



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE MÁSTER

Mecanismos para la geolocalización de usuarios en Twitter.

Máster Universitario en Dirección TIC para la Defensa

ALUMNO: Benjamín Andrés Pintos

DIRECTORES: Milagros Fernández Gavilanes

CURSO ACADÉMICO: 2021-2022

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE MÁSTER

Mecanismos para la geolocalización de usuarios en Twitter.

Máster Universitario en Dirección TIC para la Defensa
Especialidad de Sistemas y Tecnologías de la Información

Universida_{de}Vigo

RESUMEN

En este documento se tratan los dos métodos principales descritos en la literatura académica sobre la geolocalización en Twitter: el basado en el análisis de texto y el de análisis de las redes de usuarios. Se profundiza en las técnicas que emplea cada uno de ellos, especialmente aquellas relacionadas con el procesamiento del lenguaje natural, algoritmos de Machine-Learning y las relacionadas con el análisis de grafos.

Posteriormente, se implementan en código Python los mecanismos necesarios para llevar a cabo todo el proceso de geolocalización siguiendo algunos de los métodos descritos. Para lo cual, se obtienen diversos conjuntos de tweets utilizando la API de Twitter y se almacenan en una base de datos MongoDB. Seguidamente se lee la información relevante y se inicializan las estructuras de datos que, tras su etapa de preprocesado, se utilizarán en los algoritmos de Machine-Learning o análisis de grafos y que permitirán la clasificación de los mismos como pertenecientes a una localización u otra.

Aunque el código presentado no pretende competir en eficacia y exactitud con los métodos descritos en la literatura académica, sí que nos permite obtener una visión completa de su funcionamiento, permitiendo descender a los detalles de implementación como las librerías que se precisan, las estructuras de datos, los parámetros que determinan el comportamiento de los algoritmos de clasificación, las herramientas de visualización y presentación de resultados, etc. Obtendremos gracias a ello las conclusiones de índole práctica relacionadas con los distintos mecanismos de geolocalización y que permitan seleccionar el más adecuado en función de la utilidad final para la que se emplee tal localización.

PALABRAS CLAVE

Twitter, Geolocalización, Machine-Learning, Procesamiento de lenguaje natural, Análisis de grafos.

AGRADECIMIENTOS

A mi familia, por soportar mis ausencias y mi atención dispersa en las presencias.

CONTENIDO

Contenido	1
Índice de Figuras	3
Índice de Tablas.....	5
1 Introducción y objetivos	6
1.1 Introducción	6
1.2 Motivación y objetivos.....	7
1.3 Organización de la memoria	8
2 Estado del arte	9
2.1 Twitter.....	9
2.1.1 Conceptos fundamentales de Twitter.....	9
2.1.2 Funcionalidades relacionadas con la geolocalización.	10
2.2 El problema de la geolocalización.	12
2.2.1 Tipos de localización	13
2.2.2 Información disponible para la localización.....	13
2.2.3 Influencia de la información temporal en la localización.....	14
2.2.4 Expresión formal del problema	15
2.3 El proceso de la localización.....	15
2.3.1 Adquisición de datos.....	15
2.3.2 Preprocesamiento.....	18
2.3.3 Representación de la localización.....	20
2.3.4 Diseño del algoritmo de inferencia.....	23
2.3.5 Evaluando la eficacia de la localización	23
2.4 Principales métodos de geolocalización	25
2.4.1 Métodos basados en análisis de texto	25
2.4.2 Métodos basados en el análisis de redes de usuarios.....	43
2.4.3 Métodos híbridos	52
2.4.4 Comparación de la eficacia de los diversos métodos	55
3 Desarrollo del TFM.....	57
3.1 Metodología	57
3.1.1 Enfoque.....	57
3.1.2 Herramientas utilizadas	58
3.2 Obtención de los datos.	58
3.2.1 La API de Twitter	58
3.2.2 Métodos de recogida de datos mediante librerías que emplean la API de Twitter. 60	

3.2.3 Almacenando datos.....	61
3.2.4 Datos de referencia (“ground-truth data”)	63
3.2.5 Datos de test.....	63
3.3 Algoritmos basados en análisis de textos.....	64
3.3.1 Algoritmos basados en diccionarios toponímicos.	64
3.3.2 Algoritmos basados en Machine-Learning.	66
3.4 Algoritmos basados en análisis de redes de usuarios.....	80
3.4.1 Construcción de las redes de usuarios	80
3.4.2 Algoritmos basados en la propagación de etiquetas	84
4 Resultados	89
4.1 Sobre la obtención de datos.	89
4.2 Sobre la aplicación de algoritmos basados en análisis de textos.	90
4.2.1 Método basado en diccionarios toponímicos.....	90
4.2.2 Método basado en análisis del texto de los tweets.	90
4.3 Sobre la aplicación de algoritmos basados en análisis de redes de usuarios.	91
5 Conclusiones y líneas futuras	92
5.1 Conclusiones	92
5.2 Líneas futuras	94
6 Bibliografía.....	95
Anexo I: Código obtención datos.....	99
Anexo II: Código de localización mediante Diccionario Toponímico.....	101
Anexo III: representación localizaciones	103
Anexo IV: Código localización mediante algoritmos de Machine-Learning.....	104
Anexo V: Código localización mediante algoritmos de análisis de redes de usuarios	114

ÍNDICE DE FIGURAS

Figura 2-1 Grafo generado por SocioViz para #tokyo2020 en lenguaje español entre el 23 y el 29 de julio de 2021.....	10
Figura 2-2 Estructura JSON de un Tweet geolocalizado en la ciudad de Madrid	11
Figura 2-3 Tipos de localización (procedente de [5])	13
Figura 2-4 Esquema general del proceso de geolocalización (basado en [10])	16
Figura 2-5 Detalle de la división en áreas de Norteamérica (procedente de [7]).....	21
Figura 2-6 Problema de la cobertura del círculo (procedente de [3]).....	22
Figura 2-7 Clasificación alternativa de algoritmos de inferencia (procedente de [4]).....	23
Figura 2-8 Ejemplo de CDF (procedente de [22]).....	25
Figura 2-9 Densidad por kilómetro cuadrado de las características geográficas de GeoNames (procedente de [43])	26
Figura 2-10 Esquema del método de análisis de texto que utiliza diccionario toponímico. ..	27
Figura 2-11 Etapas del ciclo de vida de un sistema de Machine-Learning.....	29
Figura 2-12 El problema del sobreajuste (procedente de wikipedia)	30
Figura 2-13 Clasificación multiclase DAG	31
Figura 2-14 Ejemplo de clasificación mediante regresión logística (obtenido mediante adaptación del código propuesto en [45]).....	32
Figura 2-15 SVM: Maximización del margen entre hiperplano y vectores de soporte.....	33
Figura 2-16 Comparación de clasificación SVM modificando los valores de C . (Figuras obtenidas mediante adaptación del código propuesto en [45])	34
Figura 2-17 Aumentando el número de características para lograr un clasificador lineal. (procedente de [46])	34
Figura 2-18 Comparación de la clasificación de muestras con diversos kernel. (Extraída de [49])	35
Figura 2-19 Ejemplo de clasificación mediante árbol de decisión (obtenido mediante adaptación del código propuesto en [45])	36
Figura 2-20 Ejemplo de clasificación mediante bosque aleatorio (obtenido mediante adaptación del código propuesto en [45])	37
Figura 2-21 Ejemplo de iteraciones de k-means (obtenido de https://stanford.edu/~cpiech/cs221/handouts/kmeans.html)	38
Figura 2-22 Determinación del número óptimo de clases (obtenida de https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/)	38
Figura 2-23 Ejemplo de aplicación de Naive Bayes.	40
Figura 2-24 Funcionamiento conceptual de LDA (obtenido de [47]).....	41
Figura 2-25 Ejemplo de grafo.	43
Figura 2-26 Representación de la información de un grafo no dirigido.	43

Figura 2-27 Probabilidad de amistad en función de la distancia (obtenido de [33]).....	45
Figura 2-28 Niveles iniciales del árbol de decisión que se utilizará para ponderar los contactos de un usuario en el grafo de relaciones (obtenido de [33]).....	46
Figura 2-29 Extracto de la matriz de probabilidades de encaminamiento de nodos.....	47
Figura 2-30 Ejemplo de grafo simple no dirigido en el que los nodos rojos y verdes están etiquetados.	47
Figura 2-31 Extracto de la matriz T elevada a 20.....	48
Figura 2-32 Ejemplo de aplicación del algoritmo de Propagación de Etiquetas en aprendizaje semi-supervisado.....	49
Figura 2-33 Representación de un grafo con dimensión reducida obtenido mediante <i>DeepWalk</i> . (Figura obtenida de [35]).....	50
Figura 2-34 Funcionamiento del Algoritmo <i>DeepWalk</i> (obtenida de [35]).....	51
Figura 2-35 Recorridos BFS/DFS y valores de α en los recorridos de <i>Node2Vec</i> (obtenido de [36]).....	52
Figura 2-36 Grafo de menciones con supresión de celebridades y adición de los nodos adjuntos (obtenida de [37])	53
Figura 2-37 Esquema de un modelo de aprendizaje conjunto (obtenido de [39]).....	55
Figura 2-38 Comparación de los métodos de localización (obtenida de [9]).....	56
Figura 3-1 Área de obtención de los tweets	60
Figura 3-2 Representación gráfica de las localizaciones ofrecidas por <i>GeoNames</i> de <i>tweets_spain</i>	66
Figura 3-3 Nube de palabras correspondiente a los tweets de Madrid.....	70
Figura 3-4 Nube de palabras correspondiente a los tweets de Barcelona	70
Figura 3-5 Palabras más utilizadas en los conjuntos de tweets	71
Figura 3-6 Influencia del hiperparámetro C en la exactitud.....	73
Figura 3-7 Representación gráfica de la exactitud en función de C y de Γ	78
Figura 3-8 Representación gráfica de la exactitud en función de $n_estimators$ y max_depth	79
Figura 3-9 Coeficientes de las palabras más significativas en la clasificación Madrid-México.	80
Figura 3-10 Representaciones gráficas del grafo extraído de <i>tweets_mad_ext</i>	83
Figura 3-11 Grafo del conjunto <i>tweets_mad_bar</i> obtenido con Gephi	84
Figura 3-12 Comunidades en <i>tweet_madrid_ext</i> obtenidas con propagación de etiquetas....	85
Figura 3-13 Propagación de etiquetas del 10% de nodos aleatorios en el grafo <i>tweet_madrid_ext</i>	86
Figura 3-14 Propagación de etiquetas a partir de cuatro nodos iniciales.	87
Figura 3-15 Nodos en cada clase tras la propagación de etiquetas	87

ÍNDICE DE TABLAS

Tabla 2-1 Ejemplos de tweets procedentes de FourSquare (obtenida de [3])	12
Tabla 3-1 Tweets recogidos en la colección <i>tweets_spain</i>	62
Tabla 3-2 Tweets recogidos en la colección <i>tweets_stream</i>	62
Tabla 3-3 Tweets recogidos en la colección <i>tweets_spain_ext</i>	63
Tabla 3-4 Resultados obtenidos en colección <i>tweets_spain</i>	64
Tabla 3-5 Ejemplos de localizaciones ofrecidas por <i>GeoNames</i>	65
Tabla 3-6 Conjuntos de entrenamiento y test para aplicación de algoritmos ML	67
Tabla 3-7 Palabras más utilizadas en los tweets de los usuarios de Madrid y Barcelona.	69
Tabla 3-8 Informe de clasificación con Regresión Logística en Madrid-Barcelona	74
Tabla 3-9 Matriz de confusión con Regresión Logística en Madrid-Barcelona.	74
Tabla 3-10 Informe de clasificación con Regresión Logística y balanceo de muestras	75
Tabla 3-11 Matriz de confusión con Regresión Logística y balanceo de muestras.	75
Tabla 3-12 Informe de clasificación con Regresión Logística en Madrid-México	75
Tabla 3-13 Matriz de confusión con Regresión Logística en Madrid-México.	76
Tabla 3-14 Informe de clasificación con SVM para Madrid-Barcelona.	76
Tabla 3-15 Matriz de confusión con SVM para Madrid-Barcelona	76
Tabla 3-16 Informe de clasificación con SVM para Madrid/México.	77
Tabla 3-17 Matriz de confusión para SVM para Madrid/México	77
Tabla 3-18 Informe de clasificación con Bosque Aleatorio para Madrid-Barcelona	78
Tabla 3-19 Matriz de confusión con Bosque Aleatorio para Madrid-Barcelona	79
Tabla 3-20 Informe de clasificación con Bosque Aleatorio para Madrid/México	79
Tabla 3-21 Matriz de confusión con Bosque Aleatorio para Madrid/México	79
Tabla 3-22 Características del grafo <i>tweet_madrid_ext</i> obtenidas con Gephi	82
Tabla 3-23 Resultados de propagación de etiquetas en <i>tweets_mad_bar</i>	88

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

Poco después de la aparición de Internet, las redes sociales irrumpieron en nuestras vidas extendiéndose con gran rapidez y llegando a alcanzar tal importancia, que las aplicaciones que las soportan se han convertido en los principales activos de algunas de las más grandes empresas del planeta. La nueva forma de comunicación que proveen ha supuesto un nuevo paradigma en las relaciones interpersonales y ha llegado a convertirse en motivo de estudio para disciplinas científicas como la sociología. No cabe duda, por tanto, de que las redes sociales son una fuente de información muy importante sobre los humanos, sus relaciones, sus preferencias, sus pensamientos e inquietudes, sus gustos, sus tendencias, etc. La explotación de dicha información con fines científicos, comerciales o incluso políticos, es motivo de gran interés para muy diversos actores. Las observaciones obtenidas de las redes sociales pueden tener un gran peso en la valoración del éxito o fracaso de una campaña de marketing¹ o incluso en la predicción de las fluctuaciones de la bolsa [1].

En un ranking a nivel global de las redes sociales más populares, Twitter aparece en el número dieciséis con 396 millones de usuarios activos en abril de 2021². Si descontamos algunas redes que, aunque tienen mayor número de usuarios, están muy localizadas en determinados países orientales, la relevancia de Twitter aumenta. Concretamente, en España Twitter ocupa el tercer lugar del ranking, superando el 49% de los usuarios en 2020³. Sin embargo, a efectos de obtención de información susceptible de análisis, Twitter es especialmente interesante puesto que es una plataforma abierta muy sencilla de monitorizar y que proporciona un enorme volumen de datos. Actualmente se generan más de 500 millones de tweets diarios⁴.

Uno de los elementos de información que resultan de especial utilidad es la ubicación de los individuos que utilizan la red social. Un análisis de la opinión que tienen los clientes de un producto, obtenido a partir de los tweets, nos dará información de qué está pasando, pero para

¹ Extraído de “Twitter Analytics” <https://business.twitter.com/en/advertising/analytics.html> [Disponible a 23/12/2021].

² Extraído de “Most popular social networks worldwide as of October 2021, ranked by number of active users” <https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/> [Disponible a 23/12/2021].

³ Extraído de “Redes sociales con mayor porcentaje de usuarios e España en 2020” <https://es.statista.com/estadisticas/489153/porcentaje-de-internautas-en-las-redes-sociales-en-espana/> [Disponible a 23/12/2021].

⁴ Extraído de “Twitter Usage Statistics” <https://www.internetlivestats.com/twitter-statistics/> [Disponible a 23/12/2021].

saber cómo actuar, a menudo necesitamos conocer la ubicación de esos usuarios. Este es el objeto de lo que se denomina geomarketing, que es una de las muchas aplicaciones que tiene la geolocalización de Twitter. Pero no es la única, algunos estudios utilizan la geolocalización para, por ejemplo, analizar las evacuaciones durante catástrofes [2] o para predecir la situación del tráfico¹.

