
Recommender
for
restaurants

Hello!

I am Pablo Granatiero

This is my Experimental Final Project
In Fundamental Principles of Data Science

Hello!

I am Pablo Granatiero

This is my Experimental Final Project

In Fundamental Principles of Data Science

Supervised by

Dr. Jerónimo Hernández González

Hello!

I am Pablo Granatiero

This is my Experimental Final Project

In Fundamental Principles of Data Science

Supervised by

Dr. Jerónimo Hernández González

In Collaboration with

David Martin Suarez (Product Officer)

Objective of this experimental thesis is to build a Recommender System for the new spanish gastronomic app Velada

“ In principle one of the main purpose was to consider the parameter time. Giving particular attention to recurrency:

> to build a recommender,
> not just able to suggest
> the right restaurant,
> but able to do it
> in the right moment.

Contents

1
The app

2
The data

3
The Recommenders

4
Conclusions - Future implementations

1

The App

Description and user journey

Velada

Released on September 2021.

An app for *android* and *apple* devices.

Born to modernize the world of gastronomic guides. For the more demanding *foodies*..

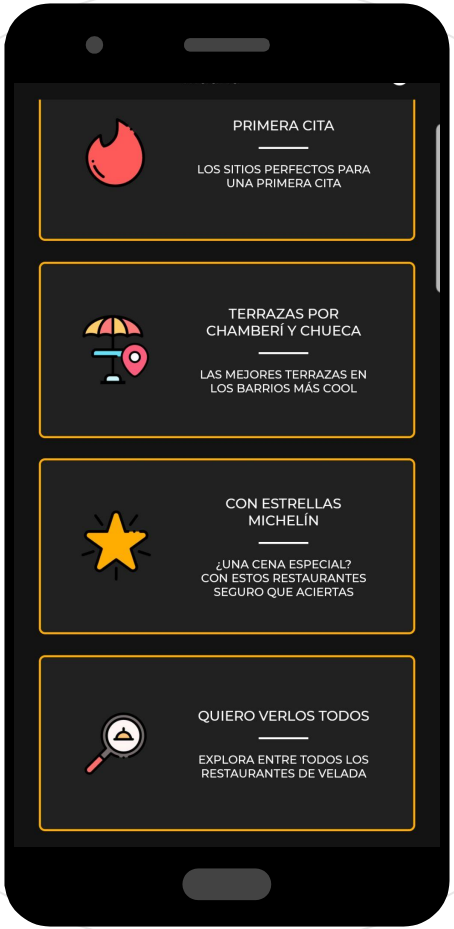
It aims to find the perfect match between a user and his restaurant:
“It is the *Tinder* of restaurants”.

it is adds to the usual features of a food application (reviews, geo-localization or food-type), filters linked with more insightful needs, like *First Date* or *Covered Terrace*.

User journey

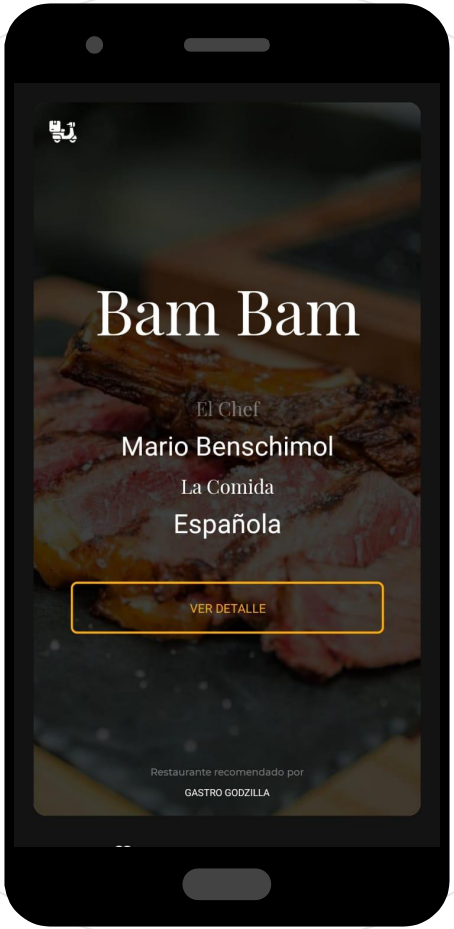
In the next slides we will see more in detail how the app works, trying to follow the user from switching on the app to choosing the restaurant.

1 Home page



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*”, …).

2 Restaurant card



The user enters to a restaurant card of a restaurant under the given filters.

