UNIVERSITÀ DEGLI STUDI DI TRENTO

Department of Information Engineering and Computer Science

Bachelor's degree in
Computer Science

Final Dissertation

Life event detection on social media

Supervisors

Alberto Montresor

Daniele Miorandi

Student

Nicolò Pomini

Academic year 2017/2018

Acknowledgments

I dedicate my bachelor's thesis to my beloved grandmother.

I want to thank all the people that have supported me during this three years, especially my family, my aunt Tiziana and my girlfriend Arianna.

A special thanks goes to those that made possible this thesis: my university supervisor Alberto Montresor; Daniele Miorandi, who guided me every day during the internship and the drafting of the thesis; all the U-Hopper family, to gave me the chance to work in such a fantastic environment. In alphabetical order: Carlo, Christian, Daniele, Diego, Federico, Mozhdeh and Rossana.

Contents

Abstract

Nowadays digital marketing heavily relies on the segmentation of an audience at both aggregated and at the single user's level. A life event is a very important event occurring in someone's life, such as a wedding or the birth of a child. It may have a relevant impact on the person's life, guiding many of her choices in terms of purchases, services and mobility. If identified, life events can represent a huge business opportunity and this is why they are extremely relevant for marketeers.

The best and easiest places to look for life events are social media: they have become one of the most used means of communication, where people share daily news, their opinions and personal life updates. The catchment area of these services has grown exponentially by size in the last few years: for example Facebook has more than 2.2 billion monthly active users as of January 2018[1].

Life event identification on social media is an open problem; there is no software solution that detects this kind of information into a social timeline. Consequently, the research question concerns the feasibility of the problem.

This thesis was developed at U-Hopper[2], a small enterprise specialized in big data analytics solutions, and the detection of life events represents for it a concrete need and a realistic future business.

The proposed solution goes a step forward with respect to the state of the art, offering a detection method based on the entire user's timeline on social media. The system that is presented uses a hybrid technique to perform its scopes: a machine learning-based method to make post classification, and a model based approach to analyze the timeline. The system is designed to work with different social networks. It does not use any text or picture analysis technique, but a method based on the Wikipedia entities, to understand the content and the meaning of photos and texts. A prototype was developed and evaluated, to understand how what has been designed behaves in reality. The machine learning part reaches the benchmarks of the state of the art, making a post classification with both precision and recall measures around 0.9, while the timeline analysis offers an accurate estimation of the period when the life event occured (in 64% of cases the life event is identified in a week range), and does a good discrimination on those people who have not lived any life events.

Finally, a new open dataset about life event classification on Twitter has been released, containing more than 5000 samples written by 45 users, with the purpose of simulating a real social timeline.

[1] as reported by Wikipedia, `en.wikipedia.org/wiki/Facebook`

[2] `u-hopper.com`

1 Introduction

This thesis work was developed during a 4 month internship at U-Hopper, with which I was able to combine a work experience in a company with the writing of the bachelor's thesis. U-Hopper is a research-intensive deep-tech SME, headquartered in Trento, providing big data-enabled solutions and technologies for the government, retail and manufacturing sectors. U-Hopper has received numerous awards for its innovative solutions, including, among the others, the Lamarck prize (2013), a EC Seal of Excellence (2015), the Innov@Retail prize (2016) and a nomination for the 2017 EC Innovation Radar Awards. Life event detection can be a significant business for the company, because it is already into the user profiling world, with a product called Tapoi[3], but above all because detect a life event into someone's life can bring many opportunity with banks, insurances, estate agents and many other businesses, which are constantly looking for such a meaningful information like a life event can be.

According to Cambridge Dictionary[4], a *life event* is *"a very important event in someone's life, such as marriage, the birth of a child, or the death of a family member"*. These kind of events are quite rare in a lifetime, and they may bring with them a big charge of stress, and big changes for those who live them. Furthermore, they do not last only for the day they happen, but there is a medium or long period that is influenced by the event: for example, a pregnancy lasts for 40 weeks, or a wedding is usually organized in several months; but also a negative life event, like the death of someone dear, causes bad feelings for a period more or less long for those who live it. The Holmes and Rahe stress scale [10] puts in relation several life events for the load of stress they cause: on a scale from 0 to 100, the most stressful event for an adult is the death of a spouse, that scores 100, but also positive events are part of this list, such as marriage, with the score of 50, and a pregnancy, with 40. In addition, a life event can be followed by another one life event: for example in Italy, 80% of births occur within a marriage in 2012[5].

Therefore, a life event is a signal of many factors in who lives it. If it is detected in time, it can be a great opportunity for many entities, such as banks, insurance companies and other kinds of ad-hoc marketing campaigns. The life events have a deep effect on the individual's spending habits and purchase patterns. For example, according to the results[6] of a study made by the University of the West of England about the relationship between life events and travel behaviour [5], households are more likely to change the number of cars at the time of life events: the "*Birth of a child increases likelihood of a non-car owning household acquiring a car and increases likelihood of a two-car owning household relinquishing a car. This suggests households seek a one car solution when having children*".

The identification of a life event into social media contents is a *machine learning* problem, and to be more precise, a classification one. Machine learning is a field of computer science which deals with allowing a computer system to "*learn with data, without being explicitly programmed*" [21]. It can be applied in many contexts, such as taking decisions, or make optimizations, forecasts and predictions. Nowadays a human being faces itself with machine learning in everyday life: home assistants, security surveillance, music and shopping suggestions, customer services are strongly powered by artificial intelligence. These services relies on data to learn how to work as good as possible: they are trained with samples of data similar to what they expect to receive by their users: the more accurate, exhaustive and in large quantities they are, the better the system learns. Therefore, data have a very central role in machine learning problems.

A classification task has the goal of assigning a belonging class to a given object. The input is composed by a tuple of *features* that characterize the object, usually made by numbers, and the output is a categorical variable, such as a "yes/no" label. In other words, it can be seen as a mathematical

[3] `www.tapoi.me`

[4] `dictionary.cambridge.org`

[5] `www.istat.it/it/files//2015/02/Avere_Figli.pdf`

[6] `travelbehaviour.files.wordpress.com/2014/06/lttb_carownbriefingnote_16-june.pdf`

function, that maps a vector $\boldsymbol{x} \in \mathbb{R}^n$ to an answer $y \in C$

$$f : \mathbb{R}^n \to C$$
$$f : \boldsymbol{x} \mapsto y$$

where C is a set of possible categories. A famous educational example of this kind of problem is the Iris flower classification[7], where the input $\boldsymbol{x}$ is composed by the sepal and petal widths and lengths in cm of the flower, and $C = \{\texttt{Iris setosa}, \texttt{Iris virginica}, \texttt{Iris versicolor}\}$. For each sample composed by 4 measures the belonging class is predicted.

The background of live event identification on social media consists in a huge stream of documents, called *posts*, each one written by a specific user and containing text, images, external links and other attachments, with a date of publication. Every user can share, comment or like someone else's post. A *timeline* is the list of posts written by a single user, sorted by decreasing date of publication. These data are accessible through the Web (in some cases the author's authorization is necessary to get them) and contain many information. There is an endless number of works that use social media data for various purposes, and in this case they are used to detect events.

1.1 Research objectives

Life event detection on social media is still an open problem; in fact, such a commercial solution to be integrated into a social system of some commercial, banking or insurance entity does not exist, not even an accademic pubblication that goes beyond a classification problem. Therefore, the reseach question is the following: *is it feasible to detect the occurrence of life events based on social media user activities?*

The goal of this thesis is to find a way to understand whether a person has lived a life event, or if she is about to live it, observing her activities on social media. Nowadays, social media are widely used all over the world: according to the Global Digital 2018 report[8], 3.196 billions of people use social media every month, sharing messages, photos and other contents that are also about their private life. They are used to deliver messages of congratulations or support, to search for information from users who have experienced certain situations or to make announcements of news or changes. Consequently, social media are one of the best platform to get reliable information about people in an easy, fast and free way.

The idea is to design a hybrid system that combines a data driven approach to analyze each post, and a model driven solution to inspect a user's timeline, in order to find out whether the user has lived a life event according to what she shared on social media. This work is focused on the detection of two life events, marriages and births of a child.

1.2 Detection vs prediction

A user's timeline is composed by all the posts published from the time she signed up to the social media up to today. From this data is possible to *detect* an event that happened in the past. Event detection is the identification of an event that occurred, into a stream of data, and now it is finished or at most it is still ongoing. Another possibility is to *predict* an event, understanding whether the event can occur in a more or less near future. Event prediction can be done in several ways, for example observing what happen in the user's past, or in timelines of users with similar age, location and preferences, or using some logical model: for example, a married person will likely have a baby in her lifetime.

Of course, the two things have a very different meaning. With event detection is possible to understand what a person lived in the past, what she might searched or needed at the time of the event, and which are her usual needs: for example, if it is known that a user had a child, her possible future vehicle will be a family car, or her new house should probably have at least two bedrooms. With event prediction is possible to anticipate some user's choices or needs, giving her some ad-hoc

[7] `en.wikipedia.org/wiki/Iris_flower_data_set`

[8] `wearesocial.com/it/blog/2018/01/global-digital-report-2018`

offers and advertising campaign: for example, if it was known that a user would probably have a child in the near future, she would likely look for a new bedroom furniture or a new bigger car.

From the machine learning point of view there is no big differences between event detection and event prediction. In both cases the usual patterns that indicate that the event is about to occur are the same. The only difference is that the detection is based on facts, and so it is possible to say whether the obtained result is correct or not, while prediction is based on suppositions, which are not always verifiable. This thesis is focused on life event detection, so on what the user has lived.

1.3 Outline

This thesis work is organized as follows: in Chapter 2 the state of the art is introduced, explaining what is missing for this purpose and what has to be changed. In addition, the data situation about life event will be shown, unfolding a solution that is different from the standard text classification. In Chapter 3 the design solution is presented, with a particular focus on the logical behind the system and on the used algorithms. Chapter 4 is about the implementation procedure and the performance evaluation of a prototype of the system, and finally, in Chapter 5 some final remarks and observations for future developments are presented.

2 State of the Art

In this chapter, the state of the art in dealing with life events on social media is presented, starting with what the research has done so far to detect them, and explaining also the current situation with data in this field.

Event detection using social networks is a common practice. The literature offers many example of event detection on a global scale based on the analysis of social media contents, such as real-time earthquake identification based on tweets [20], breaking news discovery in Twitter [11, 19], or big gigs recognition observing what is posted on Flickr [15].

Event prediction using social network contents is also pretty common: the most striking example of the last few years are the 2016 american elections. In fact, while many of the official polls made by the most famous american newspapers and televisions had always forecasted Hillary Clinton as the winner, social media reactions had been increasingly in favor of the Republicans during the election campaign[9], and the rest is history. Other cases of event prediction using social media are, for example, movie box-office [1] or Oscar-winner forecasts.

There is much less literature about life-event detection on social media, and it is almost completely focused on post classification. Another important issue about life events is related with data: it is hard to find it, and in a machine learning scenario like this problem data play a key role.

