Predicción del éxito de los mensajes de *Twitter*

Máster en Ingeniería Informática, Facultad de Informática, Universidad Complutense de Madrid

Curso académico: 2017/2018
Autor: Manuel Alejandro Rodríguez Santana
Directores: Enrique Martín Martín y Adrián Riesco Rodríguez
Fecha: 5 de julio de 2018
Convocatoria: junio 2018
Calificación: 7.5

El/la abajo firmante, matriculado/a en el Máster en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: “Detección/Estudio/Predicción del éxito/popularidad de interacciones en redes sociales”, realizado durante el curso académico 2017-2018 bajo la dirección de Enrique Martín Martín y Adrián Riesco Rodríguez en el Departamento de Sistemas Informáticos y Computación, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en internet y garantizar su preservación y acceso a largo plazo

Índice

Resumen

Título: Predicción del éxito de los mensajes de Twitter

En la sociedad actual es habitual emplear las redes sociales, como por ejemplo *Twitter,* para anunciar y/o difundir múltiples productos, descubrimientos, noticias, etc. El objetivo de las personas que realizan estas publicaciones es que lleguen a ser muy populares, para que la información llegue al mayor número de personas posible. Sin embargo, no existe una noción de popularidad definida formalmente, ni una fórmula exacta que diga cómo lograrlo.

En este trabajo se ha escogido analizar únicamente la red social *Twitter,* acotando así el problema a estudiar. El fin del trabajo consiste en poder conocer cuando un *tweet* es popular o no, para posteriormente poder predecir cuándo un *tweet* se convertirá en popular antes de publicarlo. Para ello, primero se han recopilado una serie de *tweets* durante casi dos semanas con una temática en común, Cataluña. Luego se han procesado los datos almacenados calculando unas métricas que intentan medir dicha popularidad. Y por último se han aplicado algoritmos de aprendizaje automático, intentando obtener el modelo que prediga el éxito o no de un *tweet*.

El resultado final del trabajo no ha sido del todo satisfactorio, pero se han podido obtener una serie de conclusiones que pueden ser muy útiles de cara a posibles trabajos futuros.

Palabras clave: *Twitter, tweet*, popularidad, aprendizaje automático

Abstract

Title: Prediction of success of *Twitter* messages

In our current society is common to use social networks, such as *Twitter*, to announce and/or spread multiple products, discoveries, news, etc. The purpose of the people who perform these publications is that they become very popular, with the final objective that this information reaches the greatest possible number of people. Nevertheless, it does not exist any popularity notion formally defined, nor exact formulation of how to do it.

This work has chosen to study just the *Twitter* social network, limiting the trouble to research. The goal of this work is about knowing when a *tweet* is popular or not, to predict when a *tweet* becomes popular before publish it. First of all, a set of *tweets* were collected for almost two weeks with a common topic, Catalonia. Then, saved data was processed to measure some metrics to try to measure the popularity. Finally, some machine learning algorithms were applied to the processed data trying to get a model, which could predict if a *tweet* will be successful or not.

The final result was not satisfactory. However, we could obtain some useful conclusions, which could be used in future works.

Keywords: *Twitter, tweet*, popularity, machine learning

1. Introducción

Es un hecho que en los últimos años el uso de las redes sociales no ha parado de crecer. En 2018 hay más de 4021 millones personas en internet, de las cuales más de 3196 millones usan las redes sociales mensualmente [1]. Por lo tanto, a día de hoy es extraño quien en mayor o menor medida no comparte algo de su vida por la red. Algunos de estos sitios han tenido una fama efímera y otros aún perduran en nuestra sociedad. Un buen ejemplo de los efímeros es *MySpace*. Se fundó en 2003 y fue aumentando su éxito poco a poco hasta que en 2005 es comprada por 580 millones de euros. Sin embargo, su uso fue decreciendo hasta el punto de que en 2011 fue vendida por tan solo 35 millones [2, 3]

En contraposición con *MySpace* se encuentra *Facebook*. Fundada en 2004 y actualmente la red social más usada del mundo. A pesar de las muchas quejas de sus usuarios sobre la privacidad de sus datos y de los escándalos recientes sobre este mismo tema [4, 5], cuenta con 2167 millones de usuarios activos [1]. Otro ejemplo muy famoso a día de hoy es *Twitter*. Compañía fundada en 2006 y la cual cuenta con 330 millones de personas activas [1].

1.1. Objetivos

Debido al éxito de estas redes se están generando constantemente enormes cantidades de datos. Millones de personas diferentes, desde todo tipo de lugares y situaciones, compartiendo públicamente sus relaciones, opiniones, gustos, intereses, etc. Y esto, sumado a la mejora de las herramientas para procesado de datos y algoritmos de aprendizaje automático, abre un gran abanico de posibilidades.

El objetivo de este trabajo consiste en recopilar y analizar datos de estas redes sociales. Específicamente el trabajo estará centrado en *Twitter*, ya que sus usuarios se expresan principalmente con palabras. En contraposición con otras redes sociales más enfocadas al contenido multimedia como *YouTube* o *Instagram*. Gracias a que el contenido se encuentra en formato de texto, es más fácil de analizar que un video o una imagen. Además, en *Twitter*, las personas suelen exponer sus quejas y opiniones en directo sobre el tema que se esté tratando en ese momento. Lo que nos permite obtener opiniones de la población sobre diversos temas y analizarlos.

La idea general será recopilar un conjunto de datos inicial para analizarlos. Mediante este análisis se deberá decir cuáles de esos *tweets* cogidos son “exitosos” y cuáles no. Para ello se han definido una serie de métricas, las cuales serán explicadas más adelante. Con esto lograremos saber qué *tweets han cumplido mejor su objetivo de llegar a un mayor número de personas*. Luego, con esta información deducida se tratará de obtener algún clasificador que prediga si algo que se escriba *a posteriori* llegará a un gran número de personas o si se perderá entre toda la demás información de la red.

Un estudio como el propuesto en este trabajo podría ser especialmente útil para empresas o personas que deseen que sus ideas y/o productos lleguen al mayor número de personas posibles. Proporcionaríamos una “guía de estilo” para aquellas personas destinadas a

publicar contenido por las redes sociales. Con esta ayuda podrán saber cómo cumplir el objetivo de enviar información a sus seguidores de la manera más adecuada:

- ¿Deben enviar mucha información de golpe o poca?
- ¿Es mejor escribir por la mañana o por la noche?
- ¿Es bueno nombrar a otros usuarios o no?

1.2. Plan de trabajo

El trabajo se ha realizado en tres fases. La primera parte fue la recopilación de información. Consistió en buscar cómo se puede obtener la información de *Twitter*. En esta misma parte pensamos en una forma adecuada para medir esa "popularidad" de los *tweets*. Se comenzó a trabajar sobre esta fase a finales de octubre y se terminó a principios de diciembre.

En cuanto a la segunda parte del proyecto, consistió en primer lugar en descargar la información que se usó. En segundo lugar, en almacenar de forma óptima dichos datos para que pudiesen ser consultados y procesados de manera eficiente. Y por último en calcular las métricas de popularidad que se definieron en la primera parte. Esta parte del trabajo comenzó en paralelo a la primera fase, a finales de noviembre, y terminó a finales de febrero.

En la última parte del trabajo se hizo uso de los datos almacenados en el apartado anterior para entrenar algoritmos de regresión como por ejemplo *"*Random Forest Regression*",* que serán explicados más adelante. También se probó con algunos algoritmos de clasificación como *"*Naive Bayes*".* Con esto se pretendió conseguir una herramienta que pueda predecir si lo que escribamos llegará a muchas personas o no. No obstante, era importante que la opción desarrollada fuera escalable para que pueda ser empleada con grandes cantidades de datos. Esta última fase comenzó nada más terminar la fase anterior, a finales de febrero y se declaró el trabajo como finalizado a principios de mayo para poder escribir este documento y entregarlo en junio.

En los próximos capítulos de esta memoria se podrá encontrar una explicación de cómo y por qué se hizo todo lo que se hizo en este trabajo. Además, se realizará dos sugerencias de posibles trabajos futuros. Una breve descripción de cada capítulo sería:

- **Capítulo 2:** Explicación de todas las herramientas y algoritmos empleados en el trabajo, para que sea más fácil para el lector entender de todo lo que se habla más adelante.
- **Capítulo 3:** Explicación de las métricas que se decidieron emplear en este trabajo y que serán las que definan si un *tweet* es popular o no.
- **Capítulo 4:** Explicación del desarrollo para el cálculo de las métricas explicadas en el capítulo anterior.
- **Capítulo 5:** Explicación del desarrollo y utilización de las métricas calculadas para usar algoritmos de aprendizaje automático y así poder obtener predicciones.
- **Capítulo 6:** Explicación de las limitaciones y los problemas encontrados antes y durante el trabajo. Este apartado servirá para entender muchas de las decisiones tomadas en el transcurso del trabajo.

- **Capítulo 7:** Explicación de las conclusiones de este trabajo y de las posibles líneas de trabajo futuro

1. Introduction

It is a fact that in recent years the usage of social networks has never stopped growing. In 2018 there are more than 4021 million people on the internet, of which more than 3196 million are using social networks monthly. Therefore, today is strange who is not sharing something on internet to a greater or lesser extent. Some sites have had an ephemeral fame and others still have it. A nice example of ephemeral fame is *MySpace*. It was founded in 2003 and it was increasing its success little by little until 2005, when it was bought for 580 million euros. However, its usage was decreasing until it was sold for only 25 million [2, 3].

An opposite example to *MySpace* is *Facebook*. It was founded in 2004 and currently it is the social network most used in the world. Despite having numerous complains of its users about their personal data privacy and its recent well-published legal issues about the same topic [4, 5], it has 2167 million active users [1]. Another example today is *Twitter*. This company was founded in 2006 and it has 330 million active people [1].

1.1. Objectives

Due to the success of these social networks, huge amounts from data are being generated. Millions of different people, from all kinds of places and situations, are sharing their relationships, opinions, pleasures, etc. This opportunity, added to the improvement of the processing data tools and machine learning algorithms, opens a range of possibilities.

The aim of this work is to collect and analyze data of these social networks. Specifically the work will focus on *Twitter*, mainly because its users express themselves with words, contrary to other social networks, which are focused in multimedia content such as *Youtube* or *Facebook*. Given its text format, it is easier to analyze than a video or a picture. Furthermore, on *Twitter,* people usually expose their complains and opinions about any topic instantly. It enables us to get society opinions about a lot of topics to process them.

The general idea is to collect an initial data set of *tweets* to process them. By means of this analysis, it can be seen which tweets are more "successful" than others. For this purpose, metrics have been defined, which will be explained later. By the end of this study, we will get to know what *tweets* have better met their goal of reaching a large number of people. In this way, the deduced information will help us to create some classifiers and then predict if a *tweet* will be read by many people or if it will be quickly forgotten.

Studies like this could be very useful for companies or people which want their ideas and/or product reach to many people. We would provide a “style guide” for those people who are uploading data on internet. This “style guide” will help the people to complete theirs objectives, they will be able to share information in a proper way:

- Should they write a lot of information or just a little?
- Is it better to publish their content in the morning or in the night?
- Is a good idea mentioning other users?

1.2. Work plan

The research was carried out in three phases. The first part was the collection of information. It was about getting the information from *Twitter*. In addition, we thought about the best way to measure the *tweets*'s "popularity" too. This part started at the end of October and it ended at the beginning of December.

In the second project's part, first, the information was downloaded. Then, we had to save the data optimally, since it should be possible to consult and analyze it efficiently. Finally, we calculated the popularity measures, which were defined in the first project's part. This phase started at the end of November, it was done in parallel with the first phase and it ended at the end of February.

In the last project's part, we used the collected information in the previous part to train regression algorithms, such as "Random Forest Regression" for example, which will be explained later. Also, we tried with some classification algorithms, such as "Naive Bayes". The intention was to create a tool, which will be able to predict if something that someone publishes on *Twitter*, would reach many people. Nevertheless, an important fact was that the developed solution had to be scalable, because it could be used with huge data collections. This last phase started when the previous one ended, it was at the end of February. The work was declared ended at the beginning of May, because we had to write this document for June.

In the chapters to come of this document, a reason of how and why we did what we did in this project will be available. Furthermore, we will do two proposals of future work. Below there is a brief description of every chapter:

- **Chapter 2:** Explanation of everything that we used in this work, tools and algorithms. This explanation will make it easier to understand by the reader.
- **Chapter 3:** Explanation of the metrics used in this work and which will predict if a *tweet* is popular or not.
- **Chapter 4:** Explanation of the development to calculate the metrics explained in the last chapter.
- **Chapter 5:** Explanation of development and usage of the calculated metrics to employ them in machine learning algorithms and make predictions.
- **Chapter 6:** Explanation of the limitations and issues during the study. This chapter will serve to understand a lot of things about the decisions what we made.
- **Chapter 7:** Explanation of the conclusions of this work and possible lines of future work.

2. Preliminares

En esta sección se explicará las herramientas empleadas en este proyecto y algunos conceptos necesarios para entender el trabajo realizado.