Sin embargo, los usuarios son cada vez más conscientes de la importancia de esta información y son más reticentes a exponer su privacidad. Así pues, aunque Twitter permite que los usuarios geolocalicen sus tweets, el porcentaje de los que lo hacen es muy bajo. Hasta tal punto es así que la propia red social eliminó en junio de 2019 la posibilidad de geolocalizar con precisión la ubicación de los tweets, debido a que los usuarios no la utilizaban². Sin embargo, el interés en conocer la ubicación de los usuarios sigue existiendo, aunque sea más difícil de obtener. Este documento trata precisamente esa cuestión ¿qué mecanismos existen para obtener la localización de un usuario en Twitter?

Desde el punto de vista de la Defensa y la Seguridad, es fácil encontrar recomendaciones para que los usuarios desactiven las opciones que facilitan su localización a través del uso de las aplicaciones de redes sociales, de control de la actividad deportiva, etc.³. Obviamente, si algo es peligroso para nosotros en manos del enemigo, también se da la situación inversa. Sin embargo, no hay que olvidar que Twitter, los operadores de telefonía o servicios de internet tienen mucha más información sobre la ubicación de los usuarios que la que podemos obtener mediante cualquier análisis de la actividad pública de los mismos. Por ello, si se justifica, probablemente ante autoridades judiciales y por razones de Defensa y Seguridad, la geolocalización de los usuarios no precisa la utilización de los métodos que se van a tratar aquí. Debería ser posible instar a los citados proveedores a proporcionar la información completa de los usuarios. Así pues, hay que precisar que en este documento únicamente tratamos métodos que hacen uso de la información pública de los usuarios: sus perfiles, o sus tweets.

1.2 Motivación y objetivos

El objeto de este documento es presentar los distintos mecanismos que se han descrito en la literatura académica, explicando las metodologías más significativas, facilitando al lector interesado una visión genérica del estado del arte y, al mismo tiempo, una visión detallada del funcionamiento de algunos de los algoritmos más utilizados. Este documento aspira, por tanto, a constituir una herramienta útil para todos aquellos que quieran, no sólo conocer los mecanismos de geolocalización de forma conceptual, sino implementarlos, mediante código de programación. Al hablar de mecanismos, lo interpretamos en el sentido más amplio posible. Se trata tanto de analizar los métodos actuales desde un punto de vista genérico, como de abordarlos desde un punto de vista práctico que nos permita comprender en su totalidad el proceso: desde la obtención de los tweets, su almacenamiento, la creación de las estructuras de datos en código de programación, la utilización de las distintas librerías que nos facilitarán el preprocesamiento y posterior tratamiento de los datos, la representación gráfica de los resultados, hasta la evaluación de la exactitud en la geolocalización.

No es objetivo del trabajo la obtención de algoritmos eficaces de localización que compitan en exactitud y eficiencia con los existentes en la literatura académica. Lo que se pretende es

¹ Extraído de “From Twitter to Traffic Predictor” <https://www.cmu.edu/news/stories/archives/2021/may/traffic-prediction.html> [Disponible a 23/12/2021]

² “Twitter will remove precise location tagging in tweets, citing lack of use” <https://techcrunch.com/2019/06/18/twitter-will-remove-location-tagging-in-tweets-citing-lack-of-use/> [Disponible a 23/12/2021]

³ Un ejemplo lo podemos encontrar en “Use of Geo Location-Capable, devices, applications and services” <https://www.marines.mil/News/Messages/Messages-Display/Article/2472175/use-of-geo-location-capable-devices-applications-and-services/> [Disponible a 23/12/2021]

mostrar los distintos mecanismos, procedimientos y herramientas de código que pueden ayudar a llevar a cabo dicha tarea.

Por otro lado, además de la propia utilidad de la geolocalización de los tweets, los mecanismos utilizados nos aproximan a algunas de las áreas de la informática que hoy en día están de actualidad, como el Big Data o el Machine-Learning.

1.3 Organización de la memoria

En el capítulo 1, realizamos una breve introducción resaltando la importancia de Twitter como fuente de información y mencionando las posibles utilidades de la geolocalización de los usuarios. También exponemos los motivos y objetivos del presente trabajo, así como su organización.

En el capítulo 2 “Estado del arte”, se revisará la literatura académica relacionada, se planteará el problema de la geolocalización, se describirán las distintas fases que tiene su resolución y se clasificarán los distintos métodos existentes. Además, se realizarán algunos apuntes sobre la eficacia de cada uno de ellos.

En el capítulo 3 “Desarrollo del TFM” se ejemplificará el proceso de resolución de la geolocalización, obteniendo conjuntos de tweets aleatorios y aplicando sobre ellos los algoritmos mencionados en el apartado anterior. El objeto de este apartado es la implementación del código necesario para efectuar el proceso completo de geolocalización, introduciendo las herramientas necesarias para poder abordar el problema.

En el capítulo 4 “Resultados” analizaremos los datos obtenidos al aplicar los algoritmos del apartado anterior sobre los conjuntos de datos, tratando de comparar los diferentes mecanismos empleados en la geolocalización.

En el capítulo 5 “Conclusiones” se realiza la síntesis del trabajo y se incluyen las posibles líneas de actuación de futuros trabajos.

Finalmente, los anexos de este documento incluyen el código que implementa los distintos algoritmos utilizados y que puede servir de punto de partida para modelos más perfeccionados.

2 ESTADO DEL ARTE

2.1 Twitter.

2.1.1 Conceptos fundamentales de Twitter.

Twitter fue creado por Jack Dorsey en marzo de 2006. Actualmente es una red social que utilizan casi 400 millones de personas de todo el mundo. Los usuarios de Twitter escriben y publican tuits que pueden ser leídos por el resto de usuarios. Tuit es la palabra en español para denominar a lo que, en el mundo anglosajón se denomina “tweet” (podría traducirse como piada o gorjeo). Se trata, por tanto, de una palabra derivada de la transcripción fonética de la palabra en inglés. Aunque la palabra tuit está reconocida por la R.A.E, en este documento preferimos utilizar el término tweet.

Los tweets inicialmente estuvieron limitados a 140 caracteres, pero a partir de 2017 se amplió a 280. Este tamaño reducido obliga a los usuarios a condensar la información en una especie de titular. Aun así, los usuarios pueden incluir imágenes, vídeos o hipervínculos junto con el texto.

En Twitter se denomina seguidor (“*follower*”) a aquellos usuarios que están suscritos a la cuenta de otro determinado y reciben notificación de sus publicaciones. Se denomina amigos (“*friends*” o “*followees*”) a aquellos a los se sigue. Ambos conceptos son similares, pero se diferencian en el punto de vista considerado.

Cada usuario puede mostrar su interés en seguir las publicaciones de uno o más usuarios. Los tweets que se publican en el perfil del usuario se envían a sus seguidores. La intención es que los grupos de usuarios que se siguen entre sí compartan mensajes rápidos y frecuentes. También los utilizan personalidades o instituciones públicas y privadas para divulgar noticias u opiniones. Así, por ejemplo, el expresidente de los Estados Unidos Barack Obama tiene la cuenta de Twitter con más seguidores, casi 130 millones, y la “CNN Breaking News” es la agencia de noticias más seguida con algo más de 55 millones. Es decir, los grafos de las redes de seguidores pueden ser bastante grandes y, además, las relaciones entre usuarios no siempre son recíprocas. Es común encontrar usuarios con muchos seguidores (“*celebrities*”) que, o bien no siguen las publicaciones de sus seguidores, o lo hacen en mucha menor proporción.

Los usuarios pueden valorar un tweet marcándolo como de su gusto (“*me gusta*” / “*like*”), o compartirlo con sus seguidores (“*retweet*”), pudiendo añadir un comentario opcional. También es posible responder a un tweet, mencionar a otro usuario (@*usuario*) o compartirlo en otra red social. Incluso existe la posibilidad de enviar tweets directos a otro usuario, o a un grupo de hasta cincuenta usuarios. Todas estas funcionalidades contribuyen a crear conversaciones entre

usuarios, que no son más que tweets relacionados entre sí. Probablemente la forma más habitual para lograr esta interrelación es creando una etiqueta resumen (“*hashtag*”), que tiene la forma “*#etiqueta*”. De ese modo, todos los tweets que tengan dicha etiqueta se vincularán entre sí y resultará más sencillo seguir el hilo de las publicaciones de los usuarios que la utilicen. Si una etiqueta aparece en muchos tweets puede llegar a incluirse en las listas con los temas del momento (“*trending topics*”). Más tarde, otros usuarios podrán realizar búsquedas en dichas listas, para averiguar de qué se está hablando y participar en la conversación.

Las conversaciones sobre un determinado tema se pueden mostrar en grafos en los que los vértices son los usuarios y las aristas representan los retweets o las menciones. La Figura 2-1, sirve para ejemplificar la utilidad de estos grafos. Ha sido generado por una aplicación web denominada SocioViz¹, y en ella se muestran los usuarios relacionados con un hashtag concreto. Los vértices con más aristas dirigidas hacia sí mismos corresponden a los usuarios más citados o con más retweets, es decir, podrían representar a los usuarios más influyentes. El grosor de los grafos indica el número de menciones o retweets que hace un usuario de otro. El diámetro de los vértices, nos indica qué usuarios han publicado más tweets sobre dicho hashtag, es decir, los más activos.

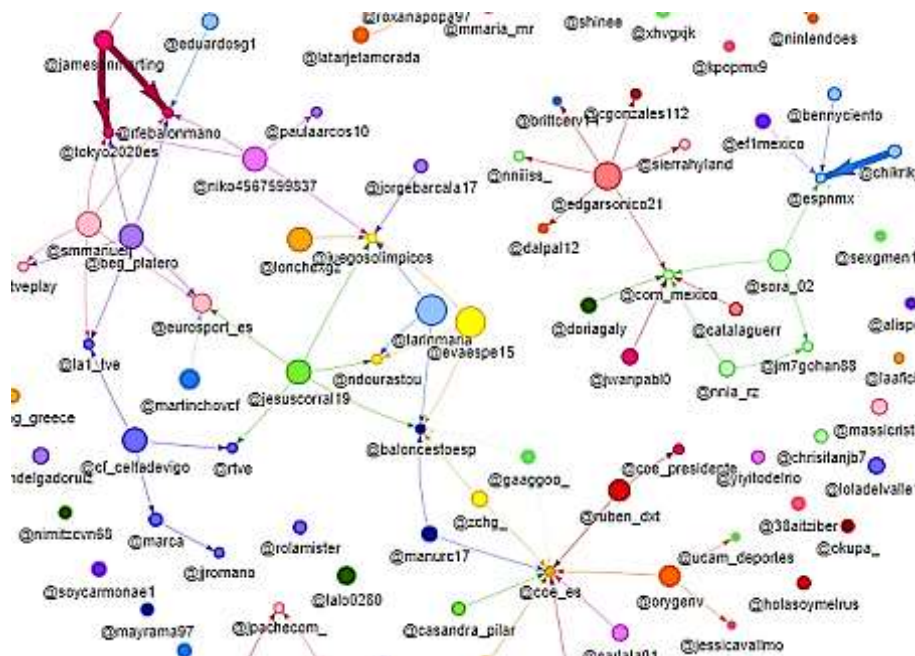


Figura 2-1 Grafo generado por SocioViz para #tokyo2020 en lenguaje español entre el 23 y el 29 de julio de 2021.

Un grafo similar al anterior se puede obtener a partir de las relaciones genéricas de seguimiento y amistad (“*followers/followees*”) entre los usuarios, sin ceñirse a un tema concreto.

Por último, reseñar que los usuarios pueden mostrar un perfil público en el que se incluya una imagen, un encabezado, una descripción personal. La información mostrada a los demás usuarios incluirá, además de lo indicado, la fecha en la que se unió a Twitter, el número de usuarios a los que siguen y cuantos les siguen a ellos.

2.1.2 Funcionalidades relacionadas con la geolocalización.

En Twitter, es posible modificar la configuración para que el perfil de usuario incluya un determinado texto en el campo “localización”. Aquí el usuario puede escribir libremente lo que considere oportuno, desde el lugar donde habitualmente reside, con diferente granularidad, hasta

¹ Obtenido mediante la API disponible en <https://socioviz.net/>

expresiones con poco o ningún sentido. Es posible encontrar desde la dirección postal completa, hasta frases del tipo “Planeta tierra”, pasando por indicaciones de la ciudad, o únicamente el país. También hay usuarios que aportan información del tipo “En la red” o similar. Así pues, además de que no todos los usuarios incluyen información en dicho campo, los que sí la incluyen lo hacen de forma poco significativa. En cualquier caso, el perfil del usuario puede contener en algunos de los campos de texto, información de la cual extraer una posible localización.

Actualmente, cuando nos damos de alta como usuarios, Twitter rellena automáticamente un campo de nuestro perfil denominado “país”. Esta información la utiliza para “personalizar nuestra experiencia” y la extrae de diversas fuentes como la dirección IP, la del GPS de nuestro dispositivo u otra información relacionada con las redes inalámbricas a las que estuviera conectado el teléfono móvil¹. El usuario, posteriormente, si tiene especial interés, podrá modificar ese dato. Lo relevante es que esa información no está disponible a través de la API de Twitter, sino que se considera información privada.

```

"place": {
  "id": "206c436ce43a43a3",
  "url": "https://api.twitter.com/1.1/geo/id/206c436ce43a43a3.json",
  "place_type": "city",
  "name": "Madrid",
  "full_name": "Madrid, Espa\u00f1a",
  "country_code": "ES",
  "country": "Espa\u00f1a",
  "contained_within": [],
  "bounding_box": {
    "type": "Polygon",
    "coordinates": [
      [
        [-3.8890049, 40.3120713],
        [-3.5180102, 40.3120713],
        [-3.5180102, 40.6435181],
        [-3.8890049, 40.6435181]
      ]
    ]
  },
  "attributes": {}
},

```



Figura 2-2 Estructura JSON de un Tweet geolocalizado en la ciudad de Madrid

Los tweets de los usuarios pueden contener información sobre la localización desde la cual el usuario genera los tweets. Anteriormente, los tweets podían incluir la ubicación precisa que consistía en una latitud y longitud desde la que se publicaba el tweet y que era extraída del dispositivo GPS que incorporaban muchos teléfonos móviles. Sin embargo, dado el escaso uso de dicha utilidad, Twitter dejó de proporcionarla en junio de 2019². Aun así, los usuarios todavía pueden incluir su localización al publicar un tweet. El modo de hacerlo es: primero, activando la funcionalidad en la configuración de la propia aplicación de Twitter y, posteriormente al redactar

¹Extraído de “Cómo cambiar la configuración de país” <https://help.twitter.com/es/managing-your-account/how-to-change-country-settings> [Disponible a 23/12/2021]

² “Twitter will remove precise location tagging in tweets, citing lack of use” <https://techerunch.com/2019/06/18/twitter-will-remove-location-tagging-in-tweets-citing-lack-of-use/> [Disponible a 23/12/2021]

el tweet, seleccionando la ciudad, el vecindario, o únicamente el país, con la granularidad que desee, de una lista de posibilidades sugerida, pero sin indicar las coordenadas geográficas precisas. La lista de lugares sugeridos procede de la base de datos generada por la aplicación FourSquare¹. Esta aplicación permite a los usuarios crear ubicaciones, opinar sobre ellas y compartirla con sus amigos. De este modo consigue trabajar con una base de datos de ubicaciones muy amplia, tanto que proporciona servicios de localización, no solo a Twitter, sino también a Snapchat², WeChat³ o Tinder⁴.

Los tweets asociados a una localización contienen metadatos de los cuatro puntos que delimitan una caja en la que se encuentra dicha localización. Esta información es accesible a través de la API de Twitter. A modo de ejemplo, en la Figura 2-2 se muestra la estructura de datos, con formato JSON, que se ha obtenido de un tweet geolocalizado en Madrid. La representación de la caja en el mapa se puede ver a la derecha. Las coordenadas son las ofrecidas por FourSquare. Parece que la caja resultante recoge un área excesiva al norte de Madrid.

