Universitat de Barcelona

Fundamental Principles of Data Science Master's Thesis

Twitter users during the 2020 Spanish lockdown. Characterizing the change in users' activity and interactions

---

Author:
Bernat Esquirol Juanola

Supervisors:
Dr. Emanuele Cozzo
Dra. Luce Prignano
Prof. A. D. Guilera

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

October 7, 2020

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Twitter users during the 2020 Spanish lockdown. Characterizing the change in users' activity and interactions

by Bernat Esquirol Juanola

In this Thesis we measure the behavioural change in Twitter users during the Spanish lockdown in the context of the COVID19 crisis and across different user types. We will create two different models that simplify user behaviour, as well as look at evalute their ego-networks features through time.

Acknowledgements

Thanks Luce and Emanuele for great advices.
Thanks Elisabet for the patience and the reviewing effort.

Chapter 1

Introduction and context

This thesis can also be found as a series of blog posts at https://bernatesquirol.github.io/tfm-docs and the code at https://github.com/bernatesquirol/tfm

The goal of this work is to evaluate the differences between Twitter user behaviour before and during the 2020 Spanish lockdown, that went from 13th March 2020 to May 2020. The dataset we will analyse is a collection of tweets from 17.868 Spanish Twitter profiles, categorized in 3 types: politicians, journalists and random profiles.

In order to measure the impact in several ways we will create different user-level models that will measure some behaviour characteristic over time. Then we'll aggregate the results for all users and discuss the changes over time and between user types. This work will have three different parts:

- Time-series analysis (Chapter 2)
- Bayesian model (Chapter 3)
- Ego networks analysis (Chapter **??**)

First we will take a look at the data from a global perspective as well as explain how we gathered it. In order to measure the impact of lockdown in user's activity, we want to get the most important changes in behaviour for every user and compute overall measurements. We will create two simplifying models: one based on time-series intervention analysis and another based on a Bayesian model. Finally we will introduce the concept of ego-network and we will investigate their evolution during the period of lockdown. All of this will be done while trying to unveil the differences of the fitted models among the groups of users and study their characteristics.

Database

We have a database with the details of each user, from the endpoint `users/show` in Twitter API. This details include the number of followers/friends, total number of tweets/likes, and so on. We will define several computed user features in the thesis, and add them as new columns to users' database. They will all be styled like **`this`**.

We also have a *light* version of every user's timeline. A timeline is the collection of tweets a user has created. We use the endpoint `statuses/user_timeline` from Twitter API to get the latest 3200 tweets for a given user.

There are several features we could extract directly from the timeline that are useful to characterize users (Arnaboldi et al., 2017b):

- **`social_ratio`**: the ratio of tweets that involve in some way other users
- **`retweet_social_ratio`**: how many social tweets are a retweet
- **`reply_social_ratio`**: how many social tweets are a reply
- **`quote_social_ratio`**: how many social tweets are a quote
- **`like_tweet_ratio`**: the number of tweets divided by the number of likes (we would also need the like timeline)
- **`frequency`**: how many tweets per day the user posts

Of course we only have a small proportion of users timeline and all of these features are relative to our sample. So these two fields are also important:

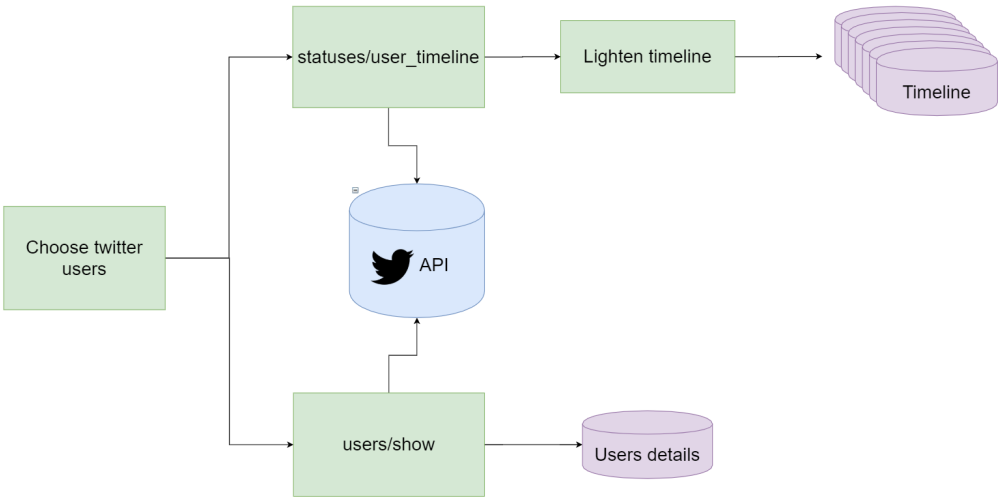

FIGURE 1.1: Data flow

- `observed_start`: date at which the observation of the timeline starts
- `observed_end`: date at which the observation of the timeline ends

Users

Journalists

There is one common feature in the journalism world: the Twitter profile of the news media that a journalist writes for, follows her on Twitter. And in fact, media pages nearly always only follow very relevant people and their own journalists. Another typical thing among journalists is that they use Twitter for professional use. It is also typical for a journalist to have some variation of the word *journalist* in their bio. This two factors allow us to find journalists very easily: We search the top social media pages in Spain and we pick their friends (users being followed by the media account), and check these that have in their bio some version of *periodista*, *journalist*,.... We have around 1.000 journalists profiles.

Politicians

We picked the members of the Congreso XIV Legistlatura list that contains all the members in Spanish congress during the February-June 2020 period. There are only 350 congressmen in Spain, and not all of them have a Twitter account. We have around 300 politicians profiles.

Random users

Getting random users from Twitter for a given location (we only want Spanish users) is not an easy task. The problem with getting random users with the endpoints of the Twitter API, is that we will most likely introduce some kind of bias. In our method we relax several assumptions in order to get random users.

Our method works as follows:

1. We call the Standard search API with an empty `query=''` and a `geocode='39.8952506, -3.4686377505768764, 600000km'` (minimum circumference that includes Spain), and `result_type=recent` and `count=100`
2. We filter those 100 tweets such that the creator of the tweet has a minimum of followers, friends and actions. The user u_0 will be the creator of the tweet. If we take directly user u_0 as our random user we would have a bias towards high tweet frequency.
3. Instead what we do is to get one of the followers or the friends of u_0. We can get as much as 5000 in one API call to `friends/ids` or `followers/ids`. Our final user will be a randomly chosen one out of the 5000.

Our method gets two types of random users: *random_friends* are those that we find using `friends/ids` call, and *random_followers* using `followers/ids`. The first group biased towards following many of people and the other biased towards having many followers. Having these two sets is not ideal, and they are also biased, as we can see in 1.2, but we avoid having biases in the frequency of tweet, a feature that we will be analysing deeply. We have around 8.000 random-friends profiles and 8.000 random-followers profiles.

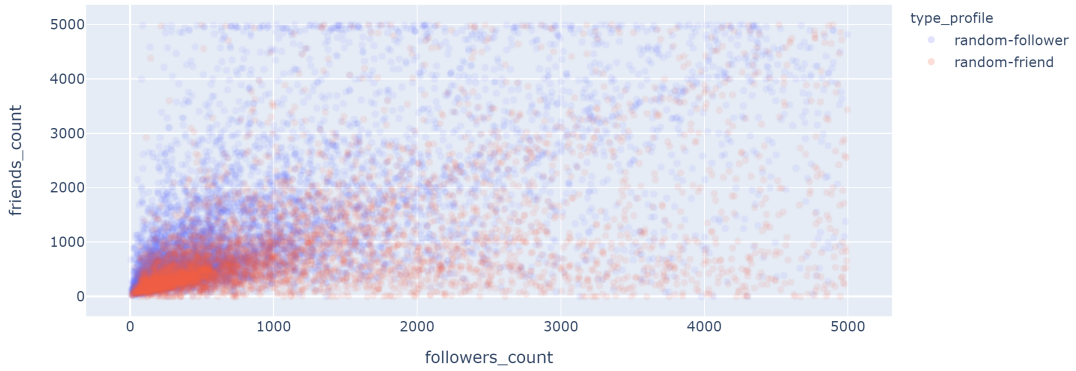

FIGURE 1.2: Bias in random profile types (max 5000 followers/following)

With the label `type_profile` we will set the user type for each user in the user profile database. The four possible values will be: `politician`, `journalist`, `random-friend` and `random-follower`.

Temporality

There is one issue that affects the data we gathered, specifically the user timelines, and can introduce some biases. Twitter API, can only give us the last 3200 tweets from a user, that means that we will have different starting dates from different users, and the starting date will be correlated with the frequency of tweet of the user, as we won't get as much back in time with the profiles with high tweet frequency. It is important to know this bias and take it into account, as we will plot a lot of global timelines, for all users, and for user types. We will divide the time series value for the user starting date bias, every time that we plot a timeline.

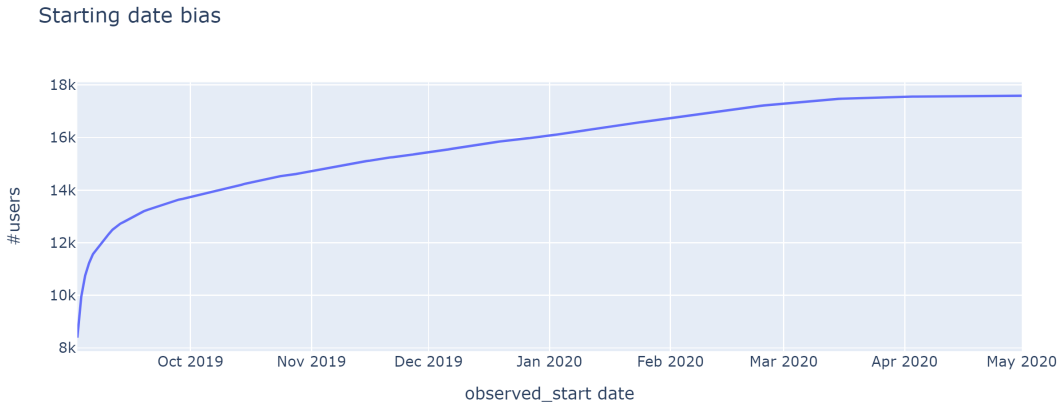

FIGURE 1.3: Bias in the starting date of our observations

Chapter 2

Time series analysis

This chapter can also be found at https://bernatesquirol.github.io/tfm-docs/timeseries.html

In this section we will try a first approach to measure the impact of the Spanish lockdown in the number of tweets a user posts.

Seasonality and trends

We are interested in knowing whether users present seasonality in their Twitter activity. We pick several timelines as example (700) and we calculate the autocorrelation plot (`acf`). Then we calculate the mean over all `acf`s. In 2.1 we can see spikes at each lag k multiple of 7. Thus we can assume tendency towards a weekly frequency period.