2.1. Twitter

Twitter es una red social basada en *microblogging*. Consiste en escribir pequeños relatos que puedan ser contados mediante pocos caracteres. En un principio, cada *tweet* escrito en esta red podía tener como máximo 140 caracteres. Sin embargo, recientemente han aumentado ese límite al doble, 280 caracteres.

Este tipo de contenido tiene una serie de ventajas frente a los blogs tradicionales. Permite que se pueda subir contenido de forma más rápida, debido a que lleva menos tiempo su preparación. Esto permite subir contenido en tiempo real de lo que está sucediendo en ese momento y también fomenta subir más cantidad de contenido. Además, obliga al creador a pensar muy bien qué quiere poner y cómo lo quiere poner, gracias a que tiene un límite [6, 7].

Se pueden dividir sus 330 millones de usuarios activos en dos grupos. Por un lado estarían los *influencers*, que son las personas que logran que su contenido lleguen a una gran cantidad de personas. Por este motivo, actualmente forman parte de campañas de marketing muy populares por internet [8]. Luego, por otro lado tendríamos a los "usuarios normales", que son aquellas personas cuyo material solo suele llegar a su pequeño círculo y que además suele consumir el contenido de algún *influencer*.

Las herramientas que ofrece *Twitter* para que la gente use su plataforma y sea atractiva para los *influencers* son las siguientes [9].

- ***Usuarios**:* Cada persona dispone de su propia cuenta de usuario mediante la cual podrá *escribir tweets,* ser seguido o seguir a otras personas, y realizar diversas acciones como dale a "me gusta"*, retweet,* responder, citar, etc.
- ***Seguir**:* Cuando un usuario A comienza a seguir a otro B, todo lo que escriba o haga B le aparecerá a A en su *Timeline* y se podrá enterar en tiempo real de lo que está haciendo esa persona.
- ***Timeline**:* Es la interfaz principal de la aplicación. A cada usuario le aparece en ella el contenido de la gente que siga (ordenado por fecha y hora) y la publicidad que cree mejor para ti la plataforma. Esto es lo primero que te aparece nada más iniciar sesión con tu cuenta en *Twitter*.
- ***Tweet***: Es como el creador de contenido sube sus *tweets*. Principalmente está formado por texto que no supere los 280 caracteres. Sin embargo, también se le pueden añadir, imágenes, videos y encuestas, que no contarán para el número de caracteres total. Además, cada uno dispone de su propio contador de "Me gusta", *retweet* y número de respuestas.
- ***Retweet* (RT)**: Es cuando un usuario A escribe un *tweet* y luego otro usuario B le da al botón de *retweet*. Esto provocará que a todos los usuarios que sigan a B les

aparecerá en su *timeline* el *tweet* de A aunque no le sigan. Una persona puede quitar y poner el *retweet* varias veces a un *tweet* para que a sus seguidores les aparezca varias veces en su *timeline*.

- **Cita**: Es algo muy parecido a un *retweet*, pero en este caso a tus seguidores en lugar de aparecerle solo el *tweet* de la otra persona, también le aparece el texto que hayas decidido poner.
- **Respuestas:** Son los *tweets* que se ponen como contestación a otro *tweet.* Luego, se puede ver todo ordenado como en una conversación. Además, puede haber respuestas de respuestas, por lo que se pueden generar diversos debates de un mismo *tweet* original.
- ***Me gusta****:* Es simplemente un contador de veces que usuarios únicos (un mismo usuario no puede darle dos veces) le han dado al botón de “me gusta” a un *tweet*. Esto no provoca que aparezca en la *timeline,* sencillamente es un contador para que el creador pueda hacerse una idea de la gente a la que le gusta lo que haya puesto.
- ***Hashtags:*** Son “términos clave” que deben comenzar por el carácter “**#**”. Además, se pueden hacer búsquedas por *hashtags* y obtener los *tweets* de cualquier persona, los sigas o no, pero que contengan el *hashtag* que hayas buscado. Estos *hashtags* pueden ser cualquier palabra, sencillamente con que muchas personas los usen, ya podrán ser consultados. Es una herramienta típicamente usada para clasificar *tweets* por temas.

Figura 1: Ejemplo de *Timeline* de *Twitter*

En la figura 1 podemos ver un ejemplo de mi *Timeline* donde se aprecian varios de los elementos comentados anteriormente. En el primer círculo rojo podemos ver cómo el usuario “Phoebina” hizo un *retweet* de un *tweet* del usuario “WowChakra”, el cual yo no sigo, y por ello me aparece su *tweet*. En el segundo círculo rojo podemos ver un icono que

significa el número de respuestas que tiene ese *tweet*. Si estuviéramos interesados podríamos entrar y ver la conversación entera de forma ordenada. Luego los iconos que aparecen a continuación son el de *retweet* y el de *"*me gusta*"* respectivamente. Y por último, en el tercer círculo rojo podemos ver como la revista de videojuegos *Meristation,* está empleando *Twitter* para promocionarse. Además, emplea *hashtags* para clasificar ese *tweet* dentro de dos categorías "Zelda" y "BreathoftheWild", que son el nombre del juego que están promocionando en su noticia de su página web.

2.2. Recopilación de tweets

Una vez se ha entendido qué es *Twitter* y qué es lo que ofrece, el siguiente paso es cómo coger sus datos para poder analizarlos y procesarlos. Para esto se ha empleado la biblioteca de *Tweepy* [10]. Es una biblioteca para poder recopilar los *tweets* de *Twitter* a través de su API oficial en Python. Para utilizarla lo único que hace falta es instalarla en el ordenador y acceder a la API de *Twitter* oficial para obtener una clave de desarrollo de una aplicación que te permita acceder a sus datos.

A la hora de comenzar con la descarga de los *tweets* se podía elegir entre dos modalidades. La primera es escuchar a través de zonas geográficas. Y la segunda es escuchar mediante palabras clave. Debido a que el objetivo era centrarse en un tema en específico, la opción elegida para este trabajo fue la segunda. El tema elegido fue el de Cataluña, ya que por las fechas en las que se recopilaron los datos era un tema muy activo, por la fuerte corriente independentista del momento. Entonces, las palabras clave por las cuales se escuchaban los *tweets* eran: [cC]atalunya, [cC]ataluña y [cC]atalonia.

Después de nueve días con el *script* de *Python* en funcionamiento, se descargaron 43.4 GB. La fecha exacta en la que se puso en funcionamiento el *script* fue el 10 de octubre del 2017 y se apagó el 19 de octubre del 2017. Estas fechas coincidieron con la declaración de independencia ilegal de Cataluña, por lo que hubo una fuerte actividad en las redes sobre este tema y se pudieron recoger una gran cantidad de datos.

Un ejemplo del formato *JSON* en el cual se descargan los *tweets* de la API es el siguiente:


```
{
   "created_at":"Wed Oct 11 02:53:42 +0000 2017",
   "id":917946278864130049,
   "id_str":"917946278864130049",
   "text":"Dice Videgaray que no Reconocerá a Cataluña
           como Pais Independiente,Sr Videgaray yo ni siquiera
           Reconozco a México como Pais Independiente!!",
   "source":"<a href=\"http://twitter.com/download/iphone\
             rel=\"nofollow\">Twitter for iPhone</a>",
   "truncated":false,
```

```
"in_reply_to_status_id":null,
"in_reply_to_status_id_str":null,
"in_reply_to_user_id":null,
"in_reply_to_user_id_str":null,
"in_reply_to_screen_name":null,
"user":{
   "id":626983640,
   "id_str":"626983640",
   "name":"jesus octavio ",
   "screen_name":"jesusoct",
   "location":null,
   "url":null,
   "description":null,
   "translator_type":"none",
   "protected":false,
   "verified":false,
   "followers_count":163,
   "friends_count":372,
   "listed_count":0,
   "favourites_count":24,
   "statuses_count":880,
   "created_at":"Thu Jul 05 02:38:38 +0000 2012",
   "utc_offset":null,
   "time_zone":null,
   "geo_enabled":true,
   "lang":"es",
   "contributors_enabled":false,
   "is_translator":false,
   "profile_background_color":"C0DEED",
   "profile_background_image_url": "http://abs.twimg.com/images
                                    /themes/theme1/bg.png",
   "profile_background_image_url_https": "https://abs.twimg.com
                                          /images/themes/theme1/
                                          bg.png",
   "profile_background_tile":false,
   "profile_link_color":"1DA1F2",
   "profile_sidebar_border_color":"C0DEED",
   "profile_sidebar_fill_color":"DDEEF6",
   "profile_text_color":"333333",
   "profile_use_background_image":true,
   "profile_image_url": "http://pbs.twimg.com/profile_images/2891
                         675262 /4cb61ba1ed4fc00c32c2db56c2ba3cf4
```

```
                                _normal.jpeg",
      "profile_image_url_https": "https://pbs.twimg.com/profile_
                                  images/2891675262/4cb61ba1ed4fc00
                                  c32c2db56c2ba3cf4_normal.jpeg",
      "default_profile":true,
      "default_profile_image":false,
      "following":null,
      "follow_request_sent":null,
      "notifications":null
   },
   "geo":null,
   "coordinates":null,
   "place":null,
   "contributors":null,
   "is_quote_status":false,
   "quote_count":0,
   "reply_count":0,
   "retweet_count":0,
   "favorite_count":0,
   "entities":{
      "hashtags":[],
      "urls":[],
      "user_mentions":[],
      "symbols":[]
   },
   "favorited":false,
   "retweeted":false,
   "filter_level":"low",
   "lang":"es",
   "timestamp_ms":"1507690422155"
}
```


Este ejemplo es uno de los más pequeños que se pueden encontrar. Esto es debido a que es un simple *tweet, no es* de tipo cita o *retweet.* Si fuera de alguno de esos tipos aparecería dentro el *tweet* del que proviene. Entonces, puede darse el caso de que se generan *JSONs* mucho más grandes con muchos *tweets* encadenados. Además, si el texto que contiene fuera mayor de 140 caracteres (antiguo límite de *Twitter*) se generaría otro atributo que es "***extended_tweet***" y ahí habría varios atributos más sobre el *tweet* completo.

Ahora se procederá a explicar algunos de los atributos más importantes que contiene el *JSON* de ejemplo [11]:

- ***created_at***: Es la fecha en la cual se ha escrito el *tweet. Se encuentra en formato UTC, la cual* contiene la siguiente información ordenada y separada por espacios:
    - Día de la semana abreviado en inglés.
    - Mes del año abreviado en inglés.
    - Día del mes en formato numérico.
    - Hora, minutos y segundos en formato numérico separados por dos puntos.
    - Zona horaria en formato numérico.
    - Año en formato numérico.
- ***id*** e ***id_str****:* Es la *id* única por *tweet* que lo representa, en formato numérico y *string* respectivamente.
- ***text***: Aquí se encuentra el texto del *tweet* siempre y cuando no sea superior a 140 caracteres. En el caso de que sea superior, solo habrá una parte y para ver el texto completo habría que consultar el atributo "***full_text***" dentro del atributo "***extended_tweet***" que no se encuentra en este ejemplo.
- ***in_reply_to_status_id*** e ***in_reply_to_status_id_str***: Contiene *id* única del *Tweet* al que haya respondido si es una respuesta, en formato numérico y *string* respectivamente. En caso de que no lo sea, contienen *null*, como se puede ver en el ejemplo.
- ***in_reply_to_user_id*** e ***in_reply_to_user_id_str***: Contiene la id única del usuario al que se está respondiendo si es una respuesta, en formato numérico y *string* respectivamente. Al igual que en el caso anterior, si no lo es, contiene un *null*.
- ***user***: Es un objeto con varios atributos sobre el usuario que haya escrito el *tweet*. Algunos de estos atributos son:
    - ***id*** e ***id_str****:* Es la *id* única por usuario que lo representa, en formato numérico y *string* respectivamente.
    - ***name****:* El nombre del usuario.
    - ***location***: La localización del usuario si la ha puesto. En caso contrario como en el ejemplo, aparece *null*.
    - ***verified***: Es un *Booleano* que indica si la cuenta del usuario está verificada o no.
    - ***followers_count***: Número de seguidores que tiene ese usuario.
    - ***geo_enabled***: *Booleano* que indica si el usuario tiene activado o no la geolocalización, para que cuando escriba un *tweet*, se almacene en este desde donde se ha escrito.
- ***geo***: Es un campo que ya se encuentra obsoleto y el cual contiene las coordenadas geográficas desde donde se ha subido el *tweet* en formato [latitud, longitud].
- ***coordinates***: Lo mismo que el atributo anterior, pero este no se encuentra obsoleto y tiene el formato [longitud, latitud].
- ***is_quote_status****: Booleano* que indica si el *tweet* es una cita o no.
- ***quote_count****:* Número de citas que ha recibido el *tweet.*
- **reply_count**: Número de respuestas que ha recibido el *tweet*.
- ***favorite_count****:* Número de "me gusta" que ha recibido el *tweet.*
- ***retweet_count***: *Número de retweet* que ha recibido el *tweet*.
- ***entities****:* Es un objeto que contiene información sobre las distintas "entidades" que puede contener un *tweet*, que son las siguientes:

    - ***hashtags**:* Lista de objetos que contiene información sobre los *hashtags* que contiene el *tweet.*
    - ***urls**:* Lista de objetos que contiene información sobre las *urls* que contiene el *tweet.*
    - ***user_mentions**:* Lista de objetos que contiene información sobre los usuarios que menciona el *tweet*. Para que considere que mencionas a otro usuario, se tiene que poner un "@" delante del nombre de la personas que quieras mencionar.
    - ***symbols**:* Lista de objetos que contiene información sobre los *"*símbolos*"* que contiene el *tweet.* Para que se considere un "símbolo", debe cumplir con la siguiente expresión regular: `\$[a-z]{1,6}([._][a-z]{1,2})?` [12].
- ***lang**:* Idioma en el cual está escrito el *tweet*.