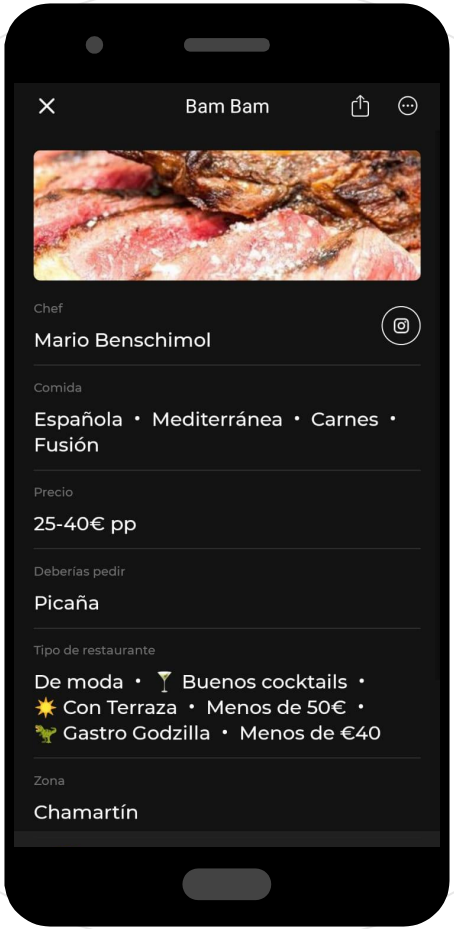
Here he can choose:

swiping left he skips the restaurants;

swiping down he saves it in his personal favourite list;

pushing *view* button he goes inside the restaurant page.

3 Restaurant page



Here a series of actions and information about the specific restaurant are accessible.

Here the user can:

go to the restaurant **website** or its position on *Google* ***map***;

go to the restaurant ***Instagram*** page;

save it into his **favourite** list, **reserve**, **call** or request a **delivery**.

4 Filters list

can push the button on the upper right corner, opening the filters menu.

Here he can customize his search activating three different filters:

Vibes (insightful needs like *Good brunch*, *special occasions* or first date),

Food-Type (like *Italian* or *Japanese*);

neighborhoods.

2

The Data

Description and analysis

Raw data files come from two different sources:

Raw data files come from two different sources:

From the app itself

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.

Raw data files come from two different sources:

From the app itself

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.

Collected by *Google Analytics*

Different dataframes describing the different actions the users do interacting with the app. They are dynamic data and each file is related to the information collected in one day of activity.

> 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.

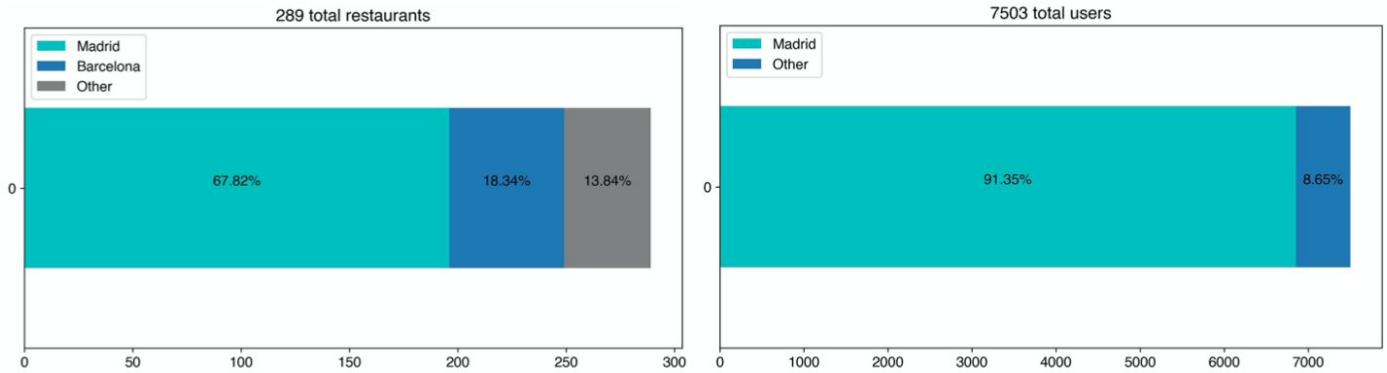
> You can find in the Github repository the program to convert raw data in an exactly ready-to-use dataframes.

Analyzing Data coming from the app itself:

It mostly regards restaurants. In particular each restaurant is defined by these five features plus its city.

- `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.

Restaurants' and users' cities



(A) All the restaurants in Velada.

(B) All the restaurants in Velada.

- **the total number of restaurants is 289.**
- **Of these 196 restaurants from Madrid, 189 are active**, in the meaning that users made with them one interaction or more
- **the total number of users is 7503**
- Of those users **~ 91% are from Madrid**, where by this we mean users that interacted only with Madrid restaurants

Analyzing Data coming from Google Analytics:

It collects all the interactions between users and restaurants saw in the user journey. A timestamp defines the instant they happened.