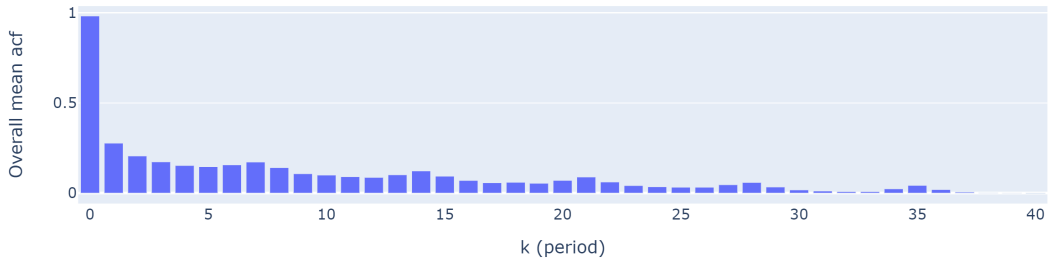

FIGURE 2.1: Mean acf for 700 samples

Stepwise segmented regression analysis

Our goal is to try to measure the effect of an intervention (lockdown) in the overall set of users. To do that we cannot only see the overall changes in the system, as we would average out global behaviour changes in opposite direction. We need to look at the user level, analyse its behaviour and then add up the information from all users to extract valuable insights. A good way to analyse behaviour of a user over time in a complex social system is to do stepwise segmented regression analysis (Britt, 2015). That means we will get a simplified version of the timeseries, one that just gets the *essential* information: points where to break the time series and levels of number of tweets per day, such that they describe accurately the real timeline. In order to do this we will not only find the optimal breakpoints to divide the timeseries in, but also the number of breakpoints that makes the model less complex while describing the model correctly. The mean value between breakpoints will be the level of that interval. The function `breakpoints` in `R` package `strucchange` gives us exactly that (Garziano, 2017). Although it could be done in `python`, there is no efficient method to get the same result, and as we are dealing with a lot of timeseries, we will stick to `R` with the wrapper for python `rpy`.

We will go pick a user timeline as an example 2.2 to see how this function works internally.

The following code creates the breakpoints and retrieves the level of each interval:

```
from rpy2 import robjects
import rpy2.robjects.packages as rpackages
```

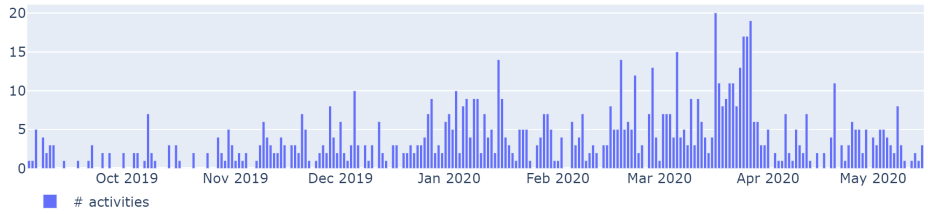

FIGURE 2.2: Activity of user 1000092194961838080

```
rpackages.importr('strucchange')
def get_breakpoints_and_levels(id_user):
    path = '../data/models/{}'.format(id_user)
    file = open(path, 'rb')
    model = pickle.loads(file.read())
    freq = model['freq']
    formula = robjects.Formula('freq_tweet ~ 1')
    env = formula.environment
    vector = robjects.FloatVector(freq.values)
    env['freq_tweet'] = robjects.r['ts'](vector,
                                         start=freq.min())
    breakpoints = robjects.r['breakpoints'](formula)
    fitted = robjects.r['fitted'](breakpoints, breaks=len(breakpoints[0]))
    return {freq.index[int(i)]:fitted[int(i)] for i in [0.0]+list(breakpoints[0])}
```

Internally what the function `breakpoints` does is it optimizes a piecewise linear fit, for each i number of breakpoints, up to n (maximum number of breaks); and it compares among the i's which has the lowest Bayesian Information criterion (BIC). BIC is a common criterion to do model selection among finite set of models. It is a way to measure the maximum likelihood of a function (goodness of fit to the real data) with the minimum amount of complexity (without overfitting). We can see the results for this in 2.5. We will establish the maximum number of breaks in $n = 5$.

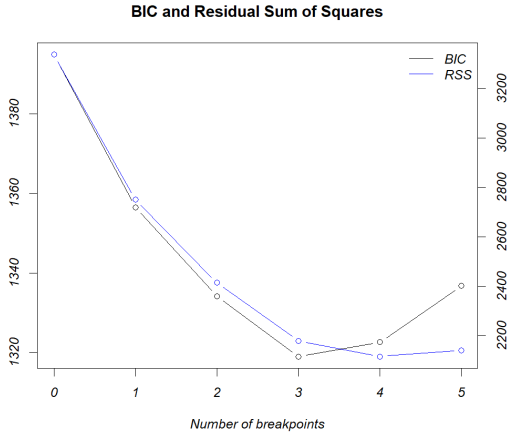

FIGURE 2.3: BIC for the optimized interval for each number of breakpoints. In the case of the example the best description of the timeseries with the least breakpoints is with 3 breakpoints (4 intervals)

Another `R` function that appears in the code is `fitted`, retrieves the level of each interval, also computed by the `breakpoints` function.

As we did this for every profile in the dataset, we can see some general insights. In order to do this we will compute the jump value for every timeline, that means, for each breakpoint, how much the tweet frequency of the user changed from the previous value. Normalizing this jump value for the mean frequency of the timeline will help avoiding frequency biases in the overall picture.

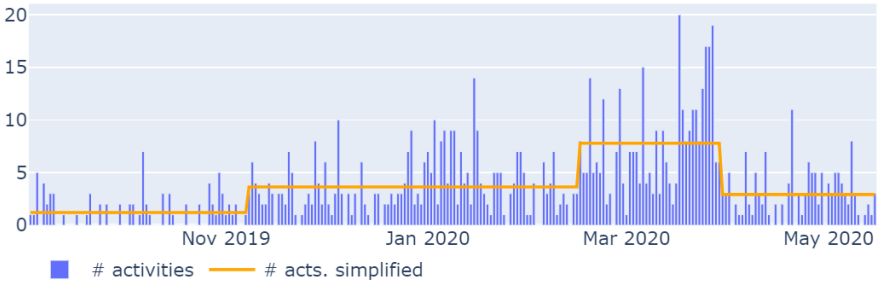

FIGURE 2.4: Activity of user 1000092194961838080

$$
\begin{aligned}
\bar{T} &:= \text{observed mean} \\
\tau_0 &:= \text{starting point} \\
\tau_i &:= \text{date of the breakpoint} \\
L_i &:= \text{mean of the interval } [\tau_i, \tau_{i+1}) \\
J_i &:= L_i - L_{i-1} \\
J_i' &:= \frac{J_i}{\bar{T}}
\end{aligned}
$$

In the case of the example:

$$
\begin{aligned}
\bar{T} &= 3.57 \\
\tau_0 &= \text{08-Nov-19}, J_0 = 2.43, J_0' = 0.68 \\
\tau_1 &= \text{16-Feb-20}, J_1 = 4.16, J_1' = 1.16 \\
\tau_2 &= \text{30-Mar-20}, J_2 = -4.86, J_2' = 1.37
\end{aligned}
$$

With the label **`levels_dict`** we will add a dictionary with keys τ_i and values L_i as a feature in our users database.

Analysis

In this section we will analyse certain aspects of the features we have computed so far.

Seasonality

We have computed the **`seasonal_decompose`** of each timeline in our database, that means decomposing the activity in (additive) trend/seasonal/residual series. In the plot in 2.5 we can see the differences in seasonality between user types. We can see a Tuesday-Friday activity cycle in politicians that aligns well with typical political weekly communication cycles.

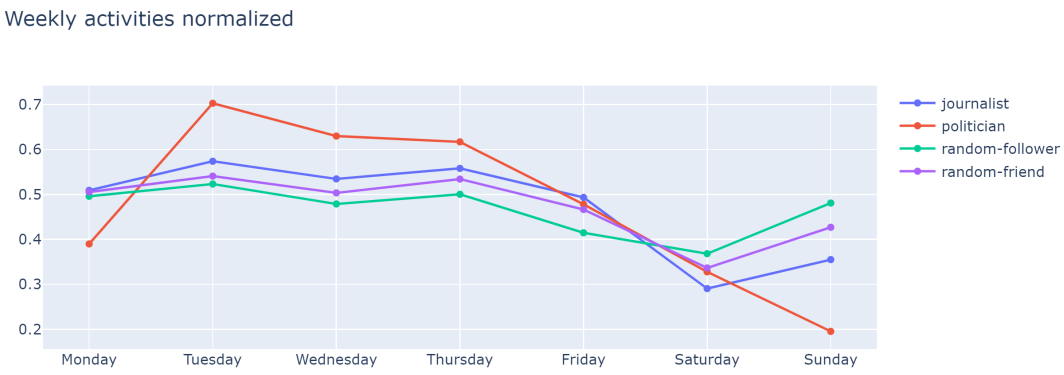

FIGURE 2.5: Weekly seasonal activity per user type

We can also look at overall activity per week in the whole system. We will plot the number of tweets per week for all our timelines. We will compute how many tweets per week a user created and we will normalize between weeks (so the more active week for a user gets a 1 and the least active week gets a 0). If we sum all of these levels of activity per week (divided by the starting date bias) for all users we get 2.6. We see in 2.6 how in March the activity overall reaches levels that hadn't been achieved since October 2019. But of course this plot is not the best way to measure the impact of COVID19, as some of the users may have decreased the activity during the lockdown and some others may have increased. If we only compute the activity in the system we don't see this changes.

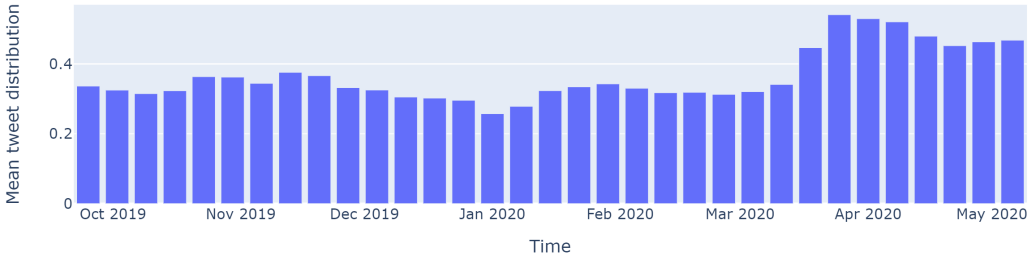

FIGURE 2.6: Activity per week for all our users

Breakpoints

Here we plot the number of breakpoints for each user type. We can see that it is quite similar among user types. We can also see a trend towards 2-3 breaks and not more, and few users reach the 5 breakpoints. Consequently we can say our maximum number of breakpoints was good enough.