Como se ha mencionado, los usuarios no son muy propensos a geolocalizar los tweets que publican, por lo que no es tan habitual encontrar este tipo de información. Sin embargo, aplicaciones como FourSquare se pueden configurar para que los usuarios que hacen *check-in*, publiquen un tweet de forma automática. Los usuarios de FourSquare son incentivados mediante la concesión de puntos cada vez que hacen *check-in*, para conseguir unas distinciones o medallas virtuales.

Tweet id	Texto
t ₁	I'm at Mac @ Bukit Panjang Plaza
t ₂	I'm at ITE College Central MacPherson Campus Main (201 Circuit Road)
t ₃	Birthday dinner (@ Ambush @ JP w/2 others)
t ₄	Watching "Hello Stranger" (@ Golden Village Cinema 9 @ Plaza Singapura)

Tabla 2-1 Ejemplos de tweets procedentes de FourSquare (obtenida de [3])

En la Tabla 2-1 se observan ejemplos de tweets publicados a partir de *check-ins* en FourSquare. Algunos de ellos, simplemente informan de la localización de los usuarios, mientras que otros incluyen información de las actividades que realiza el usuario. En algunos casos la localización aparece abreviada como *JP* que se corresponde con un centro comercial de Singapur.

2.2 El problema de la geolocalización.

En la literatura académica, escrita fundamentalmente en inglés, podemos encontrar tres términos relacionados con la geolocalización. La distinción en el significado de uno y otro, la podemos encontrar en [4]. El primero de los términos es "*geocoding*", que podemos traducir por geo-codificación. Se trata de transformar un texto con un formato bien definido de una localización en una representación espacial válida, como pueden ser unas coordenadas geográficas o una referencia precisa en un mapa. El segundo es "*geoparsing*" o geo-análisis, que tiene el mismo objeto que el anterior, pero parte de un texto libre sin formato ni estructura concreta, como podrían ser los tweets. Por último, "*geotagging*", o geo-etiquetado, es el que trata

¹ "Foursquare Places on Twitter" <https://foursquare.atlassian.net/servicedesk/customer/portal/20/article/935002579> [Disponible a 23/12/2021]

² <https://www.snapchat.com/> [Disponible a 14/01/2022]

³ <https://www.wechat.com/> [Disponible a 14/01/2022]

⁴ <https://tinder.com/> [Disponible a 14/01/2022]

de asignar la representación espacial utilizando modelos estadísticos y métodos de Machine-Learning.

En este documento, utilizaremos el término genérico de geolocalización de forma indistinta para referirnos a cualquiera de ellos.

2.2.1 Tipos de localización

Cuando se habla de geolocalización en Twitter es necesario precisar a qué nos estamos refiriendo. Según [5] podemos distinguir entre tres localizaciones de interés, tal y como se observa en la Figura 2-3. Un primer tipo de localización es la mencionada por el usuario en un tweet. Se trata de un lugar, con una granularidad concreta, al que el usuario se ha referido al publicar un tweet. Puede ser un lugar próximo al que vive o puede ser un lugar al que le gustaría ir de vacaciones, o que ha visitado un familiar. No se puede asegurar que siempre tenga relación directa con la ubicación del usuario.

Un segundo tipo es la localización del lugar donde reside el usuario. De nuevo, podemos entenderlo con distinta granularidad y hablar de la calle concreta, de la ciudad, etc. Representa la ubicación del usuario a largo plazo. Sin embargo, este dato no es inmutable, algunas personas cambian de domicilio con cierta frecuencia.

El tercer tipo es la localización de la publicación. Se trata del lugar desde el que se publicó el tweet. Habitualmente estará relativamente cerca del lugar donde vive el usuario, aunque no siempre tiene que ser así.

Entre los tipos de localizaciones mencionados no hay uno más preciso o más correcto que los demás en términos generales. El que sea más relevante uno u otro tipo dependerá del propósito de la geolocalización. No es lo mismo tratar de averiguar la situación del tráfico, o de un accidente, que un análisis de los clientes de un comercio. En el primer caso necesitamos conocer el lugar desde el que se realizó la publicación. En el segundo más bien estamos interesados en la zona donde viven. Aun así, uno de los más utilizados es el segundo, habitualmente denominado “*home location*”.

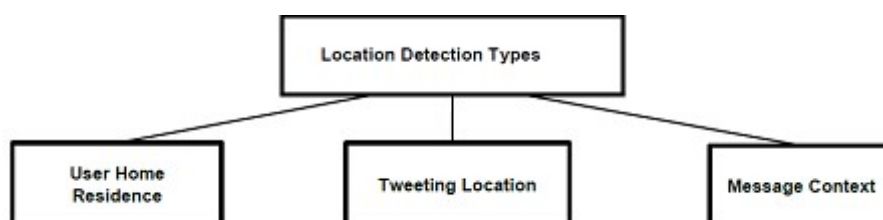


Figura 2-3 Tipos de localización (procedente de [5])

2.2.2 Información disponible para la localización

Las fuentes más habituales de las que se puede inferir la localización de un usuario son, por un lado, la información que este ha publicado en su perfil. Por otro lado, el contenido del conjunto de mensajes que ha publicado el usuario o, para algunas aplicaciones, algún mensaje concreto de dicho conjunto.

Sin embargo, hay otra información que puede ser útil, sino para ubicar al usuario, para aumentar la precisión obtenida a partir de otros elementos. Las personas que siguen (“*followers*”), o aquellas a las que sigue (“*followees*”) un usuario, pueden ayudar a localizarlo. En muchos casos estas relaciones se establecen entre personas que se conocen, porque se ven con cierta frecuencia, por lo que podemos suponer que viven cerca. Este argumento es uno de los más explotados para resolver el problema y da lugar a toda una metodología, como veremos más

adelante. Hay que reseñar que, para que sea de aplicación, se precisa conocer, al menos, la localización de alguno de los nodos del grafo que representa esta red, a partir de la cual, poder inferir la del resto de nodos.

2.2.3 Influencia de la información temporal en la localización

La información temporal también puede ser relevante y, puesto que no la vamos a tratar con mayor detalle en posteriores apartados, seguidamente nos referiremos brevemente a algunos de los métodos que la utilizan.

En una secuencia de tweets, si conseguimos geolocalizar uno de ellos y si la diferencia de tiempo con el siguiente no es excesiva, podemos localizar el segundo con más facilidad. Este enfoque es el que se utiliza en [6] para geolocalizar los tweets dentro de la ciudad de Singapur y Jakarta con suficiente precisión como para asociarlos a alguno de los lugares (“venues”) identificados en FourSquare. Los autores se basan en que, eligiendo un millar de tweets al azar publicados en dichas ciudades, en torno al 50% de los usuarios publicaban un segundo tweet separado temporalmente un máximo de treinta minutos, y el 95% se espaciaban menos de 10 kilómetros. Tratando de aprovechar esta información implementan un método que, aseguran, mejora la eficacia de otros mecanismos entre un 4% y un 40%.

Otro estudio relacionado con la información temporal es [7], en el que se analiza, por ejemplo, la precisión de la localización de los tweets emitidos a distintas horas del día, mostrando que sobre las cinco de la mañana se logra la máxima precisión y que, a partir de las siete de la tarde, cae rápidamente.

En el mismo estudio se analiza la durabilidad de los modelos utilizados para predecir las localizaciones. Puesto que todos utilizan un conjunto de datos de entrenamiento para configurar los modelos de predicción, la representatividad de dicho modelo puede verse afectada con el transcurso del tiempo. En las gráficas presentadas por los autores, se percibe una pérdida de precisión de casi veinte puntos porcentuales en un periodo de tres meses. Curiosamente, también se observa un aumento en la precisión a partir de agosto de 2015 que los autores no pueden explicar, llegando a conjeturar que podría deberse a efectos estacionales que sugerirían que la precisión dependería, de algún modo, de los meses del año en que se publiquen los tweets (¿quizás en los meses de verano del hemisferio norte, donde se obtuvieron las muestras, es más difícil localizar un tweet porque hay más personas de vacaciones publicando fuera de su lugar de residencia?). La variación estacional de la precisión en la geolocalización podría ser motivo de un estudio más detallado.

Por otro lado, en el perfil del usuario también se puede guardar información sobre el huso horario que utiliza (“*utc_offset*” y “*time_zone*”) lo cual constituye información importante para localizarlo, al menos, dentro del grupo de países que tienen el mismo huso. Bien es cierto que este campo de información Twitter lo considera obsoleto (“*deprecated*”)¹ y, por tanto, en la actualidad no es habitual que los usuarios lo actualicen, puesto que la aplicación no lo permite directamente. Así, por ejemplo, este campo estaba vacío (“*null*”) en todos los casos de los más de treinta mil tweets recogidos para este trabajo en julio de 2021.

Aunque los tweets publicados en todo el mundo tengan el sello de tiempo (“*timestamp*”) referido al horario UTC, todavía es posible extraer información relevante. En [8] se utiliza el timestamp como una de las características que utilizará una red neuronal para clasificar a los usuarios como pertenecientes a una determinada región. El argumento subyacente es que es poco probable que un usuario publique sus tweets de madrugada, por lo que es lógico suponer una cierta desviación en la longitud en la que se ubica el usuario.

¹“Data dictionary: Standard v1.1” <https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/user> [Disponible a 23/12/2021]

Una aproximación similar se utiliza en [9], en la que para determinar el lugar común de residencia de los usuarios, se otorga más peso a los tweets publicados a partir de determinada hora del día, en los que la intuición nos dice que es más probable que el usuario se encuentre en su domicilio.

2.2.4 Expresión formal del problema

En gran parte de la documentación académica se utiliza una nomenclatura para definir formalmente el problema de la geolocalización. Inspirándonos en [10], [11] y [12], trataremos de hacerlo aquí. Supongamos que denotamos por I el conjunto de la información que es posible obtener de la API de Twitter para geolocalizar la ubicación habitual de un usuario (“*home location*”). Podemos decir que I está compuesta de los siguientes datos ($I = P \cup F \cup T \cup M$):

- P : Perfil del usuario.
- F : Red de relaciones seguidor/seguído (“*follower/followee*”).
- T : Información temporal, u otras procedentes de los metadatos del tweet, como el idioma.
- M : Conjunto de los contenidos publicados por los usuarios en Twitter.

A los usuarios U , los dividiremos entre aquellos que tenemos previamente geolocalizados U^T , y que constituyen la referencia (“*ground-truth*”) que se empleará posteriormente tanto para entrenar el modelo de predicción, como para testarlo, junto con aquellos usuarios de los que hemos podido inferir su localización U^I y los que no U^U . De modo que $U = U^T \cup U^I \cup U^U$.

Del mismo modo, al conjunto de localizaciones de los usuarios lo llamaremos L , algunas de las cuales son reales L^T , asociadas a U^T , tal que para cada usuario $U_i \in U^T$ existe $L_j \in L^T$ que es la localización de: $U_i: L_j = Loc(U_i)$. Otras serán las inferidas L^I , asociadas a U^I de un modo similar, y aquellas que no conocemos L^U . Así pues $L = L^T \cup L^I \cup L^U$. Al conjunto de las localizaciones conocidas, podemos llamarlo $L^K = L^T \cup L^I$.

El problema de la geolocalización se puede enunciar como la búsqueda de la función:

$$f: (U, I, L^K) \rightarrow L^U$$

Se trata pues de encontrar localizaciones desconocidas, partiendo del conjunto de usuarios, de la información disponible y de ciertas localizaciones conocidas. Lo significativo, es que la localización no depende únicamente del usuario U_i y que se precisa un conjunto de localizaciones conocidas previamente L^K .

Además, la función $Loc(U_i)$, debe tratar de maximizar la precisión y, además poder aplicarse al mayor número de usuarios posible para minimizar $|U^U|$.

2.3 El proceso de la localización

La Figura 2-4 muestra un esquema general del proceso de localización, que detallaremos a lo largo de este apartado.

2.3.1 Adquisición de datos

El proceso de adquisición de datos se apoya en la utilización de la API de Twitter. A través de ella, es posible obtener los tweets que ha publicado un usuario, su perfil, los usuarios a los que sigue (“*followees*”, o “*friends*” en la terminología de la API), los usuarios que le siguen (“*followers*”), etc.

La API proporciona dos estructuras de datos tipo diccionario de las cuales obtener la información: el “*tweet*”¹ y el “*user*”². Cada una de ellas tiene diversos atributos relacionados que contienen la información citada. Hay que reseñar, sin embargo, que algunos de ellos han sido señalados por Twitter como obsoletos, por lo que no deberían utilizarse como fuente de información. En apartados posteriores se hablará sobre estas estructuras de datos y la forma de acceder a ellas.

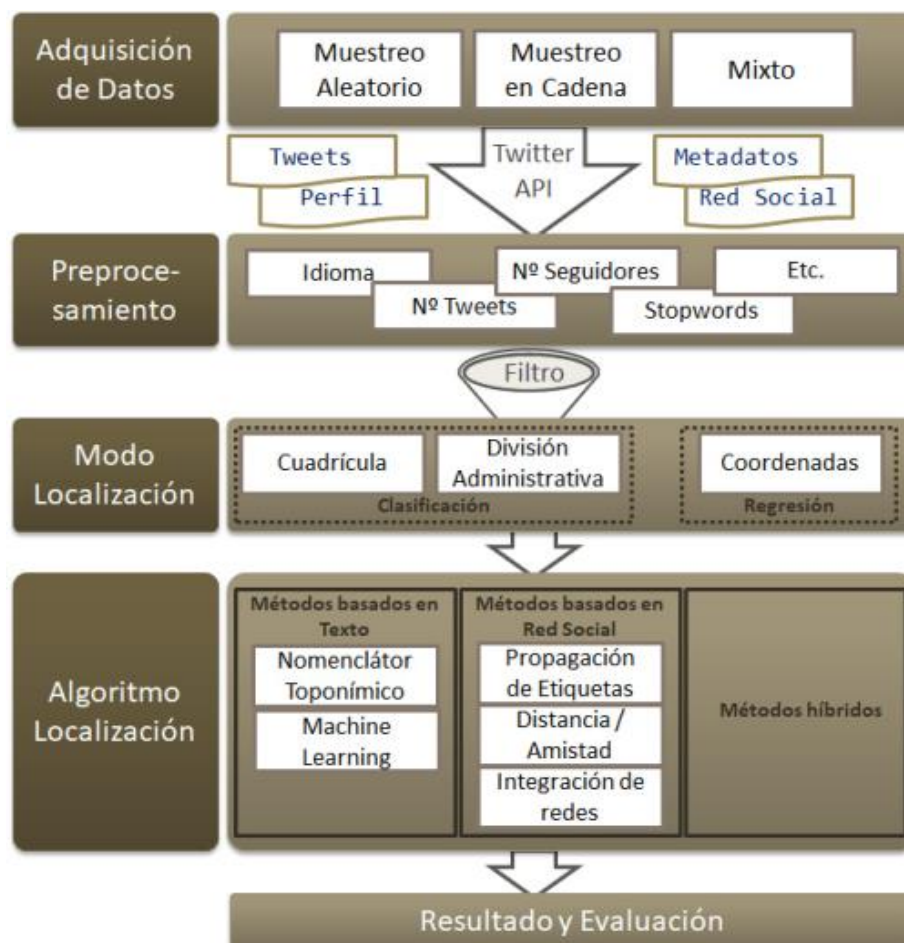


Figura 2-4 Esquema general del proceso de geolocalización (basado en [10])

Para poder desarrollar la mayoría de los algoritmos de localización se precisa un conjunto de datos sobre los que ya se conozca el resultado a obtener. Este conjunto de datos es el denominado de referencia (“*ground-truth*” o “*gold standard*”). Ese conjunto se subdivide, de nuevo, en otros dos: un subconjunto de entrenamiento, a partir del cual se desarrolla el algoritmo de localización, y un subconjunto de test, a partir del cual se comprueba su desempeño. Resulta pues primordial obtener ese conjunto de datos previamente georreferenciados.