Overall interactions:

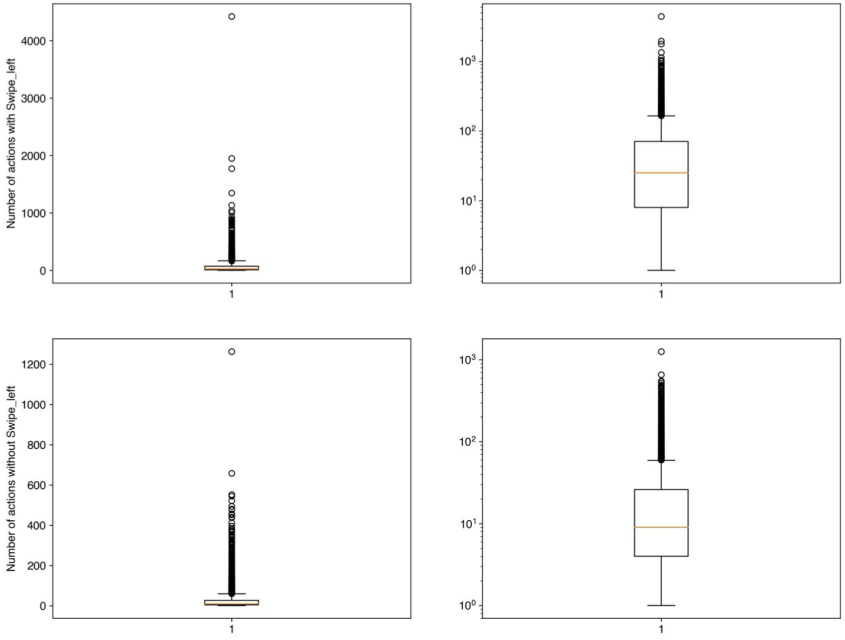
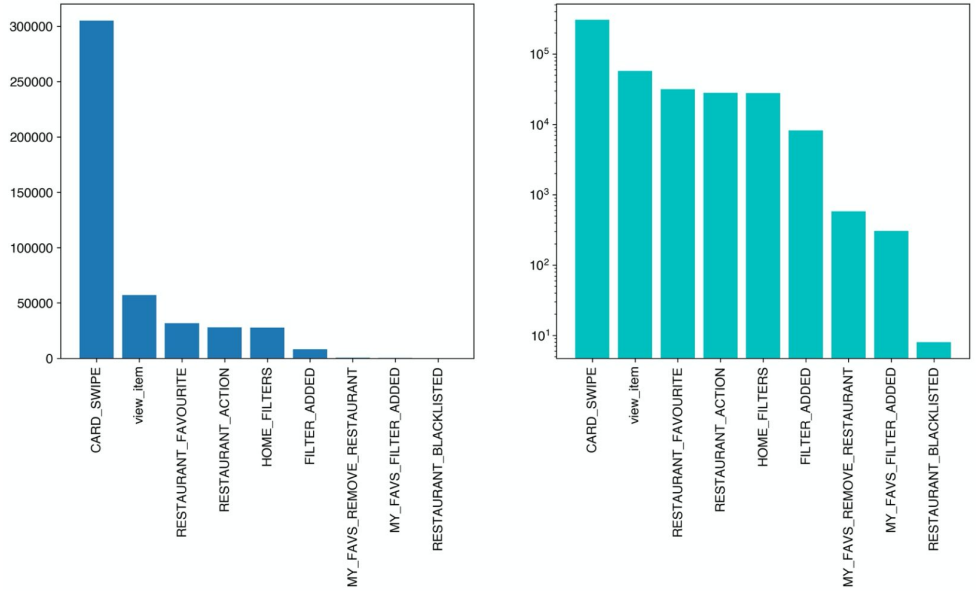


FIGURE 3.1: A box plot of the total number of users' interaction with the app. On the right in logarithmic scale and on the bottom without considering the Swipe Left action.

The mean number of actions per user is ~ 61.13 with the 75% of all users between 1 and 71 actions. **If we do not consider *swipe left*, the mean reduces to ~ 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.

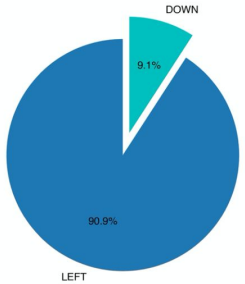
Overall interactions:



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

The most common action is by far ***card swipe***: swipe restaurants is the most entertaining action. ***View item*** is, with almost an order of magnitude less, the second most common action. Followed by **restaurant favourite** (good news for a Recommender) and **restaurant action** (we will see them in next slide). Then the filters and the black lists.

The big majority of swipe is left (skipping the restaurant).