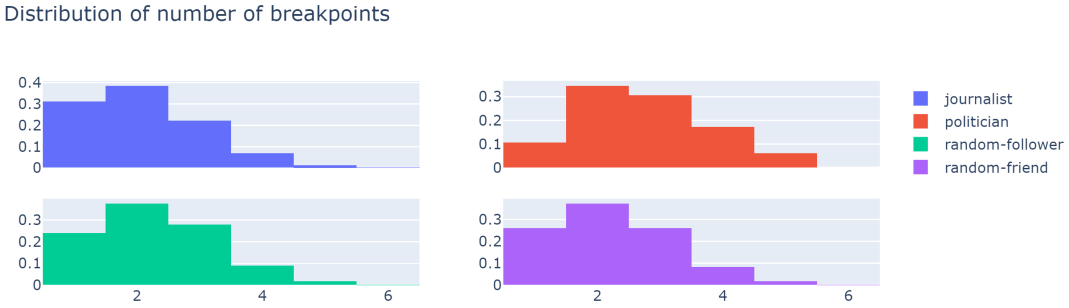

FIGURE 2.7: WWeekly seasonal activity per user type

Levels

We can see in 2.8 the sum of J' for all users in our dataset. This values give us the first way to measure the change in user behaviour to COVID19. This is a more accurate measure of what happened in the Oct-19 and May-20 period. The green lines represent users changing its activity to a higher level, and red ones to a lower level. We can see a green increase in the beginning of the confinement and a drop in late April.

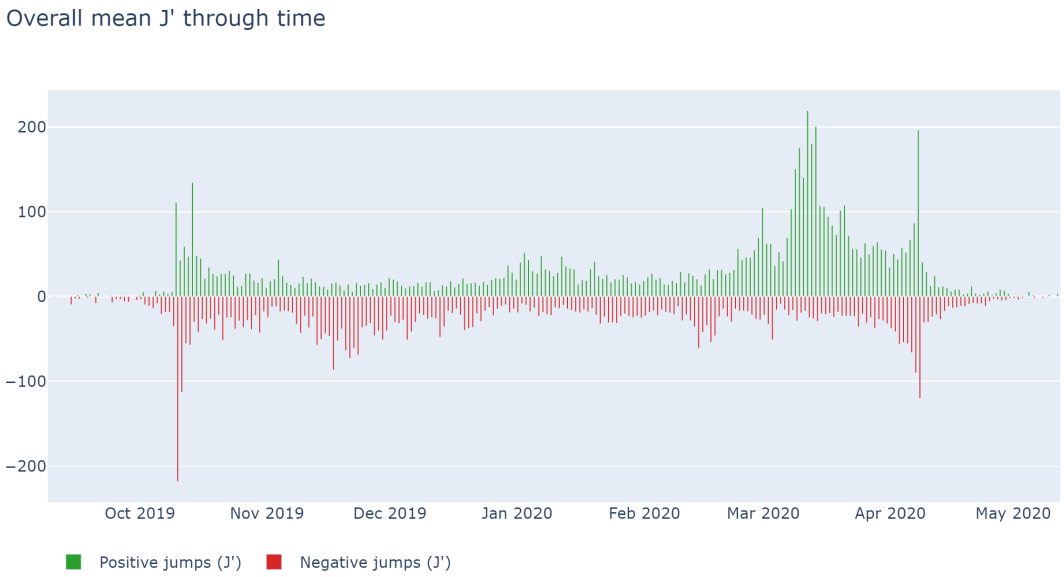

FIGURE 2.8: Sum of jumps between levels (J') from all timelines

FIGURE 2.9: Sum of jumps between levels (J') from all timelines per user type

Chapter 3

Bayesian analysis

This chapter can also be found at https://bernatesquirol.github.io/tfm-docs/bayesian.html

In this chapter we will try a different approach to measure the impact of the Spanish lockdown in Twitter user activity, with the same dataset. Our goal is to simplify the activity data of the user so we can measure the most important changes in all users behaviour and the impact of lockdown and COVID19 in Twitter. We aggregate our data by days.

Prior

In order to measure the impact of the Spanish lockdown in Twitter, we will analyse multiple users timelines and we will fit a *Bayesian switchpoint* model for each timeline. This is a very basic model that assumes that there is one day (τ) the user changes its behaviour, either increasing the frequency of tweet or decreasing it. The fact that we have just one switchpoint will be great for our analysis, as we have a huge collection of timelines, and we want to extract a *minimalist* model.

We will call T_i to the expected number tweets for the day i created by a given user. Although there is evidence that under certain circumstances human interaction follows a non-Poisson process (Pareto, Weibull, Log-Normal), there is a wide range of social interaction models that assume that communications among individuals are randomly distributed in time, and thus Poissonian (Zhang et al., 2016).

Here T_i will be modelled following a Poisson distribution and at day τ the parameter of the distribution will change from λ_1 to λ_2. The Poisson distribution is defined as:

$$P(Z = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad k = 0, 1, 2, \ldots$$

The parameter λ must be positive, and if we increase λ we add more probability to large k's, and if we decrease it, small k will have more probability. One important fact about the Poisson distribution is that the expected value of the distribution $E(\text{Poisson}(\lambda)|\lambda) = \lambda$.

These two λ's will be modelled following Exponential distributions. The Exponential distribution always gives a positive number, it is continuous like the parameter in the Poisson and has a really straight forward expected value. Is a continuous defined by:

$$f_Z(z|\alpha) = \alpha e^{-\alpha z}, \quad z \geq 0$$

And its expected value is $E(\text{Exponential}(\alpha)|\alpha) = \frac{1}{\alpha}$.

So the prior for our model is, with $n =$ days (Davidson-Pilon, 2015):

$$
\begin{aligned}
\lambda_1^{(0)} &\sim \text{Exponential}(\text{rate} = \alpha) \\
\lambda_2^{(0)} &\sim \text{Exponential}(\text{rate} = \alpha) \\
\tau &\sim \text{Uniform}[\text{low} = 0, \text{high} = 1) \\
&\text{for } i = 1 \dots n : \\
\lambda_i &= \begin{cases} \lambda_1^{(0)}, & \tau \leq i/n \\ \lambda_2^{(0)}, & \tau > i/n \end{cases} \\
T_i &\sim \text{Poisson}(\text{rate} = \lambda_i)
\end{aligned}
$$

We have only one hyperparameter in this Bayesian graph that is α. From the expected values of the distribution we have:

$$E(T_i) = E(\text{Poisson}(\lambda_i)|\lambda_i) = E(\lambda_i) = E(\text{Exponential}(\alpha)|\alpha) = \frac{1}{\alpha}$$

And as we want $E(T_i) = \bar{T}$, we will have our initial rate for the λ_i's be: $\alpha = \frac{1}{\bar{T}}$.

With these steps we converted our prior in an informative prior. We could have also created two α_i, one for every λ_i, but the values of those α_i wouldn't be so clear and unbiased.

This model will fit a τ that will be the day of the *distribution change*. Our prior for this τ will be a uniform distribution, as we want all days to have the same probability to be the switch point. We expect to see a the change of behaviour near the lockdown dates.

Model fit

The implementation is done with Tensorflow probability (TFP) in Python 3.6. To fit the data into the model and get our posterior we will use Markov Chain Monte Carlo algorithm. This algorithm finds the most probable parameters for a given prior by sampling from the prior and fitting the data. When we specify the priors, we create a n-dimensional surface that can be modified in certain ways, depending on the parameters. The MCMC algorithm travels the space of all possible surfaces and finds which have the higher probability to have generated our data. There are several types of MCMC algorithm that change how the optimization works, we will use Hamiltonian Monte Carlo. We need to create our prior in the TFP language so the `tfp.mcmc.HamiltonianMonteCarlo` function can optimize it:

```
def joint_log_prob(count_data, lambda_1, lambda_2, tau):
    tfd = tfp.distributions
    # alpha is the initial value of both exponentials
    alpha = (1. / tf.reduce_mean(count_data))
    # exponential lambda
    rv_lambda_1 = tfd.Exponential(rate=alpha)
    rv_lambda_2 = tfd.Exponential(rate=alpha)
    # uniform tau
    rv_tau = tfd.Uniform()
    indices = tf.cast(tau *
                  tf.cast(tf.size(count_data),dtype=tf.float32)
                          <= tf.cast(tf.range(tf.size(count_data)),
                  dtype=tf.float32),
              dtype=tf.int32)
    # switch
    lambda_ = tf.gather([lambda_1, lambda_2],
                        indices=indices)
    # T_i
    rv_observation = tfd.Poisson(rate=lambda_)
    # return joint log probability of the model
    return (
         rv_lambda_1.log_prob(lambda_1)
         + rv_lambda_2.log_prob(lambda_2)
```

```
        + rv_tau.log_prob(tau)
        + tf.reduce_sum(rv_observation.log_prob(count_data))
    )
```

Now, we need to create the kernel where tensorflow optimizes the function. The kernel will output some results about the convergence of the algorithm and the samples the algorithm found to λ_i and τ.

Model test

We will fit a series of artificial data, some of which will follow the prior, and some of which won't, to see how well the model behaves. In order to create artificial timelines, we will create a key valued dictionary τ_i and L_i, that will be similar to breakpoints and levels dictionary we created in the segmented regression analysis section (2) in the timeseries model. The dictionary has as keys (τ_i) the dates of the breakpoints and as values ($\lambda_i = L_i$) the value of the expected tweet activity for the following period. The first key is always the beginning of the timeseries.

We will use `pvalue` to get a notion of model fitness to the real data. We used an adaptation of the frequentist Kolmogorov-Smirnov test. We compute different artificial datasets based on the output of the model, and average the `pvalue` of the KS-test between those artificial datasets and the real data (in the case of the following examples is the real data is also artificial).

Single switch

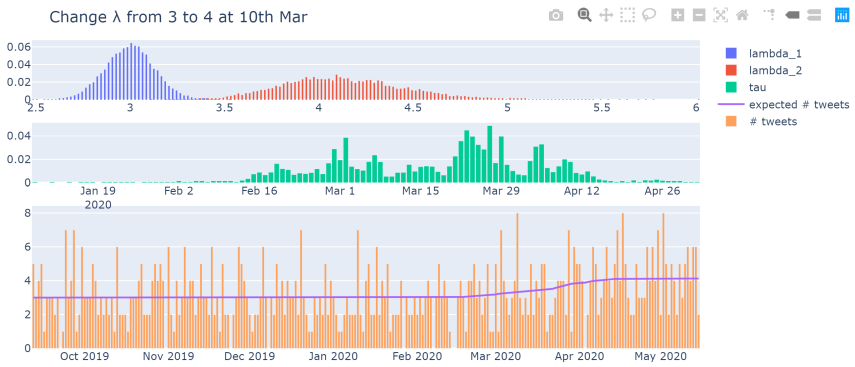

FIGURE 3.1: `{2019-09-12: 3, 2020-03-09: 4}` `pvalue=0.56`

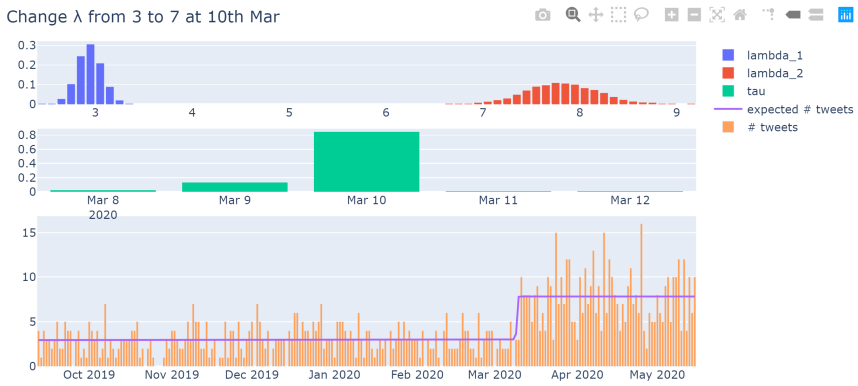

FIGURE 3.2: `{2019-09-12: 3, 2020-03-09: 7}` `pvalue=0.77`

We can see two immediate observations from figures 3.1, 3.2, 3.3:

- We will have clearer distributions for τ when the change in the λ_i's is large, as we expect τ to be only one day. The more unclear τ is, the wider will be λ_2 distribution.
- We will have a thinner distribution in λ_i when the change happens in the middle. Else we will have more uncertainty (as we have less data) in one of them.