2.1 Life event detectors on social media

Almost all the models proposed by the literature are classifiers that take a textual post as input and give as output a label, indicating whether the text talks about a life event. Each classifier is specialized to a single life event, for example a detector of posts about weddings. Some models are focused on a single post at a time, while few others [3, 16] consider also the conversation linked to the main post to understand the context and the importance of the speech. The feature extraction is done mainly in two ways: working only with the text itself, using bag-of-words or bag-of-N-grams [2, 8, 13], or using a semantic analyzer, to combine the previous method with semantic information, such as sentiment score, formality with which the text is written, entities and topics contained, etc [12]. In addition to that, some models consider also *external* features, like user's features (number of friends, age, location, post frequency, etc.) or post success [9].

All these models have several weaknesses for the purpose of this thesis, which are briefly described

[9] techcrunch.com/2016/11/10/social-media-did-a-better-job-at-predicting-trumps-win-than-the-polls

now. First of all, they just give a piece of information – saying whether a post is about or not to a life event – not going further with it. Secondly, there is no consideration of the user's timeline, of her behavior on social media, and no comparison with other posts published by the same user: in fact the samples used in papers are fetched more or less randomly from the social networks, without any user profiling intent, and consequently each analyzed post is completely disconnected with the other ones. Furthermore, analysing contents in this way gives no insight about how much the event may have lasted, and it is not understandable whether the event was in the past and now it is over, it is just passed, it is occurring or it is coming in the near future. Another point worth of noting is that a post related to a life event may not be concerned with the author (e.g. participate in a wedding instead of getting married), or a post classified as about a life event could be a false positive: in both cases the decision to say that this person has lived a life event is based only on a single content, and this is a result of not consider the user's timeline. Thirdly, only text contents are considered: however, according to Mark Zuckerberg, CEO and founder of Facebook, "*Most of the content 10 years ago was text, and then photos, and now it's quickly becoming videos*"[10]. Not only are posts composed by more than only text, but many of them have no text at all. Furthermore, the meaning of an image or a video can give a strong clue to understand what the post is about: for example, an image containing a pink ribbon can easily be interpreted as a female baby announcement, without even look at the text. Last but not least, every model cited above uses Twitter contents only: this platform allows to share texts with at most 280 characters (until late 2017 the limit was 140) and it is used more for business purposes rather than personal life sharing. On the other hand, most of the contents on this social network are free to access and to obtain for non-commercial purposes, without having an explicit user authorization.

2.1.1 A step further

The literature offers a model that goes a step further of those highlighted above: the work made by Cavalin, Gatti, Pinharez for the *IBM InfoSphere BigInsights* platform [4]. Starting from life event classification on a sample of tweets, it performs a user entity matching on an existing database of clients, aimed to decide which is the best approach to offer them some services or some products. Once a user that texts about a life event is identified, her information are used to match her into a database of clients.

In common with our model there is a big move forward from a simple classifier, but with an important difference: it solves the reverse problem of ours. Its goal is to get a list of users that posted about a life event in a given time window, while ours is to understand if a user has posted about a life event and in which time period. In this paper a wide range of posts written by many authors is taken to analyse the presence of a life event, not considering user's timeline at all, while our model is focused on a single user analysis. This brings all the problems highlighted above in the previous section. Furthermore, also this model is concentrated only on Twitter contents.

2.2 Other interesting methods

In addition to the previous models, the literature offers several works that are methodologically interesting. They are briefly described now.

As already mentioned, life event detection is a branch of event identification. The core of the business is therefore the identification of events from a stream of time-stamped documents coming from social networks. In concrete, the job made by Vavliakis, Symeonidis and Mitkas [23] organizes documents from various sources according to the event they describe, assigning to the event an importance, while the one made by CC Chen, MC Chen and MS Chen [6] tracks how much activity is related to an event from a global stream of documents. However, also these two models work on global scale, not analyzing a specific user, and they have an implicit issue: they consider the importance of the event based on the *noise* the event brings with him. The more people talk about an event, the more important it is. Therefore every global event, such as the Olympics, will be classified as much more important than a wedding or a birth of a child.

Another interesting study is the one proposed by Li, Ritter and Jurafsky [14], a model-driven

[10] `www.fastcompany.com/3057024/mark-zuckerberg-soon-the-majority-of-content-we-consume-will-be-video`

system to infer user's attitudes or preferences reasoning over user's attributes and social network graphs. It builds, for every person taken into analysis, a series of predicates for her attributes (location, gender, education), relationships (married, friends) and preferences (what she likes/dislikes), which can be used to detect important events on her timeline or in friends/relatives social profiles. This work is interesting because is the only one that is not completely driven by data and machine learning, but it offers a logical model, and also it combines multiple social networks (Twitter and Google+). On the other hand, of course, this model is not designed to deal with life events.

A last interesting study is about topic sentiment analysis using the hashtag graph in Twitter [24]. It demonstrates that hashtags offer additional information to texts, classifying tweets and users in categories: the use of a specific hashtag may be connected with a specific life event. This kind of study, however, has more sense on global scale analysis, like searching for users who show interest in a specific topic: it may be not so useful in a context of life event detection, which is concentrated on a single-user analysis.

2.3 The dataset problem

One of the biggest issue in dealing with life events is the lack of data. For an artificial intelligence powered solution a great amount of data is needed, to train a machine learning system and to evaluate it. Furthermore, very few benchmarks exist, so it is hard to understand which is a good result to reach and what are the minimum expectations.

Many users do not share with the world what happens to them in their private life, in fact most of the posts on social networks are not about users' personal life: according to [16], in a set of 20000 samples, only 1.7% of them are concerned with some life event. There is also a strong subjectivity in sharing a life changing event: every user has her way to announce a personal news and it's hard to find a pattern that identifies a life event announcement. For these reasons, is not easy to build a dataset big and reliable enough to train the system as good as possible. Furthermore, among the papers cited above, only one [9] published the dataset build for the experiment[11].

The literature presents two different ways to fetch and label the data. Some authors prefer to fetch only contents that contain specific keywords (such as *"bride"* for marriage) [9, 12], labelling each content *by hand* as about or not to the life event itself; some other search for contents randomly, labelling it in a more automatic way, considering a text related to the life event if it contains at least one keyword [7, 8, 16]. The first method requires a bigger effort than the second one, because data is selected before the download, and once downloaded is analyzed by a human, who assigns to each post a relation with a life event. On the other hand, bigger work leads to better precision: a human classification is not based on the content only, but also on the meaning of the text. Of course this approach for huge datasets (those used into the cited papers with *automatic* labelling space from tens of thousands to millions of samples) is not recommended.

In case more than one life event is taken into consideration, this kind of classification becomes a multi-label classification. For example, if marriage and birth of a child are considered, a tweet that speaks about a wedding will have the *marriage* label setted to `true`, and the *having children* label setted to `false`. According to [2] and to Wikipedia[12], the easiest way to perform this type of classification is doing a set of binary classification: to do that, a dataset for each life event is necessary. Another problem is represented by how many languages the classifier wants to support: for each of them is necessary a satisfying number of samples, to allow the classifier to learn any patterns or idioms related to each language. In conclusion, if the methodologies exposed by these works were followed, a dataset for each life event and for each language would be necessary. For example, a system that classifies posts about marriage and birth of a child written in english and italian, would need 4 datasets: marriage-english, marriage-italian, child-english and child-italian.

The solution chosen to overcome the language issue was inspired by Tapoi, the user profiling software made by U-Hopper, which works with Wikipedia[13] contents to understand what a user has talked about in her social profiles [22]. The idea is to use Wikipedia articles to identify words, topics

[11] `reellives.net/rl-data/uploads/2015/06/a692044.csv`
[12] `en.wikipedia.org/wiki/Multi-label_classification`
[13] `www.wikipedia.org`

and concepts inside a phrase: each post will contain a set of articles, called *entities*, and categories, called *topics*, which will be the features for the classification. Instead of using the classical bag-of-words with texts, a *bag-of-Wikipedia-entities* will be applied. The procedure is the following: firstly, a dataset in a given language has to be provided; secondly, each post of the dataset has to be analyzed by some semantic analyzer, to extract Wikipedia entities and concepts; thirdly, a machine learning classifier is trained using Wikipedia entities and concepts as feature. Once the classifier is ready, the procedure of analyzing posts with a semantic analyzer must be done with every post to classify. In case a post is written in a language that is not the one of the training set, its entities and concepts have to be translated into those belonging to the Wikipedia version of the training set language.

In addition to that, is possible to use Wikipedia entities not only in texts, but also with images. In fact, there are several image analyzers that are able to find out entities into pictures. In this way is not necessary to create a training set also for images, but it is enough to use the same dataset of texts.

3 System design

In this chapter the design process is presented, starting from the requirements and the description of the working logic. After that, a more schematic view is provided, explaining how the system is designed and how the components work and communicate each other. At the end, the algorithms used to compute the outputs are presented and explained.

Seen as a black-box – in Figure 3.1 – the system takes two inputs: all the user's activities in the social networks she is registered in, and a life event to search for, and it returns as output a yes/no answer to the question "*has this user lived this life event?*", with a time reference estimation attached.

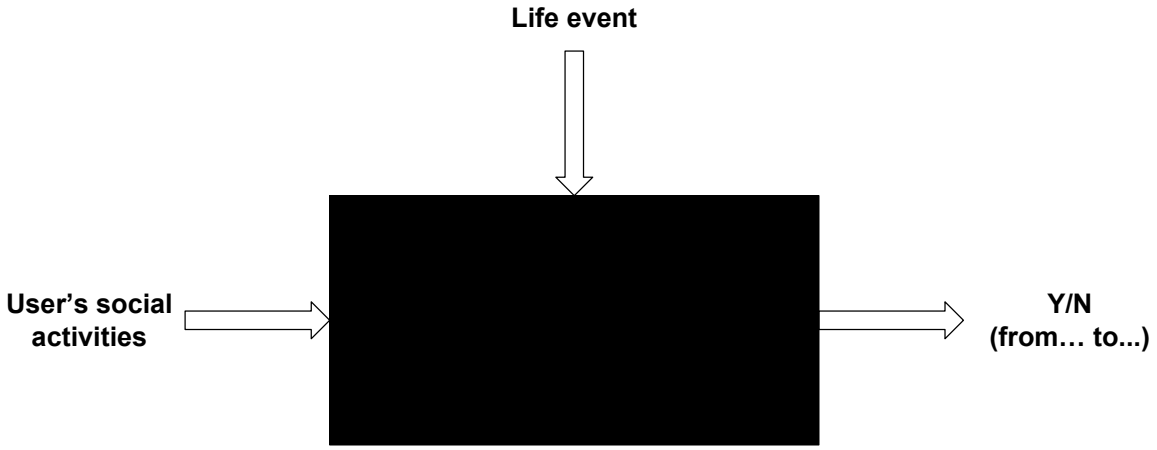

Figure 3.1: A black box view of the problem.