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

Restaurants actions
-actions inside a restaurant's page-

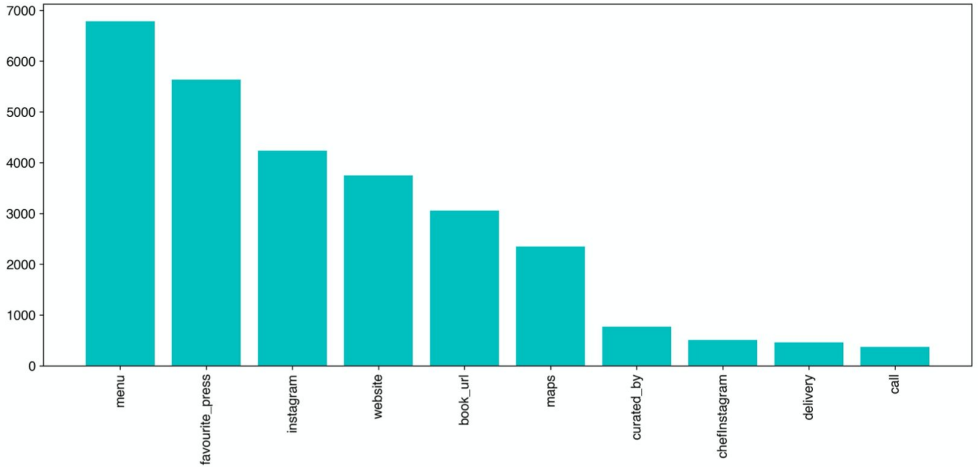
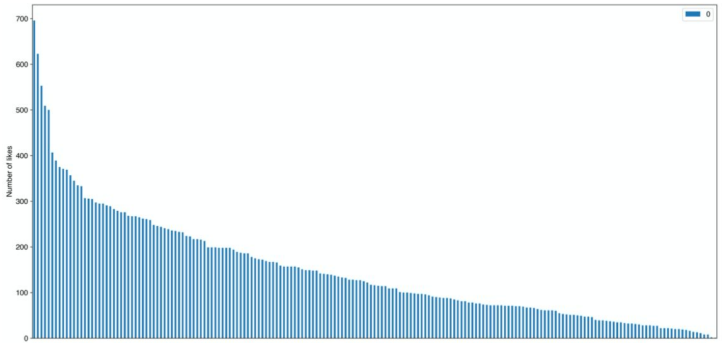


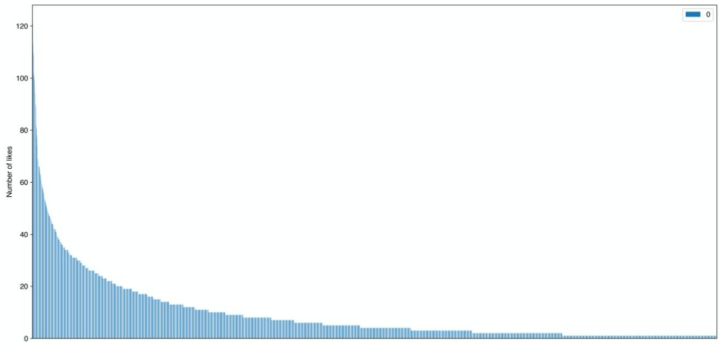
FIGURE 3.4: The total number of interactions the users can do inside a restaurant page.

The most common actions are the ***menu*** button and the ***favourite*** button. 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.

Likes



(A) The number of like for each restaurant.



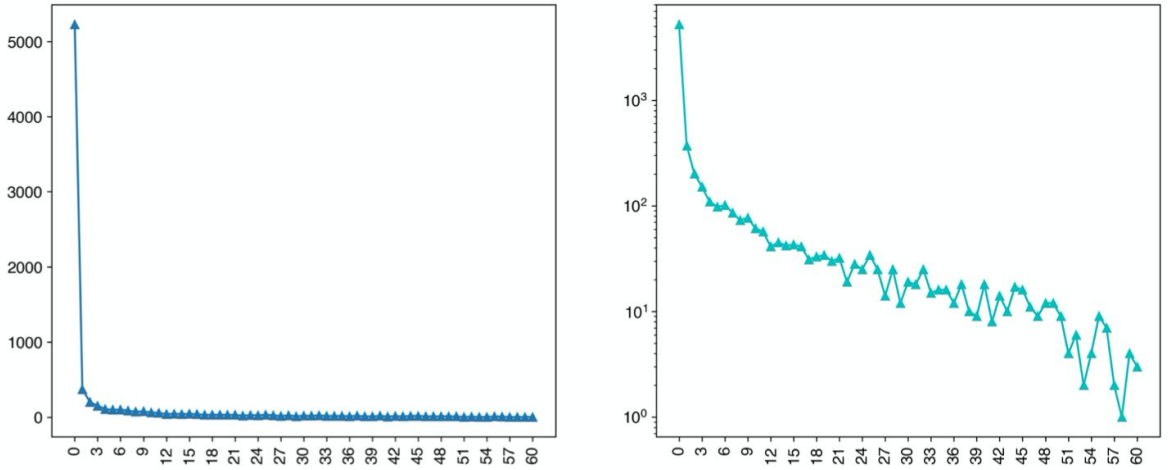
(B) The number of like for each user.

