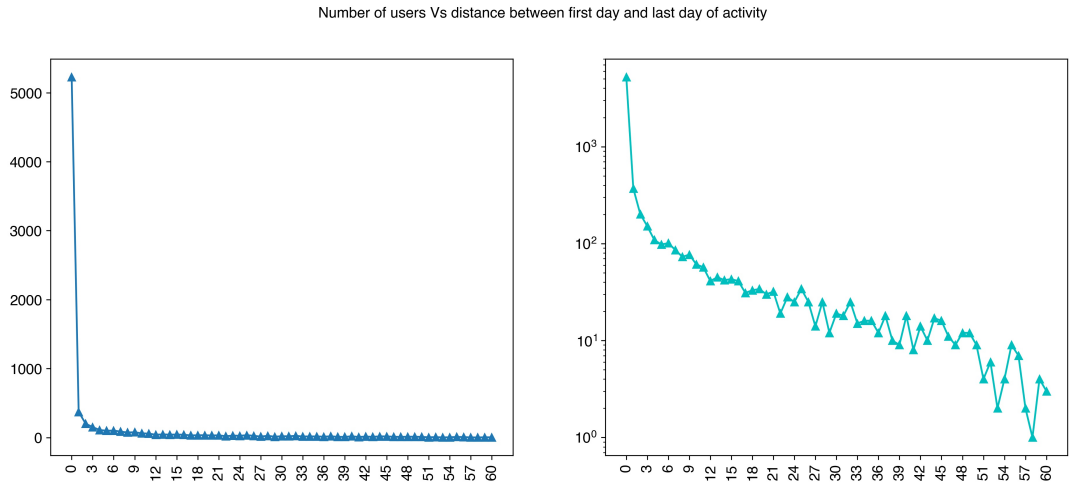


(A) The number of like for each restaurant.

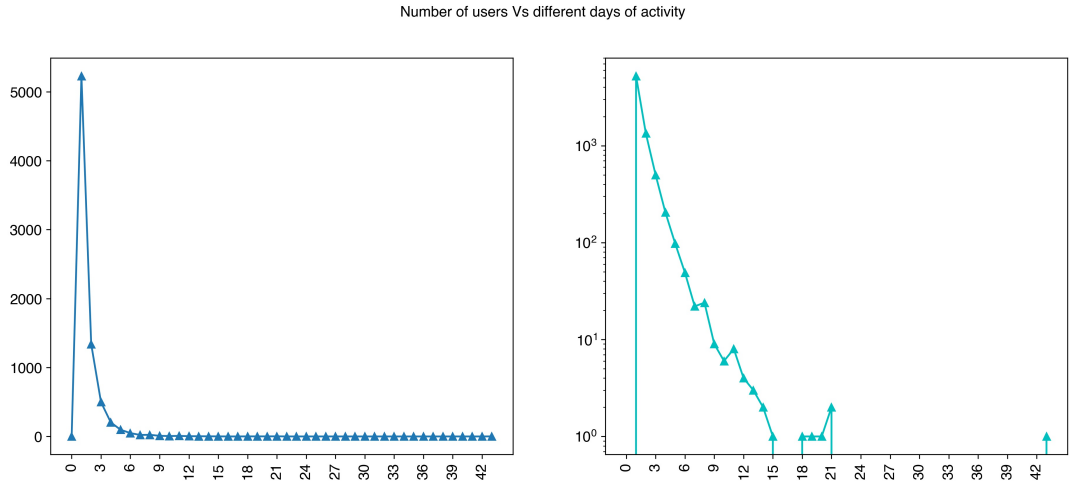


(B) TThe number of like for each user.

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



(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.

Chapter 4

The Recommenders

In this section we will describe in detail the Recommenders we built. In the first section we will see the pre processing and the decision 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. Finally we will show the Recommenders we built and results we obtained with them. A last section is about future implementations and possible developments.