3.1 Requirements

The main requirement is that the software solution has to be able to decide whether the user taken into analysis has lived a given life event. The decision has to be taken by analyzing her activities on the social networks she is registered in. The time required to perform this computation should not be long, although data come from an online stream of contents.

Other requirements expect to let the addition of a new user to analyze, or to add and remove a social media account to an already existing user.

The set S of life event that can be searched for should be defined a priori, and it will be:

$$S = \{\texttt{Getting married}, \texttt{Having children}\}$$

3.2 Logical components

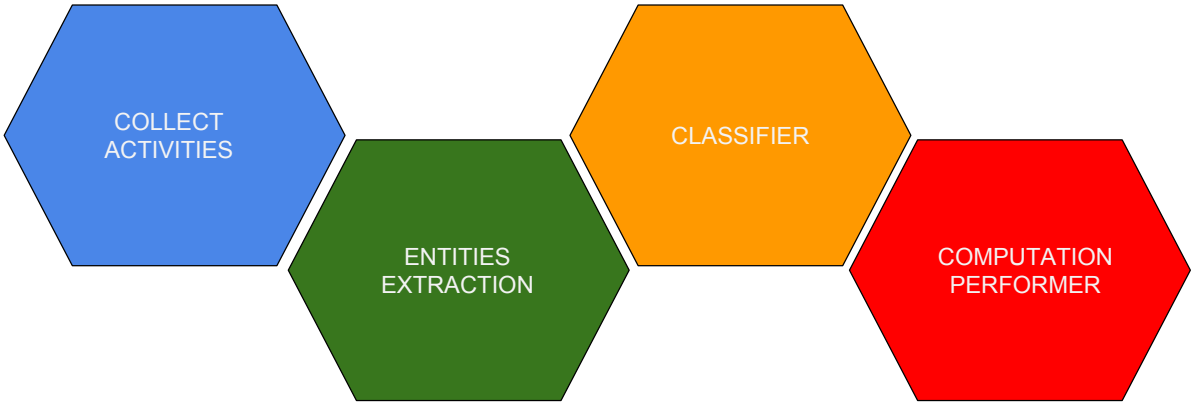

Figure 3.2: The solution design in a nutshell.

This section describes the specifications of the logical functionalities of the system. Each functionality is a group of actions whose purpose is to carry out a section of the job required to go from row data - a user identifier - to a piece of information - the answer to the question *"Has this person lived a life event?"*. The steps are the following, as shown in Figure 3.2:

1. Collect activities from social networks.
2. Extract semantic entities from social network contents.
3. Classify each content.
4. Decide whether the user has lived the life event relying on his/her contents.

Each step relies on what the previous steps do: for example, to execute the third one, the first and the second ones must have already been executed at least once. The number of executions required to perform the entire computation from scratch is probably not the same for each part: in fact, the first two steps are coupled together, and they need to be performed a consistent number of times before fetching an entire user's timeline. Once the whole timeline is downloaded, just a single call for the third and the fourth steps is needed to complete the computation. In the following subsections the logic of the proposed methodology is illustrated, providing a textual description of what each component has to do, of the input, the output and the parameters.

3.2.1 Collect activities

This part has the goal to fetch user's contents on social networks, using the services for developers offered by the official APIs of each social network. The data of interest are some pieces of user's information, such as name, birthday, number of friends/followers, subscription date, her location and language, and of course all her contents: all the posts she wrote, with attachments, external links and the publication date, with also some other useful metadata, such us the number of likes, replies and shares. Some social media – like Twitter – offer a free API plan with some restrictions over time: for example, to retrieve user's tweets is possible to make 1500 requests every 15 minutes. In case the limit is reached, this part should suspend itself waiting for a new time slot: for this reason is suggested to run this function in a dedicated thread or process, in order not to cause a bottleneck.

As input this functionality needs a series of IDs that identify the user among all the social networks she uses, and for each social platform is necessary to know at which point the data of a given user has been downloaded in any previous download. In fact, due to the huge amount of data, social network services return a small quantity of data for each request; therefore, after the first request, the point from which start to download data must be specified.

As output, a list with *new* user's post is expected. In case it is the first time that the data of a user is downloaded, a list of user's information is expected too, otherwise n new posts are enough. The term *new* means that all the fetched post were not previously downloaded by the system, turning out to be new for it, even if they can be dated far in the past. In other words, a new post for our system may not be new for the social network, but is simply a post that was not included in any of the previous download for that given user. Last but not least, the point the data has been fetched for the user must be updated.

The number n of post to download could be given as parameter. It would be better if the number is *small*, not to overload the system. For example, Twitter allows to fetch up to 200 tweets a request. The default parameters should be setted equal to the maximum limit imposed by the API.

3.2.2 Entities extraction

This part has the task to add semantic information to the data that was previously downloaded from social networks. What is obtained from the previous part are only raw texts and images, the goal is now to understand what the user has talked about into her posts. To do that, some external semantic analyzers are used: this kind of services extract entities, topics and sentiment starting from a text or a image. An *entity* is a person, an object or a concept that has an article on Wikipedia; a *topic* is a Wikipedia category to which an entity belongs to. For example, the phrase *"I'm studying computer science at the university"* has two entities - `Computer science` and `University` - and the following list of topics: `Electrical engineering, Electronic engineering, Computer engineering, Computer science, Educational stages, Higher education, Types of university or college, Universities and colleges, Youth`. These new metadata added to the information obtained previously will be useful in the following steps to understand whether a post is about a life event or not. This functionality has also the delicate task to deal with the request rate limits of the external analyzers: in fact many of these API have strict limitations for free plans, allowing only a small amount of requests in a certain time window. In case the limit runs out, this part of the system has to suspend himself waiting for another time window to send new requests, keeping in memory all the computation requested in the meantime.

A post, composed by text, images or external links is expected as input. Furthermore, a list of posts could be accepted as input, but in this case the number of remaining requests must be handled carefully.

As output, a list of entities, topics and sentiment scores is returned for each post analyzed. The sentiment score is made by a floating point number, that ranges from a minimum value - such as -1.0 - to a maximum value - like 1.0. Instead, entities and topics can be represented by a string or an `URI`, for example a `Wikipedia URI`.

By default, everything that is returned by the external analyzer could be given as output, together with a confidence for each entity found inside texts or photos. In addition, this part of the system should provide the possibility to consider only the *top entities* (e.g. the most meaningful, or those with the highest confidence), and also the possibility to set a minimum value of confidence under which an entity is discarded.

3.2.3 Post classifier

This part has the fundamental purpose to decide whether a post is about or not to a certain life event. Its task is to extract some features from the posts previously downloaded, and apply some machine learning algorithm to take this decision. As explained in Section 2.3, the standard approach to classify textual contents in multiple languages is to have a great number of examples for each language; in this thesis a different approach is attempted, using entities and topics instead of single words, for two reasons. In this way is enough to train the classifier with a single dataset in a given language L - english - and for all the contents that are not in the L language is just necessary to map all the entities

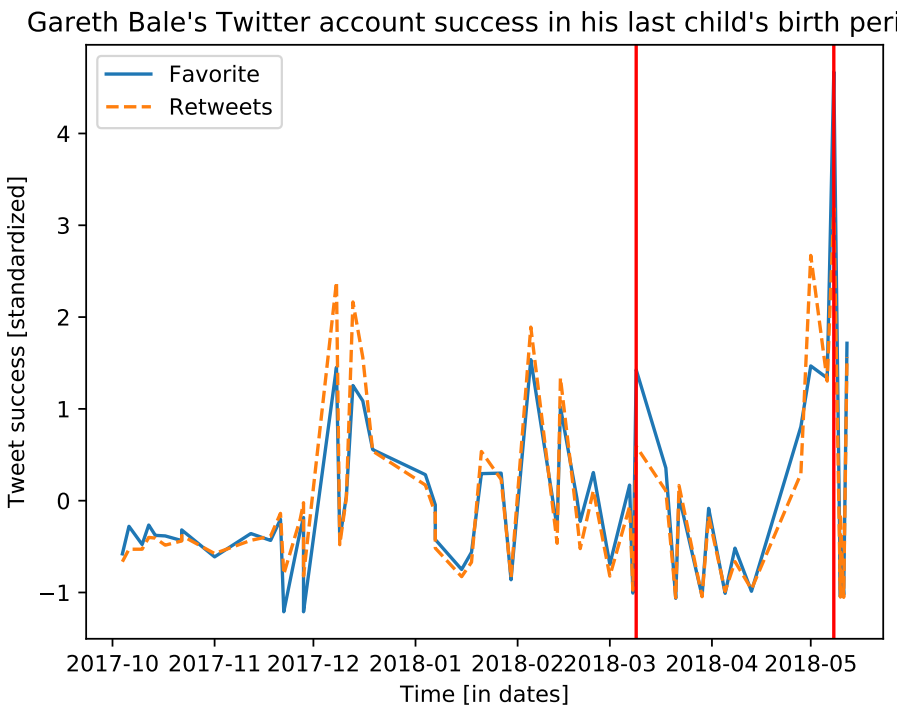

Figure 3.3: The success – number of likes and shares – of the posts standardized and plottet over time. The two vertical lines represent the moments when the user talks about a life event. In both cases, the success is very high.

in the respectives of L. In addition to that, with this technique is possible to consider both photos and texts as the same thing: they are just *entity containers*, and it's not necessary to treat them separately. All the key entities for a life event exist in several languages, so this mapping procedure should easily be successful: for example, the english entity `Infant` for the birth of a child event is translated in 82 different languages. English is chosen as main language because the english version of Wikipedia has a number of articles at least 5 times bigger than any other language. A point worth of noting is that the classifier uses only the vertices of the Wikipedia graph, most of which exist in several languages, and not the graph structure, which is different for each Wikipedia version.

At this point the current situation is, for each user taken into analysis, a list of posts - made by texts, attachments and images - enriched with semantic entities, topics and sentiment score. The input for the classifier is a tuple (or a list of tuples) made by a post, composed as just described, and a life event of interest, for which we want to predict if the post is about it. A life event could be represented by an unique string, e.g. "`GETTING_MARRIED`" for a wedding, or alternatively by an integer ID.

As output, the probability with which the post (or the list of posts) concerns with the life event is expected.

The feature extraction method can be decided a priori, as described earlier, or it could be setted as parameter: in this last case is possible to use only *text features* - the words that compose the text of each post - or use only *entity features* - the semantic entities for reasoning on the meaning of the post, or use them both together. This choice should be made paying attention to the available datasets to train the classifier: in fact, if *text features* are chosen, at least a different dataset for each language supported has to be provided, as explained in Section 2.3.

3.2.4 Timeline analyzer

This last function has the goal to analyze the user's timeline to discover a life event in it. Since a life event is something important and rare into a user's life, the assumption is that also in her social life her behaviour is different when the event is approaching or is in progress. For this reason some unusual patterns, related to the life event itself or to the average success of the posts, are searched

for. The idea is that a user does not publishes contents about, for examples, a new baby, very usually, but only when something important is about to happen. In addition to that, according to [9], a post about a new baby announcement gets probably more feedbacks compared to a normal post, such as congratulation messages. As also shown in Figure 3.3, when the user talks about a life event, the success of her post tends to be high. The detection of a life event is based on these routine changes, which can be various: a slow but constant increase of interest before the event, or a sudden peak when the event is happening. In general, the relative frequency of activities related to the life event itself is monitored.