En [13] se menciona que el porcentaje de tweets geolocalizados es, según estudios previos, del 0.85%. Sin embargo, en la muestra que realizan los autores sobre 30 millones de tweets obtienen que dicha cifra asciende al 3.1%. En dicho documento se ofrecen datos sobre la edad, sexo, nacionalidad, y otros datos, de los usuarios que geolocalizan sus tweets y de aquellos que tienen los servicios de localización activados, aunque después no los utilicen. En cualquier caso,

¹“Data dictionary: Standard v1.1” <https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/tweet> [Disponible a 23/12/2021]

²“Data dictionary: Standard v1.1” <https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/user> [Disponible a 23/12/2021]

lo importante es que el porcentaje no es muy alto y será necesario obtener gran cantidad de tweets para que el conjunto de referencia sea significativo.

Una referencia a las técnicas de muestreo necesarias para obtener el conjunto de datos se puede encontrar en [14]. La primera de las técnicas que mencionan es el muestreo por tema (“*topic-based sampling*”) en el que se obtienen tweets relacionados con una etiqueta (“*hashtag*”) o tema concreto. La API de Twitter nos permite este tipo de búsquedas y nos ofrecerá un conjunto de tweets relacionados con la etiqueta solicitada. El problema de este método es que la elección de la etiqueta dificulta la deseada aleatoriedad de la muestra. Esto nos lleva al método del muestreo aleatorio (“*random sampling*”) en el que se obtiene una muestra de los tweets que se están publicando en este momento. La API de Twitter nos ofrece la posibilidad de muestrear el flujo (“*stream*”) de los tweets, pudiendo acceder con la modalidad gratuita a una porción del 1%¹ de los casi 500 millones de tweets diarios. El problema de este método de muestreo es que únicamente se obtienen datos de los usuarios activos. Aquellos que no publiquen o que lo hagan poco frecuentemente tendrán pocas probabilidades de aparecer en el flujo de tweets.

Otra de las técnicas mencionadas en dicho documento es la denominada como muestreo en cadena en el esquema y que en la bibliografía en inglés se denomina “*snowball sampling*”. Esta técnica de muestreo parte de una lista de usuarios definida, pero reducida, que se va extendiendo a partir de relaciones entre los usuarios como las de las redes de amistad (“*followers / followees*”). Con este método se pretende evitar el inconveniente del muestreo de los usuarios poco activos, puesto que no es preciso que publiquen nada, simplemente que sigan a alguno de los usuarios de la lista inicial. En esta lista, típicamente se incluyen personalidades o instituciones. Sin embargo, de nuevo, la aleatoriedad no está garantizada.

Para evitar los inconvenientes específicos del muestreo directo del flujo de datos y del muestreo en cadena, se pueden realizar combinaciones de ambos. Este tipo de métodos aparece con el nombre de “Mixto” en la Figura 2-4.

Algunos de los académicos relacionados con la materia han permitido el acceso a través de internet a los tweets utilizados en sus investigaciones. En algunos casos, dichos conjuntos han sido utilizados por otros, e incluso se les ha puesto un nombre para identificarlos. Algunos de estos conjuntos son los siguientes:

- GEOTEXT²: El conjunto de datos está descrito en [15]. consta de 377.616 tweets procedentes de 9475 usuarios geolocalizados obtenidos en marzo de 2010.
- UTGEO2011³: Descrito en [16]. El conjunto ya filtrado consta de 38 millones de tweets de 449.694 usuarios recogidos entre septiembre y noviembre de 2011. Este conjunto se ha denominado de diferentes formas en documentos académicos posteriores como, por ejemplo, NA o TWITTER-US.
- TWITTER -WORLD⁴: Descrito en [17], con 12 millones de tweets en idioma inglés de 1.39 millones de usuarios recogidos entre septiembre de 2011 y febrero de 2012.

¹ Únicamente el 1% del flujo (“*stream*”) de tweets es accesible si se utiliza la API v1.1, tal y como se especifica en “Filtered Stream” <https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/migrate> [Disponible a 01/01/2022].

Si en lugar de acceder al flujo utilizamos la función “*Search*” podremos recopilar tweets con una antigüedad máxima de 7 días, tal y como se indica en “Search Tweets: Standard v1.1” <https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets> [Disponible a 01/01/2022]

² “Geot-tagged Microblog Corpus” <http://www.cs.cmu.edu/~ark/GeoText/> [Disponible a 23/12/2021]

³ En “Supervised Text-based Geolocation Using Language Models on Adaptive Grids” https://github.com/utcompling/textgrounder/wiki/RollerEtAl_EMNLP2012 [Disponible 23/12/2021] se indica cómo obtener los datos.

⁴ Según [13], los datos están disponibles en <http://www.csse.unimelb.edu.au/~tim/etc/coling2012-geo.tgz>. Sin embargo, en la fecha de elaboración de este documento, dicha dirección no existía. Existe una página relacionada con la Universidad de Melbourne que actualmente tiene la url <https://www.cis.unimelb.edu.au>

2.3.2 Preprocesamiento

El preprocesamiento comprende todas las operaciones que realicemos para filtrar los datos recogidos y seleccionar aquellos que vayan a ser útiles para los algoritmos de localización que se implementen posteriormente. Ya hemos visto que, por ejemplo, para constituir el conjunto de referencia se precisan tweets geolocalizados, y únicamente un porcentaje muy bajo lo están.

Un tipo de procesamiento se basa en el valor de alguno de los metadatos que ofrece la API de Twitter de cada uno de los tweets. Algunos de los ejemplos más comunes pueden ser:

- De los datos del perfil de usuario:
 - Valor de localización no nulo.
 - Número de amigos o seguidores mayor o menor que un cierto umbral.
 - Número de tweets, incluyendo retweets, que ha puesto el usuario mayor que un mínimo.
- De los datos del tweet:
 - Fecha de creación del tweet dentro de algún periodo concreto.
 - Si el tweet es un retweet o una contestación a otro usuario.
 - Si el tweet tiene geolocalización, ya sea mediante una coordenada geográfica o mediante la designación de un lugar que contendrá una caja de coordenadas delimitadora y otra información como: país, nombre del lugar o tipo de lugar.
 - Idioma del tweet, por si únicamente queremos utilizar tweets de un determinado idioma.

Otro tipo de procesamiento que se aplica al texto de algunos de los campos, bien del perfil de usuario, como por ejemplo del campo descripción, o bien del propio tweet, donde típicamente se procesa el campo de texto. Este tipo de procesamiento es más complejo y, aunque tiene características comunes al procesamiento de lenguaje natural (“*Natural language processing*”) o PLN, en [18] se sugiere que hay ciertas diferencias a tener en cuenta. Sugieren que los métodos tradicionales de PLN ofrecen escasa eficacia, debido a las particulares características de los tweets: lenguaje informal, utilización de abreviaturas, emoticonos, url’s, hashtags, etc. El objetivo de este procesamiento previo es la obtención de un vector de características que sea procesable, normalmente, por los algoritmos de Machine-Learning. Cada posición del vector se asocia con una determinada palabra y adquiere el valor de la frecuencia con que aparece dicha palabra en los tweets de un usuario. Por ejemplo, si en el conjunto de los tweets de un usuario aparece la palabra “playa” y “metro” con cierta frecuencia, quizás se pueda asignar una probabilidad de localización mayor para la ciudad de Barcelona que para la de Madrid. El problema es que trabajar con vectores de tantas columnas, como posibles palabras distintas tengan los tweets de los usuarios, sería muy costoso computacionalmente. De ahí la necesidad del preprocesado.

En el preprocesado de textos existen diversos métodos que se suelen emplear combinándolos entre sí, con el fin de obtener un vector de características lo más significativo y reducido posible:

- “*Tokenization*”

Es el proceso por el cual se divide la frase en sus partes significativas y se identifican las entidades individuales que la componen. Se trata fundamentalmente de identificar los límites de las palabras buscando los espacios o los signos de puntuación para poder separarlas. Las entidades con significado a considerar se denominan n-gramas. Un 1-grama consta simplemente de una palabra, mientras que un 2-grama o bigrama consta de dos. Tal y como se indica en [19], el proceso de “*tokenization*” es más complicado en el ámbito de las redes sociales que el que se utiliza para los textos de los periódicos o las publicaciones científicas. Los autores de [19], que han desarrollado una aplicación denominada “*TweetMotif*” para agrupar tweets por temas, utilizan expresiones regulares que permiten el

tratamiento diferenciado de los textos habituales en Twitter que incluyen caracteres como @, #, o las abreviaturas y emoticonos.

- Etiquetado (“*Part-of-speech (PoS) tagging*”)

Se trata de identificar el tipo gramatical de la palabra en función de la posición en la frase, detectando así nombres, url’s, etc, o incluso extrayendo segmentos de frases que se refieren a un momento temporal: pasado, presente o futuro.

- Eliminación de palabras no significativas (“*stopwords*”).

Este tipo de palabras son aquellas poco significativas para el significado general de la frase, tales como conjunciones, preposiciones, artículos, interjecciones, etc. Cada idioma tiene obviamente su propio subconjunto de “stopwords” que es conveniente eliminar, para reducir el tamaño de los vectores de características. En estos vectores se contabiliza el número de veces que se utiliza cada una de las palabras, por lo que contabilizar las ocurrencias de palabras sin significado útil para la clasificación únicamente incrementará el coste que tendrá el procesamiento.

- “*Term frequency – inverse document frequency*” (TF-IDF).

Se trata de una opción alternativa a la eliminación de las “stopwords”. En este caso, en el vector de características no se indicará directamente el número de ocurrencias de una palabra sino el valor de una expresión matemática relacionada. Las palabras que aparecen en muchos documentos son penalizadas en la expresión, puesto que no ayudan a diferenciar unos de otros. Esto es lo que ocurrirá con las “stopwords”, pero también con otras palabras que, aún con significado propio, sean poco relevantes por tener un uso frecuente.

TF-IDF es, por tanto, una medida numérica que indica lo relevante que es una palabra, o término t , para un determinado documento d dentro de un conjunto de documentos o colección D . Se puede calcular de diversas maneras, pero en términos generales es el producto de dos factores: TF e IDF:

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D)$$

- $TF(t, d)$ es número de veces que aparece una palabra concreta en un documento d .
- $IDF(t, D)$ es el valor que expresa lo común que es una palabra, o término t , en un conjunto de documentos D . Se calcula con la expresión:

$$IDF(t, D) = \log\left(\frac{|D|}{1 + |d \in D: t \in d|}\right)$$

Donde $|D|$ es el número total de documentos de la colección y $|d \in D: t \in d|$ es el número de documentos de la colección en los que aparece el término t . Para evitar posibles divisiones por cero, se suma en el denominador el valor 1.

- Obtención-raíz (“*Stemming*”).

Esta técnica consiste en considerar iguales aquellas palabras que tengan la misma raíz. De ese modo se evita considerar de forma separada la palabra árbol de la palabra árboles. En general se trata de eliminar sufijos, pero también prefijos y terminaciones verbales en función del idioma utilizado¹. Un término similar es el de obtención-concepto, o conceptualización (“*lemmatization*”), en el cual no se obtiene la raíz de las palabras, sino la palabra que expresa el concepto. Por ejemplo, ante la palabra numeral, si utilizamos la

¹ Por ejemplo, para el idioma inglés se han desarrollado algunos algoritmos de stemming concretos como “Porter stemmer”, su variante “Snowball stemmer”, o también “Lancaster stemmer”. Todos ellos se pueden encontrar en la librería *nlTK* de Python que incorpora además funcionalidades para otros idiomas como el español.

obtención-raíz podríamos obtener numer, mientras que si utilizamos obtención-concepto obtendremos número.

Como se ha mencionado, las palabras significativas identificadas dentro del tweet podrán ser procesadas posteriormente por otros algoritmos para, por ejemplo, clasificar el tweet dentro de una categoría temática, tal y como se explica en el punto 2.4.1.2.2 al hablar de *Latent Dirichlet Allocation*.

Es importante resaltar que el principal objetivo del preprocesamiento es reducir y conformar los datos para que los algoritmos que trabajen con ellos lo hagan de la forma más rápida y efectiva posible. Eso implica que el preprocesamiento será particular y específico para el algoritmo concreto que se vaya a utilizar.

Por otro lado, el preprocesamiento es una etapa muy importante. El hecho de filtrar los datos de entrada de una forma u otra produce un conjunto de datos diferente, y el resultado del algoritmo que los procesa, también es distinto. Aunque el algoritmo que se encarga del procesado sea el mismo, si la etapa de preprocesado es diferente, el sistema, en su conjunto, también lo es.

Algún ejemplo concreto de preprocesamiento lo podemos encontrar en [20], donde para filtrar a los usuarios que pudieran ser personalidades o programas (“bots”), descarta aquellos que tengan más de 30.000 seguidores. Igualmente, se elimina a los usuarios poco activos, considerando como tales a aquellos que tengan menos de cuatro tweets en un período de 100 horas.

En [21], para obtener la “home location” de un usuario, se requieren al menos tres tweets geolocalizados y cuya distancia máxima entre ellos no supere los 30 km. También se filtran localizaciones que indican posiciones cerca de las coordenadas 0° de latitud y 0° de longitud, que se asocian a un mal funcionamiento del GPS del usuario al publicar el tweet, o tweets publicados por un mismo usuario, con unas posiciones e intervalos de tiempo que implican desplazamientos a más de 1000 km/h, lo que sugiere que han sido realizados por bots.

En [16], con la intención de eliminar del conjunto de datos los tweets de los bots y de los *spammers*, suprimieron todos aquellos usuarios que seguían a más de 1000 usuarios o que fueron autores de más de 1000 publicaciones en un periodo de tres meses que duró la recogida de datos.

2.3.3 Representación de la localización

La representación de la localización es el modo en que el algoritmo de geolocalización ofrecerá su resultado. Es decir, ante un conjunto de datos de entrada, ofrecerá una predicción de localización. Esta predicción podrá tener la forma de coordenadas geográficas –números para la latitud y longitud-, o bien, un código, o un literal, que identifique un lugar geográfico.

Esta última opción, que está relacionada con los métodos de clasificación de Machine-Learning, es bastante flexible. Se puede optar por escoger las regiones administrativas de un país, o bien, dividir el mismo país en áreas, con la misma o distinta superficie, identificadas por un código. El algoritmo ofrecerá a su salida la clase –nombre de la región o código que identifique el área- correspondiente.

La primera opción, ofrecer coordenadas geográficas, es propia de algoritmos de Machine-Learning regresivos que se tratarán en apartados posteriores. También es una opción cuando el propósito de la geolocalización precisa este tipo de datos numéricos como salida. En tal caso, la clasificación sigue siendo útil, puesto que cada una de las clases que identifican un área geográfica concreta, siempre pueden ser reducidas a unas coordenadas obteniendo, por ejemplo, su punto central.

Para el caso de los métodos de clasificación, es muy importante seleccionar bien las áreas. Para calcular el error que se comete en la localización de un usuario se suele obtener la diferencia entre su posición real y la ofrecida por el algoritmo de geolocalización. Si la posición ofrecida por el algoritmo es el centro del área en la que se ha ubicado al usuario, áreas muy grandes producirán errores muy grandes. Por otro lado, tampoco es bueno utilizar divisiones que estén muy desequilibradas en cuanto a número de usuarios, porque el aprendizaje de los algoritmos de Machine-Learning no será eficiente en esos casos. Estas son las razones por las que, en ocasiones, se opta por una selección adaptativa de las áreas de clasificación. El particionado adaptativo tiene en cuenta la densidad de usuarios en cada región y reduce el tamaño de las divisiones para que los lugares con mayor densidad se correspondan con divisiones más pequeñas, mientras las menos densas se correspondan con divisiones más grandes. En [8] se utilizan dos parámetros, a modo de umbrales, para definir el tamaño de las divisiones. El primero es el límite máximo de usuarios por división: $T_{\max}$. El segundo es un tamaño máximo de división, así, aun cuando no se alcance el máximo de usuarios, se impide que las divisiones sigan creciendo en tamaño para evitar que sean excesivamente grandes. Se trata pues de una solución de compromiso que consigue que las divisiones no sean ni muy grandes, ni muy pequeñas, aun a riesgo de que haya algunas áreas con pocos usuarios. En realidad, el tamaño máximo se corresponde con un determinado nivel en la partición que utiliza una librería específica de Google: (“*Google’s S2 geometry library*”¹) que agrupa, en hasta 30 niveles, las proyecciones de la superficie terrestre sobre un cubo, de modo que tengan un tamaño similar. Por ejemplo, al nivel 12 le corresponden áreas que varían entre tres y seis kilómetros cuadrados aproximadamente. Las divisiones conforman un árbol jerárquico donde un área de un determinado nivel se subdivide en cuatro del siguiente nivel.