Luego, en el caso de que el *tweet* sea un *retweet*, aparecería en el *JSON* el atributo "**retweeted_status**", el cual contiene dentro el *JSON* del *tweet* al cual se le ha hecho *retweet*. En el caso de que fuera una cita, aparecería en este caso el atributo "**quoted_status**" y también contiene dentro el *JSON* del *tweet* al que citan. Para más información sobre los atributos que puede contener o no y que representan, se puede consultar la siguiente referencia [11].

2.3. MongoDB

Una vez extraídos los *tweets*, el siguiente paso era procesarlos y almacenar lo que fuera útil en alguna base de datos. La selecciona fue *MongoDB*, debido a que es gratuita, *open-source*, muy flexible, tiene un buen rendimiento y se podría escalar fácilmente si en el futuro se quisiera portar toda la infraestructura a la nube [13].

MongoDB es una base de datos *noSQL* orientada a documentos. Estos documentos se encuentran en formato *BSON* (*binary JSON*). Básicamente el formato *BSON* es la extensión del formato *JSON*, cualquier fichero escrito en este segundo podría ser almacenado como *BSON*. Sin embargo, aporta algunas ventajas como por ejemplo campos ordenados o más eficiencia al codificar y decodificar en distintos lenguajes [14]. Además, soporta más tipos de datos, algunos destacables son [15]:

- **Fechas en *Tiempo Unix* o *POSIX*** (milisegundos desde el 1 de enero de 1970 hasta la fecha que sea, representada como un entero de 64 bits).
- **Datos binarios** para poder almacenar por ejemplo imágenes.
- **Expresiones regulares**.
- **Código *javascript*** *(con alcance).*
- ***ObjectId*,** son ids únicas que sirven para identificar a un registro de forma inequívocamente. Algunas de sus propiedades son:
    - Están ordenadas.
    - Son rápidas de generar.

Sus principales características (motivos por los cuales se seleccionó esta y no otra base de datos) son [13]:

- Tiene un **gran rendimiento**. Su modelo de datos de documentos en *BSON* le permite reducir las actividades de entrada y salidas del sistema. Además, soporta la utilización y creación de índices para aumentar la velocidad de las consultas (herramienta muy empleada en este trabajo).
- Proporciona un **lenguaje de consultas enriquecido** que soporta operaciones de lectura y escritura. También soporta agregación de datos, consultas geoespaciales y búsqueda de textos.
- Dispone de una **gran disponibilidad**. Se puede emplear replicación de datos y proporciona seguridad ante fallos [16].
- Garantiza una **buena escalabilidad**. Otorga la posibilidad de distribuir y replicar los datos entre distintos nodos del cluster, permitiendo así distribuir la carga y evitar la sobrecarga de .un nodo en específico del sistema [17].
- Es **gratuita** y **open-source** bajo la licencia de *Free Software Foundation's GNU AGPL v3.0 [18, 19]*.

2.4. Neo4J

Dado que algunas de las métricas de popularidad que vamos a considerar se calculan mediante un grafo, se hace uso de Neo4J. *Neo4J* es la base de datos orientada a grafos más grande y con mayor comunidad activa. Cuenta con más de 3 millones de descargas y aumentando en un ratio de 50 000 por mes. Además, gracias a su comunidad activa se producen numerosos eventos y Meetups por año [20].

En Neo4J los datos se representan en lugar de por tablas como en el modelo relacional, como nodos y aristas. Estos elementos pueden ser de diferentes tipos y contener distintos atributos. Por ejemplo, se podría tener dentro de la misma base un nodo tipo "Película", otro nodo tipo "Persona" y dos tipos de relaciones, "Director" y "Actor". De esta forma se puede obtener un grafo donde relacionar las personas con las películas sabiendo si han actuado o dirigido.

A parte de su gran comunidad, otras de las razones por las cuales se seleccionó *Neo4J* y no otra base de datos orientada a grafos son: en primer lugar, que cuenta con el lenguaje de consultas de grafos *Cypher*. Este lenguaje proporciona una sintaxis sencilla y muy legible, mediante la cual se pueden realizar consultas sobre el grafo. Está inspirado en SQL, es declarativo y sigue el concepto de coincidencia de patrones. También permite realizar operaciones DML como inserciones, actualizaciones o borrados [21]. Un ejemplo de este lenguaje es este:

```
MATCH (t1:Nodo {id:$id})<-[r:RELTYPE *..]-(t2:Nodo) RETURN t2 as tweet
ORDER BY t2.level ASC
```

En este ejemplo, dada una *id,* devuelve todos los nodos hijos del nodo con la *id* que le hayamos pasado, independientemente del tipo de relación que tenga. Además se ordenan por el nivel de forma ascendente. Este ejemplo es una consulta realizada en este trabajo

para devolver todos los *tweets (que serían nodos)* que deriven de un *tweet* principal, independientemente de si es una respuesta, una cita o un *retweet*.

En segundo lugar, otro de los motivos por el cual se escogió Neo4J, es que supuestamente proporciona operaciones de escritura y lectura muy rápidas sin sacrificar en la integridad de los datos. En tercer lugar, proporciona una arquitectura escalable, lo que también lo hacía muy favorable para este trabajo. Y por último, en su versión "*Community Edition*", es *open-source* y gratuita bajo la licencia "*GNU General Public License (CPL) v3*" [20, 22, 23], parecido a *MongoDB*.

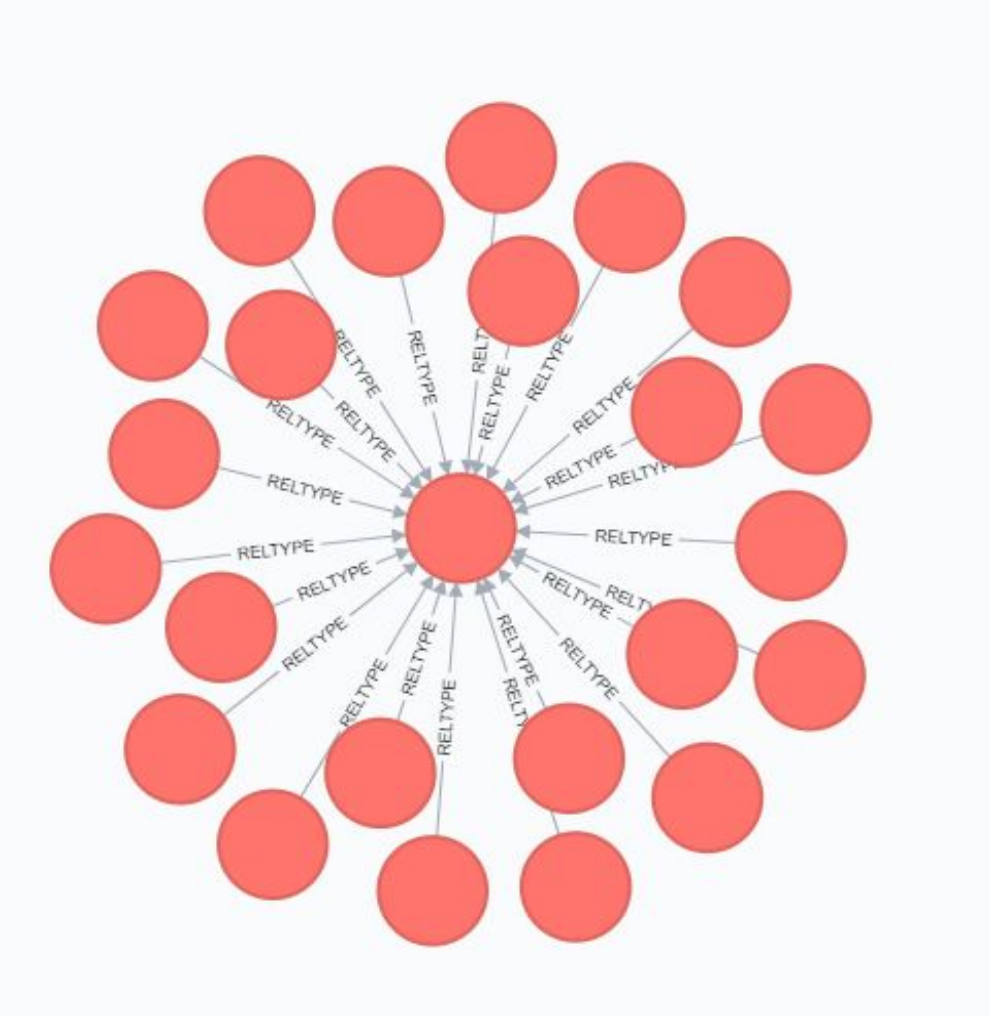

Figura 2: Ejemplo de un grafo de un *tweet* y sus *retweets*

En la figura 2 se puede observar un ejemplo de un grafo muy típico de *Twitter.* El nodo central representa el *tweet* original que puede haber escrito cualquier persona. Y luego, los nodos de alrededor representan cada uno un *retweet* realizado por alguna otra persona. En la imagen se han limitado el número de nodos que se mostraban, pero en realidad serían muchos más, *el tweet* central tenía 64 retweets.

2.5. Spark

Spark es una tecnología que proporciona un sistema de cómputo basado en clúster. Sus bibliotecas están disponibles para cuatro lenguajes de programación, *Java*, *Python*, *R* y *Scala*. Pone a la disposición de sus usuarios multitud de herramientas, algunas de las más destacables son [24]:

- *Spark SQL* para hacer consultas *SQL* sobre los datos aunque no se encuentren sobre una base de datos relacional.
- *SparkML* para trabajar todo lo relacionado con el aprendizaje automático.
- *Spark GraphX* para el tratamiento y procesamiento de grafos.
- *Spark Streaming* para el procesamiento de grandes cantidades de datos en tiempo real de forma eficiente.

Dentro de *Spark* se pueden almacenar los datos de dos formas. La primera de ellas es la antigua, mediante *RDDs*. La segunda manera, usada en este trabajo, es mediante *DataFrames*. Los DataFrames son una colección de datos organizados en columnas con nombres, conceptualmente son muy parecidos a la típica tabla de una base de datos relacional. Sin embargo, proporciona algunas mejoras de eficiencia gracias al motor optimizado de *Spark SQL* *[25, 26]*.

En cuanto a las características que hicieron que fuera esta la tecnología seleccionada para usar en el trabajo fue que es una tecnología basada en un sistema de clúster como ya se ha comentado. Esto permite que la solución desarrollada puede ser fácilmente escalable a grandes cantidades de datos portandolo a la nube y habilitando unas cuantas máquinas para ello. Además, es *open-source* y cuenta con una licencia *Apache Licence 2.0* [27].

La última característica importante es que cuenta con una comunidad bastante activa y que actualmente se encuentra en crecimiento, el equipo de desarrollo sigue trabajando en ella activamente. El número de líneas de código no ha parado de aumentar desde 2012, llegando a los dos millones (1.3 millones son de código de verdad, quitando comentarios y saltos de línea) en mayo de 2018 [28].

Figura 3: Gráfica del uso de Apache Spark hasta mayo de 2018 [28]

2.6. SparkML

Una vez se tenían los datos moldeados y almacenados en las bases de datos con sus métricas calculadas, el siguiente paso era analizarlos para sacar conclusiones. Para ello se empleó la última herramienta de este trabajo, *Spark*, con su biblioteca nueva de *DataFrames ML*, en lugar de la antigua de *RDDs MLlibs*.

Una vez se tienen los datos almacenados en *DataFrames*, estos se pueden usar para emplear el resto de herramientas que proporciona *Spark* como la consulta de datos mediante *SQL* o emplear funciones sobre toda la colección de datos, o sobre columnas específicas. No obstante, lo más relevante para este trabajo era poder emplear los algoritmos de regresión y de clasificación que proporciona la biblioteca de *SparkML*.

2.7. Aprendizaje automático

Como se ha comentado en el subapartado anterior, se han empleado, distintos algoritmos de aprendizaje automático que proporciona la biblioteca de *SparkML*, tanto de regresión, como de clasificación. Por lo tanto, aquí se procederá a explicar brevemente cómo funciona *grosso modo* cada uno de los algoritmos empleados.