4.1 Preprocessing

As we introduced in the last chapter, the first decision we took is to restrict our data to the Madrid restaurants, indeed we saw in 3.1a they represent the majority of the total restaurants taking by now almost all the app interactions.
With an eye at the figure 3.2 we choose to filter the data as to take into account the users that have a number of interactions $N : 10 \leq N \leq 1000$, as to have a minimum quantity of information and at the same time removing the outliers.
Actually our data doesn't have an explicit feature such as score, rate or *bought*, we already said that we tried to derive it from our existing feature, for instance: we creates a boolean feature called *Is_Positive*, it takes **True** as a value if the restaurant is in the user's favourite list, i.e.

```
if (event_name == RESTAURANT_FAVOURITE or
action_string_value == favourite_press or
dir_string_value==DOWN) and
(event_name != MY_FAVS_REMOVE_RESTAURANT or
event_name != RESTAURANT_BLACKLISTED))
```

it takes **True** even if the users consumed the item, i.e.

```
if (action_string_value == book_url or
action_string_value == delivery or
action_string_value==call)
```

And takes **False** in all the other cases.

From here on out, if a user has *Is_Positive*==**True**, we will simply say that he likes the restaurant.
As starting point, filtering again our data, we select 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 i takes boolean values depending the user actually likes the give restaurant. The Users-Items matrix:

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

in our case UI 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**. From this we can extract meaningful statistics for the users:

```
count = 3027.00
mean = 9.23
std = 13.31
min = 1.00
25% = 2.00
50% = 4.00
75% = 11.00
```

4.2 The Metrics

In order to evaluate the models we choose three different metrics: **Accuracy**, **Precision@K** and **Recall@K**. To define them we say $Rec_K(u) = \{r_{i_1}, \dots, r_{j_K}\}$ the set of the K recommended restaurant and $Real_N(u) = \{r_{i_1}, \dots, r_{j_N}\}$ 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 comment with the app owner.

4.3 The base models

The first two models we built are two models we will use as a parameter to compare recommenders with each other: a random recommender and a 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 choosed 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 not likes. To test this Recommender 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**.
We anticipate here the conditions in which mostly of our experiments will be performed: we set $K = 5$, we took the decision to further filter our data, considering users and restaurants with at least 5 likes as to have enough information, we used the aforementioned metrics, meaning these metrics over all the users, performing the experiment over 5 different train-test splits and meaning over them. Under these assumptions we obtained for the Random Recommender:

$$\overline{Acc} = 0.01493 \tag{4.5}$$
$$\overline{P@K} = 0.01672 \tag{4.6}$$
$$\overline{R@K} = 0.03137 \tag{4.7}$$

4.3.2 Most Popular K Recommender

The second base model we tested is the *Most Popular K*, another simple one, largely used in the cold start.
After having obtained the same UI matrix 4.1, for a given user u we define again $R(u) = \{r_1, \dots, r_n\} : UI(u, r_i) = 0$ and we sorts it in a way that the first restaurant in R_u is the one that received more like, the last is the one that received less. I.e. 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 defined the set of the Most Popular K suhhested for the user u, $Rec_K(u) = \{r_{1_i}, \dots, r_{K_j}\}$.
Under the same test conditions of section 4.3.1, we obtained:

$$\overline{Acc} = 0.10110 \tag{4.8}$$
$$\overline{P@K} = 0.08281 \tag{4.9}$$
$$\overline{R@K} = 0.17081 \tag{4.10}$$

4.4 The Collaborative Filtering

4.4.1 Binary Collaborative Filtering

In this section we will show in detail our Collaborative Filtering Recommender, introduced in chapter one. They can be item based or user based depending the way we do the prediction. In the present work we choose to use the item based model.
This kind of Recommender is based on the sparse binary matrix 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 restaurant he still not liked, 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, that let us define the degree of similarity of two restaurants:

$$sim(r_i, r_j) = \frac{(r_i, r_j)}{|r_i||r_j|} \tag{4.11}$$

the bigger $sim(r_i, r_j)$ the more similar should be the users' tastes on r_i and r_j, with $0 \leq sim(r_i, r_j) \leq 1$.