One of the key elements of a *Recommender Model* is the presence of an **explicit rate**, 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 unequivocally positive and we will simply call them *like*. We detect as positive features:

- ***Saving in favourite*** *list* (in any way it happens)
- **Book**
- **Delivery**
- **Call**

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.

Time 1

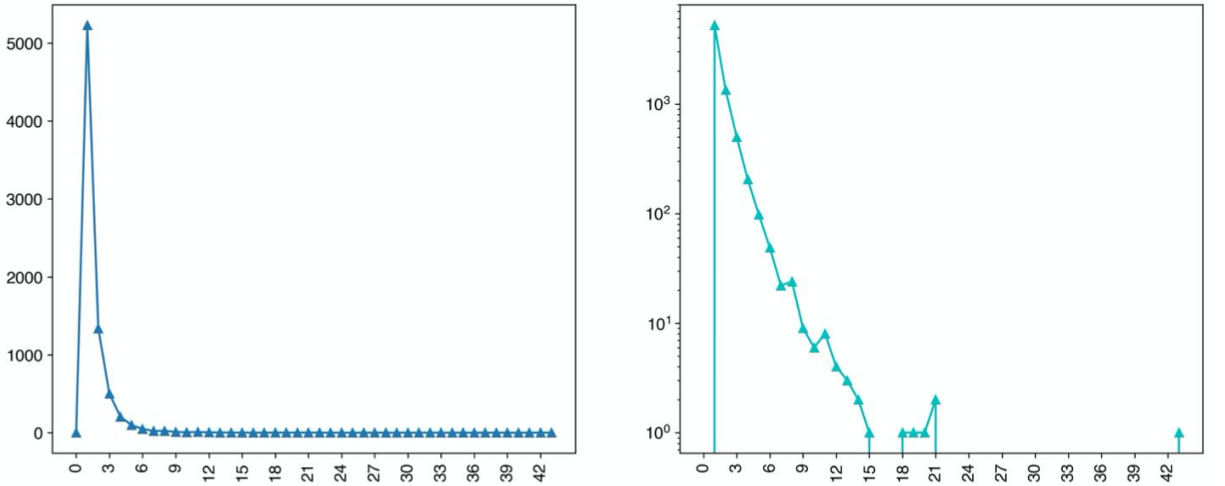


(A) On the y axis the number of users, on the x axis the distance between the first and the last day of usage.

Dynamic aspects: almost **two months of data**. From the first timestamp, **2021-03-07**, 13:01, to the last timestamp, **2021-04-30**, 21:59.

We plot **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 time length of 5 days ($O(10^2)$).

Time 2



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

Dynamic aspects: almost **two months of data**. From the first timestamp, **2021-03-07**, 13:01, to the last timestamp, **2021-04-30**, 21:59.

How often they use the app: 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, focusing our work on the static part of the recommender system, with these premises, we obtained some result which leaves us hopeful for the future.

3

The Recommenders

Description and results

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.

It analyzes patterns of user behaviours about products to provide highly personalized and customized recommendations that suit a user's taste.

Main families of Recommender Systems:

Non Personalized

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

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.

Collaborative Filtering

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.**

The Recommenders we built: Preprocess

The first decision we took is to **filter** our data to the restaurants (and interactions) located in **Madrid**.

We choose to filter the data as to take into account the **users that have a number of interactions *N* : 10 ≤ *N* ≤ 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*. **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 (book, delivery, call). And takes **False** in all the other cases or if the restaurant has been **removed** from favourites or **blacklisted**.

The Recommenders we built: Binary User-Item Matrix

Selecting 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.

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

With our data *U I* 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**.

The Recommenders we built: the metrics we choose

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**.

Precision@K: the number of restaurants guessed over K.

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

Recall@K: the number of restaurants guessed over N.

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

Accuracy: the number of restaurants guessed if K=N, over N.

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

The Recommenders we built: the base models

Two models we will use as a baseline to compare recommenders with each other.

Random Recommender

It takes as input a *user_id u* and it gives as **output *K* restaurants randomly chosen** from $R(u) = \{r_1, \dots, r_n\} : UI(u, r_i) = 0$
ri is a restaurant that the user *u* still does not like.

Most Popular Recommender

We sort the restaurants in *R(u)* by the number of likes. 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**

$Sort\_R(u)$: $Rec_K(u) = \{r_{1_i}, \dots, r_{K_j}\}$

The Recommenders we built: the Collaborative Filtering

Is **based on the sparse binary user-items *UI* matrix**:

Each restaurant is described by a *N* dimensional sparse vector r = (0, … 0, 1, 0 … , 0), 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 *ri* and *rj*, a function *d*(*ri*, *rj*).