2.7.1. Regresión

2.7.1.1. Regresión lineal

Se basa en relacionar de manera lineal una variable dependiente con una o más variables independientes, y una componente aleatoria. Se puede expresar mediante la siguiente fórmula:

$$Y_i = \alpha_0 \sum_{j=1}^{k} \alpha_j X_{ji} + \varepsilon_i$$

Donde:

- Y_i es la variable dependiente (que sería el valor de popularidad del *tweet*).
- X_{ji} son las variables independientes (que serían los atributos del *tweet*).
- α_0 y α_j son los parámetros desconocidos, los parámetros que se calculan durante el entrenamiento, para que luego el modelo pueda hacer las predicciones.
- ε_i es el error asociado a la medición de i del valor X_{ji}.

Luego, con el conjunto de datos de entrenamiento, donde deben estar todas las variables dependientes (Y) con sus variables independientes (X) se calculan los parámetros desconocidos α_0 y α_j a través del sistema de ecuaciones que se genera. Con esto calculado, ya se tendrá el modelo entrenado para que le puedan llegar solamente las variables independientes (X) y poder calcular Y gracias a los parámetros desconocidos ya calculados [29, 30].

2.7.1.2. Regresión lineal generalizada

La regresión lineal generalizada está pensada para unificar en el mismo contexto teórico cualquier modelo que siga un modelo de distribución de la familia exponencial, como es el caso por ejemplo de la distribución Gaussiana, la cual emplea la regresión lineal explicada en el algoritmo anterior. Sigue la siguiente fórmula [29, 31, 32]:

$$E(Y) = g^{-1}(\vec{X} \cdot \vec{\alpha})$$

Donde:

- E(Y) es el valor esperado para la variable dependiente.
- g representa la función de enlace, podría ser por ejemplo la de distribución Gaussiana como para el caso de la regresión lineal.

- $\vec{\alpha}$ vector que representa los parámetros desconocidos.
- $\vec{X}$ vector que representa las variables independientes.

2.7.1.3. Regresión mediante bosques aleatorios (*Random forest regression*)

Son una combinación de varios árboles de regresión. Un árbol de regresión consiste en averiguar qué atributos de los parámetros de entrada son los más relevantes para tener en cuenta en cada paso y luego clasificar de forma numérica (regresión) o categórica (clasificación). Un ejemplo de árbol de regresión para este caso podría ser el de la figura 4:

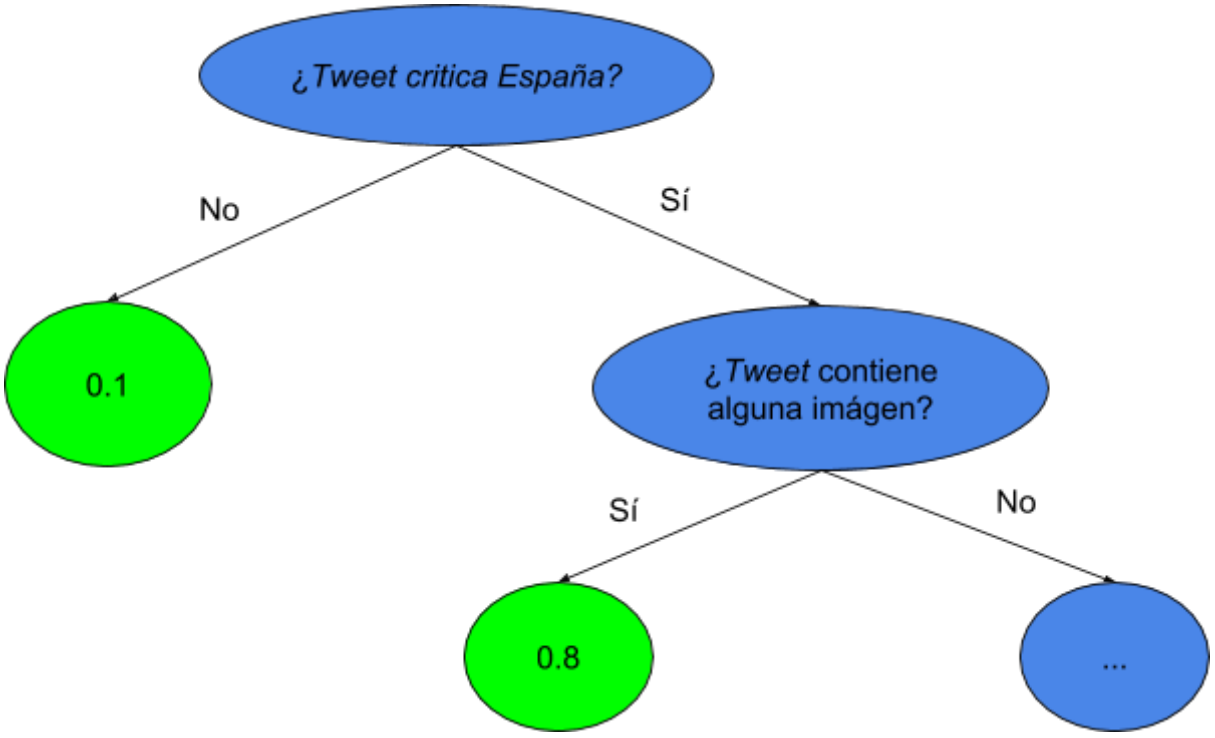

Figura 4: Fragmento de un ejemplo de árbol de regresión inventado sobre la popularidad de un *Tweet*.

En el árbol de regresión de la figura 4 se puede ver cómo el primer atributo que se ha considerado más importante es la temática del *Tweet*. En el caso de que no se critique a España el *tweet* tendrá poca popularidad. No obstante, si critica a España y además tiene alguna imagen será bastante popular. Y en el caso de que critique a España y no tenga alguna imagen continuaría estudiando otros atributos que ya no aparecen en el ejemplo.

Volviendo al algoritmo de la regresión mediante bosques aleatorios, consiste en combinar varios árboles de regresión como el anterior, con el objetivo de evitar el sobreentrenamiento del algoritmo. Primero se entrenan varios árboles de regresión por separado, donde hay una componente aleatoria que provoca que cada árbol sea diferente. Luego, se combinan las predicciones de cada uno y se comprueba cuál es mejor solución teniendo en cuenta lo que haya predicho cada árbol por separado [29].

Además, gracias a la implementación que tiene la biblioteca de *Spark ML* de este algoritmo, se le pueden pasar parámetros tanto continuos como categóricos.

2.7.2. Clasificación

2.7.2.1. Naive Bayes

Es un clasificador simple que se basa en el Teorema de Bayes y en probabilidad condicional. Asume que cada uno de los parámetros de entrada es independiente del resto y calcula la probabilidad condicionada de cada uno:

$$P(c_i \mid X_i, X_2, \ldots, X_n)$$

Después de calcular todas las probabilidades, se obtiene la mayor para ver qué clase tiene la mejor probabilidad. Se obtiene el resultado mediante la siguiente fórmula:

$$max_{c_i \in C} : (P(c_i) \prod_{j=1}^{n} P(X_j \mid c_i))$$

Donde:

- C son los valores posibles que puede tomar la variable que se quiere predecir, es decir, la clase. En este trabajo ha sido 1 para los *tweets* populares y 0 para los no populares.
- X_j son los valores que toma cada uno de los atributos, para ir calculando la probabilidad condicionada con cada uno.

Esto sería el algoritmo de *Naive Bayes* básico. Sin embargo, hay varios tipos con pequeñas variante. En el caso de la biblioteca de *Spark ML* soporta las variantes de "*Naive Bayes multinomial*" y de "*Naive Bayes Bernoulli*" [29].

2.7.2.2. Linear Support Vector Machine (SVM)

Consiste en construir un hiperplano en el espacio, sin importar el número de dimensiones del problema. Sin embargo, una limitación importante es que es un clasificador binario, es decir, aunque no importa el número de dimensiones del problema, solo puede clasificar entre dos clases. Esto es debido a que traza una línea (en el caso de dos dimensiones) o un hiperplano (en el caso de más de dos dimensiones) de separación entre las dos clases. Luego, todo lo que esté a un lado pertenece a esa clase y todo lo que se encuentre en el otro pertenece a la otra clase [29, 33].

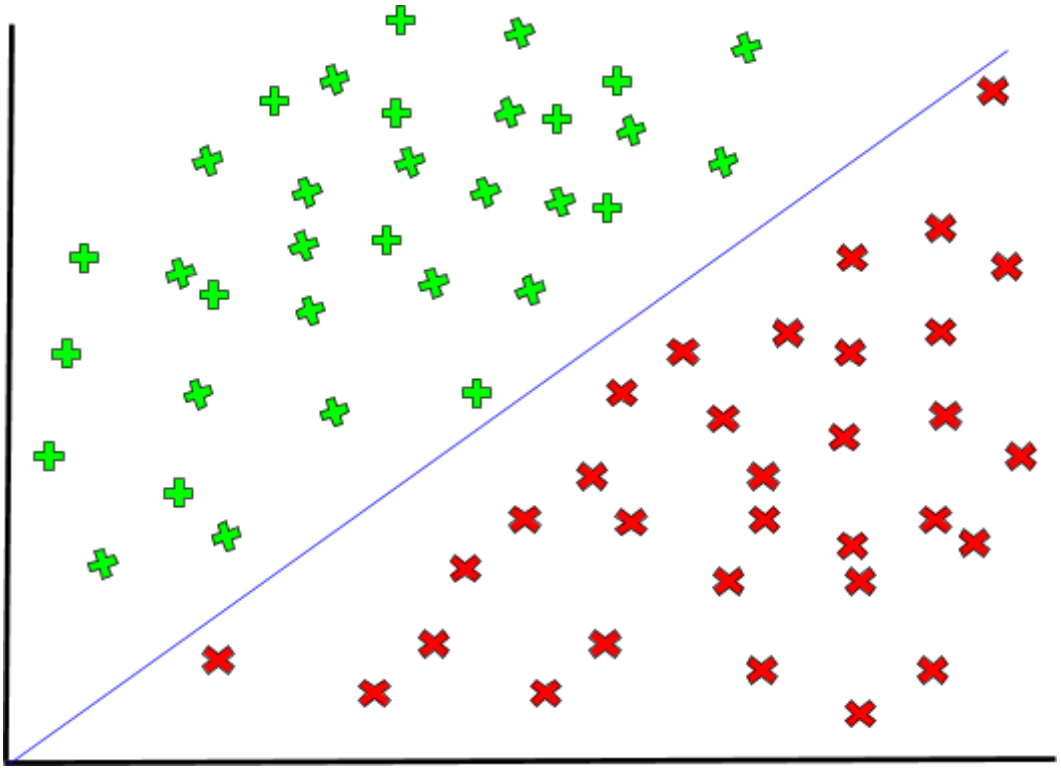

Figura 5: Ejemplo de una línea de separación entre dos clases del algoritmo “Linear Support Vector Machine”.

En la figura 5 se puede observar un ejemplo de cómo se ha calculado una línea de separación en un algoritmo de Support Vector Machine entre la clase de las cruces rojas que podrían ser los *tweets* no populares, y las cruces verdes que podrían ser los *tweet* populares. Entonces, cuando se tenga un nuevo *tweet* se comprobaría a qué lado de la línea se encuentra y esa sería su clasificación.

2.7.2.3. Clasificación mediante árboles aleatorios (Random forest classifier)

Es similar al algoritmo de “regresión mediante bosques aleatorios”. La implementación de *Spark ML* de este algoritmo permite que se pueda obtener un resultado, binario, multiclase o de regresión [29]. Esto es debido a que los árboles pueden ser de regresión o de clasificación sin problema y por lo tanto los bosques aleatorios funcionarán de la misma manera para clasificación y para regresión.

3. Métricas de popularidad

El primer paso en este trabajo consistió en definir de manera objetiva qué era un *tweet* exitoso y qué no. Lo más lógico inicialmente es pensar que si un *tweet* tiene mucha visibilidad será popular. Lo que lleva a pensar que lo más importante para que un *tweet* tenga éxito es que lo escriba alguien con muchos seguidores. No obstante, esto no es tan simple: si nos centramos solamente en las personas con mayor número de seguidores, no todos sus mensajes tienen la misma repercusión. Por ejemplo, tenemos el caso del famoso *youtuber* "elrubius", quien cuenta con nada menos que con 11,4 millones de seguidores en *Twitter* y ha escrito miles de *tweets* [34]. A pesar de su gran número de seguidores tiene mensajes con solamente 1700 *retweet* por ejemplo, pero también tiene otros con cientos de miles. Incluso uno de sus *tweets* ha entrado en el top 10 de los mayores *retweets* [33].

Por lo tanto, ya podemos comenzar a deducir que debe haber algo más aparte del número de seguidores para que un *tweet* se vuelva popular y sea exitoso. Ahora, si miramos el lado contrario, personas que no sean famosas y su número de seguidores sea mucho más pequeño, también podemos encontrar casos de mensajes muy populares. Tenemos el ejemplo del famoso vestido blanco y dorado, o azul y negro. Una usuaria de *Tumblr* no muy conocida colgó una imagen de un vestido preguntando por su color y generó un debate enorme [36].