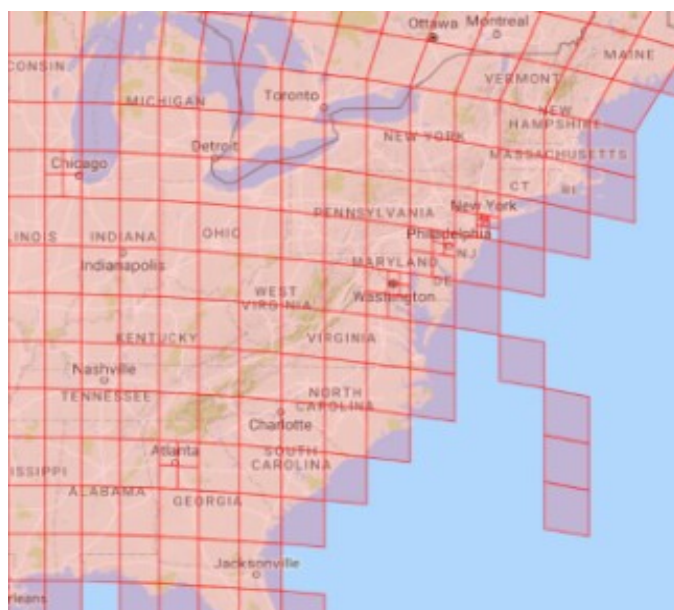


Figura 2-5 Detalle de la división en áreas de Norteamérica (procedente de [8])

En la Figura 2-5 se puede observar cómo para ciertas ciudades como, Nueva York, Washington, Atlanta o Chicago las divisiones son más pequeñas, al contener un mayor número de usuarios, mientras que en otras zonas las divisiones han alcanzado su nivel máximo determinado por $L_{\min}$ en las que el número de usuarios es significativamente menor.

El procedimiento para construir estas divisiones parte del conjunto de datos de entrenamiento, en el que se conoce la ubicación real de los usuarios. Cada uno de los usuarios se

¹ La descripción de esta librería se puede encontrar en “S2 Geometry” <https://s2geometry.io/> [Disponible a 23/12/2021]. Existen implementaciones para lenguajes Go, Java y Python.

asigna a una de las zonas de mayor nivel de las definidas en la librería Google-S2. Estas zonas se corresponden con las hojas de una posteriormente se van fusionando entre sí las zonas de más nivel del árbol, siempre que no alcancen el máximo de usuarios determinado por $T_{\max}$ o alcancen el área máxima determinada por las divisiones con nivel $L_{\min}$.

En [17] se opta por una representación basada en ciudades en lugar de por las divisiones de cuadrícula, en las que, según los autores, existe una variabilidad demasiado grande entre las que son apropiadas para regiones densamente pobladas y las que lo son para las escasamente pobladas. Por otro lado, reconocen que la representación basada en ciudades es inapropiada para clasificar a los usuarios ubicados en áreas rurales. Este inconveniente es considerado como menor puesto que la gran mayoría de los usuarios de Twitter se encuentran en las ciudades. Las clases asociadas como ciudades son extraídas inicialmente de la base de datos *GeoNames* pero se procesan de tal modo que van agrupando las poblaciones cercanas hasta un máximo de 50km, dentro de una misma región administrativa, y considerando únicamente las que tienen más de 100.000 habitantes. Aplicando un algoritmo, basado en esos criterios, identifica 3709 ciudades en el mundo. Esta selección constituye el conjunto de clases con las que trabajará el algoritmo de geolocalización.

Un enfoque diferente lo podemos encontrar en [4], en la que la propia recogida de los datos se realiza de modo que estos ya estén geolocalizados utilizando una herramienta de búsqueda que proporciona la propia API de Twitter. Esta herramienta permite realizar búsquedas en el histórico de tweets especificando un círculo, mediante unas coordenadas del centro y un radio. La API nos devuelve los tweets que Twitter sabe, basándose en información privada y sin que estén geolocalizados por los usuarios, que han sido emitidos desde su interior. Este método permite, no solo obtener un conjunto de referencia geolocalizado, sino que también establece, desde el principio, la representación de la localización. En el documento mencionado se subdivide Gran Bretaña en círculos de un radio diferente. Más pequeños para las áreas densamente pobladas. La definición del tamaño y ubicación de dichos círculos trata de optimizar diversos criterios, como la eficiencia en la recogida de los tweets o el minimizar el área de solapamiento entre círculos para evitar que un mismo tweet pueda ser asignado a dos de ellos.

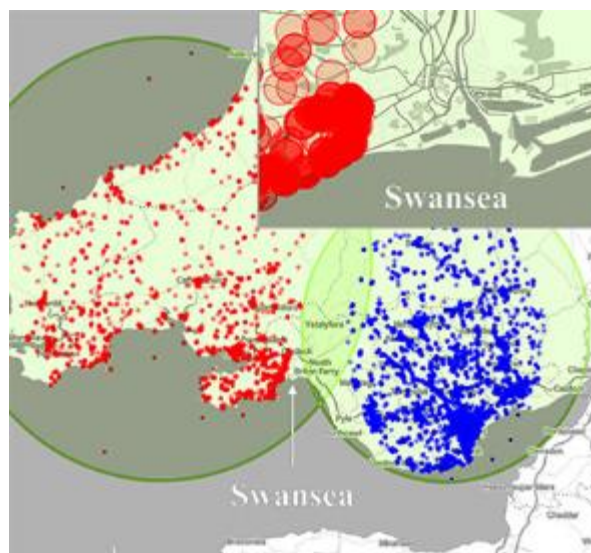


Figura 2-6 Problema de la cobertura del círculo (procedente de [4])

Una de las dificultades a las que tuvieron que enfrentarse los autores fue lo que denominaron el problema de cobertura del círculo (*“circle coverage problem”*). Se detectó que, al definir círculos relativamente grandes, la herramienta de búsqueda de Twitter no recogía los tweets correspondientes a los bordes del círculo. Si ese borde coincidía con un área densamente poblada

como el centro de una ciudad, dejaban de recogerse una cantidad significativa de tweets. Este problema, parece ser un defecto de la propia API y no es posible saber cuándo se solucionará.

En la Figura 2-6 se observan dos de los círculos en el que se han marcado con puntos la ubicación de los tweets que los usuarios sí habían geolocalizado. Se observa que, en el borde de ambos círculos, donde se ubica la ciudad de Swansea, no hay ningún punto, a pesar de ser una de las zonas más densamente pobladas de la región. Esa anomalía lleva a la conclusión de que, por alguna razón desconocida, entre los tweets que recoge la API no se incluyen los ubicados en las proximidades del borde de los círculos.

2.3.4 Diseño del algoritmo de inferencia

El algoritmo de inferencia es la parte más importante del sistema de geolocalización. En los apartados anteriores hemos recogido los datos, los hemos procesado y hemos determinado el formato de salida del algoritmo. Ahora es el momento en que se termina de definir el algoritmo, y para ello es necesario determinar qué método de entre los posibles se va a emplear. Según [10] existen tres posibilidades: los métodos basados en análisis de textos, los que se basan en las redes de usuarios y los que híbridos o mixtos que utilizan parcialmente ambos. Esta es la clasificación que utilizaremos para describir los distintos algoritmos, sin embargo, existen otras posibilidades, como la que se describe en [5], donde se clasifican en algoritmos basados en procesamiento de lenguaje natural (PLN), los que emplean diccionarios de topónimos y los que utilizan Machine-Learning, tal y como se puede ver en la figura Figura 2-7.

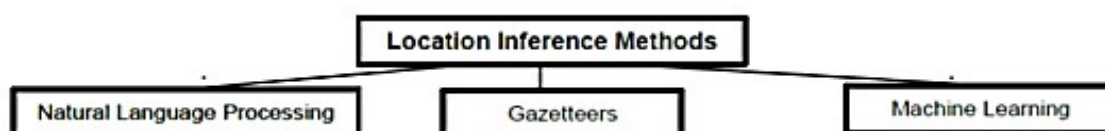


Figura 2-7 Clasificación alternativa de algoritmos de inferencia (procedente de [5])

Dado que esta parte es la fundamental en el proceso de geolocalización se desarrolla con más detalle en el apartado 2.4 de este documento. Se menciona simplemente aquí por mantener la secuencia mostrada en la Figura 2-4.

2.3.5 Evaluando la eficacia de la localización

Todos los métodos descritos en la literatura académica utilizan algún tipo de valor que cuantifique su eficacia. Como ya se ha sugerido anteriormente, hay dos tipos de medidas relevantes. La primera, que podemos denominar precisión, es lo bueno o malo que sea el resultado ofrecido por el algoritmo de geolocalización. Esto es, lo alejado que esté la respuesta, de la ubicación real. La segunda, denominada exhaustividad, se refiere a la cantidad de usuarios sobre los que el algoritmo es capaz de predecir una ubicación. Algunos algoritmos siempre ofrecen una ubicación, aunque sea mala, por lo que se puede decir que tienen una exhaustividad del 100% a costa de una disminución en la precisión. Otros algoritmos pueden optar, por lo contrario, ofreciendo gran precisión pero escasa exhaustividad.

Los datos que se utilizan para cuantificar estas medidas se obtienen aplicando el algoritmo a evaluar sobre un subconjunto de los datos denominado subconjunto de test, del que se conoce la ubicación real previamente. Este subconjunto de datos, además, no puede haberse utilizado para entrenar, si fuese necesario, el algoritmo a evaluar.

Algunas de las métricas más utilizadas, que podemos encontrar en [22], [10] y [23] son las siguientes:

- Mediana de los errores de distancia (“*Median error distance*”): Es la mediana de las distancias entre las localizaciones predichas y las reales de los tweets utilizados como test.
- Exactitud (“*Accuracy*”): Utilizado en algoritmos de clasificación en los que se comprueba si la ubicación real de los tweets se encuentra dentro de la zona identificada por alguna clase. La precisión cuantifica porcentualmente el número de aciertos.
- Existe una variante de esta medida (“*Precision@k*”¹) en la que se mide el número de predicciones que se encuentran entre las k clases más probables predichas por el algoritmo.
- Exactitud a n kilómetros (“*Accuracy at n kilometers*”): Mide el porcentaje de ocasiones en las que la localización real y la predicha no difieren más de una cierta distancia. Así, por ejemplo, *Acc@166* indica la precisión para una distancia de 166 kilómetros.
- Media de los errores de distancia (“*Mean error distance*” o “*Average error distance*”): Es la media de las distancias entre las localizaciones predichas y las reales.
- Mediana de la distancia máxima de error (“*Median-max error distance*”): Es la mediana de los errores más altos obtenidos para la localización de todos los usuarios. Este valor proporciona un valor intuitivo de la precisión esperada puesto que la mitad de las predicciones tendrán un error máximo de, al menos, esa cantidad.
- Cobertura (“*User coverage*”): Es el porcentaje de usuarios de los que es posible, utilizando un determinado algoritmo, inferir su localización. El concepto también es aplicable al número de tweets que se es capaz de geolocalizar (“*Post coverage*”).

Otra de las métricas más utilizadas se ha convenido en abreviar como AUC (“*area under the curve*”) y se basa en la función de distribución acumulativa (“*Cumulative Distribution Function*”: CDF). CDF muestra el porcentaje de predicciones que tienen un error menor de x kilómetros de la ubicación real. CDF se define formalmente como:

$$F(x) = P(\text{distancia} \leq \log(x))$$

Esta función es muy descriptiva puesto que incluye las medidas de *Acc@n* para diversos valores de n .

En la Figura 2-8 se muestra un ejemplo de la función CDF. Las diferentes líneas corresponden a diferentes algoritmos que se quieren comparar. Se puede ver, por ejemplo, que el algoritmo *Dav11* tiene casi un 50% de predicciones con un error menor de 10 kilómetros.

Se observa que el eje horizontal utiliza una escala logarítmica. La razón por la que es así, es la siguiente: supongamos que tenemos cuatro sistemas de localización en los que la probabilidad de geolocalización para una determinada distancia es del 70%. El primero obtiene dicho porcentaje para distancias de hasta 1 km, el segundo hasta 10km, el tercero para 100km y el cuarto para 109km. Comparándolos entre sí, el primero es un orden de magnitud mejor que el segundo, aunque la diferencia de distancia sea solo de 9 km. Mientras que el tercero y el cuarto, a pesar de que también hay 9km de diferencia, se puede decir que son prácticamente similares en cuanto a eficacia. Puesto que AUC utiliza CDF, los valores más bajos de x tendrán más peso en el cálculo total del área.

¹ En este documento nos referiremos a “*Accuracy*” como exactitud y “*Precision*” como precisión.

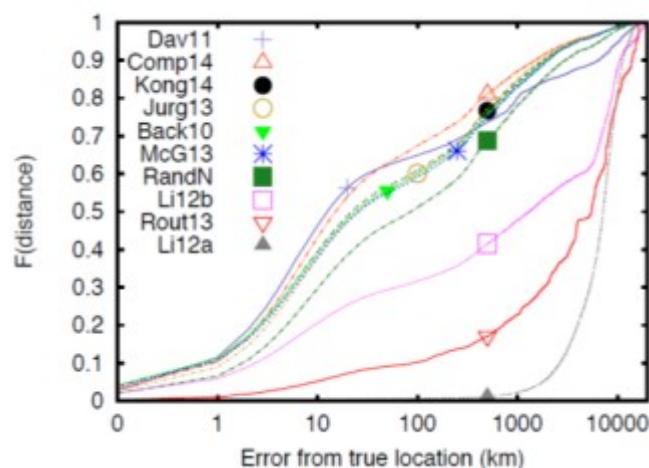


Figura 2-8 Ejemplo de CDF (procedente de [23])

Por otro lado, AUC normaliza la magnitud de los valores situándolos dentro del rango $[0, 1]$. Si AUC vale 1 significa que todas las predicciones de localización se realizan sin error, mientras que si vale 0 indica que las predicciones están lo más lejos posible, en la superficie terrestre, de la ubicación real. Al normalizar esta medida se puede utilizar para comparar fácilmente la eficacia de distintos métodos.

2.4 Principales métodos de geolocalización

2.4.1 Métodos basados en análisis de texto

Los métodos basados en análisis de texto utilizan la información contenida en los propios tweets o en el perfil del usuario para relacionarla con una ubicación geográfica.

En consonancia con el esquema de la Figura 2-4, los métodos basados en análisis de texto se subdividen en dos: los que utilizan algún diccionario toponímico (“*gazetteer*”) y los que emplean Machine-Learning, incluyendo aquellos que utilizan fundamentalmente técnicas estadísticas.

2.4.1.1 Métodos de análisis de texto que utilizan diccionario toponímico.

Un diccionario toponímico, que es la traducción por la que hemos optado de la palabra inglesa “*gazetteer*”, es una base de datos en la cual se recogen una gran cantidad de lugares con su información relacionada, incluyendo, además del nombre, sus coordenadas geográficas y otros datos de interés. Lugar se entiende en un sentido amplio y puede referirse a un país entero, o a un comercio concreto. Estas bases de datos están accesibles a través de internet y habitualmente proporcionan un servicio que permite la realización de consultas. Este servicio es gratuito en algunos casos y en otros no. Además, el número de consultas permitido se restringe a un cierto número por unidad de tiempo.

Algunas de estas bases de datos proporcionan una API para realizar las consultas mediante programación. Entre ellas se pueden citar: *MapQuest Open Geocoding API*¹, *Google Geocoding API*², *ArcGIS API*³, *OpenStreetMap API*⁴, *Bing Maps*⁵ o *GeoNames*¹.