In the binary case, the most used distance is the cosine distance: $$d(r_i, r_j) = 1 - \frac{(r_i, r_j)}{|r_i||r_j|}$$

Alternatively we can define: $$sim(r_i, r_j) = \frac{(r_i, r_j)}{|r_i||r_j|}$$

Using the similarity of *ri* with others restaurants we can assign a score to all the restaurants *r* ∈ *R*(*u*): $$score(u, r) = \frac{\sum_{r_i} sim(r_i, r) UI(u, r_i)}{\sum_{r_i} sim(r_i, r)}$$

We will then **sort all the restaurants *r* ∈ *R*(*u*)** based on their scores *score*(*u*, *r*). **The first *K* restaurants** in the list are the restaurants we are **recommending** to *u*.

The Recommenders we built: the Content Based Model

We will represent the restaurants in a different way. The restaurants in Velada are defined by a series of features: **Vibes(30), Price-range(6), Food-Type(24), Stars(4), Neighborhood(19).**

The restaurant *r* is represented as a *One-Hot-Encode* of these five features, obtaining a vector ***Vr* = (0, ... ,1, ... ,0)** where $V_r \in R^{N_{feat}}$ and *Nfeat* is the total number of different values the five features can assume, in this case ***Nfeat* = 83**.

We then **normalized** each part of the vector corresponding to a **particular feature to a l1 norm.** Finally the obtained **vector is normalized in an l2 norm.**

We initialize an empty **user vector Vu** for the user *u* **in the same space of *Vr*.**
We then select the **set of restaurants the user *u* liked, r : *UI*(*u*, *r*) = 1.**
We **add one to each component *Vu*(*i*)** each time a restaurant **has *Vr*(*i*)≠0.**
We normalize twice the vector *Vu*, in an l1 for each category slice, and in l2 the whole vector.

We obtained a representation of the user comparable with items.
The nearest *K* restaurants, using **cosine** distance, are the restaurants we will recommend.

The experiments we did: setup (unless specified)

- **Madrid** users and restaurants (interactions)
- Filtered data with a number of actions ***N*, s.t. 10 ≤ *N* ≤ 1000**
- Users and restaurants with **at least 5 likes** as to have enough information
- The number of recommended restaurants as ***K* = 5**
- A dataframe with all the likes of all the user/restaurants couples. We splitted it into train and test set with a **test size of 0.1**
- To have more consistence of results we repeated the experiment for **5 different splits**

The experiments we did: base models results

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

These results are not surprising. Indeed in general, **the number of restaurants** a user u really like **in the test set, is N s.t. *N* < 5**, so ***P*@5 < *R*@5**. As we will see in **all the cases** *P*@5 < *R*@5.

Furthermore **by measuring if *Accuracy* ≤ *P*@5**, we can evaluate how much $|RecN(u) \cap RealN(u)| \leq |RecK(u) \cap RealK(u)|$, 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.**

The experiments we did: Collaborative Filtering results

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: **passing from 3027 Madrid users and 189 restaurants to 1450 and 187 restaurants.**

The experiments we did: Collaborative Filtering results

Adding the "neighbors" parameter

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

Rather than summing over all the restaurants, we **will sum just over the *Nneigh* restaurants more similar to *r*, basing the similarity on cosine distance.**

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:

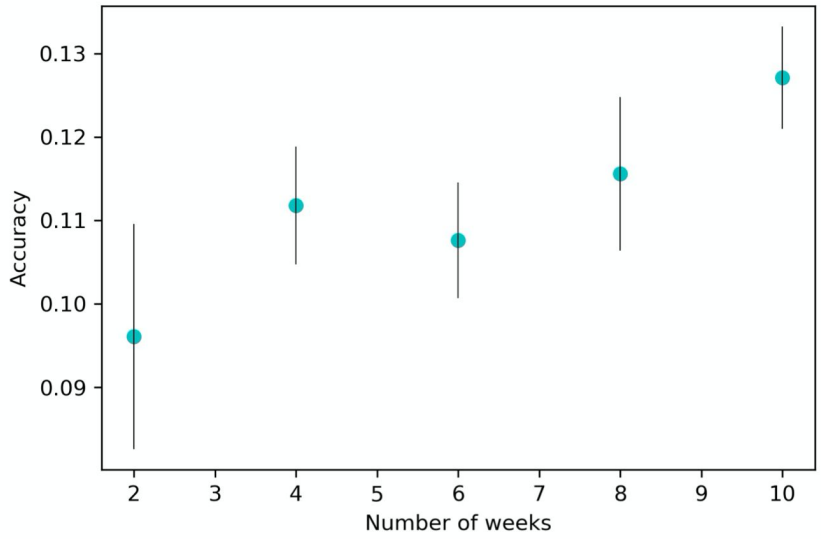
$$\overline{Acc} = 0.12907 \; ; \; \overline{P@K} = 0.11547 \; ; \; \overline{R@K} = 0.22402 \; ; \; \overline{\sigma_{acc}} = 0.004642$$

The experiments we did: Collaborative Filtering results

Rising the volume of data through time

Analyzing the change in accuracy, taking the different data set through time: passing **from just 205** users (indeed big sigma) and **3243 likes** of a two weeks data set, **to the 1450 users and 24740 likes** of full dataset.