At this point, a timeline is composed by a list of post, each one with the probability of being about the life event, and with the other metadata, like date of publication and post success. Let p be the probability given by the classifier about a post, that post is considered related to the life event if $p > p_1$. If the previous condition is false, but $p_2 \leq p \leq p_1$, the post success is taken into consideration to decide if the post is about the life event: let n the number of post of a user, and

$$\mu_{\text{likes}} = \frac{1}{n}\sum_{i=1}^{n} post_i.likes \quad \mu_{\text{shares}} = \frac{1}{n}\sum_{i=1}^{n} post_i.shares \quad \mu_{\text{sentiment}} = \frac{1}{n}\sum_{i=1}^{n} post_i.sentiment$$

the average of likes, shares and sentiment score among all the user's post. If

$$post.likes > \mu_{\text{likes}} \wedge post.shares > \mu_{\text{shares}} \wedge post.sentiment > \mu_{\text{sentiment}} \tag{3.1}$$

the post is considered related to the life event, otherwise no.

For each day d the user has been active on social media (those days in which the user published something), the frequency f is computed as follows:

$$f(d) = \frac{r(d)}{all(d)} \tag{3.2}$$

where $r(d)$ is the number of posts related to the life event written the day d, and $all(d)$ is the count of all posts published the day d.

Of course, the timeline is already targeted with the life event taken into consideration. In other words, the life event to monitor is chosen in the previous steps: at this point the timeline is labelled only for a single life event. Once the frequency has been computed for every day, it can be plotted over time, and at the moment f passes a given threshold α, the life event is begun. When a minimum number of days β in which $f(t) > \alpha$ are found, the life event can be considered as detected. By the time in γ days there are no *active days*, the life event can be seen as over.

As input, is expected a list of probabilities, each one with a timestamp, that represents the user's timeline in function to a life event.

As output, a list with time ranges in which the life events have been detected is expected (this range can be composed by the date of the first and the last post related to the life event). Of course, the user may have not lived the life event, so in this case the list will be empty.

There are several parameter for this phase:

- a pair of probabilities p_1 and p_2 to consider a post directly related with the life event, or to check post success to take that decision. By default, $p_1 = 0.6$ and $p_2 = 0.4$.
- the threshold α for the frequency f over which the life event is considered as started.
- β, the minimum number of *active posts* to consider a life event detected. This parameter serves to correct any errors in post classification. In fact, is possible that a text or an image is misclassified: using this parameter is possible to avoid single and isolated errors.
- γ, the maximum time between two *active days* to consider them as related to the same life event. This parameter has the role to put an end to a detection, and consequently to split two consecutive life events.

3.3 Interfaces of the components

In this section the API of each component are presented. Internal APIs are very important to allow system scalability and to facilitate any future changes to the logic or to the functionalities. Firstly, some data types used inside the system are formalized, and secondly, the API are listed.

The first data type represents a *user*, which is a subject of a computation. The object `User` is so composed:

User		
Integer	userID	%The ID of the user inside the system. It is a unique number
String	name	%The name of the user
SocialNetworkUser[]	socials	%Array of social networks in which she's registered in

where a `SocialNetworkUser` object is so composed:

SocialNetworkUser		
Integer	socialID	%The ID of the user inside this specific social network
String	name	%The name of the user in this social network
Integer	from	%ID of the first post downloaded
Integer	to	%ID of the last post downloaded

The fields *from* and *to* are needed to understand where to begin a download request. Finally, the concept of *post* is now formalized in an object call `Post`: it includes all the useful data for this purpose, such as date of creation, number of likes and shares, sentiment score, entity and topic sets and the probability to be about a life event. At the beginning of the analysis, many of these attributes are empty, and each logical step adds a piece of information. Each post is uniquely identified by the couple (ID, social networks).

Post		
Integer	postID	%The ID of the post in the social network
String	socialNetwork	%The social network from where the post comes from
User	author	%The author of the post
Date	created	%The creation date of the post
Integer	likes	%Number of likes received
Integer	shares	%Number of shares
Float	sentiment	%Sentiment score
Set	entities	%Set of strings representing the entities found in post text or images
Set	topics	%Set of strings representing the topics found in post text or images
Float	probability	%Probability of the post to be about a life event

Now the API of the four logical component are presented. Each one can be seen as a web interface that accepts and returns JSON objects. Every time that an object described just above here is used, only the parameters to uniquely identify an instance of these objects are passed or returned as parameters. For example, only a `userID` is used to indicate an `User` instance.

3.3.1 Collector API

The collector needs a `User` instance to which fetch her contents online, and a count that indicates how many activities to download from each social network.

Input:

- `userID`, Integer.
- `count`, Integer.

Output:

- a list of `Post` identified by `postID` and `socialNetwork`.

Below an example of input and output.

```
//input
{
  "userID": 2565227499,
  "count": 200
}
```

```
//output
[
  {
    "postID": 667069250268479488,
    "socialNetwork": "Twitter"
  }
]
```

3.3.2 Entity extraction API

The extractor needs a `Post` instance to which extract entities, and returns the same instance enriched.

Input and output are the same:

- `postID`, Integer.
- `socialNetwork`, String.

Below an example in JSON format. Both input and output are the same.

```
{
  "postID": 667069250268479488,
  "socialNetwork": "Twitter"
}
```

3.3.3 Post classifier API

The classifier takes a life event on which the classification is based, and a list of `Post` instances and adds the attributes `probability`, that indicates the probability with which each post is about the life event. It returns the same instances with the probability added.

Input:

- `lifeEvent`, String.
- a list of `Post`, each one identified by `postID` and `socialNetwork`.

Output:

- a list of `Post`, each one identified by `postID` and `socialNetwork`.

Below an example of input and output.

```
//input
{
  "lifeEvent": "Having children",
  "posts": [
   {
     "postID": 667069250268479488,
     "socialNetwork": "Twitter"
   }
  ]
}
```

```
//output
[
  {
    "postID": 667069250268479488,
    "socialNetwork": "Twitter"
  }
]
```

3.3.4 Timeline analyzer API

The timeline analyzer needs a list of `Post`, which represents the user's *timeline*. Each one has to have a probability of being about a life event. It returns a list of pairs of dates, each one indicating the beginning and the end of a life event found in the timeline. The list can be empty.

Input:

- a list of `Post`, each one identified by `postID` and `socialNetwork`.

Output:

- a list of pairs of dates.

An example of input and output is presented below.

```
//input
[
  {
    "postID": 667069250268479488,
    "socialNetwork": "Twitter"
  }
]
```

```
//output
[
  {
    "from": "2018-3-8",
    "to": "2018-5-24"
  }
]
```

3.4 The detection algorithm

In this part the detection algorithm, whose logic was previously explained in Section 3.2.4, is defined in pseudocode. It is assumed that all the parameters previously highlighted in Section 3.2.4 are defined, and that all the three previous logical steps have already been executed. Let n be the number of posts in the timeline taken into analysis, and let the data type `Post` be defined as in Section 3.3.

The algorithm is composed by two main parts. In the first part, the goal is to compute the average number of likes and shares received by the user among her posts, and the average sentiment score. After that, all the posts are grouped by date of publication, creating a dictionary with the days in which the user has published something as keys (k indicates a single key), and as values the tuple (posts related with the life event on day k, all posts on day k). This is explained in Algorithm 1.

Algorithm 1 Compute the relative frequency of activities related to the life event.

Require: posts must be sorted by date

1: **function** COMPUTEFREQUENCY(Post[] posts)
2: $AVGLikes \leftarrow \frac{1}{n}\sum_{i=1}^{n} p_i.likes$
3: $AVGShares \leftarrow \frac{1}{n}\sum_{i=1}^{n} p_i.shares$
4: $AVGSentiment \leftarrow \frac{1}{n}\sum_{i=1}^{n} p_i.sentiment$
5: $frequencies \leftarrow \{\}$
6: $count \leftarrow 0$
7: $total \leftarrow 0$
8: **for** $i \leftarrow 1$ to n **do**
9: $total \leftarrow total + 1$
10: **if** $posts[i].probability > p_1$ **then**
11: $count \leftarrow count + 1$
12: **else if** $posts[i].probability \geq p_2 \wedge posts[i].likes > AVGLikes \wedge posts[i].shares > AVGShares \wedge posts[i].sentiment > AVGSentiment$ **then**
13: $count \leftarrow count + 1$
14: **if** $i = n \vee posts[i].created \neq posts[i+1].created$ **then**
15: $frequencies[posts[i].created] \leftarrow (count, total)$
return $frequencies$

For each day k, *count* and *total* are computed, which represent the number of posts related to the life event and the total number of post on that day respectively. At line 10 the decision to trust the classification directly is taken. In case it is not trusted, the condition at line 12 checks whether the probability is greater or equal than p_2 and if number of likes, shares and sentiment score are all over the average of the user, the post is considered related to the life event, as explained in Formula 3.1. The last condition at line 14 is to understand whether a post is the last one for a day: this happens when the current post is the last one in the collection, or when the next one has a different date of creation.

In the second part, the dictionary created in the first part is analyzed, computing for each day k the frequency of activeness as in Formula 3.2 and analyzing this latter to understand if there are significant changes on it. This is explained in Algorithm 2. Also here it is assumed that the days in the dictionary are analyzed in increasing order.

The symbol $\perp$ represents a null type. The idea is to find some time intervals that represent a life event, each one recognized by a *start* and an *end*. For each day in which the user published something,

Algorithm 2 Decide whether a user has lived a life event

```
1:  function DetectLifeEvents(frequencies)
2:      start ← today()
3:      end ←⊥
4:      count ← 0
5:      found ← {}
6:      for all k ∈ frequencies do
7:          f ← frequencies[k].count / frequencies[k].total
8:          if f > α then
9:              if k < start then
10:                 start ← k
11:                 end ← k
12:             if (k − end) < γ then
13:                 end ← k
14:                 count ← frequencies[k].count
15:             else
16:                 if count ≥ β then
17:                     found ← found ∪ {(start, end)}
18:                 start ← k
19:                 end ← k
20:                 count ← 1
21:     if count ≥ β ∧ end ≠⊥ then
22:         found ← found ∪ {(start, end)}
        return found
```

the relative frequency f is computed (at line 7) as shown in Formula 3.2. If f is greater than the threshold α and it is the very first day encountered, *start* will point to it. For every other following day k in which $f > \alpha$ and k comes less than γ days later than the current *end*, *end* is overwritten, and putted equal to k. When an active day is too far from the last active day, the conditions to add a life event are checked (line 16): if at least β active posts were previously found, the life event from *start* to *end* is added, and these variables are resetted to detect another interval. In the end, at line 21, is verified whether there is an interval left open, and in this case, if there are enough posts, a last life event is added.