¹ “Access the tools and resources you need to map the world” <https://developer.mapquest.com> [Disponible a 23/12/2021]

² “Web Services Geocoding API” <https://developers.google.com/maps/documentation/geocoding/overview> [Disponible a 23/12/2021]

³ “APIs, Tools and Location Services” <https://developers.arcgis.com> [Disponible a 23/12/2021]

⁴ “ES: Página principal” https://wiki.openstreetmap.org/wiki/ES:Página_principal [Disponible a 23/12/2021]

⁵ “Bing Maps Documentation” <https://docs.microsoft.com/en-us/bingmaps/> [Disponible a 23/12/2021]

GeoNames es una base de datos gratuita contiene unos 25 millones de nombres geográficos de los que 11.8 millones corresponden a elementos únicos, entre los que se encuentran 4.8 millones de lugares poblados. Las coordenadas que ofrece utilizan el sistema geodésico WGS84². Existe un acceso comercial y otro gratuito. Este último está limitado a 20.000 créditos diarios por usuario registrado y 10.000 créditos por hora³. Cada tipo de consulta consume una cantidad de créditos que varía entre 0.1 y 4⁴. Esta base de datos es utilizada en diversos documentos académicos relacionados con la geolocalización, como por ejemplo [17], [7], [24], [25], [21], etc.

El principal inconveniente de este tipo de bases de datos, en especial las que han sido creadas mediante proyectos colaborativos de voluntarios, es que la mayor parte de las entradas pertenecen a unas ciertas áreas geográficas, mientras que otras, tienen mucha menor densidad de entradas, tal y como se observa en la Figura 2-9.

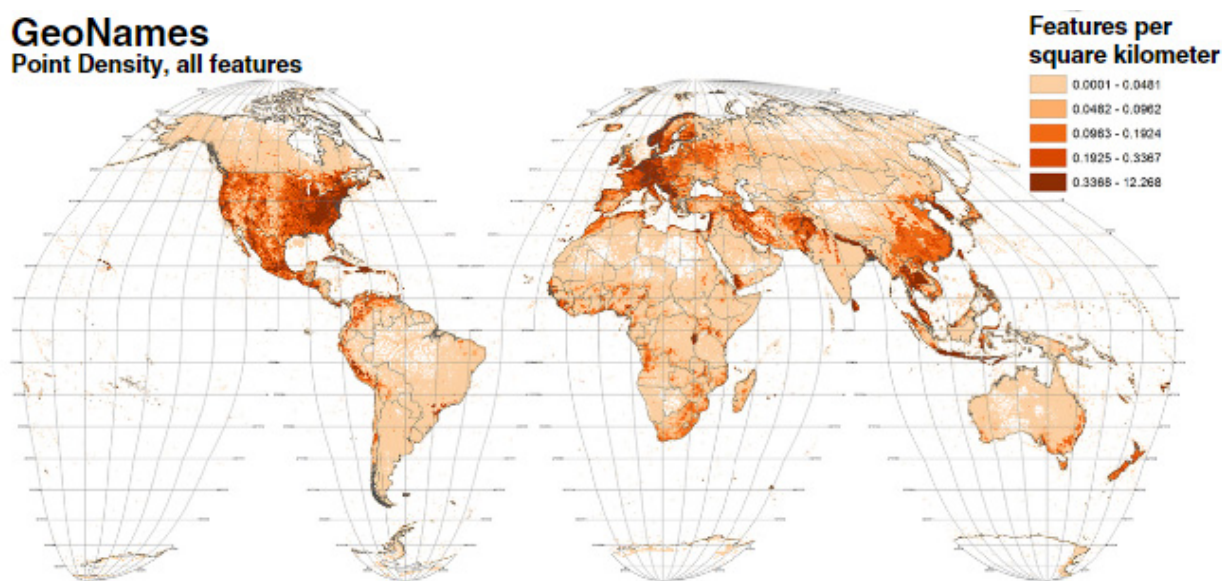


Figura 2-9 Densidad por kilómetro cuadrado de las características geográficas de GeoNames (procedente de [44])

El funcionamiento de estos algoritmos se basa en identificar lo que en [24] se denomina LIW (“*Location Indicative Words*”), tanto en los campos del perfil de usuario, como en los propios tweets. En [26], se analiza el perfil de usuario, cotejando las palabras existentes con la base de datos DBpedia⁵ para comprobar si coincide con alguna entrada clasificada como un lugar. El texto del usuario se coteja con la DBpedia, palabra a palabra, en grupos de dos (bigramas), y hasta en grupos de tres palabras (trigramas). Si alguna de estas combinaciones se corresponde con una entrada de la DBpedia caracterizada como un lugar, entonces dicho lugar será uno de los candidatos para geolocalizar al usuario.

Los LIW, tal y como se utilizan en [24], no tienen por qué ser palabras que indiquen lugares. Un ejemplo que utilizan es el de la palabra *dippy*. Esa palabra no está en el diccionario y no tiene

¹ “GeoNames Web Services Documentation” <http://www.geonames.org/export/web-services.html> [Disponible a 23/12/2021]

² “About GeoNames” <https://www.geonames.org/about.html> [Disponible a 23/12/2021]

³ “GeoNames Data” <https://www.geonames.org/export/> [Disponible a 23/12/2021]

⁴ “GeoNames Webservices Credits” <http://www.geonames.org/export/credits.html> [Disponible a 23/12/2021]

⁵ La DBpedia es un proyecto de la Universidad de Leipzig que extrae datos de la Wikipedia para crear una base de datos de contenido estructurado en forma de web semántica. Se puede encontrar información sobre la DBpedia en <https://es.dbpedia.org/wiki/attach/III%20Jornadas%20esDBpedia/FAQ%20DBpedia%20y%20fact-sheet%20esDBpedia.pdf> [Disponible a 23/12/2021]

sentido para la mayoría de los norteamericanos, sin embargo, se usa en la ciudad de Pittsburgh para designar una preparación específica de huevo frito. Este tipo de palabras se puede considerar, por tanto, LIW y, sin embargo, no tiene relación con los métodos que utilizan un diccionario toponímico. Otro ejemplo podría ser la palabra *metro*, circunscrita a la geolocalización de tweets en España. En algunos casos, la palabra haría referencia a una medida de longitud y no se podría considerar un LIW, pero en otras haría referencia a un tipo de transporte urbano que existe en relativamente pocas ciudades.

Sin embargo, existen otro tipo de LIW que sí tienen relación con lugares y que, por tanto, son susceptibles de encontrarse en un diccionario. Ejemplos de estos casos, podrían ser palabras como *Cibeles*, *Rambla*, *Triana*, etc. Que, aunque no sean exclusivas de una ciudad concreta, pueden acotar la búsqueda.

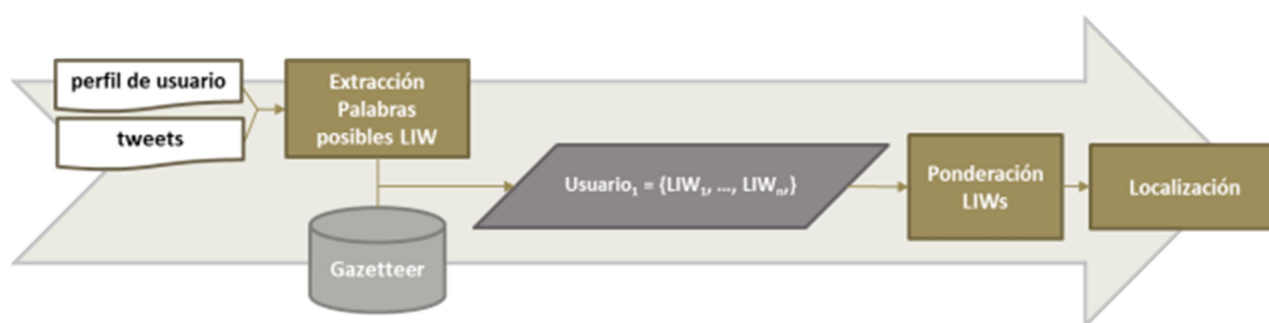


Figura 2-10 Esquema del método de análisis de texto que utiliza diccionario toponímico.

En la Figura 2-10 vemos un esquema del proceso de localización. El proceso de extracción de palabras que posiblemente sean LIW puede ser más o menos complejo. Una acción básica consistiría en la eliminación de las palabras conectoras: artículos, preposiciones, etc. Cuantas más palabras se eliminan, menos consultas habrá que realizar al servicio del diccionario. Los LIW que se vayan obteniendo se van asociando al usuario correspondiente. Seguidamente hay que valorar la lista de LIW determinando cuáles de ellos tienen más peso en la localización final. Se puede utilizar un conteo de palabras y asignar la ubicación al topónimo que aparezca más veces. Otra opción podría ser, desechar las ubicaciones que no tengan relación con las demás. Por ejemplo, si todas las ubicaciones son de España y solo una es de otro país, podemos eliminarla. Para este proceso, es de gran ayuda la clasificación jerárquica de las categorías de los diccionarios. Así, una ubicación puede ser un barrio, de una ciudad, de una región, de un país. *GeoNames*, por ejemplo, permite hasta cinco órdenes de división administrativa¹.

Los métodos basados exclusivamente en diccionarios toponímicos, aunque relativamente sencillos de implementar, no se encuentran en la literatura académica. Probablemente porque se considera que tienen poca eficacia, además de que el proceso de consulta al diccionario puede ser costoso. Es más habitual realizar algún tipo de combinación con otros métodos. Por ejemplo en [26], se utilizan las localizaciones inferidas de las cuentas que sigue un determinado usuario, ponderadas de diversa forma, para obtener la localización final. Es decir, se utiliza la información de la red de usuarios.

Aunque la utilización de los diccionarios toponímicos no sea el núcleo del algoritmo de localización, es posible su empleo en otros procesos fundamentales. Por ejemplo, se puede utilizar para la construcción del conjunto de datos de referencia (“*ground-truth*”). Partiendo de las localizaciones que los usuarios declararan sobre sí mismos, se puede utilizar el diccionario para extraer la ubicación con la granularidad deseada (país, ciudad, etc.) e incluso las coordenadas geográficas. Otra posibilidad es la determinación de la representación de la

¹ “GeoName Feature Codes” <https://www.geonames.org/export/codes.html> [Disponible a 23/12/2021]

localización. En [17], se utiliza *GeoNames* para establecer 3709 posibles ciudades alrededor del mundo. Esas ciudades serán la que el algoritmo de localización, posteriormente, determine como ubicación de los usuarios.

2.4.1.2 Métodos de análisis de texto que utilizan Machine-Learning.

2.4.1.2.1 Descripción general de los métodos de Machine-Learning.

Como cualquier algoritmo, los de Machine-Learning (ML) ofrecen un resultado a su salida, a partir de unos datos de entrada. El resultado depende, tanto de los datos como de las instrucciones. Una característica distintiva de los algoritmos de ML es que precisan ejecutarse un cierto número de veces, con algunos datos de entrenamiento, para ajustar el valor de determinados parámetros internos. Ese periodo es el que se denomina de aprendizaje. Posteriormente, el algoritmo se alimentará con datos reales, ya no de entrenamiento, y ofrecerá un resultado, aplicando en los cálculos que realice los valores de los parámetros obtenidos durante la mencionada etapa de aprendizaje. El efecto es que da la impresión de que el computador ha aprendido cuál es la respuesta a partir de un conjunto de muestras de ejemplo.

Normalmente, los algoritmos de ML son alimentados por tuplas o vectores de características que representan algún elemento de la realidad. Por ejemplo, la entrada de un algoritmo que detecte si un correo es spam o no, podría tener un vector de características con varios valores: el primero contaría el número de veces que aparece la palabra *barato* en el texto del correo; el segundo contaría las ocurrencias de la palabra *oferta*; el tercero tendría el valor *1* si el remitente está en nuestra lista de contactos y *0* si no lo está; etc. La selección de las características, su representación en formato numérico y, en algunos casos su normalización, son tareas necesarias para poder conformar las tuplas que vayan a alimentar a los algoritmos de ML. Para muchas aplicaciones de ML, estas tuplas de características se denominan *muestras*. Por otro lado, la salida de los algoritmos de ML también suele ser de tipo numérico, aunque en ocasiones los números representen características cualitativas. Esto es así porque los ordenadores trabajan más eficientemente con números que con otro tipo de datos.

Ahora que ya conocemos algunos conceptos básicos, ya podemos tratar de explicar qué hacen los algoritmos de ML. En general, se utilizan cuando no hemos encontrado una manera absolutamente determinista para resolver un problema. Quizás porque el problema tiene una gran cantidad de variables que influyen en su salida y no somos capaces de estipular exactamente cuál es su influencia, ni podemos cuantificar la forma en que están relacionadas. Así pues, lo que hacemos es extraer todas las características que nos parecen relevantes y ofrecérselas al ordenador en un formato con el que pueda trabajar fácilmente. El ordenador las procesa y nos ofrece un resultado de salida. En muchas ocasiones ese resultado es una predicción, como el caso del detector de spam de correos, en el que este nos dice: *con los datos extraídos de este correo y con lo que he aprendido del conjunto de correos con que fui entrenado, creo que este correo es, o no, spam*. Hay varios conceptos clave aquí. El primero es que la corrección o no del resultado de salida depende de las características con las que hayamos decidido que trabaje el algoritmo. Si no son las más adecuadas, las predicciones pueden ser erróneas. El segundo es que el algoritmo tratará de emular lo aprendido: *si con estos datos de entrada se obtuvo una determinada salida, con estos otros datos, que se parecen mucho, la salida debe ser muy semejante*. Es decir, las predicciones del algoritmo dependerán también de los datos que se utilicen durante su aprendizaje. Si los datos del aprendizaje no son representativos de los que se presentarán en la etapa de explotación, las predicciones podrán ser erróneas.

Por supuesto, la forma en que trabajan los sistemas de ML que hemos descrito en letra cursiva en el párrafo anterior, tienen que implementarse mediante código, convirtiéndose en cálculos y comparaciones, que es lo que básicamente sabe realizar un ordenador. Para determinar si un vector de características se *parece* a alguno de los aprendidos previamente se utiliza una

función matemática que exprese esa diferencia, como, por ejemplo, el sumatorio de la distancia entre puntos de un espacio n -dimensional. El resultado del algoritmo será aquel que minimice esa diferencia. En los algoritmos de ML solemos encontrar una función, denominada de *coste* que debemos minimizar, modificando, durante el aprendizaje, algunos de los parámetros internos que multiplican a cada una de las características del vector de entrada. Esos parámetros ponderan la importancia de cada característica en el resultado final, por lo que se les suele denominar *pesos*. Existen también otros parámetros que no se modifican mediante el proceso de aprendizaje, sino que los establece el usuario para determinar, por ejemplo, la tasa de aprendizaje –velocidad de modificación de los *pesos* al tratar de minimizar la función de *coste* progresivamente–, o la condición de finalización de la búsqueda de ese mínimo, como, por ejemplo, el número de iteraciones de alguno de los bucles. Esos parámetros que fija el usuario se suelen denominar *hiperparámetros*.

Una pregunta interesante que podemos hacernos sobre los *pesos* es con qué valor los inicializamos al comienzo del algoritmo, es decir, que importancia concedemos a priori a cada una de las características a la hora de realizar la clasificación. La respuesta, sorprendentemente, es que no importa en exceso. De hecho, los pesos se suelen inicializar con valores aleatorios con una distribución normal de media cero y una desviación típica pequeña, para que sus valores sean cercanos a cero. El algoritmo, en sucesivas iteraciones en las que analizará una muestra, irá actualizando progresivamente dichos pesos hasta que la función de coste converja hacia un valor mínimo y no sufra prácticamente variación si se analiza una nueva muestra.