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 4.3.1:

$$score(u,r) = \frac{\sum_{r_i} sim(r_i,r)UI(u,r_i)}{\sum_{r_i} sim(r_i,r)} \tag{4.12}$$

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

4.4.2 Experiment 0

As a setup for the first model 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. As to have more consistence of results we repeated the split five times, obtaining these averaged quantities:

$$\overline{Acc} = 0.09634 \tag{4.13}$$
$$\overline{P@K} = 0.0885 \tag{4.14}$$
$$\overline{R@K} = 0.19127 \tag{4.15}$$

Over all the $0.1\dot{N}$ couples we obtained almost the 10% of probability to recommend to a user a restaurant that he truly like, in a way it is in his favourite list or he actually try it.

4.4.3 Experiment 1: more filters

With this second experiment we can see that filtering again the data, neglecting the users with few information, we improve our results. Using just the users and restaurants with almost 5 likes as we did for the Random and for the Most Popular, under the same train-test conditions of the previous experiment we obtained:

$$\overline{Acc} = 0.11420 \tag{4.16}$$
$$\overline{P@K} = 0.10683 \tag{4.17}$$
$$\overline{R@K} = 0.21050 \tag{4.18}$$

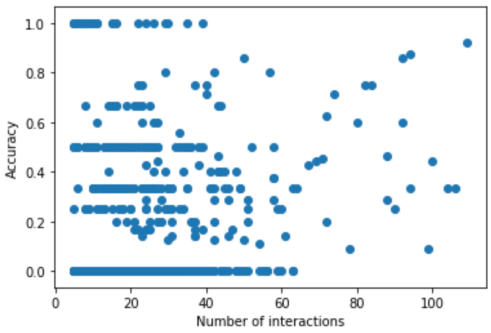
We improve a lot our accuracy, the price to pay is that with this setup we are able to make predictions for less users and restaurants, for instance: The number of madrid users with almost one like are 1450 and the number of madrid restaurants with almost five like are 187.

4.4.4 Experiment 2: adding the parameter N_{neigh}

We obtained another improvement in the model, introducing a new parameter N_{neigh}. N_{neigh} is the number of neighbors we will consider, evaluating the score with the

	accuracy	p@K	r@K
10	0.129073	0.115472	0.22402
20	0.11948	0.110331	0.211469
30	0.117991	0.108725	0.215721
40	0.118476	0.109914	0.216132
50	0.114764	0.106511	0.207582
60	0.114473	0.10655	0.200903
70	0.116923	0.105479	0.203334
80	0.115455	0.108218	0.206183
90	0.1121	0.104415	0.208768
100	0.116194	0.106477	0.210769
110	0.109679	0.104649	0.20031
120	0.110166	0.106096	0.20939
130	0.112392	0.103267	0.199294
140	0.116652	0.108558	0.211707
150	0.118867	0.108258	0.208159
160	0.116592	0.107534	0.209741

(A) Errors decreasing with N_neigh.



(B) On the x axis the number of likes in the train set, on the y axis the accuracy of the given user.

FIGURE 4.1: Results of experiment 2.

equation 4.12. 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 similarity on 4.11.

In figure 4.1a is shown the inverse relationship between the accuracy and N_neigh. In particular, with the setup of the Experiment 1, adding the parameter $N\_neigh = 10$, we obtained another big improvement in accuracy, this time for free, without sacrificing users:

$$\overline{Acc} = 0.13000 \tag{4.19}$$

$$\overline{P@K} = 0.11547 \tag{4.20}$$

$$\overline{R@K} = 0.22402 \tag{4.21}$$

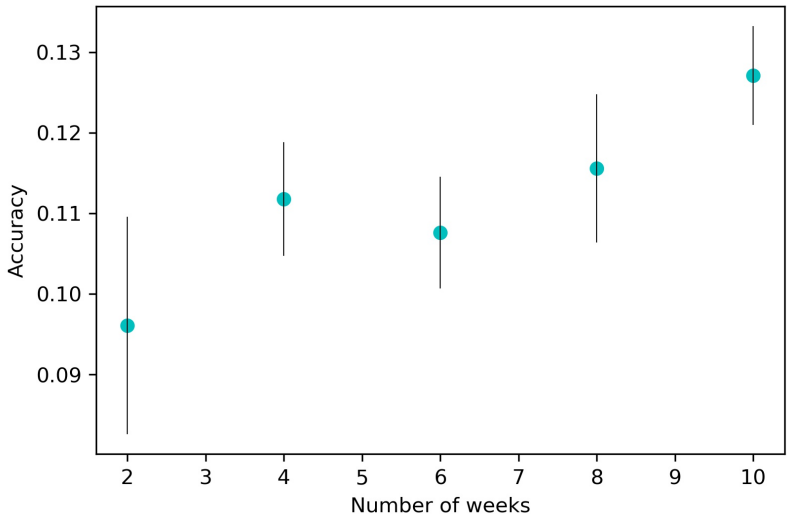


FIGURE 4.2: Change in accuracy adding data every two weeks.

FIGURE 4.3

4.4.5 Experiment 3: time evolution

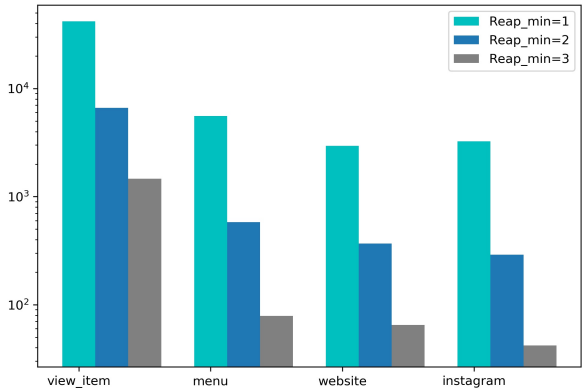
We further investigated binary collaborative filtering analyzing the change in accuracy with time, as to evaluate if, evolving with time, i.e. raising the number of users, we reach to improve our model. In order to do that we evaluate the accuracy under the same condition of the last experiment 4.21, but taking the different data set through time, every two weeks:

- **Week two**: The number of Madrid users after 2 weeks with almost five like is 205. The number of likes is 3243;
- **Week four**: The number of Madrid users after 4 weeks with almost five like is 532. The number of likes is 8496;
- **Week six**: The number of Madrid users after 6 weeks with almost five like is 858. The number of likes is 13808;
- **Week eight**: The number of Madrid users after 8 weeks with almost five like is 1347. The number of likes is 22855;
- **Week ten**: The number of Madrid users after 10 weeks with almost five like is 1450. The number of likes is 24740;

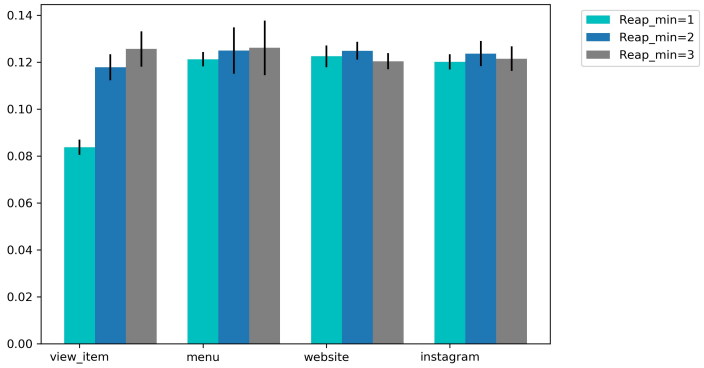
Finally in figure 4.2 is shown how, adding users and interactions to the model, the Collaborative Filtering is able to learn more about users tastes, improving each two week the quality of our recommendation.

4.4.6 Experiment 4: 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: with the same condition of experiment 4.21, 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



(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 4.4

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 action and $1 \leq Rep\_min \leq 3$, obtaining 45 different settings. Looking at figure 4.4b we can see that, with these data and this model, there is no evidence that adding other actions actally improves the accuracy. In all the setup the value of the accuracy is comparable with the one without features. It appears 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 4.4a, it has to be noted that actually the number of users that did an actions more than three time is of the order of magnitude $\leq O(10^2)$, carrying not much more information than the normal binary case.