The complexity of both Algorithms 1 and 2 is $\Theta(n)$ in time, because each post is analyzed only once, in both functions. In case posts are not sorted by date, the computational cost will be dominated by the sorting procedure, becoming $\Theta(n \log n)$.

3.5 Exposed APIs

In this section the user endpoint APIs offered by the system are defined. These interfaces are RESTful APIs, which expose the main resources available in the system: `User`, `Download Request` and `Computation`. For each resource, the possibilities to retrieve and create new instances are offered, and for the user resource is also possible to edit an instance. The structure of the `User` resource is described in Section 3.3. A `Download Request` is meant to download user's contents from social media, and it is executed asynchronously, as explained in Section 4.1.1. Finally, a `Computation` is an analysis on a user's timeline with a given life event, as described in Section 3.4.

Each response has a field `"ok"` that indicates whether everything went well. In case an error occurs, the field `"error"` gives a human readable explanation of the error, while the HTTP status code indicates what kind of problem occurred (for example a 404 error in case the user requires a resources that doesn't exist).

The complete API specification can be found on Apiary:

`lifeevents.docs.apiary.io`

4 Implementation and evaluation

This chapter presents how a prototype of the system has been developed, giving a panoramic view of the interaction between components and a detailed focus on the solution chosen to deal with external services with a limited number of requests in a certain time range. After that, the development techniques for the classifier are presented, along with the performance evaluation of the latter one, and finally an evaluation about the detection process is provided.

For the evaluation, confusion matrices and some measures derived from them are used. A confusion matrix is a table layout used to show the results of a classification algorithm. Its columns represent the prediction output by the algorithm, while the rows represents the real class of the samples. In case of binary classifications, like those in this thesis, the matrix has 4 cells: the top-left one is a counter of the *true positives* (TP) samples, the top-right for the *false negatives* (FN), bottom-left for *false positives* (FP), and finally bottom-right for *true negatives* (TP). The sum of the four counters gives the number of samples analyzed. On these numbers, 4 measures are used:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$
$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$
$$\text{F}_1\text{-Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

4.1 Component interactions and communications

Overall, the whole system can be divided in two big parts: the one to get the data, the other to analyze them. The four logical steps previously explained in Section 3.2 are translated into four components placed in the deepest points of each request, as shown in Figure 4.1 and 4.2. These four parts have different roles and behaviours: some of them are autonomous, while some others depend one from the other. Each one provides its own API, as shown in Section 3.3, to allow the system to be scalable.

The request handler is the user's endpoint of the system, the only access point from the outside. This component allows managing the addition and modification of the users to be analyzed, and starts a content download or a computation of the timeline.

The download request will be described in detail in Section 4.1.1, because due to the use of external services it is not immediate to explain.

The computation request is much easier, because it only works with data within the system, without any external service. Each request is fulfilled with the personal data of the user in analysis present at the moment of the request. This process can be seen in the lower part of Figure 4.1. User's posts are firstly classified as related or not to the life event taken into analysis, and then data is given to the timeline analyzer part to detect the life events into user's timeline, as explained in Section 3.2.4.

4.1.1 A queue system to download user's contents

The most complex request is a `/downloadrequest`[14], because it has to face itself with external APIs over HTTPS, which have free plans with some request limits over a time range. Due to this issue, each part needed to fulfill this kind of requests has to be able to suspend itself without interrupt the whole system, in case request number runs out. In other words, when the function for downloading contents on a social network, or the one that deals with the external semantic analyzer, reaches the limit of requests, it has to wait for another time slot stopping only itself and nothing else. For this reason

[14]For further details about the request endpoints, visit the official API: `lifeevents.docs.apiary.io`

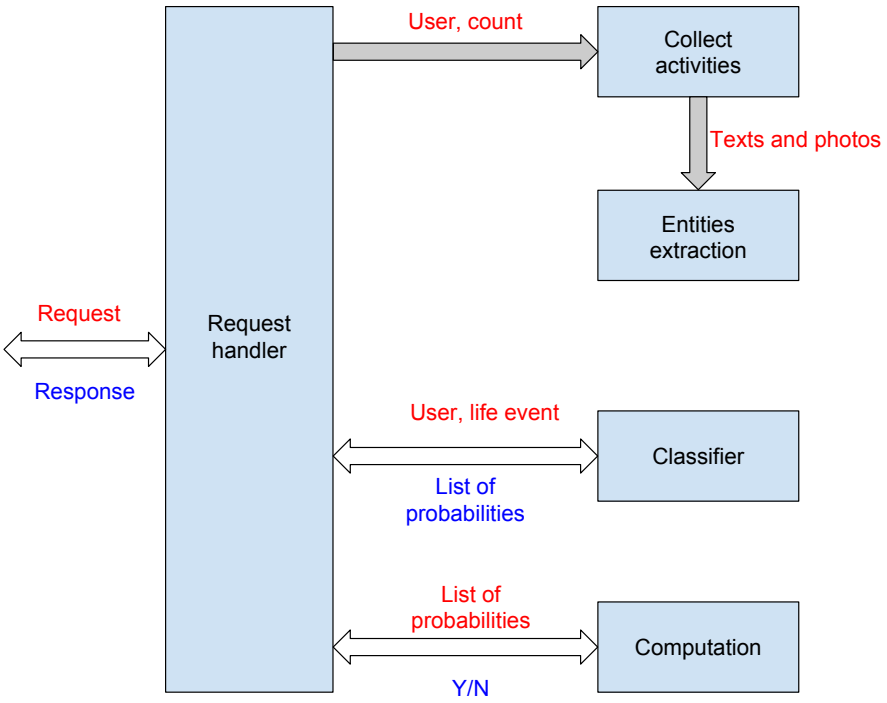

Figure 4.1: A global view of the system with the data exchanged between the components. Grey arrows are queues of inputs, while white arrows are messages of a class method call.

data is not passed with a standard class method call, but it is inserted into a queue. In this way the component that asks for an external service does not need to care if the downloader or the semantic analyzer are already busy or suspended. There will be a queue for each module that has to interface itself with an external service, so one for each social media and one for every semantic analyzer (one for texts and one for pictures). This decision is necessary to avoid bottlenecks as much as possible: in case a social network API reaches the limit, the downloads in the other social networks can be carried on, and so also the following requests can start. The only thing that is necessary to track is when each part of the Download Request finishes, in order to have always a current status for every request of download: when at least a data stream of a request is taken over by some downloader or analyzer, the status is *in progress*, while if all data streams of a requests are waiting into a queue, the status is *waiting*. By the time each stream passed into its specific downloader and analyzer successfully, the status is *completed*. The status timeline can be seen in Figure 4.3.

For example, let us suppose to download contents made by a user who is signed up to Twitter, Facebook and Instagram. The request is collected by the API endpoint, which verifies if the user exists into the database of the system, and in case it exists the endpoint gets from the same database, for each of the three social media, the user ID on that social network and the last post downloaded in a previous request, if there is one. After that, the tuple (user ID, starting point, number of post to download) is queued in the respective social media queue, and an immediate response is given to the API request, to let the user know that her request has taken into consideration. At this point, three streams of data – one for each social media – are created, and they go on independently: each of them waits its turn into the downloader queue, then data is downloaded and saved into the database. At this point other two streams for each downloaded post are created, one for images and one for texts: the tuples (post ID, text) and (post ID, photos) are enqueued in the text semantic analyzer and in the image semantic analyzer respectively. When all these streams pass both downloader and analyzers, the request is *completed*. This schema can be seen in Figure 4.4.

Even though this architecture avoids as much as possible bottlenecks, a download request can take a very long time to be completed, even days, because each post from any social media has to pass through the semantic analyzers, which have a very restricted number of requests a day.

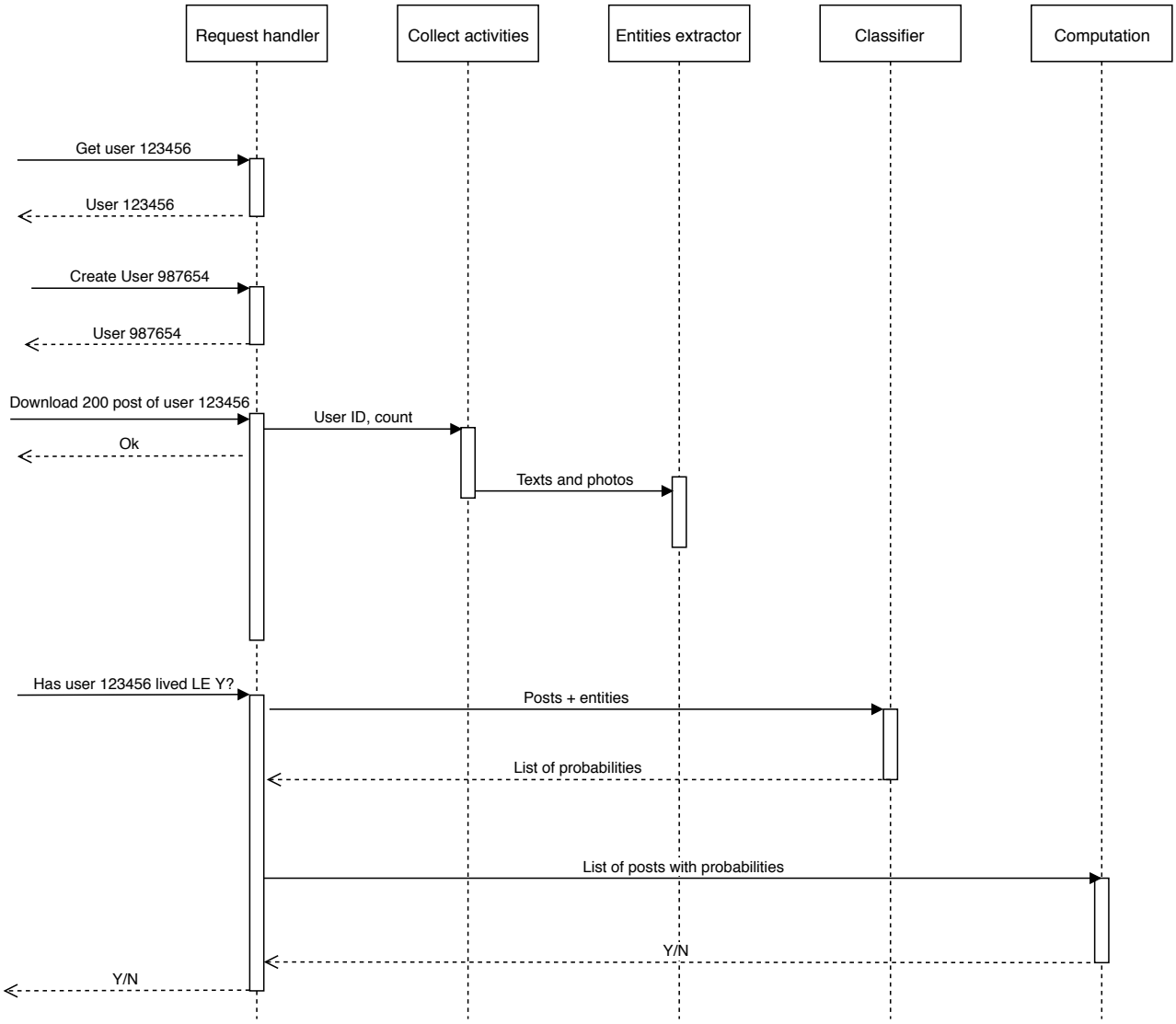

Figure 4.2: The interaction among components over time.