En la Figura 2-11, se muestran las etapas de construcción de un sistema de ML. Se observan las etapas previas, relacionadas con la selección y preprocesado de las características. También la etapa de aprendizaje, en la que el sistema, a partir de un conjunto de datos preseleccionado, ajustará sus parámetros internos, típicamente los *pesos*.



Figura 2-11 Etapas del ciclo de vida de un sistema de Machine-Learning

En la etapa de evaluación, el usuario tiene que valorar si el algoritmo de ML es, o no, el más adecuado para afrontar el problema, valorando algunas medidas de rendimiento u observando los resultados ofrecidos. Puesto que existen diversos tipos de algoritmos, cada cual, con características propias, y, dado que no se puede aseverar que uno sea mejor que otro en todos los casos, el usuario puede optar a priori por cualquiera de ellos. También puede, manteniendo el mismo algoritmo, ajustar los *hiperparámetros*.

Finalmente, en la etapa de Predicción/Explotación, se utilizará el sistema con datos nuevos, desconocidos hasta el momento y, en todo caso, distintos de los utilizados para el aprendizaje, para obtener la salida o predicción.

En general, los algoritmos de ML se clasifican en tres grandes grupos:

- Algoritmos supervisados: Aquellos en los que el usuario conoce cuál debe ser el resultado que se considera correcto para cada una de las muestras utilizadas en el aprendizaje. El algoritmo procesará las muestras y comparará su salida con la correcta para modificar los pesos de los parámetros, ajustándose progresivamente.
- Algoritmos no supervisados: En este tipo no existe un resultado correcto previo, sino que el algoritmo utilizará diversas técnicas, normalmente estadísticas, para determinar diversos grupos de muestras que presenten características semejantes. Posteriormente, durante la explotación el algoritmo tratará de determinar a cuál de dichos grupos pertenecen los nuevos datos de entrada.

- Algoritmos de aprendizaje recompensado o por refuerzo: Este tipo de algoritmos trata de determinar cuál es la mejor acción a tomar en un entorno establecido, con el fin de maximizar alguna noción de premio o recompensa. Uno de los ejemplos habituales es el de los programas que actúan como oponentes en juegos, como el ajedrez o el go. Puesto que este tipo no se emplea para la geolocalización de usuarios o tweets, no trataremos de él en adelante.

En ciertos casos, los algoritmos de ML supervisados ofrecen una salida numérica, como un número natural o real, resultado de alguna operación de cálculo. Este tipo de algoritmos se denomina de *regresión*. Un ejemplo de sistema de regresión sería aquel que intenta predecir el precio del petróleo en el mercado, a partir de diversos índices. En contraposición están aquellos algoritmos que ofrecen como salida la denominación de una categoría o clase, al que se predice pertenece la entrada. Aunque en ocasiones la clase se nombra con un número, podría nombrarse con una letra o una palabra, puesto que no es indicativo de ninguna cantidad. Estos algoritmos se denominan de *clasificación*. Un ejemplo de estos, sería un sistema que tratara de identificar la subespecie de una flor a partir de las medidas de sus elementos.

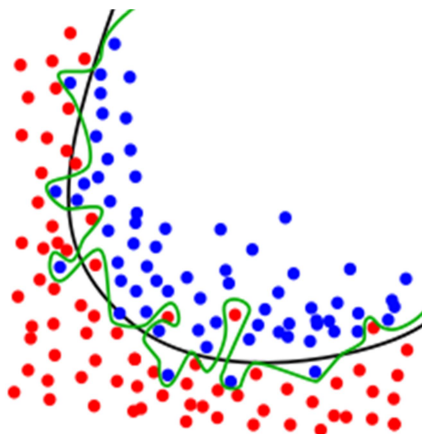


Figura 2-12 El problema del sobreajuste (procedente de wikipedia)

Hasta ahora, hemos empleado casi indistintamente, la palabra sistema de ML o algoritmo de ML. Podemos precisar que el sistema incluye todas las etapas mostradas en la Figura 2-11, mientras que el algoritmo de ML es más bien el código que ejecuta el ordenador para obtener la salida o la predicción, excluyendo cualquier preprocesado previo o análisis posterior. Esta distinción es útil puesto que el mismo algoritmo nos servirá para diversos propósitos. Tanto para clasificar especies de flores como para determinar si un correo es spam. El algoritmo trabajará de la misma forma con datos, sin importar lo que éstos representen. El sistema, por el contrario, incluye, por ejemplo, la etapa de selección de las características con las que trabajará el algoritmo, por lo que sí le interesa lo que representan los datos.

Uno de los problemas de ML, independientemente del algoritmo que se utilice, es el sobreajuste (*“overfitting”*). Aparece cuando tratamos de encontrar un clasificador tan preciso para unos datos de entrenamiento concretos, que al utilizarlo en explotación con datos nuevos (*“unseen data”*), su rendimiento disminuye drásticamente.

En la Figura 2-12, se puede observar que para separar las dos clases, el clasificador representado con la línea verde consigue una precisión perfecta con estos datos de entrenamiento. El clasificador representado por la línea negra tiene una precisión menor, puesto que clasifica erróneamente más muestras. Sin embargo, se puede considerar que es más adecuado, puesto que se ajusta mejor al conocimiento que pretendemos que adquiera el algoritmo para clasificar dos grupos de muestras, ignorando la posible variación aleatoria de las mismas.

Para evitar el sobreajuste, los algoritmos suelen incluir algún hiperparámetro que se puede ajustar manualmente. Este proceso de análisis y ajuste es propio de la etapa de Evaluación, mostrada en la Figura 2-11.

Algunos de los algoritmos ML están diseñados para trabajar únicamente con dos clases (clasificación binaria). Si tenemos muestras correspondientes a más clases, entonces es necesaria una adaptación. Los métodos más habituales son:

- Uno contra todos/el resto (“one-versus-all” OvA / “one-versus-rest” OvR). Se trata de descomponer el problema de clasificación multiclase en varios problemas de clasificación binaria. Suponiendo tres clases, rojo, verde y azul, primero ejecutaremos el algoritmo para valorar si la muestra a analizar pertenece a la clase rojo, considerando la otra posible clase como la verde y la azul juntas. Posteriormente, repetiremos la ejecución para la clase azul, considerando juntas la roja y la verde, etc. Finalmente, se escogerá aquella clase que ofrezca un mejor valor en la función de coste. El número de ejecuciones consecutivas es igual al número de clases.
- Uno contra uno (“one-versus-one” OvO). La idea es similar a la anterior, pero en lugar de comparar cada clase contra el resto, lo hacemos con cada una individualmente. Si tuviéramos cuatro clases, $k=4$, tendríamos que repetir la ejecución del algoritmo 6 veces, lo que se corresponde con la expresión $(k*(k-1))/2$.
- Grafo acíclico dirigido (“directed acyclic graph” DAG). Si en la estrategia OvO, al ejecutar el algoritmo entre las clases A y B, por ejemplo, se obtuviera que el resultado es la clase A, podemos considerar, al haber desechado la clase B, que ya no es necesario comparar las clases B-C, B-D, etc. De ese modo, se crea un grafo de posibles comparaciones que nos permite resolver el problema con $k-1$ ejecuciones. El grafo necesario para cuatro clases, se muestra en la Figura 2-13.

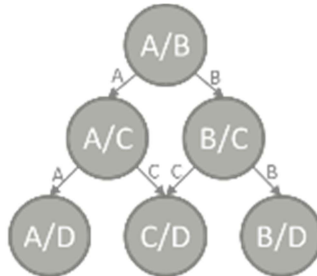


Figura 2-13 Clasificación multiclase DAG

2.4.1.2.2 Algoritmos de Machine-Learning.

En este apartado trataremos brevemente sobre algunos de los algoritmos de ML más conocidos que también se utilizan en la geolocalización de los tweets o de usuarios, como veremos más adelante.

• Regresión logística.

Se trata de un algoritmo de clasificación supervisado, a pesar del nombre. En su versión más simple permite clasificar las muestras en dos conjuntos (clasificación binaria), pero puede extenderse para trabajar con más conjuntos. La separación que realiza de las muestras es de tipo lineal, al igual que otros algoritmos como el *Perceptron* o *ADALine*.

En la Figura 2-14, se han clasificado muestras de tres tipos de flores, en función de dos de sus características. Las clases se identifican en el gráfico por estar contenidas en un área de distinto color. Las muestras que no están rodeadas por un círculo son las que se han utilizado para el aprendizaje. Las que sí lo están, son las utilizadas para test. Se observa que no todas las

muestras han sido correctamente clasificadas. Pero también se observa que, mediante una separación lineal, ni siquiera el conjunto de datos de entrenamiento se puede separar perfectamente (nótese las muestras en la frontera entre las clases 1-2 o verde-azul). Este hecho no es un problema de funcionamiento del algoritmo, sin embargo, si se observa que el conjunto de muestras no puede separarse, razonablemente, mediante líneas rectas, entonces deberíamos utilizar otro algoritmo que no fuera “lineal”.

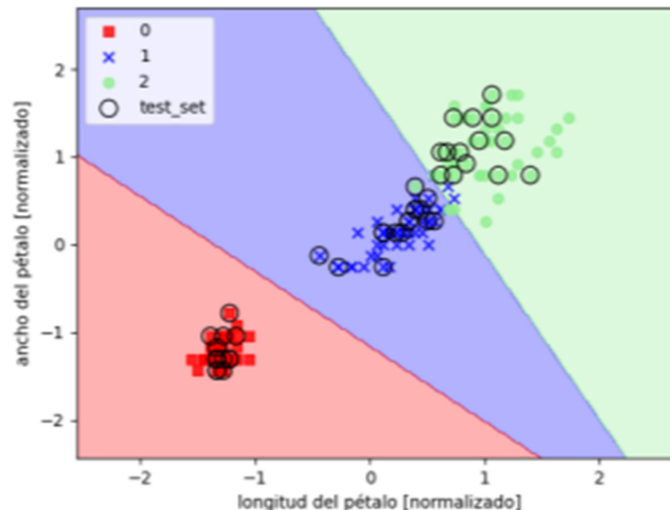


Figura 2-14 Ejemplo de clasificación mediante regresión logística (obtenido mediante adaptación del código propuesto en [45])

Hay que resaltar que, en el espacio bidimensional que implica el hecho de seleccionar únicamente dos características, es fácil e intuitivo visualizar las líneas de separación entre clases. Si hubiéramos escogido tres características las fronteras de separación serían planos, en lugar de líneas. En general, los subespacios de $n-1$ dimensiones se denominan *hiperplanos*. Si utilizásemos más características, que suele ser lo habitual, la representación gráfica no sería tan sencilla.

Describir el fundamento de este algoritmo cae fuera del alcance de este documento. Basta mencionar que la función de coste que utiliza es la denominada *función sigmoide*¹, que es la inversa de la *función logit*² que está relacionada con las probabilidades de que se produzca un suceso (por ejemplo que una flor con determinadas características pertenezca a una clase determinada).

Este algoritmo es utilizado en numerosos campos, desde la medicina a la meteorología. Una característica distintiva del mismo es que puede determinar la probabilidad de que se produzca un hecho. Si estuviéramos tratando de determinar si va a llover o no en un lugar, a partir de ciertas mediciones, el algoritmo de regresión logística podría ofrecer una respuesta binaria (sí o no), clasificando los datos como hemos visto anteriormente. Pero, además, nos ofrece la probabilidad numérica de que lo haga. De hecho, el algoritmo de clasificación binaria clasificará la muestra como perteneciente a la clase “A” si la probabilidad es mayor de 0.5, mientras que asignará la clase “B” si la probabilidad es menor. Lo interesante es que para una muestra determinada puede calcular que la probabilidad es, por ejemplo, 0.83. De ese modo, la predicción de lluvia no será simplemente de que lloverá, sino que nos informará de que lo hará con una probabilidad del 83%.

¹ “Sigmoid function” https://en.wikipedia.org/wiki/Sigmoid_function [Disponible a 23/12/2021]

² “Logit” <https://en.wikipedia.org/wiki/Logit> [Disponible a 23/12/2021]

Para evitar el sobreajuste, el algoritmo de regresión logística puede incorporar un hiperparámetro que introduzca una penalización ante valores excesivamente altos de los pesos asociados con alguna característica concreta. Dos ejemplos de esta técnica se denominan *Lasso Regression* o *Regularización L1* y *Ridge Regression* o *Regularización L2*¹.

- Máquinas de Vectores Soporte (“Support Vector Machines”: SVM).

SVM también es un algoritmo de clasificación lineal con aprendizaje supervisado. Las muestras que se encuentran más próximas al hiperplano que delimita las clases se denominan *vectores soporte* (“*support vectors*”) y, el algoritmo trata de maximizar la distancia entre ellas y dicho hiperplano.

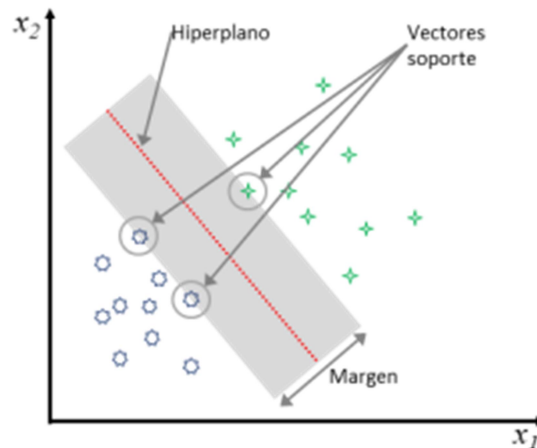


Figura 2-15 SVM: Maximización del margen entre hiperplano y vectores de soporte

En la Figura 2-15 se observa la idea fundamental del funcionamiento del algoritmo. Se trata de encontrar el hiperplano, de entre todos los posibles que maximice la distancia con los *vectores soporte*. Dicho hiperplano se denomina *hiperplano máximo de separación* (“*maximal margin hyperplane*”).

Los hiperplanos se obtienen a partir de las muestras de entrenamiento. El objeto de tratar de localizar aquel que más separado esté de ellas para evitar, en la medida de lo posible, el error de clasificación ante la presumible variabilidad de las muestras reales con que trabajará el algoritmo en su etapa de explotación.

Sin embargo, existen algunos inconvenientes. Así, si se modificara el conjunto de muestras que constituyen los vectores soporte, añadiendo por ejemplo una muestra de entrenamiento adicional ubicada en la zona gris de la Figura 2-15, el hiperplano máximo de separación se vería afectado en gran medida, lo que implica poca robustez en su definición. Además, la búsqueda de maximizar las distancias con los vectores soporte podría conducir a problemas de sobreajuste.

En la práctica, es poco habitual encontrarse con muestras claramente separables de forma lineal, en las que no existe el hiperplano de separación, así que resulta más conveniente tratar de encontrar un hiperplano más robusto y con mayor capacidad de predicción, aun a costa de que las dos clases de muestras de entrenamiento no queden perfectamente separadas. Esto nos lleva a una definición “suavizada” del hiperplano que buscamos y, por tanto, a un clasificador de márgenes suavizados (“*soft margin classifiers*”). Para determinar la magnitud de ese suavizado,

¹ Para una explicación más detallada de su funcionamiento, consultar, por ejemplo, <https://towardsdatascience.com/11-and-12-regularization-methods-ce25e7fc831c>

se introduce un hiperparámetro, llámese por ejemplo C , en el algoritmo que permitirá un cierto margen en el error de clasificación.

En la Figura 2-16 se muestran los hiperplanos de clasificación de los mismos datos que se utilizaron en la Figura 2-14 pero obtenidos con el algoritmo SVM para distintos valores de C . En este caso, un valor de C pequeño, supondrá mayor laxitud en la separación de las clases, mientras que un valor grande, supondrá lo contrario. Se observa, por ejemplo, que la frontera entre las clases identificadas con los colores rojo y azul, está mejor colocada en la figura de la izquierda que tiene un valor de C más alto. En ambos casos, sin embargo, la precisión de la clasificación, calculada como el porcentaje de muestras bien clasificadas, fue exactamente el mismo.