4.5 Content Based

The next Recommender we tested is a Content Based recommender. With this recommender we are able to use the information contained in the dataframes described in chapter 3.2. In order 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, for instance:

- `Price`: 6 different range of price;
- `Food Type`: 24 different food type categories;
- `Vibe`: 30 different kind 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 of 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 normilize twice the vector V_u, in an ℓ_1 for each category slice, and in ℓ_2 the whole vector. In this way we obtained a representation of the user comparable with the items and we can evaluate the distance of the user from all the other items: the nearest K restaurants are the restaurants we will recommend. Once again the distance is a *cosine* distance $d(u, r) = 1 - \frac{(V_u, V_r)}{|V_u||V_r|}$.
We can activate or deactivate features, selecting from the vectors, just the slice corresponding to the desired features.

4.5.1 Content Based Experiment

As experiment with the Content Based, as to try to takes advantage of all the information we have, we tried to add, the information of the users' actions. In order to do that 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 Point**: View-Items;
- **Three Point**: All the actions inside the restaurant page excepted the ones creating favourites or making an order/reservation;

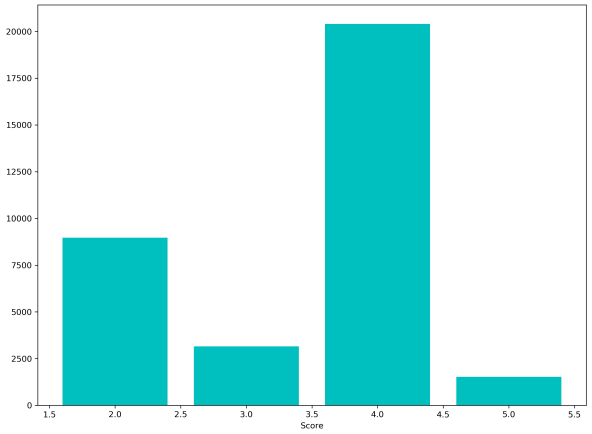
- **Four Point**: All the actions putting the restaurant in the favourite list;
- **Five Point**: All the actions generating an order or a reservation.

As we can see we choose this kind of score basing on qualitative reasons, for instance how in deepness goes the interest of the user for the restaurant (Skip it → Enter his page → More Actions on it → Favourite → Reserve).
We then consider for each couple (u, r) all the set of actions user u did on restaurant r, we assign a score on each action as we said, 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 zeros the $UI(u, r)$ that does not correspond to any actions) and neglecting Swipe Left. 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 experiment in 4.4.4, 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.

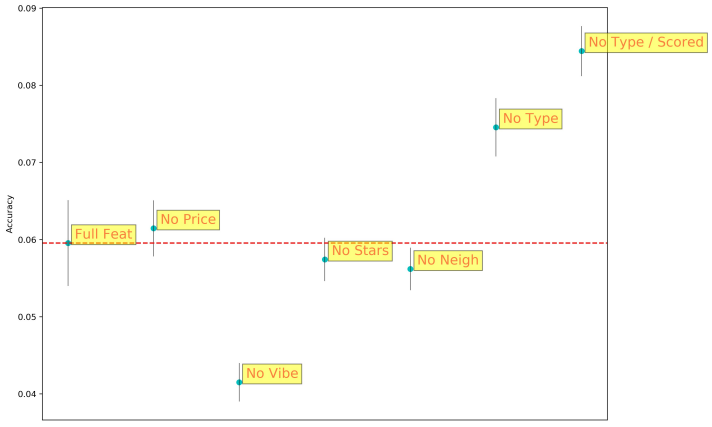
In figure 4.5b is shown the accuracy obtained, we can see that it performs worse than the Collaborative Filtering. We can also see how it behaves if we drop one feature from the model. We can see that *Price*, *Stars* and *Neighborhood* do not affect considerably the result, otherwise *vibe* improve it and *food type* worsens it. In particular we can see that, adding actions information on the model add improve the accuracy, indeed the best set up is the one without Food Type features and considering the scores in the User-Item Matrix.