Multiple switches

The real data may have more than one switch, but our model assumes there is just one.

In the examples we can see how in case of two or three changes of behaviour, the model will get only one of them. We can see how the `pvalue` is way lower in 3.6 (0.01). We can conclude we can use the `pvalue` as a way of knowing if there are some existing behaviour changes in the data that we missed in the model. When the `pvalue` is small doesn't mean that the model is wrong, the model will have chosen a breakpoint that the data will have, but probably other changes of behaviour may exist.

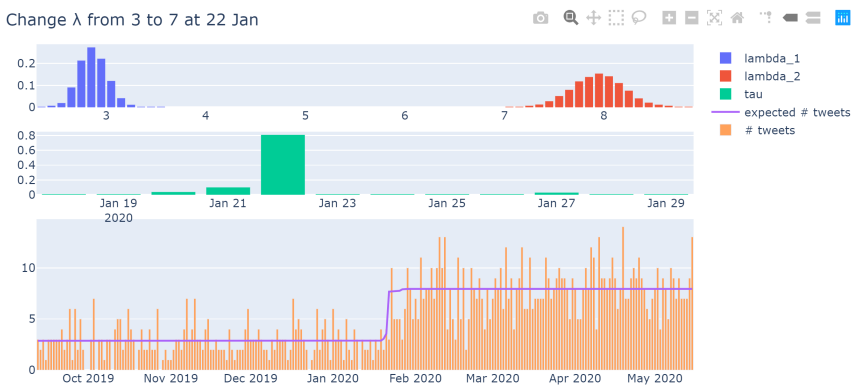

FIGURE 3.3: `{2019-09-12: 3, 2020-01-22: 7}` `pvalue=0.82`

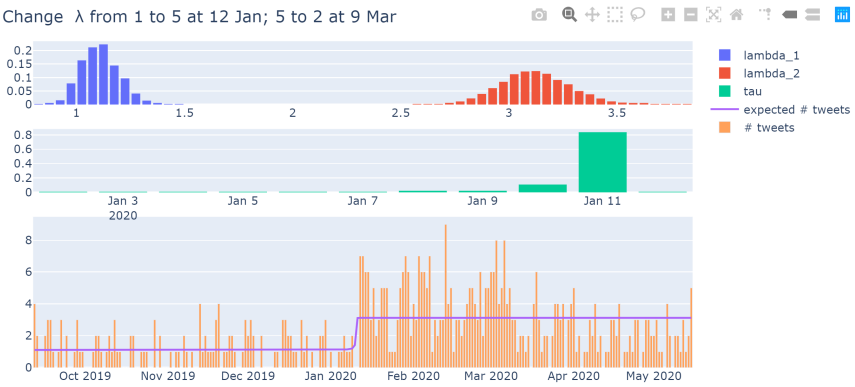

FIGURE 3.4: `{2019-09-12: 1, 2019-01-12: 5, 2020-03-02: 2}` `pvalue=0.46`

Real data

For every timeline in the database, we fitted the model previously mentioned. Let's explore some real timelines and see how the model perfoms.

The figure 3.6 shows something new. If we look at both distributions of λ_i they appear to have a binormal distribution. If we look at the τ we can also see two bumps, one in February 22nd and another in March. What is happening here, is that there are two switchpoints, the rising in February and the decreasing of activity in March. We can see how the rising in March corresponds probably to the part of $\lambda_2 \sim 7.5$ (as λ_1 needs to be lower that λ_2) and after March, we would have a $\lambda_3 \sim 5.6$, that is hidden in λ_2. This will happen whenever the λ_i are near and there are two switches.

To test the hypothesis that another breakpoint would create a better model, based on the dictionary: `{2020-02-22:6.4, 2020-02-22:7.5, 2020-03-22:5.4}`, with a second breakpoint (τ_2), and a λ_3, taking into account the bimodality of τ and λ_i. We adapted the `pvalue` test to create multiple timelines from that dictionary, and we found a higher `pvalue=0.015`, while the model with only one switchpoint has a `pvalue=0.0036`.

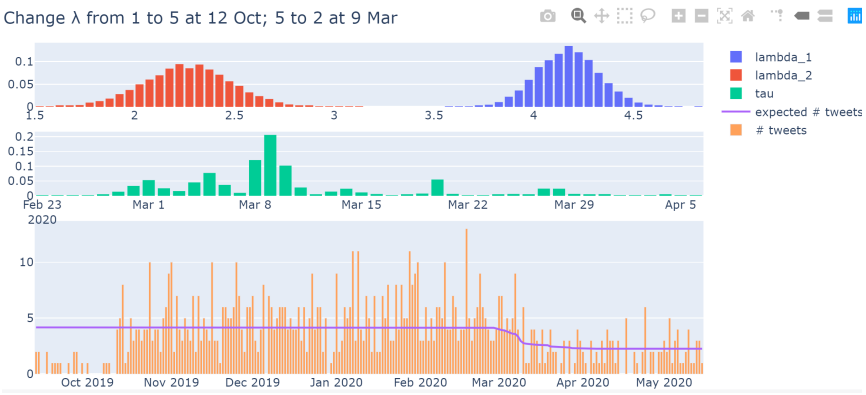

FIGURE 3.5: {2019-09-12: 1, 2019-10-12: 5, 2020-03-02: 2} pvalue=0.50

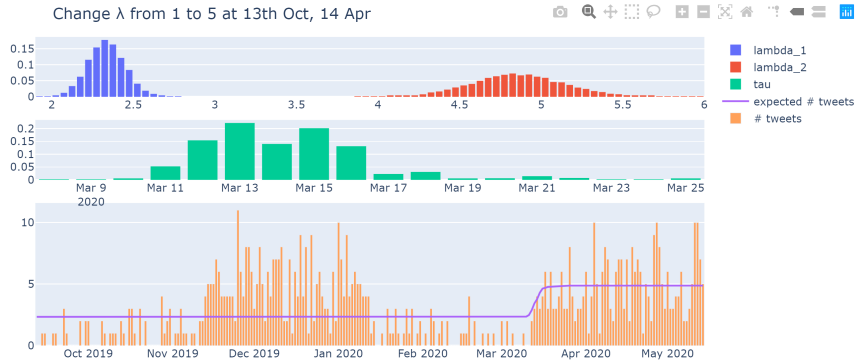

FIGURE 3.6: {2019-09-12: 1, 2019-11-13: 5, 2020-01-13: 1, 2020-03-14: 5} pvalue=0.01

Other Models

We also analysed other models for a given example. We couldn't run these models in the whole dataset, as running three models in the whole dataset would be too expensive computationally.

Sigmoid

we will present another model: the sigmoid. instead of the sudden break we had at day τ, we will have a smooth transition between λs.TFP, 2020a With the theoretical formulation of the sigmoid prior being:

$$
\begin{aligned}
\lambda_1^{(0)} &\sim \text{Exponential}(\text{rate} = \alpha) \\
\lambda_2^{(0)} &\sim \text{Exponential}(\text{rate} = \alpha) \\
\tau &\sim \text{Uniform}[\text{low} = 0, \text{high} = 1) \\
&\text{for } i = 1 \dots n : \\
\lambda_i &= \lambda_1^{(0)} + \frac{1}{1 + \exp(\frac{i}{n} - \tau)} (\lambda_2^{(0)} - \lambda_1^{(0)}) \\
T_i &\sim \text{Poisson}(\text{rate} = \lambda_i)
\end{aligned}
$$

In our data this model can remove a bit of uncertainty in the τ, as we don't need a sudden break in the data. We can see a in 3.9 the light differences between model performances in the example. As we have one progressive change, the day of the change is more probable to be before (Dec, 23) in the sigmoid model.

Multilevel model

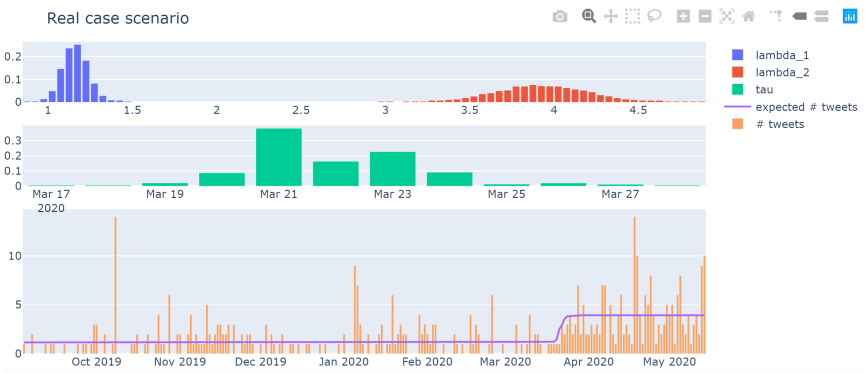

FIGURE 3.7: `user_id=455018574, pvalue=0.24`

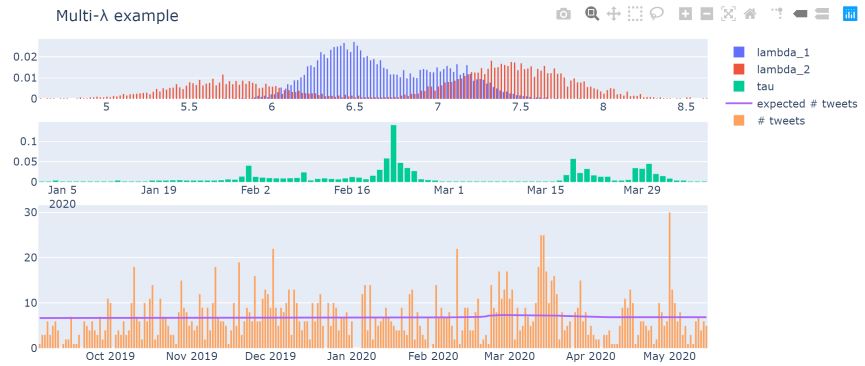

FIGURE 3.8: `user_id=112894609, pvalue=0.0036`

Another thing we could try is allowing the model to have more than two levelsTFP, 2020b. This model will resemble the model created in the timeseries breakpoints section 2. The difference here is we set up a maximum number of levels ($L_{\max}$) the model can have but no maximum amount of switches (breakpoints) between those levels.

$$\lambda_0 \sim \text{Categorical}\left(\left\{\frac{1}{L_{\max}}, ..., \frac{1}{L_{\max}}\right\} \in v^{L_{\max}}\right)$$

for $i = 1 \dots n$:

$$\lambda_i \mid \lambda_{i-1} \sim \text{Categorical}\left(\begin{Bmatrix} p & \text{if } \lambda_i = \lambda_{i-1} \\ \frac{1-p}{L_{\max}-1} & \text{otherwise} \end{Bmatrix}\right)$$

$$T_i \sim \text{Poisson}(\text{rate} = \lambda_i)$$

Similar to what we did in the splitwise segmented regression model, we will have to run it with several $L_{\max}$, creating different models. We can see in 3.10 how in 4-level model we reach the optimal amount of information with the minimum amount of levels.