Entonces, el objetivo de este punto era obtener una forma objetiva de saber cuándo un *tweet* había sido popular o no. Para ello se pensaron cinco métricas distintas que no solo se basen en el número de interacciones en bruto que ha tenido un *tweet*. Esto era un punto importante para poder identificar *tweets* exitosos tanto de usuarios con millones de seguidores, como de usuarios con solamente decenas de seguidores o incluso menos. Las métricas calculadas y utilizadas en este trabajo se explicarán en los siguientes subapartados.

3.1. Eficacia relativa

Esta métrica se encarga de medir el ratio de popularidad que ha tenido un *tweet*. En la base de datos de MongoDB se almacena como "*RS"*. Su objetivo es medir la visibilidad que ha tenido un *tweet* en base a sus interacciones, pero teniendo en cuenta el número de seguidores del usuario. Con esto se pretende conseguir saber cuándo un *tweet* ha sido popular para un usuario y que se pueda usar la métrica tanto para una persona con millones de seguidores como para alguien con muy pocos. Si por ejemplo alguien con 10 seguidores consigue un publicar un mensaje con 1000 interacciones, para él será un éxito. Sin embargo, si alguien con 10 millones de seguidores, solo consigue 1000 interacciones no será un éxito. Se calcula mediante la siguiente fórmula:

$$\frac{\alpha_1 RT + \alpha_2 QT + \alpha_3 RP}{Followers}$$

- α_1 α_2 y α_3

Donde:

- RT significa el número de *retweets* del *tweet*.
- QT significa el número de citas del *tweet*.
- RP significa el número de respuestas del *tweet*.
- α_1, α_2 y α_3 son coeficientes para darle un peso a cada uno de los elementos. En este trabajo se ha establecido el valor de uno para los coeficientes α_1 y α_2 a 1, y 0.5 para α_3. El motivo de esto es que las respuestas generan menos visibilidad que los *retweets* o que las citas, ya que no provocan que aparezca en la *Timeline* de tus seguidores directamente.
- Followers significa el número de seguidores del usuario.

El problema que tiene esta métrica es que para usuarios con muchos millones de seguidores los resultados obtenidos van a ser casi siempre muy bajos. Esto provocará que muchas veces se obtengan falsos negativos.

3.2. Potencialidad

Esta medida representa la visibilidad absoluta que puede tener un *tweet*. En la base de datos de *MongoDB* se almacena como "*visibility_value*". El objetivo de esta métrica consiste en obtener el número de personas máximas a la que ha llegado una publicación, se hayan fijado en ella o no. Corresponde a la siguiente fórmula:

$$(\alpha_1 \sum_{i \in RTID} Followers_i) + (\alpha_2 \sum_{j \in QTID} Followers_j) + (\alpha_3 \sum_{k \in RPID} Followers_k)$$

Donde:

- RTID son los identificadores de los usuarios que han hecho *retweet*.
- QTID son los identificadores de los usuarios que han citado el *tweet*
- RPID son los identificadores de los usuarios que han respondido el *tweet*
- $Followers_i$ es el número de seguidores que tiene el usuario de identificador i.

- α_1, α_2 y α_3 son coeficientes para darle un peso a cada uno de los elementos. Los valores que se les han establecido son los mismos que en la métrica anterior.

3.3. Eficacia absoluta

Esta métrica es muy parecida a la eficacia relativa y su nombre en la base de datos de *MongoDB* es "RSA". La diferencia entre la eficacia relativa y la absoluta es que en lugar de dividir entre el número de seguidores del usuario, se divide entre la potencialidad del tweet. La idea que tiene detrás esta métrica es saber realmente si un *tweet* ha sido popular o no. Por ejemplo, si alguien con 100 seguidores, escribe un un *tweet* con 1000 *retweet*s y una potencialidad de 100 000 tendrá una eficacia relativa alta porque ha llegado a muchas más personas de las que podía solo con sus seguidores, por lo que para él habrá sido popular. Sin embargo, tendrá una eficacia absoluta baja, porque de los 100 000 que pudo conseguir, solo ha conseguido 1000, lo que significa que no ha tenido tanto éxito en realidad. La fórmula empleada para calcularla es la siguiente:

$$\frac{\alpha_1 RT + \alpha_2 QT + \alpha_3 RP}{Potencialidad}$$

Donde:

- RT representa el número de *retweets* del *tweet*.
- QT representa el número de citas del *tweet*.
- RP representa el número de respuestas del *tweet*.
- α_1, α_2 y α_3 son coeficientes para darle un peso a cada uno de los elementos. Los valores que se les han establecido son los mismos que en los casos anteriores.
- "Potencialidad" es la métrica de potencialidad presentada anteriormente.

El problema que tiene esta métrica es similar al de la eficacia relativa, puede haber falsos negativos. Si una o varias personas con muchísimos seguidores citan o hacen *retweet,* elevará mucho la potencialidad del *tweet*. Esto dará lugar a que sea muy difícil obtener una eficacia absoluta alta y que puedan aparecer los falsos negativos.

3.4. Debate puro

Esta es una de las métricas que se calculan en el grafo generado en la base de datos de *Neo4J*, pero posteriormente se almacena en *MongoDB* bajo el nombre de "PD". Trata de medir el debate que se ha generado a través de un *tweet*. Para ello se suman todas las citas, *retweets* o respuestas que haya recibido un *tweet* y sus interacciones. El concepto consiste en que si un *tweet* ha generado un gran debate es que ha sido popular porque habrá llegado a muchas personas. Se ha empleado la siguiente fórmula:

$$RT + QT + RP + \sum_{i \in Childs} PD_i$$

Donde:

- RT significa el número de *retweets* del *tweet*.
- QT significa el número de citas del *tweet*.
- RP significa el número de respuestas del *tweet*.
- Childs son todas las interacciones directas que ha recibido un *tweet*. Si por ejemplo el *tweet* A recibe una respuesta, esa respuesta es un "Child" de A.
- PD_i es el valor de debate puro de cada uno de los hijos del *tweet* original. Entendiéndose como hijo las citas, *retweets* y respuestas del *tweet* original.

El problema que tiene esta forma de medir la popularidad de un *tweet* es que un hijo, por ejemplo una respuesta, nunca podrá ser más popular que su padre. Esto no debería ser así, en la vida real podría darse el caso de que una respuesta a un *tweet* sea mucho más popular que el *tweet* original.

3.5. Debate amortiguado por la profundidad

La última de las métricas calculadas consiste en un intento de mejorar la métrica anterior. También ha sido calculado en el grafo generado en la base de datos de *Neo4J* y almacenado en *MongoDB* como "MD". La idea principal es como la anterior, en detectar el debate que ha generado un *tweet* para calcular su éxito. Sin embargo, en esta métrica se tiene en cuenta también la profundidad de las interacciones. De esta manera se podría dar el caso de que un hijo sea más popular que su padre. Se ha empleado la siguiente fórmula:

$$\alpha_1 RT + \alpha_2 QT + \alpha_3 RP + \sum_{i \in childs}^{1} \frac{MD_i}{2}$$

- RT significa el número de *retweet* del *tweet*.
- QT significa el número de citas del *tweet*.
- RP significa el número de respuestas del *tweet*.
- α_1, α_2 y α_3 son coeficientes para darle un peso a cada uno de los elementos. Los valores que se les han establecido son los mismos que en los casos anteriores.
- Childs son todas las interacciones directas que ha recibido un *tweet*. Si por ejemplo el *tweet* A recibe una respuesta, esa respuesta es un "Child" de A.
- MD_i es el valor de debate amortiguado por la profundidad de cada uno de los hijos del *tweet* original. Entendiéndose como hijos las citas, *retweets* y respuestas del *tweet* original.

A continuación se pondrá un breve ejemplo para ilustrar mejor el funcionamiento de esta métrica y la anterior.

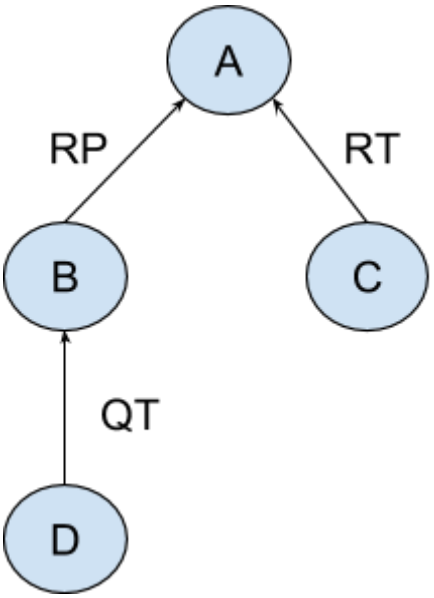

Figura 6: Ejemplo de un debate cualquiera de *Twitter.*

En la figura 6 se puede ver un grafo donde cada nodo representa un *tweet* y cada relación la interacción que los relaciona. En el caso de las hojas, D y C, tienen un PD de cero y un MD de 0 porque no han generado ningún tipo de debate. Luego, el nodo B tiene un PD de 1 porque tiene una cita y un MD de 1 también por el mismo motivo y como el PD y el MD de su hijo son 0, no hay que sumarle nada. Por último, en el caso de A, tiene un PD de 3, porque tiene 2 hijos directos (respuesta y *retweet),* pero además se le debe sumar el PD de B que es distinto de 0, es 1. El MD de A es 2, porque suma uno por el *retweet*, 0.5 por la respuesta y luego se le suma la mitad del MD de su hijo que B, lo que hace un total de 2. Entonces, podemos ver como el MD y el PD del *tweet* principal A es mayor que cualquiera de sus hijos, sin embargo, si no estuviera el *tweet* C sería distinto. En ese caso los valores de B no cambiarían, se quedaría con un PD de 1 y un MD de 1. Sin embargo, en A se quedaría con un PD de 2, todavía mayor que B, pero se quedaría con un MD de 1, quedándose con el mismo valor que su hijo B.

3.6. Unión de las métricas

Después de calcular todas las métricas comentadas anteriormente, se tomó la decisión de calcular una nueva métrica que fuera la unión de todas las demás. En *MongoDB* se ha almacenado esta medida como “AVG_M”. El motivo de tomar esta decisión es que cada una de las métricas parecía buena para una cosa, pero mala para otra. Cada una servía para medir el éxito que había tenido un *tweet* bajo una cierta circunstancia, pero no de forma general para cualquier caso. Entonces, se tomó la decisión de realizar una media ponderada entre todas las métricas.

La ponderación que se usó para todas las métricas fue de 1, excepto para la potencialidad, que fue de 0.5. El motivo de eso es que tras realizar diversas pruebas con ponderaciones distintas se descubrió que dejando esa métrica con una ponderación de 1 como las demás, influía demasiado y empeoraba los resultados de la métrica. Lo que ocurre con la potencialidad es que es una medida que contiene valores muy grandes, en torno a las

decenas de millones para algunos *tweets* de la *BBC,* que tiene 23 millones de seguidores cuando se recopilaron los datos. No obstante, también contiene valores más pequeños llegando a cero en algunos casos. Entonces, esto provoca que el rango de valores posibles sea demasiado grande y se desajuste la media de un *tweet* de forma en la cual no debería.

4. Cálculo de métricas

Para el cálculo de las métricas, el proceso se dividió en tres etapas, si obviamos los pasos intermedios de investigación y aprendizaje de las tecnologías y herramientas empleadas. La primera etapa fue la de recopilación de *tweets,* para obtener un conjunto de datos con el poder trabajar. La segunda fue el tratamiento de los datos recopilados y almacenarlos en las bases de datos de *MongoDB* y *Neo4J*, con los datos que se querían. La tercera y última etapa consistió en el cálculo de métricas como tal.

4.1. Recopilación de datos

Como se explicó en la sección de preliminares la herramienta seleccionada en este caso para acceder a la API de *Twitter* fue *Tweepy*, ya que es una biblioteca para *Python*, lenguaje de programación empleado en todo este trabajo. Era un paso importante la recopilación de datos inicial, porque de esa manera se podría trabajar siempre con el mismo conjunto de datos para poder hacer comparaciones de resultados según se iba avanzando en el trabajo. Esto es así porque la API de *Twitter* de forma gratuita no te permite acceder a datos históricos, solo te permite descargas *tweets* en tiempo real por zonas geográficas o por palabras clave.

Finalmente la opción elegida como ya se ha comentado fue la de recopilar los datos de *Twitter* por palabras clave, permitiendo así obtener datos sobre Cataluña, que ha sido el tema principal sobre el que se ha realizado el experimento. Las palabras clave seleccionadas fueron “[cC]atalunya, [cC]ataluña y [cC]atalonia”. Se recopilaron *tweets* de forma ininterrumpida entre los días 10 de octubre de 2017 y 19 de octubre de 2017, y los días 27 de octubre de 2017 y 2 de noviembre de 2017.El tipo de *JSON* que se genera puede ser visto en la sección de preliminares y se almacenaba en en una colección de *MongoDB* para posteriormente ser tratado en el mismo formato.