4.6 Conclusion

Finally in figure 4.6 and in table 4.1 (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 we saw. 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 almost 5 *likes* and received almost 5 *likes*.
The Content Based Recommender is the best one of 4.5b, i.e. the one that uses the UI matrix with scores without considering the *food type* feature.
As we can see all the models, at the date of *May*2021, after just two months collecting data, perform much better than the "Random" one, with the "Popular" and the "Content Based" having very similar accuracy and the "Collaborative" that actually, with $acc \sim \mathbf{0.13}$, is our best model in therm of accuracy and $f1$ score.
We conclude with the important information that the "Collaborative" besides being more performing, it aims also a better serendipity and restaurant diversification, indeed the Popular model, applied on the entire filtered dataset, recommends **22** different restaurants on the total of **189** active restaurants in Madrid, on the other side, the "Collaborative" recommends **166** different restaurants.



(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 4.5

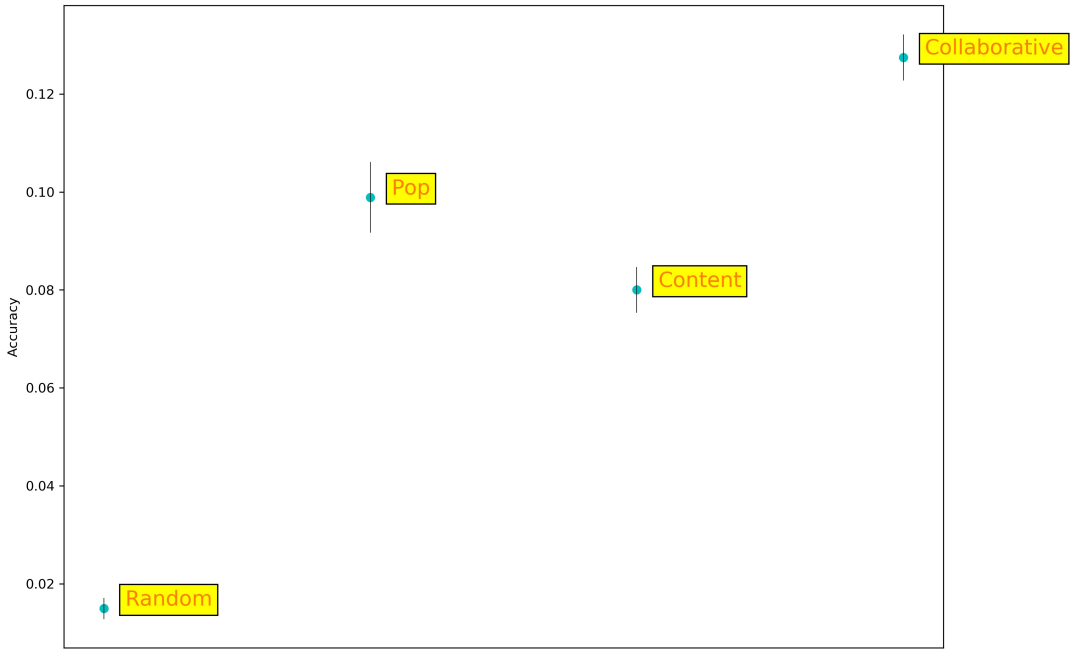


FIGURE 4.6: Comparing the accuracy of the four model: the random, the most popular, the content based evaluated with the scored UI and the Collaborative with binary UI.

FIGURE 4.7

TABLE 4.1: 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	.127453	.138785	.117733	.227204	.004642

4.7 Future Implementation

We got the best result with a binary collaborative filtering i.e., in the absence of an explicit feature, 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 or Amazon.
At a model level, two main models has to be implemented in the future, once we will have more solid users' behaviour and more solid explicit score: Matrix Factorization Models and Factorization Machine.

4.7.1 Matrix Factorization-based algorithms

A similar but slightly different approach by the ones used for the Collaborative Filtering is the SVD model. Following *MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS* and Badrul M. Sarwar, 2000, we briefly introduce SVD model. The main idea is to embedding users and items in a latent features 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 represent th dimensionality of the latent space. Let's say is K, with N users and M items:

$$\overline{UI} = PQ \text{ where } P \in R^{N \times K} \text{ and } Q \in R^{K \times M} \tag{4.22}$$

According to equation 4.22 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:

$$\hat{r}_{u,i} = P_u Q_i^T \tag{4.23}$$

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

$$min[\sum_{(u,i)=0}^{K} (r_{u,i} - \hat{r}_{u,i})^2 - \lambda(||P_u||_2 + ||Q_i||_2)] \tag{4.24}$$