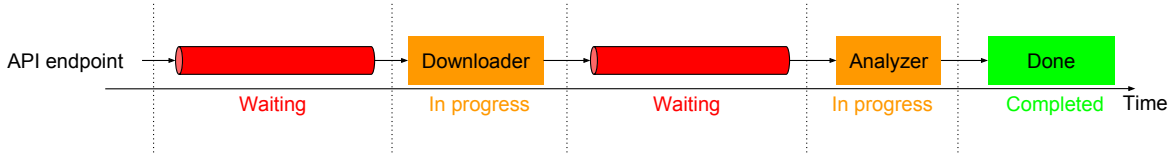

Figure 4.3: The changing of status over time of a download request.

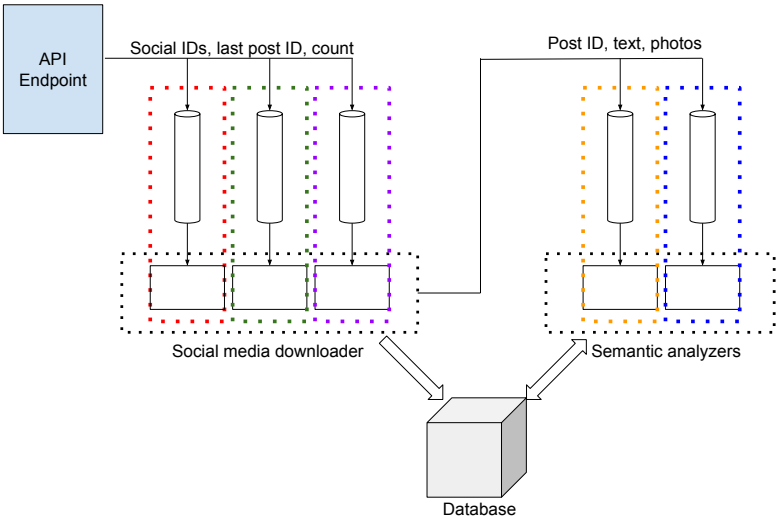

Figure 4.4: The route of a download request. From the endpoint, data is divided in independent streams, one for each social media, and each stream is enqueued in the corresponding queue. The red, green and purple frames represents three different social network download system – for example the red one for Facebook, the green one for Twitter and the purple one for Instagram. Once data is downloaded, it is saved into the database, and posts are given to the semantic analyzers, represented by the orange and blue frames: one for texts and one for images.

4.2 Implementation choices

Due to the recent Cambridge Analytica scandal that involved Facebook, and with the consequent policy updates made by the social network, develop an application that works with the products of this latter (Facebook itself and Instagram, for instance) that uses user's contents has become difficult. To do that, Facebook has to release a manual authorization for the app, and the process to obtain it requires a demo video to show what the application does, and a detailed description of how user data is used. Even if a beta development had been used, it would have been hard to find enough users available to share their data for the development of this system. For these reasons it was decided to base the prototype on Twitter.

Another point worth of noting is that the developed software is only a demo: in fact, the use of social network data for commercial purposes without explicit user authorization violates the terms of use of the latter. Furthermore, the processing of personal data in Europe is regulated by the General Data Protection Regulation (GDPR), and the use of those data must be authorized by the owner of them (the author herself), for example via *social login*.

Another issue is related with the external semantic analyzers: only a free account has been used, which implies a very strict number of requests per day (1000). This means that currently a computation can take a very long time to be executed, even days, because each post has to be analyzed by this external services. Furthermore, image analyzers have free plan with only 1000 request a month, so it was decided to use only texts of posts, and to ignore images. All the obtained result, therefore, are only related with texts of tweets. On the other hand, some tests with images were executed, showing that the use of images would give better results: in fact, several posts of life events are composed only by images, so at the current situation they cannot be classified correctly.

About the implementation, the prototype of the system is developed in Python, using some standard libraries for machine learning and mathematical purposes: the classifiers are those available in the `scikit-learn`[15] library [18], while for numerical arrays, matrices and other mathematical functions the `NumPy`[16] library [17] is chosen. Data is stored and organized using a `MongoDB` database[17].

[15] `scikit-learn.org`
[16] `www.numpy.org`
[17] `www.mongodb.com`

Last but not least, the external semantic analyzer for texts is called `Dandelion`[18] made by SpazioDati S.r.l.[19]. Some test with images were also performed, using the `Cloud Vision API` of Google Cloud[20].

4.3 Feature extraction

In this section the feature extraction technique is presented. This method is thought to work with posts from various social networks, whose structures are standardized as can be seen in Section 3.3. In general, each post has a text and a series of images that are used for the classification. According to Section 2.3, Wikipedia entities are used as features, for two reasons:

1. In this way, it is possible to work with as many languages as is wanted, using only one for training. In fact, letting L be the language of the samples in the training set, it is enough to map all the entities found in posts written in a language $L' \neq L$ to the corresponding entity in the L version of Wikipedia. All the most common entities for life events exist in many version of the online encyclopedia. For this reason, this method allows a great scalability regarding the languages supported.

2. With this method is possible to consider texts and images in the same way, seeing them as *entity containers* without differentiating them.

The feature extraction is done in this way: each entity of the samples of the training set is kept as feature, and a matrix X is created, with n rows, one for each sample to classify, and m columns, one for each feature. Each element $[i, j]$ of X is the counter of how many times the entity j is found in the post i. The number m of columns is in the order of thousands.

Of course, the features used for weddings are different from those used one for births, and consequently two different machine learning classifiers are needed, each one trained for its specific life event. To do that, the dataset taken from [9] was filtered, keeping only tweets about a wedding or a birth of a child. Then it was splitted in two files, one for each life event, and every tweet was enriched with Wikipedia entities and topics using the external semantic analyzer. In the end, the wedding dataset counted 2538 samples, the births one 2233. Each of these sets were very balanced, with half of the sample related to the life event and the other half not. Both datasets were then used to train each classifier.

4.4 Construction and performance evaluation of the classifier

In this section all the techniques used to create the classifier are explained.

According to the literature, a naive bayes classifier and a decision tree were chosen as classifiers, but due to the better performance of the first one, the second one was initially discarded. In particular, the classifier was a Multinomial Naive Bayes[21].

The very first feature extraction technique was inspired by the `tf-idf` weight function[22], which is used to understand the importance of a word into a series of documents. The idea was to select a small sample of key entities analyzing some web pages focused on the life event taken into consideration, such as a wedding planner homepage or a pregnancy blog. These k top entities were used as features of tweets, and a matrix X was created in the same way explained in Section 4.3, but with $m = k$. However, this methodology performed worse than a random classifier, with both precision and recall scores lower than 0.5, with $k = 3, 4, 5, 10, 100$.

A much better performance with the dataset, but unfortunately not with the reality, was given by using all the Wikipedia entities of the training set as features, exactly as explained in Section 4.3. Firstly, the dataset was splitted into a training and a test set, the first one with the 70% of the samples and the second with the remaining 30%; secondly, the matrix X was created. On the test set this solution had good results, as shown in Table 4.1(a) and 4.1(b), but in the reality, trying to predict if

[18] `dandelion.eu`
[19] `spaziodati.eu`
[20] `cloud.google.com/vision`
[21] `scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes`
[22] `en.wikipedia.org/wiki/Tfidf`

(a) Results on a single dataset split

Life event	Precision	Recall	F1 Score
Getting married	0.82	0.81	0.81
Having children	0.77	0.78	0.77

(b) Results on a 10 fold cross validation

Life event	Precision	Recall	F1 Score
Getting married	0.73	0.82	0.77
Having children	0.62	0.57	0.59

Table 4.1: The performance of the naive bayes were satisfactory on a very balanced dataset, but not with an unbalance situation. Unfortunatly, users' timelines are very unbalanced, and this classifier turned out to be inappropriate.

tweets from a user timeline were about or not a life event, this solution turned out to be impractical: the discrimination was too loose, and many tweets that was not related with the life event at all were considered "positives", with the precision measure that fell dramatically under 0.1. The overfitting reason was excluded, because the training set was kept aside from everything else. In addition, each sample in the dataset was written by a unique user. The most probable reason is that the dataset used for the training was too balanced compared to the real situation in social networks: in fact, while the dataset contained about 50% of tweets about a life event and 50% not, a timeline had a very small percentage of contents about a life event. For example, analyzing a portion of Gareth Bale's Twitter profile[23], it turned out that on 224 tweets taken into consideration only 2 were about the birth of a child and only one about marriage. Many other profiles considered had similar proportions.

A third approach was tried, with the purpose of improving the previous situation: train the classifier with a very unbalanced dataset, discarding randomly many of the positive samples to create a 90% negatives and 10% positives situation. As before, the results were good only with the test set: with a real profile the discrimination was too strong this time, not finding any tweet at all, maybe just because of this disproportion, where too few related samples were given and so the classifier did not see enough entities among those of positive samples.

Another very big issue was caused by the external semantic analyzer - Dandelion - used to discover Wikipedia entities inside texts. Its results turned out to be very inaccurate for such short texts written in Twitter: for example, a tweet published by a famous sports man about his wedding, with the text "*This is her, my wife*"[24] was mapped by the analyzer to a song by the British rock band *the Who* called *My Wife*, not understanding the context at all. Some other cases of misunderstanding between the text and its true meaning were found, for example the word *"congratulations"* was mapped into the entity "`Congratulations: 50 Years of the Eurovision Song Contest`", but this was not a big issue because this combination was always respected every time that word was found. Just to be sure that some words were recognized correctly, 4 keywords for each event were manually mapped into their correct entities, such as *baby* on the entity `Infant`.