4.2. Procesado de datos

Una vez se tenían los datos en crudo en una colección de *MongoDB*, se procedió a crear un flujo de trabajo que fuera recorriendo cada *tweet* y lo procesara para crear dos colecciones más en *MongoDB*. Una de las colecciones almacenaría los datos que se consideraron importantes de los *tweets* y otra almacenaría un histórico de las interacciones entre los distintos *tweets.*

Lo primero que tenía que hacer este flujo de trabajo era dividir los distintos *tweets* que podían encontrarse dentro de un mismo registro de la colección obtenida de *Twitter*. Esto significa que si por ejemplo había una cita, dentro del JSON en crudo estaría el *tweet* de la cita, pero también estaría el *tweet* citado. Por lo tanto, el primer paso era dividir en dos esos *tweets* para que fueran procesados y almacenados por separado.

El resto de pasos a seguir en este proceso consistían en procesar los distintos *tweets* y almacenar los datos que fueran necesarios en cada una de las bases de datos usadas. Un

ejemplo de la colección de *MongoDB* donde se almacenaba la información de los *tweets es el siguiente:*


```
{
   "_id":NumberLong("917721385153323010"),
   "favorite_count":82,
   "coordinates":null,
   "retweet_count":63,
   "created_at":"Tue Oct 10 12:00:03 +0000 2017",
   "replyTo":null,
   "characters":111,
   "media":[...],
   "reply_count":6,
   "text": "Turull: “És greu que es pressioni perquè les empreses
          marxin de Catalunya però ja treballem en aquesta
          qüestión\" https://t.co/ZclyOaCxD2",
   "terms_count":18,
   "isReply":false,
   "user":{...},
   "quote_count":0,
   "mentions":[...],
   "hashtags":[...],
   "urls":[...],
   "quoteTo":null,
   "isLong":false,
   "isQuote":false,
   "symbols":[...],
   "lang":"und",
   "visibility_count_quote":0,
   "visibility_count_RT":6,
   "visibility_count_reply":0
}
```


La explicación de cada campo es la siguiente:

- ***_id***: Representa la *id* en formato numérico de cada *tweet*. Además, debido a que cada *tweet* tiene su propia *id* única, se usa para que sea la *id* única de cada registro de *MongoDB.* Con esto se consigue aprovechar el índice por defecto que se crea sobre el campo *_id* de forma automática y se podrá usar para acelerar las búsquedas.
- ***favorite_count****:* El número de “me gusta” que ha recibido el *tweet*.
- ***coordinates:*** Coordenadas geográficas del *tweet* en formato [longitud, latitud] o null si no se tienen las coordenadas.
- ***retweet_count***: Número de *retweets* del *tweet*.

- ***created_at***: Fecha y hora a la que se escribió el *tweet*. Se encuentra en el mismo formato en el cual lo devuelve *Twitter*, en UTC.
- ***replyTo***: *id* en formato numérico del *tweet* al que está respondiendo o null si no es una respuesta.
- ***characters***: Número de caracteres del texto del *tweet*.
- ***media***: Lista con los distintos elementos audiovisuales que pueda contener el *tweet*. Se capturan tal cual vienen los objetos con las medias del *tweet*.
- ***reply_count***: Número de respuestas que ha recibido el *tweet*.
- ***text***: Texto íntegro del *tweet*.
- ***terms_count***: Número de términos o palabras que contiene el *tweet*.
- ***isReply***: *Booleano* que indica si el *tweet* es una respuesta o no.
- ***User***: Los datos del usuario que ha escrito el *tweet*. Se toma el objeto de la misma forma en la que aparece en el *tweet*.
- ***quote_count***: Número de citas que ha recibido el *tweet*.
- ***mentions***: Menciones que hay dentro del *tweet. Se recopila la lista de objetos tal como viene en* entities/user_mentions del *tweet*.
- ***hashtags***: *Hashtags* que hay dentro del *tweet*: Se *recopila* la lista de objetos tal como viene en *entities/hashtags* del *tweet*.
- ***urls***: *Urls* que hay dentro del *tweet*: se *recopila* la lista de objetos tal como viene en *entities/urls* del *tweet*.
- ***quoteTo***: *id* en formato numérico del *tweet* al que está citando o null si no es una cita.
- ***isLong***: *Booleano* que indica si el *tweet* es largo o no. Se basa en si el texto para *Twitter* contiene más de 140 caracteres o no.
- ***isQuote***: *Booleano* que indica si el *tweet* es una cita o no.
- ***symbols***: *Símbolos* que hay dentro del *tweet*: Se recopila la lista de objetos tal como viene en *entities/symbols* del *tweet*.
- ***lang***: Lenguaje del *tweet,* obtenido de la información de *Twitter.*
- ***visibility_count_quote***: Sumatorio de los seguidores de las personas que han citado al *tweet*. Se va calculando durante la inserción de los *tweets* para que el sumatorio de los seguidores sea exacto. De forma que si en el futuro gana o pierde seguidores este valor no sea erróneo.
- ***visibility_count_RT***: Sumatorio de los seguidores de las personas que han hecho *retweet* al *tweet*. Se va calculando durante la inserción de los *tweets por el mismo motivo que "visibility_count_quote".*
- ***visibility_count_reply***: Sumatorio de los seguidores de las personas que han respondido al *tweet*. Se va calculando durante la inserción de los *tweets, por el mismo motivo que los dos campos anteriores.*

Ahora un ejemplo de la colección de *MongoDB* sobre el histórico de las interacciones generado en los datos coleccionados:


```
{
        "_id" : ObjectId("5a835ad69f418a21446fba47"),
        "date" : "Tue Oct 10 13:43:51 +0000 2017",
        "tweetId" : NumberLong("917747507098148864"),
```

```
        "type" : "RT",
        "toTweetId" : NumberLong("917721385153323010"),
        "userId" : 337143411
}
```

La explicación de cada campo es la siguiente:

- ***_id:*** Id única de cada registro, generado automáticamente por *MongoDB*.
- ***date:*** La fecha y la hora en la cual se ha realizado la interacción, se encuentra en el mismo formato en el cual lo devuelve *Twitter*, en UTC.
- ***tweetId:*** Id única del *tweet* que ha generado la interacción.
- ***type:*** Tipo de interacción que se ha generado. Solo puede ser RT para *retweet*, QT para citas y RP para respuestas.
- ***toTweetId:*** Id única del *tweet* sobre el cual se ha hecho la interacción.
- **userId:** Id única del usuario que ha generado la interacción

Por último, dos ejemplos de qué datos se almacenan en la base de datos de grafos *Neo4J*. Primero un ejemplo de los datos de un nodo:

```
{
        "<id>" : 17349,
        "id" : 918221412141322241,
        "root" : false,
}
```

Ahora, los ejemplo de una arista, que representaría a una interacción:

```
{
        "<id>" : 11661,
        "parentId" : 918221412141322241,
        "type" : RT
}
```

La explicación de cada campo es la siguiente:

- ***<id>:*** Id única de cada nodo o arista, generado automáticamente por *Neo4J*.
- ***id:*** Id única del *tweet* que representa el nodo.
- ***root:*** *Booleano* que indica si es el nodo más alto del grafo o no. Por lo tanto, solo tendrán a verdadero este valor si es un *tweet* normal, es decir, que no es una respuesta, ni cita, ni *retweet*.
- ***parentId:*** Id única del *tweet* al cual hace referencia una mención.
- ***type:*** Tipo de interacción que se ha generado. Solo puede ser RT para *retweet*, QT para citas y RP para respuestas.

4.3. Cálculo de las métricas

Una vez se tenían los datos correctamente almacenados en *MongoDB* y en *Neo4J*, ya se podía proceder al cálculo de las métricas. Para esto se crearon algunos *scripts* extra en *Python* para calcular algunas métricas que no podían ser calculadas fácilmente durante la creación de las colecciones, como por ejemplo el ratio absoluto o la discusión amortiguada por la profundidad. Cada uno de estos *scripts* se conectaba a la base de datos que le hiciera falta, leía todos los *tweets*, calculaba las métricas con las fórmulas correspondientes y las insertaba en la base de datos. En el caso de debate puro y debate amortiguado por la profundidad los *scripts* se conectaban tanto a *Neo4J* para realizar el cálculo con la ayuda del grafo, como a *MongoDB* para luego almacenar ahí también las métricas.

Entonces, un ejemplo de los atributos que se han añadido a la colección de *tweets* de *MongoDB* después de calcular las cinco métricas es el siguiente:


```
{
   "RSA":0.00036910480898826134,
   "MD":6,
   "RS":0.00037006094791672506,
   "PD":6,
   "visibility_value":178811
}
```


La explicación de cada campo es la siguiente:

- ***RSA***: Ratio de éxito absoluto de un *tweet.*
- ***MD***: Discusión amortiguada por la profundidad.
- ***RS***: Ratio de éxito de un *tweet*.
- ***PD***: Discusión pura.
- ***visibility_value:*** Potencialidad de un *tweet*.

Ahora un ejemplo de los atributos que se le añadieron a los nodos de Neo4J después de estos *scripts*:


```
{
        "childs" : 3,
        "visibility" : 3
}
```


La explicación de cada campo es la siguiente:

- ***childs:*** Número de nodos que cuelgan de él. Este atributo representa la métrica de debate puro explicada en la sección anterior.
- ***visibility:*** Valor que representa la métrica explicada anteriormente de debate amortiguado por la profundidad.

Luego, cuando ya se tenían calculada todas las métricas, solo quedaba unirlas todas en una, como se explicó antes. Para realizar la media ponderada entre todas las métricas no se podía hacer con los valores tal y como estaban, porque cada una tenía un rango de valores muy diferente. Por lo tanto, como no se quería que ninguna métrica fuera más importante que otra, el primer paso que había que hacer era escalar todas las métricas y después realizar la media de las métricas ponderadas. Un ejemplo de los atributos que se añadieron a la colección de *tweets* de *MongoDB* es el siguiente:


```
{
   "RSA_normalized":0.0000022034964068653183,
   "MD_normalized":0.0005528934758569849,
   "RS_normalized":0.000021515729834491763,
   "PD_normalized":0.0005345211581291759,
   "visibility_value_normalized":0.002927201809947136,
   "AVG_M":0.00027294242584411884,
}
```


La explicación de cada campo es la siguiente:

- ***RSA_normalized***: La métrica *RSA* normalizada al rango [0,1].
- ***MD_normalized***: La métrica *MD* normalizada al rango [0,1].
- ***RS_normalized***: La métrica *RS* normalizada al rango [0,1].
- ***PD_normalized***: La métrica *PD* normalizada al rango [0,1].
- ***visibility_value_normalized***: La métrica *visibility_value* normalizada al rango [0,1].
- ***AVG_M***: Media ponderada de todas las métricas normalizadas.

5. Aprendizaje de la popularidad

Una vez se tenían todos los datos deseados en *MongoDB* se pudo pasar a la última fase del trabajo, el aprendizaje automático. Para eso se hizo uso de la biblioteca *SparkML*. En este punto el flujo de trabajo se dividió en 3 fases. La primera consistió en adaptar los datos que se tenían en la base de datos a un *DataFrame*. Luego, una vez se tenían los datos en el *DataFrame*, se hicieron las operaciones necesarias con las herramientas de la biblioteca para transformarlo en una estructura que se le pudiera pasar a los algoritmos de aprendizaje automático ya explicados. El *DataFrame* tenía que tener las siguientes columnas:

- **label:** En esta columna se debe poner el atributo que debe predecir el algoritmo una vez entrenado. En el caso de los algoritmos de regresión se escogió el valor de la métrica "AVG_M". Y en el caso de los algoritmos de clasificación se estableció a 0 para los *tweets* con un valor de "AVG_M" inferiores a "0.00005363273894516184 6" y 1 a los demás. Ese valor era el menor de esta métrica dentro del 20% superior, es decir, se marcaron como exitosos al 20% de los *tweets* con mayor valor de "AVG_M".
- **features:** En esta columna se debe poner un vector con todos los parámetros de entrada que tendrá el algoritmo. Para este trabajo fueron los siguientes:
    - **Characters:** Número de caracteres del *tweet* normalizados en el rango 0, 1.
    - **Followers:** Número de seguidores de la persona que escribió el *tweet* normalizados en el rango 0,1.
    - **Verified:** 0 si la persona que escribió el *tweet* no tiene la cuenta verificada, 1 en el otro caso.
    - **Media:** 0 si el *tweet* no contiene ningún tipo de elemento multimedia, 1 si contiene cualquier tipo.
    - **Hashtags:** 0 si el *tweet* no contiene ningún hashtag, 1 si contiene cualquiera.
    - **Mentions:** 0 si el *tweet* no contiene ninguna mención a otro usuario, 1 si contiene cualquiera.
    - **RP:** 0 si el *tweet* no es una respuesta, 1 si lo es.
    - **QT:** 0 si el *tweet no es una cita, 1 si lo es.*
    - **Terms_Count:** Número de palabras que tiene el *tweet* normalizados en el rango [0,1].
    - **Morning:** 0 si el *tweet* no fue escrito entre las 07:00 y las 12:59, 1 en otro caso.
    - **Midday:** 0 si el *tweet* no fue escrito entre las 13:00 y las 15:59, 1 en otro caso.
    - **Afternoon:** 0 si el *tweet* no fue escrito entre las 16:00 y las 20:59, 1 en otro caso.
    - **Night:** 0 si el *tweet* no fue escrito entre las 21:00 y las 06:59, 1 en otro caso.