The computation of 4.24 is possible using gradient descent algorithm such as the ones of the stocastic gradient descent family.
One benefit of the matrix factorization approach to collaborative filtering is its flexibility in dealing with various data aspects and other application-specific requirements. For instance we can model equation splitting it in four parts:

$$\hat{r}_{u,i} = \mu + b_u + b_i + P_u Q_i^T \tag{4.25}$$

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. And often they takes most of the value. $P_u Q_i^T$ is the user item interaction.
Another implementation 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.

4.7.2 Factorization Machine

Following the original paper Rendle, 2010, we introduce here a model that we think fits very well with Velada data structure and we think it has to be tested in the future. The starting point of the model is a different user items representation, for this purpose We introduce the sparse vector:

$$x = (0, \ ... \ , 1, \ ... \ , 0) \text{ where } x_i \in \mathrm{R}^{N+M+F} \tag{4.26}$$

Here x rather then represents a user or an item, represent a (user, item) interaction, including implicit information regarding the user and the item. Each slice of the vector carry 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 are the Michelin Star of the given restaurant, then it adds to the x vector $F = 4$ dimension 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 dimension (features) as We want to the model. The Factorization Machine model the score $\hat{r_{u,i}}$ with a second order interaction:

$$\hat{r_{u,i}} = f(x) = w_0 + \sum_{i}^{M+N+K} w_i x_i + \sum_{i,j} x_i x_j V_{ij} \tag{4.27}$$

where the weights w are to be learned by the model. For instance, if $Z = N + N + F$, the linear part is made by $Z + 1$ parameters, 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 of V approximating it 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 simplify the optimization problem related with the training of the weights. For instance, For the predict part 4.27 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:

$$\begin{cases} \frac{\partial \hat{y}}{\partial \theta} = 1 & \text{if } \theta = w_0 \\ \frac{\partial \hat{y}}{\partial \theta} = x_i & \text{if } \theta = w_i \\ \frac{\partial \hat{y}}{\partial \theta} = x_i [\sum_{1}^{N} V_{j,f} x_j - V_{i,f} x_i] & \text{if } \theta = V_{i,f} \end{cases} \tag{4.28}$$

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.

Chapter 5

Time Recommender

5.1 Time Recommender

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.

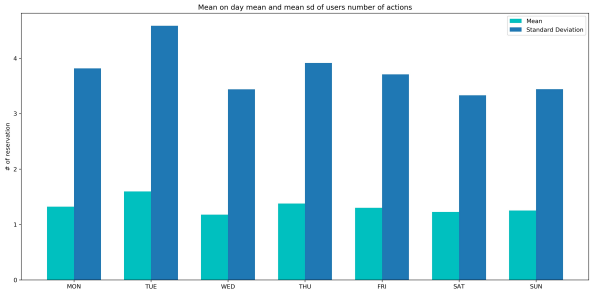
5.1.1 Introduction

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 reaching his favourite restaurant every two Friday. Detecting these frequencies of periodic behaviour has a particular relevance for an app such Velada, it aims to recommend the right restaurant at the right time. Doing this needs users' Time Series (more in general, time related information) of a certain length, that Velada, with its two months of life still doesn't have. Nevertheless in this section we try to introduce the time dependencies and how to menage the recurrent restaurants for our specific case.

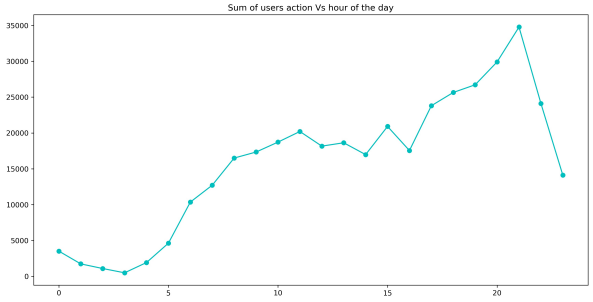
5.1.2 Time Dependencies in Velada

In order to try start implementing the *time* parameter inside our Recommender as a first thing we did some data analysis, analyzing time behaviour of users' actions and restaurants' reservations.