To avoid issues with the semantic analyzer as much as possible, as few as possible entities were used to classify a tweet: a very first approach consisted in a manual selection of features, for example the entity `Wife` for wedding, but the results were not satisfactory. The definitive approach was to let the data decide which entities to use, relying on the dataset taken from [9]. The classifier was changed, opting for a decision tree, because the previous naive bayes was too loose during the classification. A decision tree is a machine learning model used for regression and classification based on a binary tree data structure. Each node of the tree is a question relative to a feature of the objects to classify, and each leaf represent a decision, an estimation of the value of the input object. During the training, the tree is built starting from the root: at each step of the construction, the goal is to split the dataset into two, making a question on a feature of the objects, to reduce the unpredictability of the two new subsets as much as possible. Every object in input has its own path from the root to a single leaf,

[23] `twitter.com/GarethBale11`

[24] `twitter.com/petosagan/status/667069250268479488`

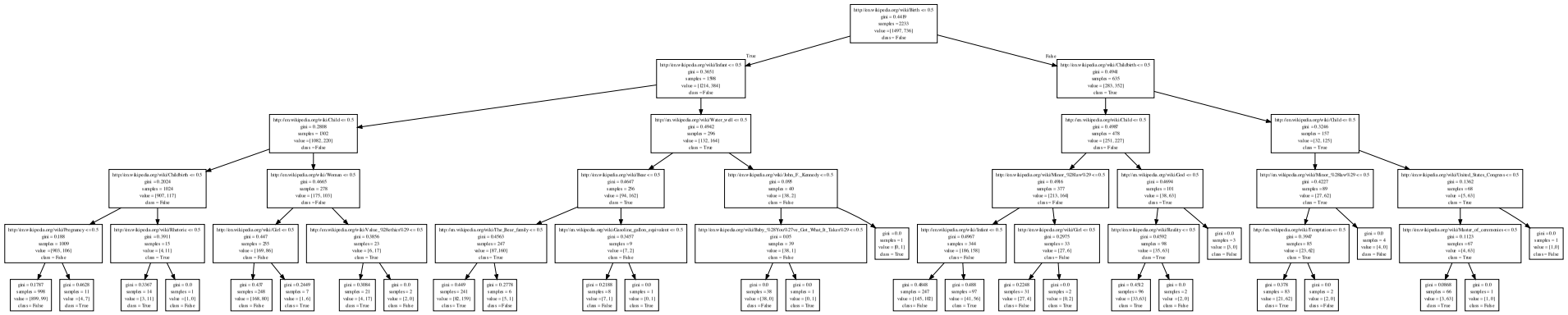

Figure 4.5: The decision tree for the birth of a child event. As can be seen, the nearest nodes to the root are decisions about entities that are strictly connected with the life event itself, and each of them makes a significant split in terms of the size of the resulting subsets, while the deepest nodes are strongly connected with the training set. For this reason the maximum depth was forced to be low.

which represents the decision. In this case each leaf contains a probability with which a sample that ends its path there is about the life event. In this way, there is a finite set of probability that are given to each post in input. The classifier is the decision tree offered by the `scikit-learn` library[25]. To reduce the overfitting on the training data, the maximum depth of the tree is limited to 5, so for each root-leaf path, at most 4 intermediate nodes could be found. The decision tree for the `Having Children` life event can be seen in Figure 4.5.

To further improve the results and to make sure that some key entities are identified, an automatic check on the texts of the posts is done. In particular, for each life event LE and for each language L, a set of keyword translation $T(LE, L)$ is defined. For example, for the life event `Having children` and for the english language, T is so defined:

$$\begin{aligned} T(\texttt{Having children}, \text{english}) = \{&(\text{"birth"}, \texttt{http://en.wikipedia.org/wiki/Birth}),\\ &(\text{"born"}, \texttt{http://en.wikipedia.org/wiki/Birth}),\\ &(\text{"baby"}, \texttt{http://en.wikipedia.org/wiki/Infant}),\\ &(\text{"child"}, \texttt{http://en.wikipedia.org/wiki/Child})\} \end{aligned}$$

The results with the dataset of [9] were a little worse than those of the bayesian classifier shown in Tables 4.1(a) and 4.1(b), but in the reality, with a test conducted on 5836 tweets taken by 44 different timelines the results were much better than the previous one, as explained in Tables 4.2(a) and 4.2(b).

(a) Results about wedding tweets

Sample	Precision	Recall	F1 Score	Support
True	0.87	0.70	0.77	134
False	0.91	0.97	0.94	5702
Avg / Total	0.90	0.90	0.90	5836

(b) Results about birth of a child tweets

Sample	Precision	Recall	F1 Score	Support
True	0.65	0.62	0.63	161
False	0.93	0.94	0.94	5675
Avg / Total	0.89	0.89	0.89	5836

Table 4.2: The performance of the decision tree with a very unbalanced dataset. On this same test set the previous classifier, the naive bayes, showed both precision and recall scores under 0.1.

4.5 Detection evaluation

Before presenting the results about the detection method it is right to make a premise. The results that will be showed now are obtained using data that are composed only by texts coming from Twitter. The reason why Twitter was used is because it has the easiest policy for download authorization: while on Facebook is necessary the explicit permission of both the social network and the user for downloading and using personal data, Twitter allows to use it for non-commercial purposes just registering a developer account. In addition to that, images were not used because semantic analyzer services for photos allow only 1000 requests a month freely, so it was not computationally possible to process thousands of posts with images. Some isolated tests with images was tried, and posts about life events that had not been classified correctly using their texts were correctly by their photos.

First of all, the tests were executed with 30 different Twitter accounts, some of them completely downloaded, some other only partially. Totally, 21 users had children in their real life and shared something about it, 8 got married and shared something too, six of which lived both a marriage and a birth of a child. In addition to that, 7 other accounts that were not concerned with these two life event were added to the samples. The reason why only 30 account were downloaded (and some of them partially) is always because of semantic analyzer free plans: in fact `Dandelion` offers 1000 requests a

[25] `scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`

(a) Getting married

Actual value \ Prediction outcome	True	False	Total
True	4	4	8
False	0	22	22
Total	4	26	30

(b) Having children

Actual value \ Prediction outcome	True	False	Total
True	12	9	21
False	0	9	9
Total	12	18	30

Table 4.3: Confusion matrices of boolean tests

day for free, and for each post two requests were necessary: one for Wikipedia entities, and one for the sentiment score. Thus, the necessary machine time to download completely an account could be greater than a week.

Secondly, the parameters setted for the detection algorithm, whose meanings can be seen in Section 3.2.4, were the following: $p_1 = 0.6$, $p_2 = 0.4$, $\alpha = 0.0$, $\beta = 2$, $\gamma = 182$.

Thirdly, two types of tests were made:

- A boolean valuation without consider the temporal factor, that indicates whether the detection reflects the fact that the subject of the analysis really experienced that event in a some time in his life.
- An evaluation on the quality of the time ranges returned.

For the first test, which is the classic test usually done with a classification problem, the results can be seen in the two confusion matrices, 4.3(a) for weddings and 4.3(b) for births respectively. Many *false negative* samples may be due to the fact that the timeline analysis algorithm considered a life event only if at least two posts about it were met. The reason is to avoid as much as possible cases in which the user talked about the life event by chance, for example posting about a movie in which there is a marriage. Another possibility is related to the exclusive use of texts in these tests.

For further details, the following are the results for wedding cases,

$$Accuracy = 0.87 \quad Precision = 1.0 \quad Recall = 0.50 \quad F_1\text{-}Score = 0.67 \tag{4.1}$$

and for births of child

$$Accuracy = 0.70 \quad Precision = 1.0 \quad Recall = 0.57 \quad F_1\text{-}Score = 0.73 \tag{4.2}$$

respectively.

The second type of test was done only on the *true positive* samples of the first test, that are those samples that lived a *life event* in their real life and for whom a time range was returned by the algorithm. Firstly, the correctness of the time ranges compared to the ground truth of the event was checked, as can be seen in Table 4.4(a) for the weddings and in Table 4.4(b) for the births. Then, for the correct results the length and the "displacement" of the intervals were analyzed, while for the wrong detection the error expressed in number of days were reported.

In particular, the results about correct intervals are shown in Table 4.5: as can be seen, the most of the intervals are shorter than a week, this because users tend to speak about life events when they are very close to leave them. Furthermore, 6 users posted about the birth of their child only the day the event happened, for this reason the beginning and the end of the interval are overlapped.

(a) Time ranges about weddings

Detection result		Ground Truth	Correctness
From	**To**		
22/09/2015	22/09/2015	03/10/2015	False
[7]09/08/2016	09/08/2016	07/11/2015	False
[3]14/05/2016	13/06/2016	13/06/2016	True
[4]15/05/2018	20/05/2018	15/05/2018	True

(b) Time ranges about births of a child

Detection result		Ground Truth	Correctness
From	**To**		
[5]13/05/2018	13/05/2018	13/05/2018	True
[6]17/03/2016	17/03/2016	20/07/2016	False
25/10/2017	17/11/2017	26/10/2017	True
07/11/2015	18/11/2015	18/12/2015	False
15/01/2015	15/01/2015	15/01/2015	True
22/03/2016	22/03/2016	22/03/2016	True
[2]09/03/2018	08/05/2018	08/05/2018	True
05/01/2012	05/01/2012	05/01/2012	True
14/07/2011	14/07/2011	14/07/2011	True
22/12/2016	22/12/2016	22/12/2016	True
16/10/2016	16/10/2016	01/04/2017	False
[1]04/01/2017	26/08/2017	30/07/2017	True

Table 4.4: The list of tests made on *true positive* samples. The first two columns represent the time range outputted by the detection, the third is the real date of the event, and the last one indicates if the result is correct or not. The numbered results are graphically represented in Figure 4.6.

Correct intervals

Life event	**Length** (days)	***Left* space** (days)	***Right* space** (days)
`Getting married`	31	30	0
	6	0	5
`Having children`	1	0	0
	24	1	22
	1	0	0
	1	0	0
	61	60	0
	1	0	0
	1	0	0
	1	0	0
	235	207	27

Table 4.5: Details about the correct intervals. *Left space* indicates the number of days between the starting point of the interval and the real date of the event, while the *right space* is the number of days from the event to the end of the interval.

Wrong intervals

Life event	**Distance** (days)
`Getting married`	11
	-276
`Having children`	125
	27
	167

Table 4.6: Details about the wrong intervals. The distance is the difference in days between the date of the real event and the nearest extreme of the interval. Positive values indicate that the detected interval finishes before the ground truth of the event, while a negative value indicates that the time range starts after the date of the event.