Ahora, es importante que se aclaren dos detalles de los parámetros anteriores:

- En el caso del número de caracteres se contaron manualmente en lugar de obtenerlos de la API de *Twitter* directamente porque había ocasiones en las cuales

ese dato no aparecía en el documento JSON. Entonces, para asegurar que se contaban todos de la misma manera se desarrolló esta funcionalidad. Además, a la hora de contar los caracteres no se tenían en cuenta los enlaces que pudiera haber en el texto.
- En el caso de la hora, se decidió emplear *One Hot Encoding*, transformando la hora en los 4 parámetros de “Morning”, “Midday”, “Afternoon” y “Night”. Se hizo de esta manera en lugar de normalizando los minutos del día por ejemplo para obtener un resultado más justo. De esta forma no habrá una separación más grande de lo que debería entre las 23:59 y las 00:01 por ejemplo.

En segundo lugar, con el *DataFrame* ya creado de esta manera se pasó a entrenar los algoritmos de regresión empleando validación cruzada simple. Para ello se dividieron los datos en tres conjuntos de 60% (entrenamiento), 20% (validación) y 20% (test). Se entrenaban los algoritmos con el conjunto de entrenamiento varias veces con distintos parámetros y se comprobaba con el conjunto de validación cuál producía mejor resultado. Luego, una vez se había seleccionado que parámetros daban mejor resultado se comprobaba con el tercer conjunto, el de test, qué resultados daban.

Para las comprobaciones entre los resultados se empleaba el error cuadrático medio. Los resultados obtenidos con los distintos algoritmos de regresión fueron los siguientes:

- **Regresión lineal:**
  - **Error cuadrático medio:** 9.982638450806017e-06
  - **Raíz cuadrada media (desviación típica):** 0.003159531365694289
  - **Error absoluto medio:** 0.00035686630923738836
- **Regresión lineal generalizada:**
  - **Error cuadrático medio:** 9.97678763250932e-06
  - **Raíz cuadrada media (desviación típica):** 0.0031586053302857134
  - **Error absoluto medio:** 0.00036601038596072725
- **Regresión mediante bosques aleatorios:**
  - **Error cuadrático medio:** 9.81288333020421e-06
  - **Raíz cuadrada media (desviación típica):** 0.0031325522070995413
  - **Error absoluto medio:** 0.00031322956623320 75

Como se puede observar son errores muy pequeños, por lo que se puede pensar que los resultados son increiblemente buenos. No obstante, si se observan más a fondo los resultados de estos modelos podemos ver que no son tan buenos. Lo que está ocurriendo es que siempre predice valores muy pequeños, es decir, absolutamente siempre predice que el *tweet* no va a ser popular. Por lo tanto, como en la vida real la inmensa mayoría de veces un *tweet* no es exitoso la mayoría de veces acierta, pero cuando el *tweet* sí es exitoso el modelo se equivoca siempre. Por lo tanto, ninguno de estos modelos resulta útil a pesar de su tasa tan pequeña de error, ya que no nos predecirá nunca que un *tweet* será popular.

En este punto se decide pasar a la última fase del trabajo: aplicar algoritmos de clasificación para ver si se obtienen mejores resultados. En este caso se emplea el mismo proceso que para los algoritmos de regresión. Se utiliza validación cruzada simple para obtener los mejores parámetros de los algoritmos. Los datos se dividen en tres conjuntos con los mismos porcentajes que antes. Sin embargo, al intentar crear los modelos siempre se

obtiene un error al emplear cualquiera de los tres algoritmos. Tanto para Naive Bayes, Linear Support Vector Machine, como para clasificación mediante árboles aleatorios se obtienen los siguientes errores:

- `java.lang.OutOfMemoryError: GC overhead limit exceeded.`
- `java.lang.OutOfMemoryError: Java heap space`

Sin importar qué se hiciera siempre se obtenían esos errores al intentar calcular cualquier algoritmo de clasificación en un ordenador en local. Después, se hizo una prueba con un conjunto muy pequeño de datos y el modelo se creó sin problemas. De esta situación se terminó deduciendo que el problema es que el cálculo de estos algoritmos de clasificación, con este conjunto de datos, necesita más capacidad de cómputo de la que se le pueda dar en un solo ordenador en local. Finalmente, se desistió de intentar calcular estos algoritmos, por lo que no se pueden mostrar resultados como de la regresión.

6. Limitaciones, problemas y soluciones

En este apartado se expondrán las limitaciones que han llevado a la toma de algunas decisiones en este trabajo. También se hablará de algunos de los problemas que han surgido durante la fase de desarrollo y que han provocado la pérdida de mucho tiempo en buscar y desarrollar soluciones.

6.1. Limitaciones

La primera y más importante limitación de todas es que el presupuesto para el desarrollo del trabajo es de 0 euros. Esto limita inmediatamente el montar arquitectura alguna en la nube. A pesar de que la mayoría de plataformas como *Amazon Web Service* o *Microsoft Azure* ofrecen parte de sus servicios gratis o dan ciertos "créditos" para comenzar a usarlos gratis y posteriormente pagar, esto no era suficiente para la cantidad de datos que se querían usar y el tiempo de cómputo que iba a ser necesario.

Esta primera limitación también ha afectado a la recopilación de datos de *Twitter*. De forma gratuita solo se pueden recopilar *tweets* de la actualidad, seleccionado si se quieren recopilar por zona geográfica o por palabras clave. No permite la obtención de *tweets* del pasado. En este trabajo se recopilaron los *tweets* por las palabras clave [cC]atalunya, [cC]ataluña y [cC]atalonia. Sin embargo, la mayoría de respuestas y citas que generan esos *tweets* no tienen esas mismas palabras clave, por lo que no se recopilan.

La falta de obtención de muchos *tweets* provoca que algunas métricas sean inexactas o incluso erróneas. En específico, las métricas más afectada por esta falta de datos son las debate puro y la de debate amortiguado por la distancia. Esto es debido a que son medidas que se basan en el grafo de relaciones que se genera en base a un *tweet.* Por lo tanto, tras recopilar el *tweet* principal que sí usa alguna de las palabras clave, luego nos faltarían la mayoría de nodos hijos. Por lo tanto, los valores de estas dos métricas se han visto muy afectadas por esta limitación en los datos.

La segunda limitación, que deriva de la primera, es la capacidad de cómputo. Como no se hace uso de ningún servicio en la nube, todas las herramientas deben estar ejecutándose en local. Se debe estar ejecutando las bases de datos *MongoDB y Neo4J*, mientras además se está realizando el tratamiento de los datos. El ordenador utilizado para todas las pruebas contaba con un procesador *Intel Core i7 6700*, el cual dispone de 4 núcleos físicos y 8 núcleos virtuales. Además se disponía de 16 GB de memoria RAM DDR4 a 2167 MHz. Con estas especificaciones durante las pruebas realizadas se alcanzaba un uso del 90% de CPU y 5 GB de memoria RAM.

Inicialmente se recopilaron 48 GB de datos de *tweets* con la idea de emplearlos todos para el entrenamiento de los algoritmos de aprendizaje automáticos. No obstante, debido a la limitación de potencia de cómputo finalmente se han empleado solamente 2.5 GB de datos, los correspondientes a los *tweets* de los días 10 y 11 de octubre de 2017. Solamente con esta cantidad de datos terminó tardando 3 horas y 59 minutos en realizar las dos primeras fases del proyecto, el procesamiento de los datos y el cálculo de métricas. Lo que suponía

un verdadero problema, cada vez que se quería cambiar algo y había que estar seguro de que estaba bien antes de realizar las pruebas con este conjunto de datos.

6.2. Problemas

En cuanto a los problemas encontrados, el primer problema de todos está muy relacionado con la segunda limitación. En las primeras etapas del trabajo, cuando todavía se encontraba en fase de desarrollo las dos primeras fases de procesamiento de los datos y cálculo de las métricas, se descubrió que la base de datos orientada a grafos *Neo4J* es muy lenta para las operaciones DML en comparación con *MongoDB*.

Al principio se hacían pruebas con un conjunto de datos pequeño (300 kB) y tardaba 2 minutos y 20 segundos en total. Luego, se probó a realizar las mismas operaciones, pero sin conectar *Neo4J* y el tiempo se redujo a 2 segundos. Para solucionar esto se invirtió mucho tiempo en optimizar todo lo posible el proceso inicial del tratamiento de datos y cálculo de métricas. Se crearon índices meticulosamente para *Neo4J,* se estudiaron las mejores formas de realizar las consultas con *Cypher* y se almacenó solamente la información necesaria. Sin embargo, lo que mejor resultado dio fue reducir todo lo posible las operaciones en *Neo4J* y aumentar las operaciones a *MongoDB*. Además, hubo algunos procesos que se decidió no hacer durante el tratamiento inicial de los datos, sino calcularlos a posteriori con otros *scripts*. Con todos esos cambios, se logró pasar de 2 minutos y 20 segundos para 300 kB de datos, a 3 horas y 59 minutos para 2.5 GB. Esto supuso una mejora en tiempo muy considerable para procesar esa cantidad de datos, puesto que de la otra forma hubieran hecho falta más de 291 horas. Este problema anterior supuso un gran esfuerzo en cuanto a tiempo, hubo que consultar mucha documentación y foros de dudas.

El segundo gran problema fue con la biblioteca *SparkML*. Durante el desarrollo del trabajo aparecieron muchísimos problemas relacionados con esta biblioteca. El primero de todos fue en la instalación en *Windows 10 Education*. A la hora de intentar usar el conector con *MongoDB* que supuestamente ya viene con la biblioteca se obtenía un error de *Java* que indicaba que no encontraba un método. Lo que ocurría es que por algún motivo desconocido faltaba un fichero ".jar" en la instalación y se tuvo que buscar por internet, descargarlo y situarlo en el directorio correspondiente.

Posteriormente, cuando ya se tenía instalado correctamente y se comenzaron las primeras pruebas, aparecieron algunos pequeños errores debido a la poca y mala documentación de la herramienta. Y para terminar con los problemas de esta herramienta, destacar que algunas funcionalidades que supuestamente deberían funcionar no lo hacían. Por ejemplo, si se emplean los *pipelines* daba un error al intentar guardar el modelo generado. Esto obligó a no usar los *pipelines,* porque poder guardar el modelo generado era algo imprescindible. Otro ejemplo de algo importante que estaba disponible pero fallaba es la validación cruzada. Se podía emplear la validación cruzada e iba bastante más rápida que la implementada en este trabajo. Sin embargo, al igual que con los *pipelines*, al intentar guardar el modelo generaba un error no controlado y se paraba la ejecución.

El último de los problemas encontrados con *SparkML* fue el ya comentado en el apartado del cálculo de métricas. No funcionaba ningún algoritmo de clasificación, devolvía los

mismos errores no controlados siempre que el conjunto de datos fuera muy grande. Posiblemente si se ejecuta el código en un cluster con más potencia funcione, pero para este trabajo no se dispuso del tiempo necesario para investigar este problema en profundidad y buscar otra solución, por lo que se optó por solamente estudiar algoritmos de regresión.

7. Conclusiones y trabajo futuro

El trabajo comenzó con una idea muy optimista: poder identificar cuándo un *tweet* iba a ser popular antes de publicarlo. Un trabajo de este ámbito con éxito hubiera atraído una gran cantidad de empresas y personas interesadas En cualquier negocio *B2C* (*business to customer*) es muy importante generar una gran cantidad de candidatos, para lo cual actualmente un medio muy popular de captación son las redes sociales. Por lo tanto, si pudieran identificar cómo deberían escribir sus *tweets* para que fueran populares antes de publicarlos y de realizar estudios previos, sería algo muy beneficioso.

No obstante, debido a las limitaciones, falta de experiencia y la falta de tiempo, la idea no se ha podido ejecutar satisfactoriamente. Los modelos de regresión aciertan la mayoría de veces, pero a costa de predecir que nada va a ser popular, por lo que no resultará de ayuda a nadie. Consideramos que este fracaso se ha debido principalmente a dos factores. El primero, consiste en que se ha analizado una cantidad de datos relativamente pequeña, 2.5 GB de datos correspondiente a 2 días. El total del que se dispone son 48 GB, que han sido recopilados durante más de 10 días.

El segundo de los factores consideramos que consiste en que se han analizado pocos atributos del *tweet*. Los atributos que se usan para los algoritmos se pueden ver en la sección de “Aprendizaje de la popularidad”, pero por ejemplo el texto del *tweet* es un campo muy poco utilizado. Del texto se cuentan los caracteres y las palabras, para identificar cuando un *tweet* es largo o no, pero no se tiene en cuenta lo que dice, ni el sentimiento de lo escrito.