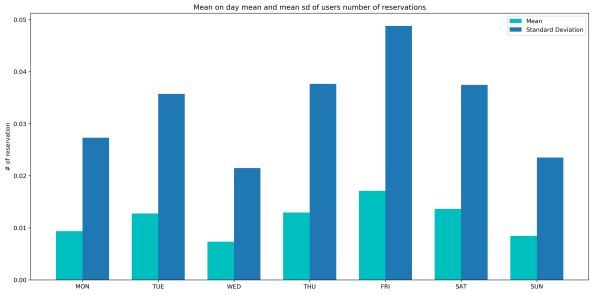
As we can see in figure 5.1 and as we saw in 3.6a and 3.6b, the user activity in this phase of the app is predominantly concentrated in a single day of usage. For example the day with more reservations is Friday, with a total number of reservations equal to 243. Of those 18 users made a reservation two 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 very still rare, for instance is happened just 99 times. Under these premise is maybe too early to built a time implementation. Anyway in this section we suggest an implementation we coded:
given a user u, rather than recommend him five restaurants he still doesn't like, we will recommend him five restaurants based on the last restaurants he reserved. For instance:
let's say we built a recommender that suggest 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's say $\mu = \overline{\Delta t}$ the mean distance in time between two consecutive reservation of the same restaurant r_i. We then modeled the probability



(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 week



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

FIGURE 5.1

p_i in a linear way s.t., if t_0 is the last reservation day $p = p(t) = \lambda(t)\frac{1}{K}$:

$$\lambda(t) = \begin{cases} 1 & \text{if } t > t_0 + \mu \\ \frac{1}{\mu}(t - t_0) & \text{if } t_0 \leq t \leq (\mu + t_0) \end{cases} \tag{5.1}$$

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 Future Implementation

Looking for some reference article in the web, the most hard thing, was to find the kind of item that can be compared to restaurants. Most of the models we found was related to short term recurrency with long length time series, like music tracks or page clicks.
In this section we will suggest as a future implementation the algorithm in Bing Xu and Chen, 2017: in this article is 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. There are two hot research hotspots in the filed of LBSN: recommendation system and mining user's life pattern. 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 with multiple periods. For example, a user goes to an area not only every Friday but also every two weeks on Monday. In this paper, is 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 in 5.1 rather the find a single μ, we should find a set $(\mu_1, \ \dots \ , \mu_n)$ 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 will follow the part of the article that describe how to evaluate the periods, neglecting the part where it describes how to store them. We think that for its semplicity and immediacy, this code could be truly implemented in a not too far future (two or three years) for the more loyal users. Following the article: 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:

$$T = \{sp | sp = (t_i - t_j) \text{ if } \ t_i < t_j\} \tag{5.2}$$

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 $(\mu_1, \ \dots \ , \mu_n)$, using the support:

$$SUP_i(t) = h_i \times \frac{sp_i}{\Delta t_0} \tag{5.3}$$

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 recommed restaurant r wich the series $S_u(t_i)$ is related to.

Chapter 6

Conclusion

6.1 Conclusion

The standard Lorem Ipsum passage, used since the 1500s "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

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).–, pp. –. DOI: http://robotics.stanford.edu/~ronnyk/WEBKDD2000/papers/sarwar.pdf.

Bing Xu, Zhijun Ding and Hongzhong Chen (2017). "Recommending Locations Based on Users' Periodic Behaviors". In: *ieeexplore.ieee.org* Volume 2017.Article ID 7871502, pp. –. 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, pp. –. DOI: http://www.bitsavers.org/pdf/xerox/parc/techReports/CSL-92-10_Using_Collaborative_Filtering_to_Weave_an_Information_Tapestry.pdf.

Dirac, Paul Adrien Maurice (1981). *The Principles of Quantum Mechanics*. International series of monographs on physics. Clarendon Press. ISBN: 9780198520115.

Rendle, Steffen (2010). "Factorization Machines". In: *ieeexplore.ieee.org* IEEE International Conference on Data Mining.–, pp. –. 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.