This model has the advantage that fits the data more accurately, but the computation time and the simplicity of the result are worse than the step-wise segmented regression model. For example we can see that the 4-state has a small spike at day 35, this is useful to analyse one day behaviour, and we wouldn't pick this one with the other model, but when we look at changes that remain over time, the linear breakpoints model probably introduces less noise.

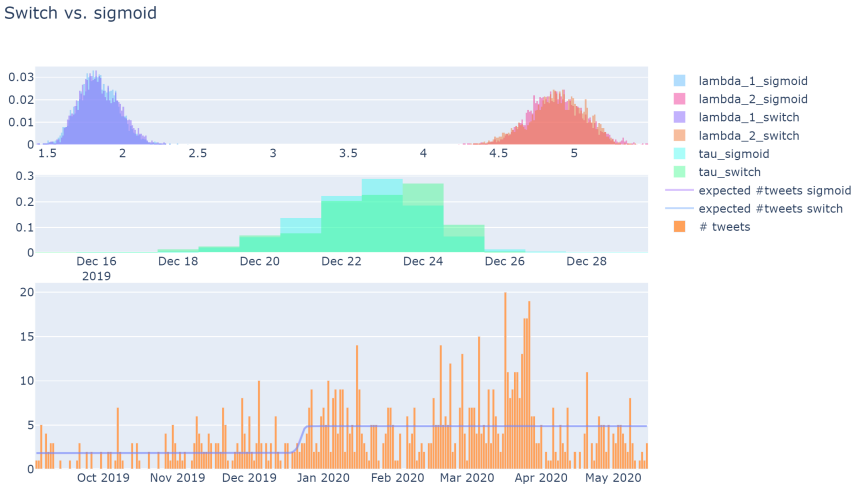

FIGURE 3.9: Multi lambda example

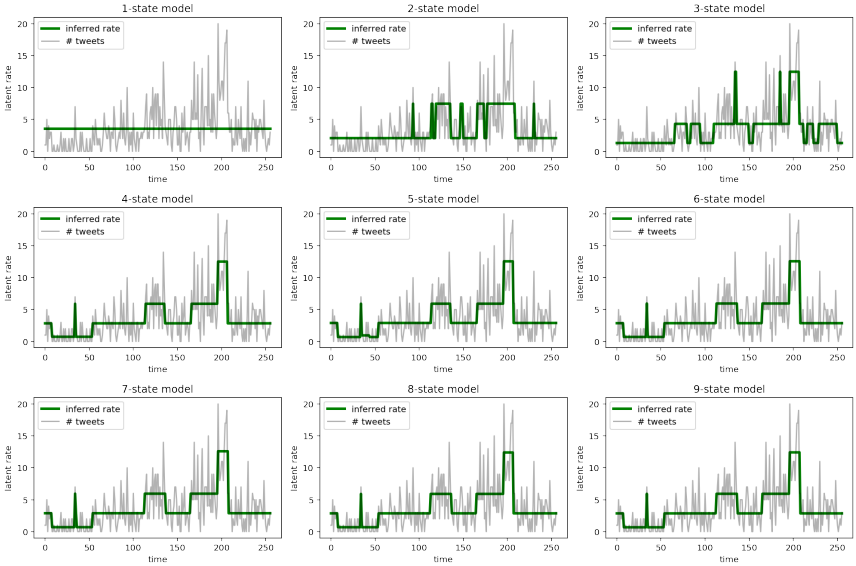

FIGURE 3.10: Example of multi-level model

Analysis

Back to our one change only model, we analysed the global behaviour of the users over time, we created a similar visualization of that one done in the timeseries section. We pick for every user the mode of their model parameters $\tilde{\lambda_1}, \tilde{\lambda_2}$, and compute the jump value, in the breakpoint $J = \tilde{\lambda}_{dif} = \tilde{\lambda_2} - \tilde{\lambda_1}$. This assumes a unimodal distribution of λ that we already have seen that is not always the case. Then we divide the jump value with the mean of the timeline (as $\lambda \sim L$ is the expected value after the breakpoint): $J' = \frac{J}{\bar{T}}$.

$$
\begin{aligned}
\bar{T} &:= \text{observed mean} \\
\tau &:= \text{distribution of the breakpoint} \\
L_1 &:= \tilde{\lambda_1} \\
L_2 &:= \tilde{\lambda_2} \\
J &:= L_2 - L_1 \\
J' &:= \frac{J}{\bar{T}}
\end{aligned}
$$

Similar to what we have done in the breakpoints section, we plot two different series, one for the positive values of J' and the other for the negative values. In 3.11 we see plotted the resulting timeline of $\tau * \tilde{\lambda}_{dif}$ for every type of user and in 3.12 the overall timeline for all the users. And as always we have to take into account the observation start date bias. As we saw previously, the bayesian model emphasizes the most important change of the user's timeline, and we don't pick smaller changes. The green lines represent users changing its activity to a higher level, and red ones to a lower level.

FIGURE 3.11: Frequency behaviour change over time

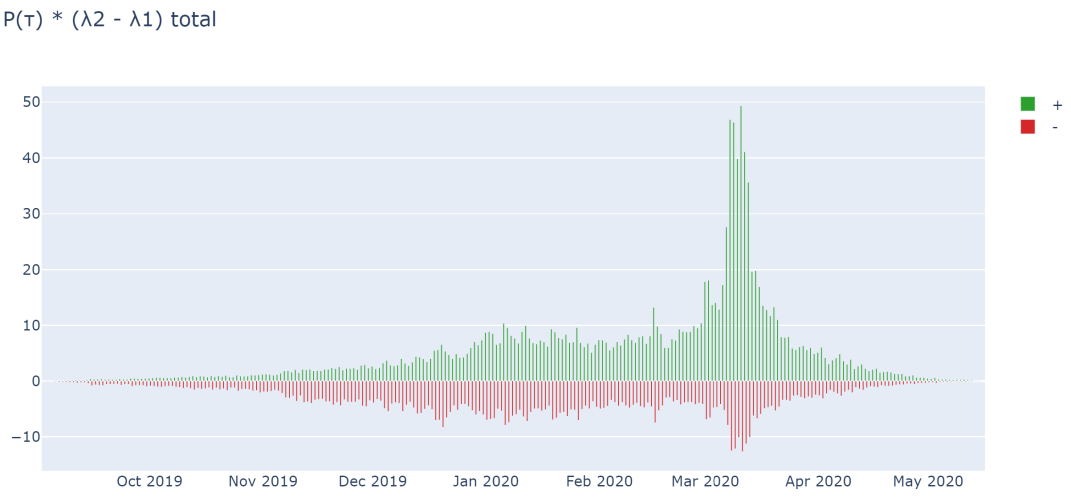

FIGURE 3.12: Overall frequency behaviour change for each user type

Chapter 4

Ego-significant-alter network analysis

This chapter can also be found at https://bernatesquirol.github.io/tfm-docs/networks.html

The ego-significant-alter networks are used in social network analysis to describe the *relational identity* of the individual. There is social cognitive science research that concludes that the self can be modelled in three components: *individual identity*, aspects of the self that are unique, *relational identity*, aspects of the self associated with other relevant individuals for the ego in the network, and *social identity*, definition of the self derived from one's membership in social groups (Constantine Sedikides, 2001). Some of the quantitative and qualitative properties of human ego-significant-alter networks are mirrored in online social networks (Arnaboldi et al., 2017a).

We present here a new way to construct an ego-significant-alter network for Twitter users. We analyse different aspects of these ego-significant-alter networks, as well as the network evolution through the COVID19 pandemic.

Concepts

Given a user u, let's define:

- $O(u)$: Others of u. The set of users that receive 2 or more retweets from u.
- $R_u(o)$: Number of retweets that receives a users $o \in O(u)$ from u
- $P(R_u) = P(\{R_u(o), \forall o \in O(u)\})$: Empiric distribution of $R_u(o)$ for all $o \in O(u)$.
- $O_s(u) = \{o \in O(u) | R_u(o) \in \text{Outliers}(P)\} O(u)$: Significative others of u. The subset of users for which its number of retweets from u represents an outlier of the $P(R_u)$ distribution

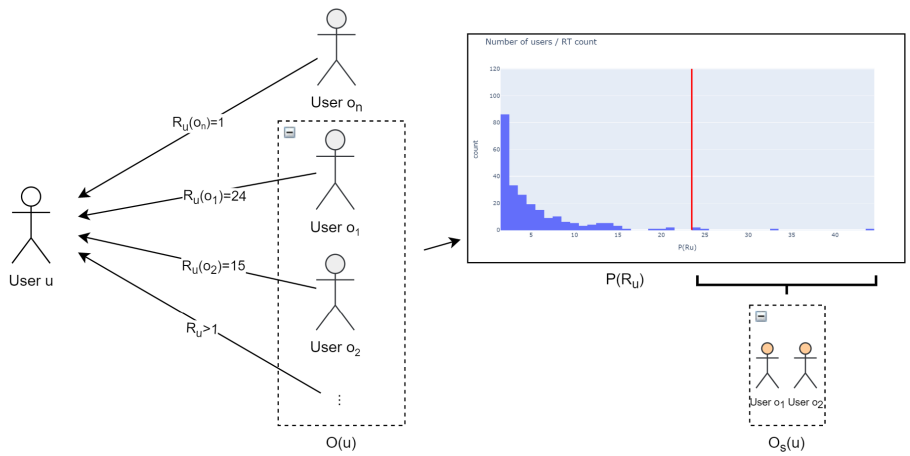

FIGURE 4.1: Significative other definition

The significative others set (O_s) are the users that receive a significant larger amount of retweets than the rest of others. Could be that the significative others set is empty, and thus we don't have an ego-significant-alter network for that particular user.