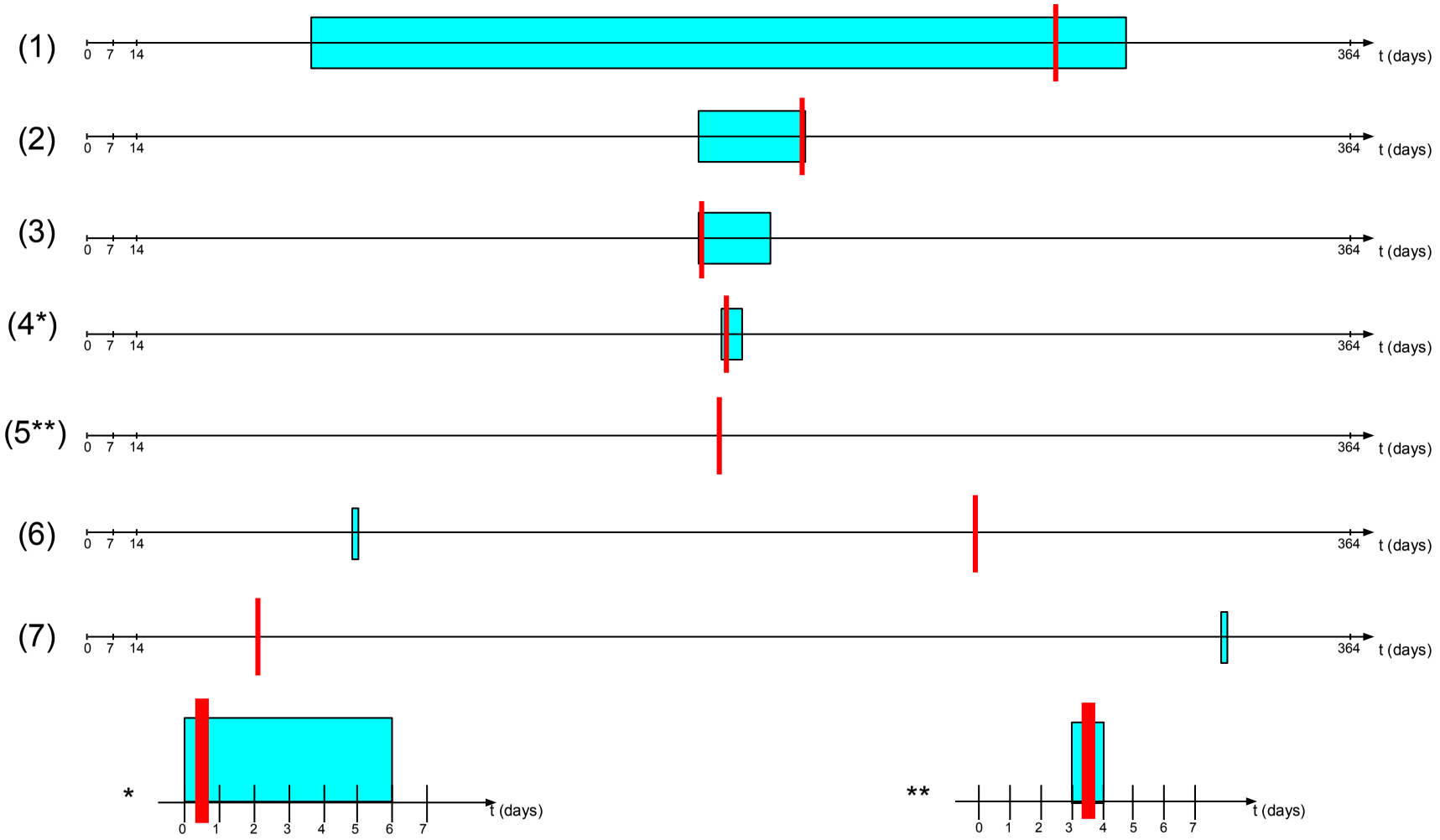

Figure 4.6: A graphical representation of the results in Table 4.4(a) and Table 4.4(b). In each time line, composed by 364 days (52 weeks), the light blue slice is the output of the detection, while the red vertical bar is the date of the real event. Case (1) shows a very large but correct detection, cases (2) and (3) are detections in the order of magnitude of the month, cases (4) and (5) are the most precised detections, with a length within a week. Finally, cases (6) and (7) are wrong detection. As can be seen in the correct detections, the ground truth stays very near to an extreme of the interval: this depends on how the user behaves in sharing news about her private life. When the the date of the real event is near to the right extreme of the interval, this means that the user announced the live event some time before its happening, while contrariwise if the event is near the left extreme, it means that the user preferred to share something later the event. The first pattern is the most common, expecially with births of child, while the second one is rarer, used sometimes with weddings.

The intervals that last for weeks or months indicate that the user decided to announce the life event some time before the incident, or spoke about it even some days after it occured. Another point worth to notice is that the intervals are very unbalanced: the life event almost always falls near an extreme of the time range, or right on it. This means that users usually talk about the event or only before it happens, or only later. In particular, in births cases is most common to announce the event with some advance, while with weddings is usual to post something about it in the days immediately following.

On wrong intervals the distance in days between the date of the real event and the nearest extreme of the interval was computed, as shown in Table 4.6. In four errors out of five, the life event was detected before the incident: this means that these users announced the coming event in advance, without posting anything right after it occured. The giant error in a `Getting married` detection, 276 days of distance between the result and the truth, is a particular case: the user wrote this post to announce the birth of his child: *"My strong wife Patricia and me are glad to introduce you Gabriel!"*[26], and the classifier marked it as a post belonging to a wedding, instead of to a birth. In this case the analysis of the attached image could have helped.

In conclusion, so much uncertainty is caused by the subjectivity with which people decide to share a personal life event on social networks. As highlighted by the tests, several users share something only when the event is upcoming, while some others make announcements with some advance.

5 Conclusions

This thesis tries to answer the research question about the feasibility of life event detection on social media, presenting a solution to detect two types of life events into social network timelines. It goes beyond the actual state of the art, because it is not limited to a post classification only, but it does a social profile analysis to discover an event. Furthermore, it uses a technique based on Wikipedia contents to understand what the user talkes about, which gives to the system an high scalability, allowing the adding of a new language to support in a more easily way than a traditional language undestranding system. In addition to that, it's possible to consider all the components of a post (text, images, external links, etc.) as *entity containers*, permitting an easier training in terms of quantity and type of data: in fact in this way it is not necessary to have one training set for texts and one for photos, but just a set of *posts* about life events, each of them can be made by texts only, picture only or both.

The classifier showed some very good results, with both *precision* and *recall* measures around 0.9, and the whole detection system had encouraging results with ample room for improvement, considering that due to GDPR issues and to the resulting new policies of social media only textual contents from Twitter were used for tests.

In addition to that, a new dataset about life event posts on Twitter has been released, containing 5836 samples composed by 45 users, 161 about weddings and 134 about births of a child. It was created deliberately so unbalance because it has the purpose to simulate how is someone's timeline on social networks, where these kinds of posts are very rare by and large. This dataset is freely available and usable with Creative Commons Attribution-NonCommercial 4.0 license, and can be found here:

`doi.org/10.5281/zenodo.1294893`

5.1 Future work

There are several aspects that can be deepened or added. First of all, this system was projected to work with several social networks, but due to the reasons explained in Section 4.2 only Twitter and textual contents were taken into consideration during the development phase. So the first possible addition can be to add the support for *Facebook* and *Instagram*, and also analyze images for post

[26] `twitter.com/manuelquinziato/status/763056696180834305`

classification. Furthermore, the latters social media are more used than Twitter to share personal contents and news, such as the own marriage or the birth of a child, and many posts about life events are full of images, or even composed by images only.

A possible way to obtain better results could be the analysis of the comments of a post: a post about a life event is usually full of comments of congratulations, nearness and support. Comments can give a better view of the context of the post, and so they can lead to a better understanding.

Another future expansion could be the addition of other life events, like graduation, buying a new home, changing a job, the death of somebody, etc. Each of them has its own meaning and brings with it a series of consequences or necessities.

In addition to that, it is possible to combine the result coming from the detection algorithm with some demographic studies or some statistical data, to refine the results. In this way it would be possible to weight the information coming from social medias with external data, provided by some statistical institute or other trusted sources.

Another possibility could be expand the machine learning capability of the system: in addition to the already current event detection, also predict a life event would give such a powerful piece of information.

In conclusion, this thesis work offers a good base for detecting life events on social networks, which can be boosted or deepened in some aspects, for example, based on the scope in which it will be used, or on what type of users it will receive. Any addiction will still follow a logic or patterns very similar to those explained in this dissertion.

Bibliography

[1] Sitaram Asur and Bernardo A Huberman. Predicting the future with social media. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Volume 01*, pages 492–499. IEEE Computer Society, 2010.

[2] Paulo R Cavalin, Fillipe Dornelas, and Sérgio MS da Cruz. Classification of life events on social media. In *29th SIBGRAPI (Conference on Graphics, Patterns and Images)*, 2016.

[3] Paulo R Cavalin, Luis G Moyano, and Pedro P Miranda. A multiple classifier system for classifying life events on social media. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*, pages 1332–1335. IEEE, 2015.

[4] Paulo Rodrigo Cavalin, Maíra Gatti, and Claudio Santos Pinhanez. Towards personalized offers by means of life event detection on social media and entity matching. In *HT (Doctoral Consortium/Late-breaking Results/Workshops)*, 2014.

[5] Kiron Chatterjee and Ben Clark. The facts are clear–life events change travel behaviour. policymakers please take note. *Local Transport Today*, 679:18, 2015.

[6] Chien Chin Chen, Meng Chang Chen, and Ming-Syan Chen. An adaptive threshold framework for event detection using HMM-based life profiles. *ACM Transactions on Information Systems (TOIS)*, 27(2):9, 2009.

[7] Smitashree Choudhury and Harith Alani. Personal life event detection from social media. 2014.

[8] Barbara Di Eugenio, Nick Green, and Rajen Subba. Detecting life events in feeds from twitter. In *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*, pages 274–277. Ieee, 2013.

[9] Thomas Dickinson, Miriam Fernandez, Lisa A Thomas, Paul Mulholland, Pam Briggs, and Harith Alani. Identifying prominent life events on Twitter. In *Proceedings of the 8th International Conference on Knowledge Capture*, page 4. ACM, 2015.

[10] Thomas H Holmes and Richard H Rahe. The social readjustment rating scale. *Journal of psychosomatic research*, 11(2):213–218, 1967.

[11] Alan Jackoway, Hanan Samet, and Jagan Sankaranarayanan. Identification of live news events using twitter. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, pages 25–32. ACM, 2011.

[12] Jayashree Khobarekar. *Detecting life events using tweets*. PhD thesis, Ph. D. dissertation, University of Illinois at Chicago, 2013.

[13] Jiwei Li, Alan Ritter, Claire Cardie, and Eduard Hovy. Major life event extraction from twitter based on congratulations/condolences speech acts. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1997–2007, 2014.

[14] Jiwei Li, Alan Ritter, and Dan Jurafsky. Inferring user preferences by probabilistic logical reasoning over social networks. *arXiv preprint arXiv:1411.2679*, 2014.

[15] Xueliang Liu, Raphaël Troncy, and Benoit Huet. Using social media to identify events. In *Proceedings of the 3rd ACM SIGMM international workshop on Social media*, pages 3–8. ACM, 2011.

[16] Luis G Moyano, Paulo R Cavalin, and Pedro P Miranda. Life event detection using conversations from social media. In *Brazilian Workshop on Social Network Analysis and Mining*, 2015.

[17] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] Swit Phuvipadawat and Tsuyoshi Murata. Breaking news detection and tracking in twitter. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 3, pages 120–123. IEEE, 2010.

[20] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860. ACM, 2010.

[21] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[22] Christian Torrero, Carlo Caprini, and Daniele Miorandi. A Wikipedia-based approach to profiling activities on social media. *arXiv preprint arXiv:1804.02245*, 2018.

[23] Konstantinos N Vavliakis, Andreas L Symeonidis, and Pericles A Mitkas. Event identification in web social media through named entity recognition and topic modeling. *Data & Knowledge Engineering*, 88:1–24, 2013.

[24] Xiaolong Wang, Furu Wei, Xiaohua Liu, Ming Zhou, and Ming Zhang. Topic sentiment analysis in Twitter: a graph-based hashtag sentiment classification approach. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1031–1040. ACM, 2011.