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.**

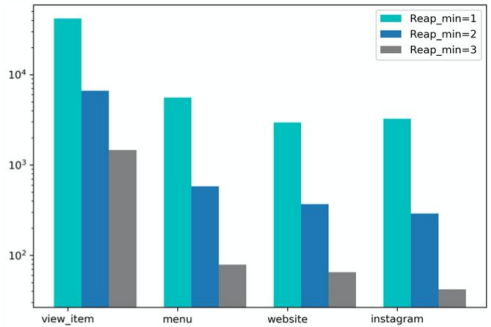


The experiments we did: Collaborative Filtering results

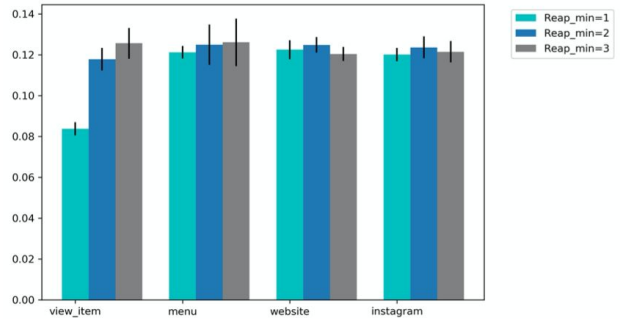
Trying to add features

We added 1 in the *UI* train matrix, for every *u* that did a specific action on restaurant *r*.
We used *view_item, menu, website, instagram,* considering how many consecutive times was pushed (Reap_min).
We performed a grid search with all the combinations of these four actions and $1 \leq Rep\_min \leq 3$, obtaining 45 different settings.

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. the number of users that did an action more than three times is very small. **All this means that this action not adds relevant information to the model**



(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.

The experiments we did: Content Based results 1

Feature importance and adding information

-Features' importance- 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 select features** from the vectors, just **slicing and removing the part corresponding to the desired features.**

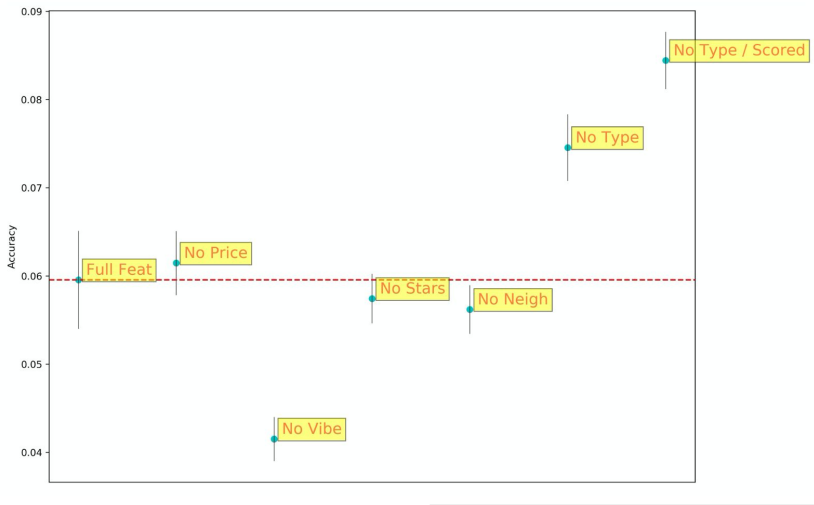
-Adding information- We assigned a score to each action in a qualitative way. **(Skip it → Enter his page → More Actions on it → Favourite → Reserve).**
We then consider **for each couple (*u*, *r*) all the set of actions** user *u* did on restaurant *r*, **we assign the score** on each action, obtaining for each (*u*, *r*) a set of scores. **We finally assign to (*u*, *r*) the max of the set**.

Score Rules

- **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.