We will use a definition of Outliers that introduces the skewness into the computation (Hubert and Vandervieren, 2008). With

$$\begin{aligned} Q_i &:= i\text{th quartile P} \\ IQR &:= Q_3 - Q_1 \\ MC &:= \text{Medcouple(P)} \end{aligned}$$

We define:

$$\begin{aligned} \text{Outliers}(P) &= \{x | x \notin [Q_1 h_l(MC) IQR; Q_3 + h_u(MC) IQR]\} \\ h_l(MC) &= 1.5e^{-4MC} \\ h_u(MC) &= 1.5e^{3MC} \end{aligned}$$

In our case we will only use the upper bound ($\text{OutlierFrontier}(P) = Q_3 + h_u(MC)IQR$), as we have a long tail distribution, with the most frequently retweeted profiles in the right.

The *ego-significant-alter network* is the directed network with with the following nodes: the ego (u), and the outliers of its retweet distribution ($O_s(u)$). We will have a directed edge between u and v if $v \in O_s(u)$. Of course the ego (u) will have outgoing edges to all the other nodes in the graph. For every outlier $o_i \in O_s(u)$, we find its outliers ($O_s(o_i)$), and if the $o_{o_i} \in O_s(u) \cup u$ we will add an edge from o_i to $o_{o_i} = o_j \in O_s(u) \cup u$.

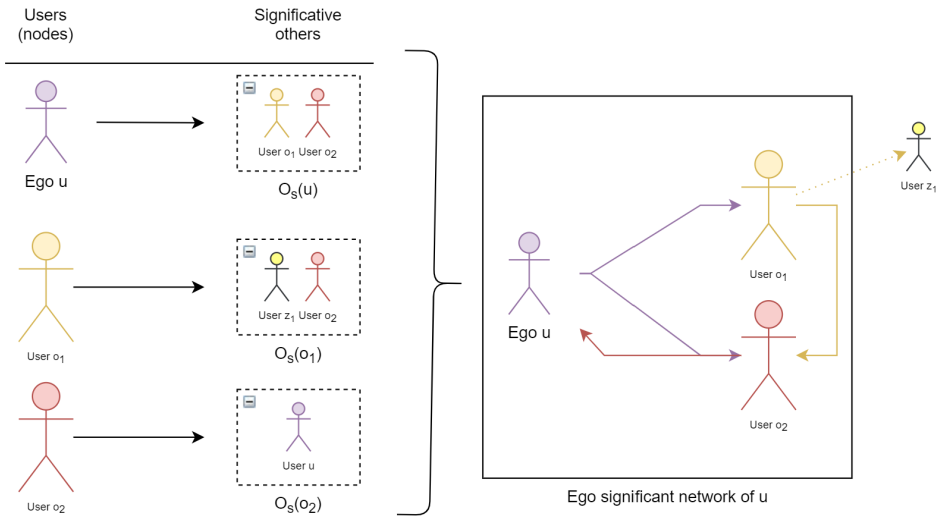

FIGURE 4.2: Schema of an ego-significant-alter network

Timeline networks analysis

We will construct a significant ego-significant-alter network for each user timeline, and we will look at overall properties of these ego-significant-alter networks. As we don't have a balanced number of users for each type, we will apply a bootstrapping technique in order to compare the distributions. This means that we will pick a random subsample of the distribution of the same size for each user group, compare among them and compute the mean of the comparisons. The plot of the histogram of the variables associated with the networks will be done in a similar fashion.

Degree

The first variable we will look at is the degree of the ego inside the network, that is, the total number of nodes of the network minus 1 (the ego). We can see how the politicians differ very much from the

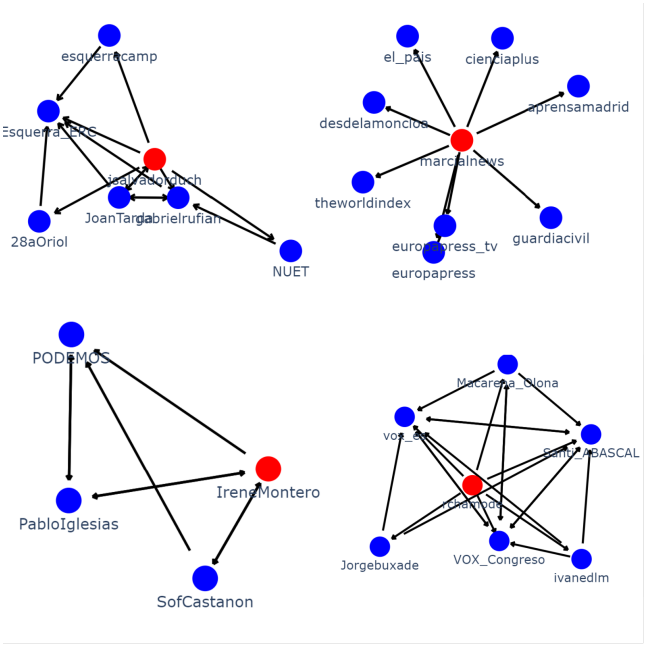

FIGURE 4.3: Examples of different ego-significant-alter networks: *Jordi Salvador Duch, Marcial Rodríguez, Irene Montero, Ricardo Chamorro*

other groups in 4.4.

$$\text{Degree}(G_u) = \#G_u - 1$$

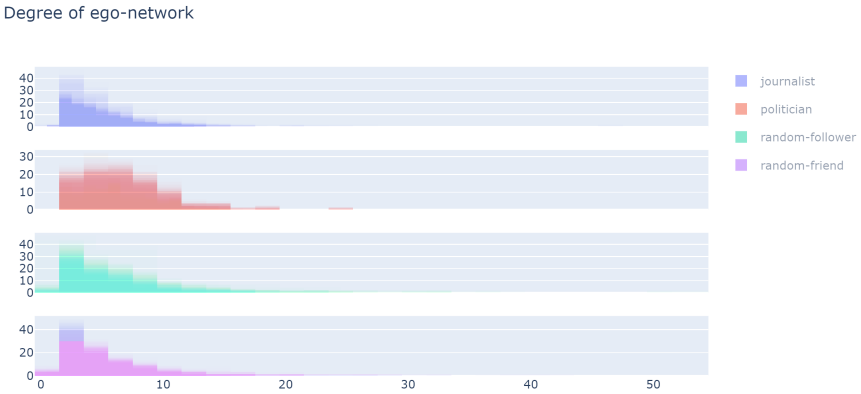

FIGURE 4.4: Degree of the ego-significant-alter network by type

P-values for sampled Kolmogorov-Smirnov test

	journalist	politician	random-follower	random-friend
journalist	0.013974	0.275131	0.117046	0.060879
politician	0.274419	0.041206	0.184481	0.257700
random-follower	0.118161	0.185019	0.008606	0.078555
random-friend	0.060738	0.259250	0.078369	0.008491

If we fit a Gamma distribution to the histograms of the different user types, we find the results plotted in 4.5 and 4.6.

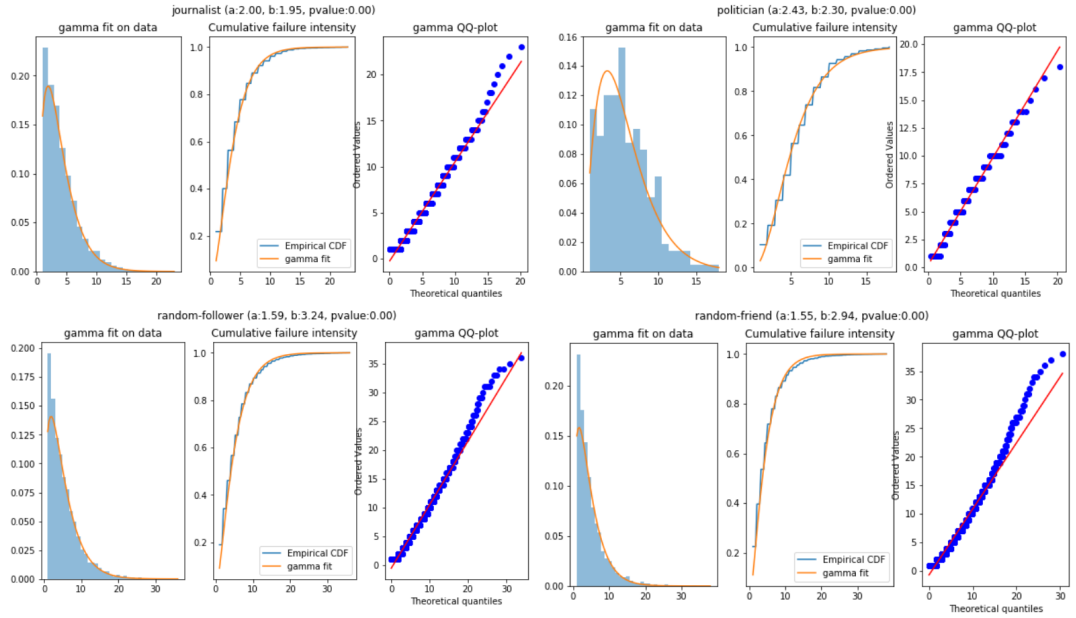

FIGURE 4.5: Fitting a gamma distribution to degree of the ego-significant-alter networks for different users

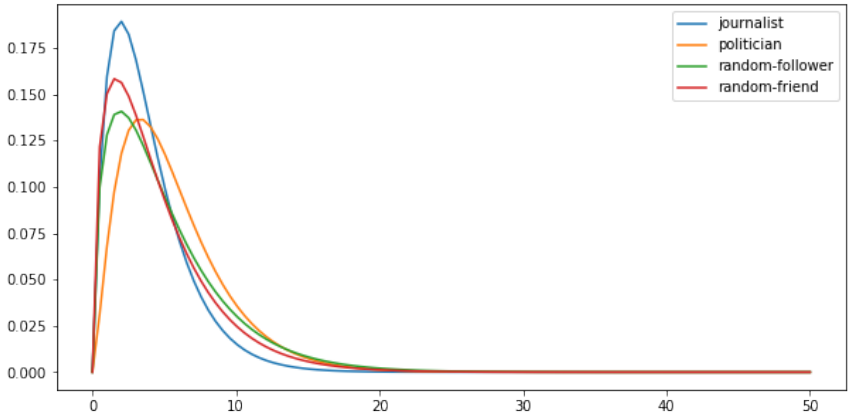

FIGURE 4.6: Resulting Gamma distribution

We can see how the plot of the fitted distributions looks like in 4.6. This means politicians tend to have more outliers than other user types. We can also see that the QQ-plots in journalists and random users are very similar, the Gamma distribution fails in the lower values of the distribution but fits correctly the long-tail.