Sin embargo, el trabajo realizado no tiene porqué haber sido inútil. Se abren dos posibilidades como líneas futuras de este trabajo. La primera línea consistiría en conservar todo el trabajo realizado, pero portarlo a una arquitectura en la nube, donde se puedan analizar una cantidad de datos mayor. Luego, donde habría que invertir más esfuerzo es en idear alguna forma inteligente y adecuada de utilizar el contenido del texto de los *tweets*. Por ejemplo se podría realizar un análisis de sentimiento sobre el texto. Esta idea se ha considerado varias veces durante el desarrollo del trabajo. Esto es debido a que en el análisis de los *tweets* de manera manual, se ha podido ver una tendencia en que los *tweets* más populares dentro de la temática de este trabajo suelen criticar a España.

La segunda de las posibles líneas futuras consistiría en idear nuevas métricas de popularidad que reflejen mejor si un *tweet* es popular o no. Por lo tanto, lo primero que habría que hacer es buscar nuevas métricas que se puedan aplicar para saber si un *tweet* es exitoso o no. Después se podrían clasificar los *tweets* de la misma manera, empleando los mismos parámetros, aunque no lo aconsejamos. Se han observado los parámetros estudiados de los 20 *tweets* más exitosos según las métricas empleadas y se han obtenido los siguientes resultados:

- Hay 69 180 con 9 o más atributos iguales.
- Hay 16 013 con 10 o más atributos iguales.
- Hay 306 con 11 o más atributos iguales.

- No hay ninguno que tenga 12 o más atributos iguales.

Se han estudiado 13 parámetros y el total de *tweets* eran 404 000 *tweets*. De estos resultados se puede observar como existen una gran cantidad de *tweets* clasificados como no exitosos muy parecidos en el número de parámetros iguales a los si clasificados como tal. Por lo tanto, debido a esta contrariedad, se puede deducir el motivo por el cual los algoritmos de regresión no dieron buenos resultados. Además, es muy probable que hubiese ocurrido lo mismo con los algoritmos de clasificación si se hubieran podido emplear. Este es el motivo por el cual no se aconseja seguir utilizando los mismos parámetros aunque se cambien las métricas. Se debería ampliar el número de parámetros a estudiar del *tweet*, como por ejemplo analizando su texto, con el fin de evitar esta contradicción en los datos que empeora los algoritmos.

7. Conclusions and future work

This project started with an optimistic idea: it should be able to identify when a *tweet* could be popular before it was published. If the work were successful, it would be interesting for many people and big companies. Any B2C (business to customer) business has an interest in common: they want to be popular because of their publications on social networks. For that reason, if they could know how they should write their publicity, before it was uploaded to the social networks, they will be very interested and it will be beneficial for them.

However, the idea couldn't be done successfully, due to limitations, lack of experience and lack of time. Regression models have a high accuracy, but this is because they always predict that anything won't be popular. Then, they will be useful for nobody. We think that this failure was caused by two factors: the first of these factors is based on the small dataset used, it was just 2.5GB corresponding to 2 days. The total amount of data was 48GB, it was recollected during more than 10 days.

The second of these factors we believe that it is because we analyzed little set attributes of the *tweet*. In the chapter "Aprendizaje de la popularidad" we described what attributes were analyzed, but the *tweet*'s text is a field little used for example. We count the numbers of characters and the number of words to know when a *tweet* is long. Nevertheless, we didn't study what the text is saying nor a sentiment analysis.

However, the project is not useless. It opens two possible future work lines. The first one could be using everything what we did, but move it to the cloud, where bigger amount of data can be processed. Then, more time should be invested to think new smart ways of use the tweet's text. A sentiment analysis could be done for example. This idea was considered many times during the project. The reason is that we did a manual analysis of the data, where we could see how the more popular *tweets* used to criticize to Spain in this topic.

The second possible future work line is to think new ways of measuring the popularity of a *tweet*, where we could see better if a *tweet* is popular or not. For this line, first of all we have to search new metrics, which could be employed to know when a *tweet* is successful. After that, we can classify *tweets* in the same way. Although, we didn't approve it, because we have observed different parameters studied of the 20 most popular *tweets* with our metrics and we got the following results:

- There are 69 180 with 9 or more equal attributes.
- There are 16 013 with 10 or more equal attributes.
- There are 306 with 11 or more equal attributes.
- There is none that it has 12 or more equal attributes.

404 000 *tweets* with 13 features were studied. we could notice that there are a lot of *tweets* classified as no successful very similar to the *tweets* classified as successful taking into account the number of equals attributes. Therefore, because of this contrariety, we could deduct the reason why the regression algorithms weren't effective. Furthermore, it's possible that we will get the same issues with classification algorithms, if we use it. For that reason, we didn't advise to employ the same attributes even if new metrics are used. The number of

tweet's parameters to study should be bigger, in order to avoid contradictory data, which would deteriorate the algorithms.

Bibliografía

1. Wearesocial (n.d): World's internet users pass the 4 billion mark. Disponible en: https://wearesocial.com/blog/2018/01/global-digital-report-2018 [Consulta: 25 de abril de 2018].
2. Wikipedia (9 de abril de 2018): MySpace. Disponible en: https://es.wikipedia.org/wiki/Myspace [Consulta: 24 de abril de 2018].
3. Spangler, Todd (n.d.): Time Inc. Buys Myspace Parent Company Viant. Disponible en: https://variety.com/2016/digital/news/time-inc-myspace-viant-1201703860/ [Consulta: 24 de abril de 2018].
4. Spangler, Todd (n.d.): Facebook under fire: How privacy crisis could change big data forever. Disponible en: http://variety.com/2018/digital/features/facebook-privacy-crisis-big-data-mark-zuckerberg-1202741394/ [Consulta: 25 de abril de 2018].
5. Lomas, Natasha (10 de abril de 2018): A brief history of Facebook's privacy hostility ahead of Zuckerberg's testimony. Disponible en https://techcrunch.com/2018/04/10/a-brief-history-of-facebooks-privacy-hostility-ahead-of-zuckerbergs-testimony/ [Consulta: 25 de abril de 2018].
6. Giraldo, Valentina (n.d.): Twitter y el microblogging: al dominio de la brevedad. Disponible en: https://marketingdecontenidos.com/microblogging-y-twitter/ [Consulta: 26 de abril de 2018].
7. 1&1 (18 de mayo de 2017): Microblogging: todo lo que necesitas saber. Disponible en: https://www.1and1.es/digitalguide/online-marketing/redes-sociales/microblogging-todo-lo-que-necesitas-saber/ [Consulta: 26 de abril 2018].
8. Mglobal marketing (n.d.): Qué es un influencer y por qué lo necesitas para mejorar tu estrategia de Marketing. Disponible en: https://mglobalmarketing.es/blog/que-es-un-influencer-y-que-aporta-a-tu-estrategia-de-marketing/ [Consulta: 26 de abril de 2018].
9. Rodríguez Machío, Álvaro (n.d.): Twitter para novatos. Disponible en: http://joventut.pallarssobira.cat/sites/default/files/adjunts/Twitter-para-novatos.pdf [Consulta: el 26 de abril de 2018].
10. Tweepy (n.d.): Tweepy Documentation. Disponible en: http://tweepy.readthedocs.io/en/v3.5.0/ [Consulta: 26 de abril de 2018].
11. Developer Twitter (n.d.): Tweet objects. Disponible en: https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object [Consulta: 28 de abril de 2018].
12. Roomann-Kurrik, Arne (12 de abril de 2013): Symbols entities for tweets. Disponible en: https://blog.twitter.com/developer/en_us/a/2013/symbols-entities-tweets.html [Consulta: el 28 de abril de 2018].
13. Documentation MongoDB (n.d.): Introduction to MongoDB. Disponible en https://docs.mongodb.com/manual/introduction/ [Consulta: 2 de mayo de 2018]
14. MongoDB (n.d.): JSON and BSON. Disponible en https://www.mongodb.com/json-and-bson [Consulta: 2 de mayo de 2018].

15. Documentation MongoDB (n.d.): BSON Types. Disponible en https://docs.mongodb.com/manual/reference/bson-types/ [Consulta: 2 de mayo de 2018].
16. Documentation MongoDB (n.d.): Replication. Disponible en https://docs.mongodb.com/manual/replication/ [Consulta: 2 de mayo de 2018].
17. Documentation MongoDB (n.d.): Sharding. Disponible en https://docs.mongodb.com/manual/sharding/#sharding-introduction [Consulta: 2 de mayo de 2018].
18. MongoDB (n.d.): MongoDB Licensing. Disponible en https://www.mongodb.com/community/licensing [Consulta: 2 de mayo de 2018].
19. GNU Operating System (19 de Noviembre de 2007): GNU affero general public license. Disponible en http://www.gnu.org/licenses/agpl-3.0.html [Consulta: 2 de mayo de 2018].
20. Neo4j (n.d.): Top ten reasons for choosing Neo4j. Disponible en https://neo4j.com/top-ten-reasons/ [Consulta: 3 de mayo de 2018].
21. Neo4j (n.d.): Cypher, the graph query language. Disponible en https://neo4j.com/cypher-graph-query-language/ [Consulta: 3 de mayo de 2018].
22. Neo4j (n.d.): About Neo4J licenses. Disponible en https://neo4j.com/licensing/?ref=open-source [Consulta: 3 de mayo de 2019].
23. GNU Operating System (29 de junio de 2007): GNU general public license. Disponible en https://www.gnu.org/licenses/gpl.html [Consulta: 3 de mayo de 2019].
24. Apache1 Spark (n.d.): Spark Overview. Disponible en https://spark.apache.org/docs/latest/index.html [Consulta: 9 de mayo de 2018].
25. Apache Spark (n.d.): Machine learning Library (MLlib) guide. Disponible en https://spark.apache.org/docs/2.3.0/ml-guide [Consulta: 3 de mayo de 2019].
26. Apache Spark (n.d.): Spark SQL, DataFrames and Datasets guide. Disponible en https://spark.apache.org/docs/2.3.0/sql-programming-guide.html [Consulta: 5 de mayo de 2018]
27. Apache Software Foundation (enero de 2004): Apache Licence Version 2.0. Disponible en http://www.apache.org/licenses/LICENSE-2.0 [Consulta: 5 de mayo de 2018].
28. BlackDuck | Openhub (n.d.): Apache Spark . Disponible en https://www.openhub.net/p/apache-spark [Consulta: 5 de mayo de 2018].
29. Apache Spark (n.d.): Classification and regression. Disponible en https://spark.apache.org/docs/latest/ml-classification-regression.html [Consultado: 6 de mayo de 2018].
30. WIkipedia (27 de abril de 2018): Regresión lineal. Disponible en https://es.wikipedia.org/wiki/Regresi%C3%B3n_lineal [Consulta 9 de mayo de 2018].
31. Wikipedia (20 de marzo de 2018): Modelo lineal generalizado. Disponible en https://es.wikipedia.org/wiki/Modelo_lineal_generalizado [Consulta 9 de mayo de 2018].
32. UC3M (n.d.): Tema 3: Modelos lineales generalizado. Disponible en http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/Categor/Tema3Cate.pdf [Consulta: 9 de mayo de 2018].
33. Wikipedia (20 de abril de 2018) : Support vector machine. Disponible en https://en.wikipedia.org/wiki/Support_vector_machine [Consulta: 6 de mayo de 2018].

34. Twitter (7 de abril de 2018): Perfil de usuario de “elrubius”. Disponible en https://twitter.com/Rubiu5 [Consultado 7 de abril de 2018].
35. El Mundo (23 de agosto de 2016): Un limón lleva a ElRubius al “top ten” de los mayores retuits de la historia. Disponible en http://www.elmundo.es/f5/2016/08/23/57bc0085e2704e30138b4616.html [Consulta: 7 de mayo del 2018].
36. Rubio Hancock, Jaime (27 de febrero de 2015): ¿De qué color es este vestido?. Disponible en https://verne.elpais.com/verne/2015/02/27/articulo/1425025733_797891.html [Consulta: 7 de mayo de 2018].

Apéndice

Todo el código que se ha desarrollado para este trabajo se encuentra disponible en el repositorio público de GitHub "https://github.com/Manwelanza/Successful_Terms_py". Cualquiera puede acceder al repositorio para descargar u observar el código. Además, se ha añadido un directorio llamado "data" en el cual se ha añadido unos cuantos *tweets* de ejemplo.

Sin embargo, es importante señalar que se ha excluido un pequeño fichero de credenciales. En este fichero se almacenaba el usuario y contraseña necesario para acceder a la base de datos de *Neo4J*. También es importante destacar que se deberá iniciar una instancia de *MongoDB* y de *Neo4J* en local antes de poder usar los scripts de Python del repositorio.

Luego, si por el contrario se quiere ignorar el código, se ha almacenado en el directorio de "models" los modelos que se han generado con los distintos algoritmos de aprendizaje automático. Se podrán copiar y exportarlos a algún proyecto en el cual se esté empleando la biblioteca *SparkML*.