The experiments we did: Content Based results 2

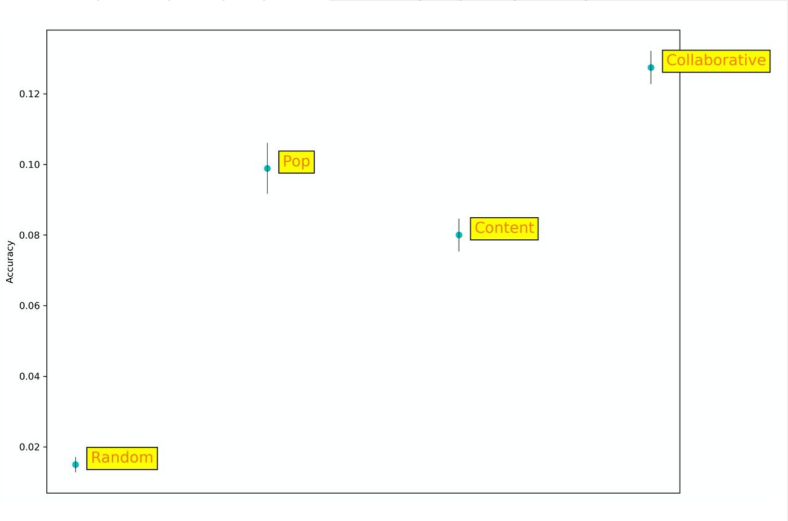
Feature importance and adding information



It performs worse than the Collaborative Filtering. *Price, Stars and Neighborhood* do not affect considerably the result, otherwise ***vibe*** **improves it and *food type* worsens it** (and this is quite wondering).

The best set up is the one without *food type* considering the rates assignated with the score rules.

The experiments we did: Comparing all models



After **two months** of data **all models perform better than the Random Model.**

The **Collaborative** that actually, with ***acc*** **~ 0.13**, is our best model.

The number of **different restaurants** suggested by **the Most Popular model**, applied to the whole data set is **22**. the number of different restaurants suggested by the **Collaborative**, that is **189.**
It means more restaurants' rotation and better user experience (serendipity).

4

Future

Implementations and conclusions

At an app level, the introduction in the app of an explicit score in the meaning of a five star scoring like the one existing in Netflix.

A more clear way to verify if a user actually went to a reserved restaurant.

> At a model level, two main implementations: regarding the static part, Factorization Machine. Then we can finally have a very first approach to the dynamic part.

Factorization Machine 1

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

The starting point of the model is a different user-item representation. We introduce the sparse vector:

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

X, rather than representing a user or an item, represents a user-item interaction. Every slice of x is a one-hot-encode of the N users, the M restaurants and the F different values of the features. The factorization machine model tries to infer the real score with a prediction that takes into account the second order interactions:

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

Factorization Machine 2

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

If ***Z = N + N + F***, are the dimensions of *x.* **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* × *Z*** matrix *Vij*, giving **a huge amount of parameters.**

The key idea of FM is to make a factorization of Vij as to reduce the dimensionality.
V is approximated by:

$$V_{i,j} = (V_i, V_j) = \sum_{p=0}^{K} V_{ip} V_{jp}$$

Passing from (*Z* × *Z*) parameters to *K* × *K*, where *K* is a parameter of the model.

Adding the time component to the recommender system

Two central but less explored questions: **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.

In both cases we need a periodic behaviour that we don't have in our data:

- we saw **all the activity** is concentrated **in the first day** of usage
- if we state that **a user enjoyed a restaurant with *delivery*, *call* or *book***, 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 **3 different Fridays**
- the **same restaurant** has been reserved by the **same user** is still very rare, it happened just **99 times**

Adding the time component to the recommender system:

Our simple model: linear increasing with time starting from the last reservation.

Let's say we built a recommender that **suggests *K* restaurants *RecK* = *r*1, ... , *rK*, without dropping from *RecK* restaurants *u* already *reserved***. In order **to suggest a single restaurant** to *u*, we initialize a **probability vector *p* = (1/*K*, ... , 1/*K*)** for a multinomial distribution *Mult*(*p*). **Let *μ* = Δ*t***, the mean distance in time **between two consecutive reservations of the same restaurant *ri***. We then modeled **the probability *pi* in a linear way**:

$$p = \lambda(t)\frac{1}{K}$$

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

If we are near the day user *u* reserved the restaurant *ri*, 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.

Adding the time component to the recommender system:

Recommender for *LBSN* (Location-Based Social Network) with multiple frequencies.

It's a very simple, easy to implement, way to learn and store multiple recurrency between user and restaurants. For example if a user use to reserve every Friday and every two Saturday.

It start with the binary series: $S(t) = 0001100101010011$

We then store all the periods: $T = \{sp | sp = (t_i - t_j) \text{ if } t_i < t_j\}$

With a support SUP, with h_i the number of times a period sp_i occurs, we can select the real periods.

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

Fixed a threshold *l*, if $l \leq SUPi(t)$ we can finally store $spi = \mu i$ in our list of periods $(\mu 1, \ldots, \mu n)$.

4.2

Conclusion

- ***Collaborative Filter*** using a binary rate is our best model **with a 13% of accuracy** score. 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.**
- 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.
- 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.**

Thank you!