Reciprocity

We define reciprocity as the number of outliers that have edges towards the ego divided by the amount of outliers.

$$\text{Reciprocity}(G_u) = \frac{\#\{v \in (G_u - u) | (v,u) \in \text{Edges}(G_u)\}}{\#G_u - 1}$$

P-values for sampled Kolmogorov-Smirnov test

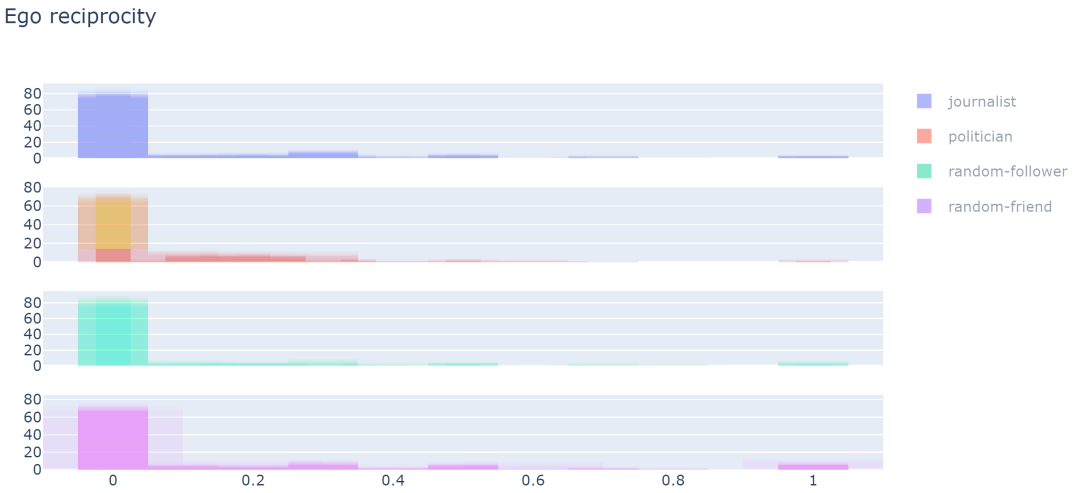

FIGURE 4.7: Ego reciprocity by type

	journalist	politician	random-follower	random-friend
journalist	0.010489	0.105550	0.040881	0.105750
politician	0.103113	0.034175	0.132675	0.157906
random-follower	0.040887	0.130812	0.006023	0.125938
random-friend	0.105946	0.157087	0.125920	0.007257

We see no clear differences between reciprocity in each type of user, as nearly 80% of the users don't have reciprocity at all from their outliers. That proportion is a bit lower with politicians.

Connections between outliers

We define outlier connectedness (or density) as the number of edges between outliers, divided by the number of possible edges between outliers.

$$\text{OutlierConnectedness}(G_u) = \frac{\#\{v \in (G_u - u) | w \in (G_u - u), (v, w) \in \text{Edges}(G_u)\}}{(\#G_u - 1) \times (\#G_u - 2)}$$

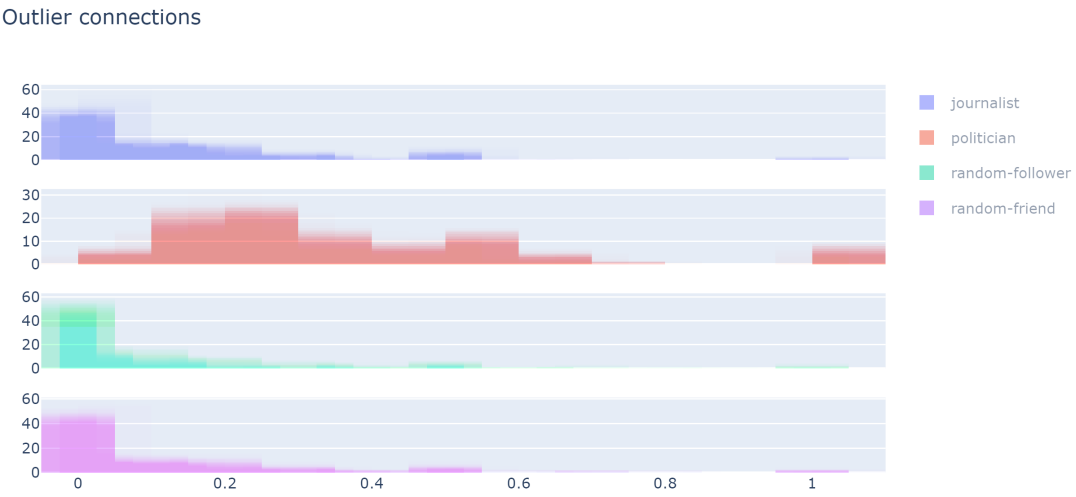

FIGURE 4.8: Connections between outliers

P-values for sampled Kolmogorov-Smirnov test

	journalist	politician	random-follower	random-friend
journalist	0.016117	0.469709	0.121097	0.059129
politician	0.469169	0.048047	0.558213	0.488959
random-follower	0.120751	0.559362	0.009477	0.076149
random-friend	0.059404	0.488519	0.076078	0.009497

Connections between outliers are way more common among politicians, as their outliers are probably other politicians in the same party.

Ego clustering

We will use the following definition of clustering for directed networks (Fagiolo, 2007):

$$\text{EgoClustering}(G_u) = c_u = \frac{1}{\deg^{\text{tot}}(u)\left(\deg^{\text{tot}}(u) - 1\right) - 2\deg^{\leftrightarrow}(u)} T(u)$$

where $T(u)$ is the number of directed triangles through node u, $\deg^{\text{tot}}(u)$ is the sum of in degree and out degree of u and $\deg^{\leftrightarrow}(u)$ is the reciprocal degree of u, and u is the ego.

P-values for sampled Kolmogorov-Smirnov test

	journalist	politician	random-follower	random-friend
journalist	0.014539	0.580375	0.110850	0.087493
politician	0.579250	0.049219	0.664144	0.630919
random-follower	0.111391	0.664238	0.008437	0.036767
random-friend	0.085648	0.631700	0.036911	0.008309

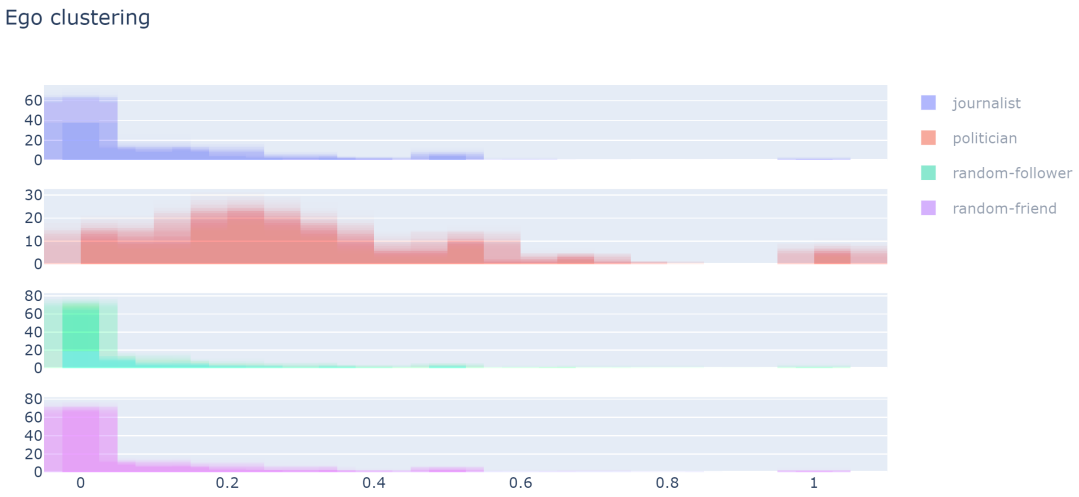

FIGURE 4.9: Clustering score for ego-node

In fact ego clustering and outlier connectedness are correlated because the ego has forward edges to all other nodes. The higher the number connections between outliers, the higher the number of triangles there will be through the ego (more clustering).

Evolving network

In this section we will measure changes in the ego-significant-alter network as activity changes. We will split user's timeline into the temporal chunks (τ_i) we found in the segmented regression analysis 2 in order to have some stationarity in the data. We will then compute where the outlier frontier is for this temporal interval τ_i. We will then divide it by the total retweeted amount ($\sum_{v \in P_{\tau_i}} R_u(v)$) during τ_i, having as a result the proportion of retweets (σ_{τ_i}) needed from a user to become outlier during τ_i. Then we will input this proportion (σ_{τ_i}) in the weekly distribution of retweet counts (P_j), week by week during the breakpoints, which will result in user's weekly outliers.

$$
\begin{aligned}
&\text{for } i = 1 \dots \#breakpoints: \\
&\quad \sigma_{\tau_i} = \frac{\text{OutlierFrontier}(P_{\tau_i})}{\sum_{v \in P_{\tau_i}} R_u(v)} \\
&\quad T_i \sim \text{Poisson}(\text{rate} = \lambda_i) \\
&\quad\quad \text{for } j = 1 \dots \#weeks: \\
&\quad\quad\quad \text{Outliers}'_{ij}(P_j) = \left\{ o | \frac{R_u(o)}{\sum_{v \in P_j} R_u(v)} > \sigma_{\tau_i} \right\}
\end{aligned}
$$

Within this framework, in order to measure the stability of the ego-significant-alter network, we will compute the following features for every week:

- **Number of outliers**: number of outliers that fall in the higher end of the frontier in the retweet counts per week.
- **Number of new outliers**: the number of new outliers that we have not seen before in previous weeks. This number will be higher at the beginning of the study, as we won't have seen that many profiles.
- **Jaccard index**: the Jaccard index is computed with the formula:

$$
J(\text{Outliers}'_{i,j}, \text{Outliers}'_{i,j+1}) = \frac{|\text{Outliers}'_{i,j} \cap \text{Outliers}'_{i,j+1}|}{|Outliers'_{i,j} \cup \text{Outliers}'_{i,j+1}|}
$$

Example

We will analyse the evolution of congresswoman Inés Arrimadas (Ciudadanos) ego-significant-alter network. In the period of time `15/08/2019-11/11/2019` we encounter, nearly every week, Albert Rivera, the leader of the party, in its outliers. Then `11/11/2019` Albert Rivera, announces its retirement from politics and steps down as the leader party, other outliers become more important, and we can see how the Jaccard index decreases and the new outliers per week increases in the following months. But we also observe that in lockdown period the behaviour of those variables changes a lot more than with the leader resignation.

Example of the network evolution of Inés Arrimadas:

Week	Jaccard	New	Total	Outliers
2019-09-30	0.5	0	3	[Albert_Rivera, CiudadanosCs, ...]
2019-10-07	0.666667	0	2	[CiudadanosCs, Albert_Rivera]
...				
2019-11-11	0	0	0	[]
2019-11-18	0	1	3	[davidmartinezg, carrizosacarlos,...]
2019-11-25	0	1	1	[JuanMarin_Cs]

Global analysis

We would like to see if our example's observed behaviour can also be found in a global scale for all the users we have in the dataset. If we look at the overall picture of the Jaccard index, we observe that, although it is very smooth (as having more users smoothes the mean and increases the variance), there are lower values in the end of December and in the beginning of March. If we observe those

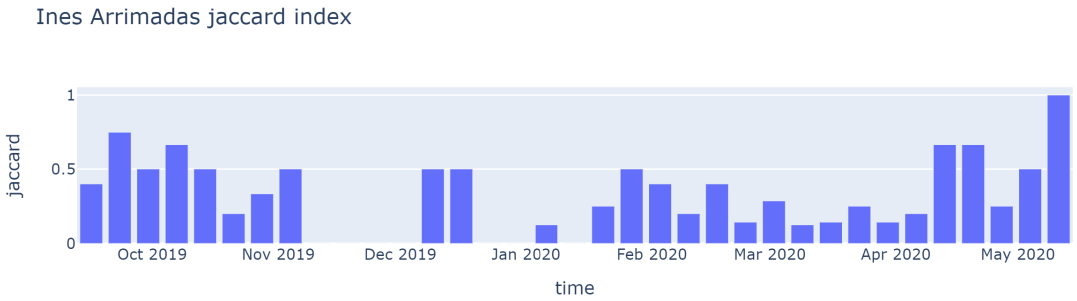

FIGURE 4.10: Jaccard index for a timeline of the congresswoman

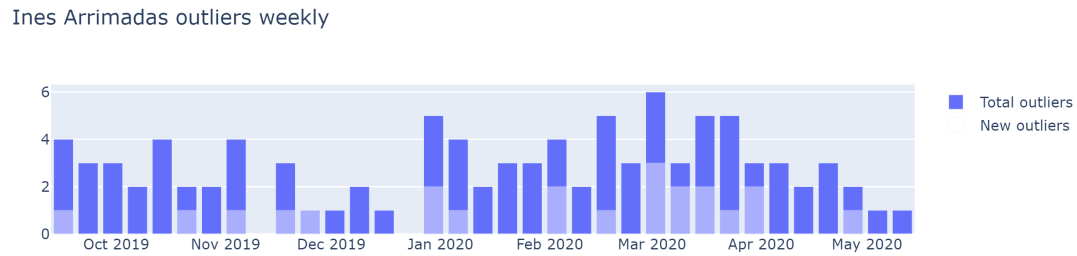

FIGURE 4.11: Number of weekly outliers of Inés Arrimadas

dates in 4.14, we see two different behaviours. While in December users have less outliers per week, probably because they retweeted less, in March there is a peak in the number of outliers of our users.

If we only look at politicians and journalist, this phenomenon is amplified, probably because we have less profiles and less variance. 4.15,4.16

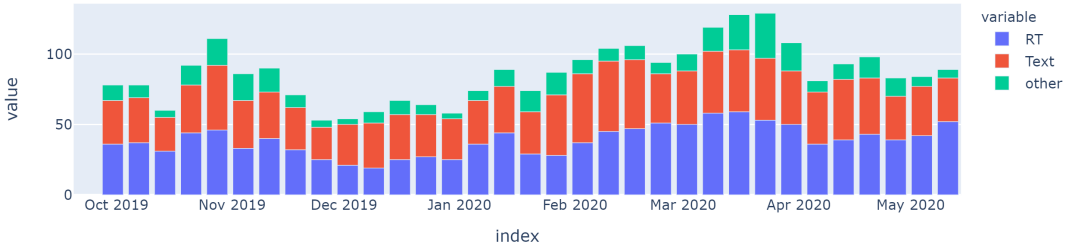

FIGURE 4.12: RT and text count of Inés Arrimadas for every week

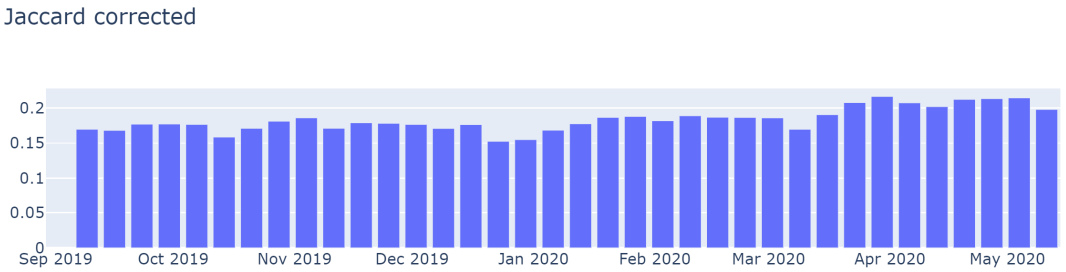

FIGURE 4.13: Global mean jaccard index

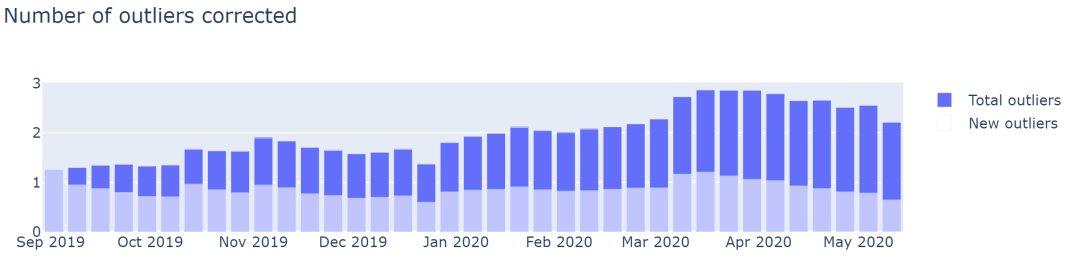

FIGURE 4.14: Global mean number of outliers

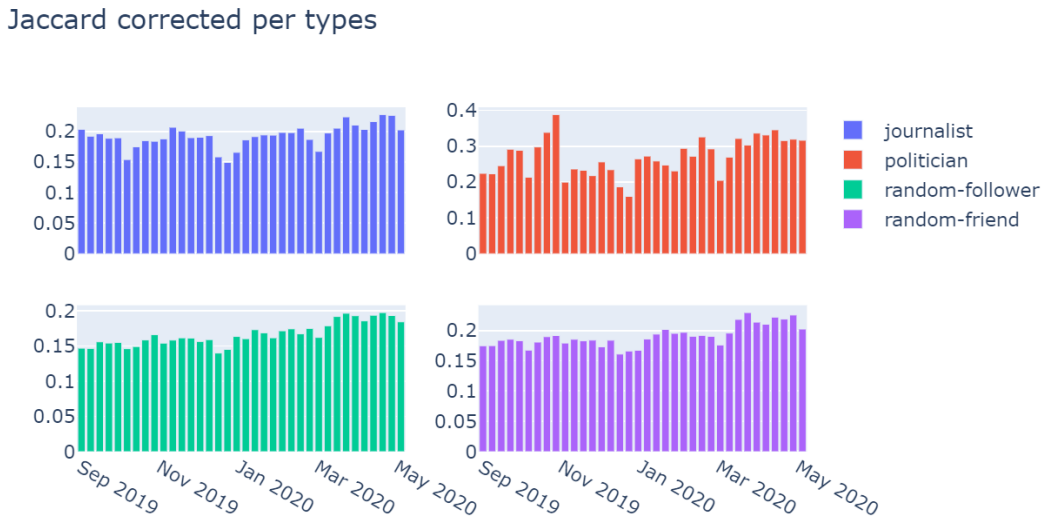

FIGURE 4.15: Global mean jaccard index by type

FIGURE 4.16: Global mean number of outliers by type

Chapter 5

Conclusions

In this work we were interested in modelling activity in Twitter during Spanish lockdown. To gain detail in the analysis we focused on single timeline models. There are several ways to model users behaviour, we investigated deeply a stepwise linear regression model and a bayesian switchpoint model. Although bayesian model focused only in the steppest change in user's behaviour and the stepwise linear regression took into account other behaviour changes, with both models we found a notable changes in the period where the users where submitted to control measures by the Spanish government. We also created and analysed the features of the *ego-significant-alter networks* in Twitter from the retweet distribution of the user, and finally investigated the evolution of these networks through activity-stationary periods.

We also characterized four Twitter profile types: *politicians*, *journalists* and *random users*. We saw how activity patters inside lockdown situation change between these user types and also in the ego-network chapter we saw that politicians ego-networks are statistically very different from random users and journalists.

Further research may include a deep analysis of other bayesian methods we saw (sigmoid, multilevel), other user's characterizations (number of followers,...), and also a deep analysis on the stability of the ego-significant-alter networks.

Bibliography

Arnaboldi, Valerio et al. (2017a). "Online Social Networks and information diffusion: The role of ego networks". In: *Online Social Networks and Media* 1, pp. 44 – 55. ISSN: 2468-6964. DOI: https://doi.org/10.1016/j.osnem.2017.04.001. URL: http://www.sciencedirect.com/science/article/pii/S2468696417300150.

Arnaboldi, Valerio et al. (Nov. 2017b). "Structure of Ego-Alter Relationships of Politicians in Twitter". In: *Journal of Computer-Mediated Communication* 22.5, pp. 231–247. ISSN: 1083-6101. DOI: 10.1111/jcc4.12193. eprint: https://academic.oup.com/jcmc/article-pdf/22/5/231/22317718/jjcmcom0231.pdf. URL: https://doi.org/10.1111/jcc4.12193.

Britt, Brian (Jan. 2015). "Stepwise Segmented Regression Analysis: An Iterative Statistical Algorithm to Detect and Quantify Evolutionary and Revolutionary Transformations in Longitudinal Data". In: pp. 125–144. DOI: 10.1007/978-3-319-18552-1_7.

Constantine Sedikides, Marilynn B. Brewer (2001). *Individual Self, Relational Self, Collective Self*. Addison-Wesley Data Analytics. DOI: 10.4324/9781315783024.

Davidson-Pilon, Cameron (2015). *Bayesian Methods for Hackers: Probabilistic Programming and Bayesian Inference*. Addison-Wesley Data Analytics.

Fagiolo, Giorgio (2007). "Clustering in complex directed networks". In: *Phys. Rev. E* 76 (2), p. 026107. DOI: 10.1103/PhysRevE.76.026107. URL: https://link.aps.org/doi/10.1103/PhysRevE.76.026107.

Garziano, Giorgio (2017). *ARIMA models and Intervention Analysis*.

Hubert, M. and E. Vandervieren (2008). "An adjusted boxplot for skewed distributions". In: *Computational Statistics Data Analysis* 52.12, pp. 5186 –5201. ISSN: 0167-9473. DOI: https://doi.org/10.1016/j.csda.2007.11.008. URL: http://www.sciencedirect.com/science/article/pii/S0167947307004434.

TFP (2020a). *Bayesian Switchpoint Analysis*. URL: https://www.tensorflow.org/probability/examples/Bayesian_Switchpoint_Analysis.

— (2020b). *Multiple changepoint detection and Bayesian model selection*. URL: https://www.tensorflow.org/probability/examples/Multiple_changepoint_detection_and_Bayesian_model_selection.

Zhang, Tianyang et al. (Mar. 2016). "A Multiscale Survival Process for Modeling Human Activity Patterns". In: *PLOS ONE* 11.3, pp. 1–9. DOI: 10.1371/journal.pone.0151473. URL: https://doi.org/10.1371/journal.pone.0151473.