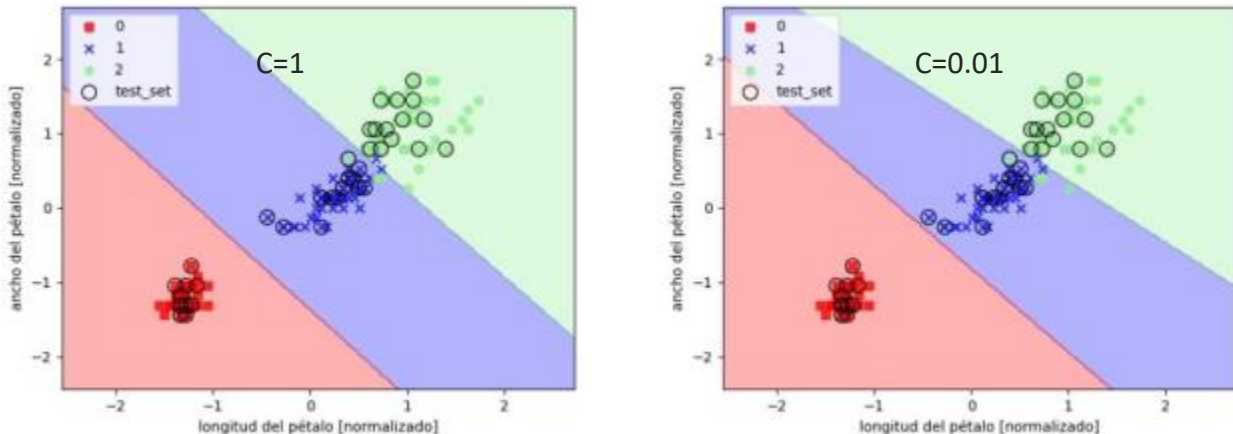


Figura 2-16 Comparación de clasificación SVM modificando los valores de C . (Figuras obtenidas mediante adaptación del código propuesto en [45])

Una de las características más reseñables del algoritmo SVM es la posibilidad de adaptarlo para clasificar muestras no linealmente separables. Un grupo de muestras con n -características supone un espacio n -dimensional en el que habrá que ubicar un hiperplano de $n-1$ dimensiones, como ya hemos visto, anteriormente. En algunas ocasiones si creamos una nueva característica, a partir de una combinación de las ya existentes, mediante ciertas operaciones matemáticas, es posible crear un nuevo espacio en el que sí sea posible una separación lineal.

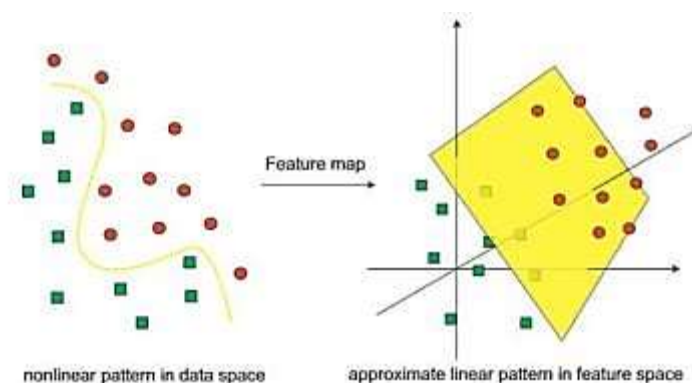


Figura 2-17 Aumentando el número de características para lograr un clasificador lineal. (procedente de [46])

En la Figura 2-17, tenemos un ejemplo en el que se ha efectuado este proceso. Existen una serie de funciones que permiten la creación de ese nuevo espacio de características sin demasiado coste computacional. El uso de una u otra función es seleccionable por el usuario, por lo que podrá utilizar aquella con la que mejor resultado obtenga. A estas funciones se les suele denominar funciones núcleo ("*kernel functions*") y, algunas de las más utilizadas son: *Polynomial Kernel* y la *Radial Basis Function (RBF)* también llamada *Gaussian Kernel*.

En la Figura 2-18 se puede ver una comparación de la clasificación de un mismo conjunto de muestras utilizando SVM con el kernel lineal, que se corresponde con el algoritmo básico, con el kernel polinómico y con el kernel RBF.

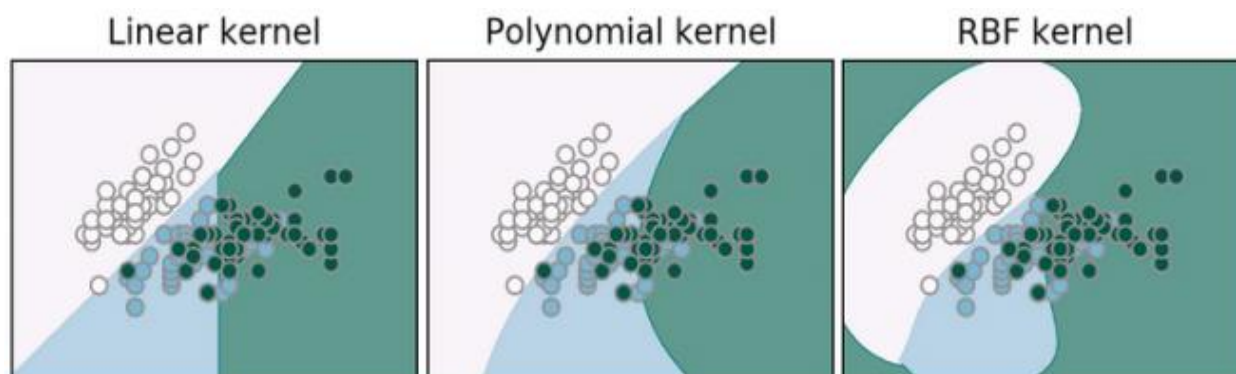


Figura 2-18 Comparación de la clasificación de muestras con diversos kernel. (Extraída de [49]).

- Árboles de decisión (“decision trees”).

Un árbol de decisión es como un diagrama de flujo en el que se va recorriendo un camino desde la raíz a las hojas, en función del resultado de las decisiones que se toman en cada nodo. El árbol comienza con un único nodo y se va ramificando, según los resultados de las posibles decisiones. Si para cada nodo únicamente existen dos alternativas se trata de un árbol binario. En ML, las decisiones que se van tomando en cada nodo hacen referencia a los valores de alguna de las características, como, por ejemplo: si la característica x_i es menor que 4, entonces continúa por la rama de la izquierda, si no, continúa por la de la derecha. En los últimos nodos, se asigna una clase a cada muestra evaluada.

Evidentemente la dificultad está en hacer las preguntas correctas en cada nodo. Para lograrlo se introducen algunos conceptos como la *ganancia de información*, la *entropía* o la *impureza Gini*.

La *entropía* mide la diversidad de clases que hay en un conjunto de muestras. Si todas pertenecen a la misma clase, la *entropía* es cero, si todas las clases están igualmente representadas en el conjunto, la *entropía* es 1. La expresión para cuantificar la *entropía* de un grupo de muestras S en el que únicamente hay dos clases A y B es:

$$E(S) = -\frac{|A|}{|S|} \log_2 |A| - \frac{|B|}{|S|} \log_2 |B|$$

Donde la clase A tiene $|A|$ elementos, la clase B tiene $|B|$ elementos y el conjunto de muestras S tiene $|S|$ elementos. De este modo, si una clase S_1 tiene 14 elementos y hay 10 de una clase y 4 de otra tendremos que el valor de entropía es $E(S_1)=0.863$. En el caso de que el número de elementos en otra clase S_2 hubiera sido 8 y 6, el valor de entropía hubiera sido mayor, concretamente $E(S_2)=0.985$. Resulta relativamente sencillo ampliar la expresión para contemplar un número de clases mayor.

Así pues, el objetivo es dividir un conjunto de muestras aplicando una condición sobre alguna de sus características de modo que la suma de la *entropía* de los subconjuntos resultantes sea la menor posible. Suponiendo un árbol de decisión binario, si sobre un nodo padre S se aplica una condición c y se obtienen dos nodos hijos P y N , cada uno con un cierto número de elementos de las clases A y B , la *ganancia de información* que se obtiene en esa división se puede cuantificar:

$$IG(S, c) = E(S) - \frac{|P|}{|S|} E(P) - \frac{|N|}{|S|} E(N)$$

Donde los cocientes $\frac{|P|}{|S|}$ y $\frac{|N|}{|S|}$ se pueden interpretar como las probabilidades de que la condición c conduzca al nodo hijo P o N , respectivamente.

Así pues, la *ganancia de información* sobre un nodo S al que se aplica la condición c , $IG(S, c)$, es la *entropía* del nodo padre, S , menos la *entropía* de los dos nodos hijos ponderada por la probabilidad de que la condición c , conduzca a cada uno de ellos. La condición c que nos proporcione la mayor *ganancia de información*, será la que haya que aplicar en cada bifurcación del árbol de decisión. En realidad, la *entropía* es una forma de medir lo heterogéneo que es un conjunto de muestras. Pero no es la única forma de medirlo. La *impureza Gini* es otra medida que se puede utilizar para calcular la *ganancia de información*. En este caso, la expresión es la siguiente:

$$G(S) = 1 - \left(\frac{|A|}{|S|} \right)^2 - \left(\frac{|B|}{|S|} \right)^2$$

Al igual que en la expresión de la *entropía*, los cocientes $\frac{|A|}{|S|}$ y $\frac{|B|}{|S|}$ se pueden interpretar como las probabilidades de que en un conjunto, existan muestras de la clase A y B , respectivamente.

Utilizando el mismo ejemplo de clases S_1 y S_2 que utilizamos para calcular la *entropía*, podemos calcular la *impureza Gini*, obteniendo $G(S_1)=0.408$ y $G(S_2)=0.489$. Al igual que la *entropía*, valores más altos de la *impureza Gini* caracterizan muestras más heterogéneas. Así pues, si en la expresión de la *ganancia de información*, sustituimos el cálculo de la *entropía* por el de la *impureza Gini*, conseguiremos el mismo propósito.

En la Figura 2-19 tenemos un ejemplo de clasificación mediante árbol de decisión. Es fácil ver que la línea horizontal que hace de frontera para la clase de color rojo, se corresponde aproximadamente con la decisión: “¿es el ancho del pétalo menor que -0.6?”. En un espacio de características bidimensional, las fronteras de separación siempre se corresponden con líneas paralelas a los ejes.

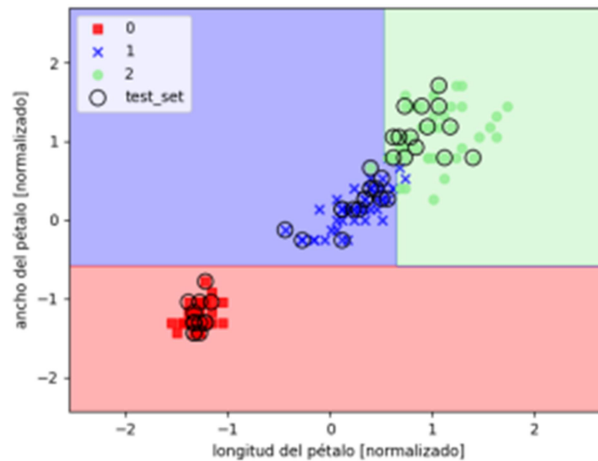


Figura 2-19 Ejemplo de clasificación mediante árbol de decisión (obtenido mediante adaptación del código propuesto en [45])

Uno de los hiperparámetros que hay que definir para los algoritmos basados en árboles de decisión es el *máximo número de subdivisiones* del árbol. En la Figura 2-19 se ha limitado a 4 niveles. Por eso hay cuatro rectas paralelas a los ejes, dos horizontales, una de las cuales bastante corta, y dos verticales. Probablemente, cuanto mayor sea el número de niveles, mejor clasificará

las muestras de entrenamiento. Sin embargo, hay que tener cuidado de no caer en el sobreajuste al excederse en el número de niveles.

El bosque aleatorio (“*random forest*”) es una forma particular de utilizar los árboles de decisión. La idea es dividir el conjunto de entrenamiento en varios subconjuntos que se utilizarán un algoritmo de decisión independiente. Posteriormente, en explotación, cada muestra trataremos de clasificarla utilizando los diversos árboles de decisión y obtendremos una respuesta que, quizás, no sea unánime. Si no lo es, podremos decidir a qué clase pertenece finalmente la muestra utilizando la técnica de voto mayoritario (“*majority voting*”) –la clase a la que ha sido asignada la muestra por el mayor número de árboles–, u otras formas más refinadas que tengan en cuenta, un valor estimado en cada árbol de la probabilidad de que una muestra pertenezca a una u otra clase (“*soft voting*”).

La ventaja de esta técnica es que los posibles errores de sobreajuste que podamos cometer en algunos de los árboles de decisión, se compensarán unos con otros, dando lugar a un modelo bastante robusto. Cuanto mayor sea el número de árboles, más robusto, pero para que las predicciones de cada árbol sean razonables, se precisa un mínimo número de muestras. Así que hay que encontrar un compromiso. Del mismo modo, para intentar lograr que el subconjunto de muestras con que se entrene cada árbol individual sea representativo, lo más habitual es subdividir el conjunto total de muestras de forma aleatoria. Además, puesto que el entrenamiento de cada grupo de muestras en cada árbol es independiente, es posible asignar núcleos diferentes del procesador para que trabajen en paralelo, con lo que se logra reducir el coste computacional.

En la Figura 2-20 se han utilizado cuatro árboles distintos y se ha limitado el nivel de crecimiento de cada uno de modo que, en cada nivel, se produzca una reducción en la impureza de 0.1.

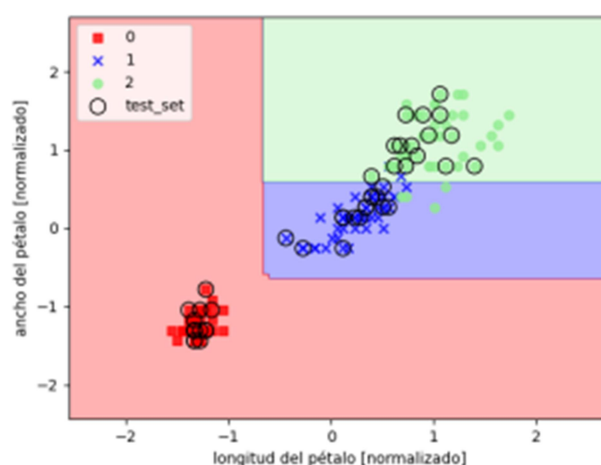


Figura 2-20 Ejemplo de clasificación mediante bosque aleatorio (obtenido mediante adaptación del código propuesto en [45])

- Agrupamiento de k -medias (“ k -means clustering”).

Este tipo de algoritmo se diferencia de los anteriores en que es no-supervisado. Es decir, a priori, no se conocen las clases correctas de los datos de entrenamiento. El algoritmo, valorando todas las características, deberá organizar las muestras en un cierto número de clases. La elección del número de clases k , a priori, es el principal inconveniente de este algoritmo.

En k -medias, se organizarán las muestras en k clases. Cada clase está suele estar representada por su *centroide*, que es la media de los valores de cada una de las características de las muestras de la clase. Inicialmente, se escogerán k muestras de forma aleatoria como centroides, posteriormente se van procesando el resto de las muestras asignándolas al centroide más cercano, que se recalcula nuevamente. Para calcular las distancias entre las muestras y el centroide se

suele utilizar la distancia euclídea al cuadrado. La distancia entre la muestra x y la muestra y , en un espacio con n características, se obtiene mediante:

$$\text{dist}(x, y)^2 = \sum_{i=1}^n (x_i - y_i)^2$$

En definitiva, k -medias es un problema que trata de minimizar la función de coste definida como la suma de los cuadrados de los errores de distancia (SCED) entre los puntos que son miembros de las clases y su centroide (μ). Con k clases, esta función se puede expresar como sigue:

$$\text{SCED} = \sum_{i=1}^k \sum_{x_j \in S_i} \text{dist}(x_j, \mu_i)^2$$

Donde el conjunto S es el que engloba a las muestras x_j que pertenecen a una de las k clases.

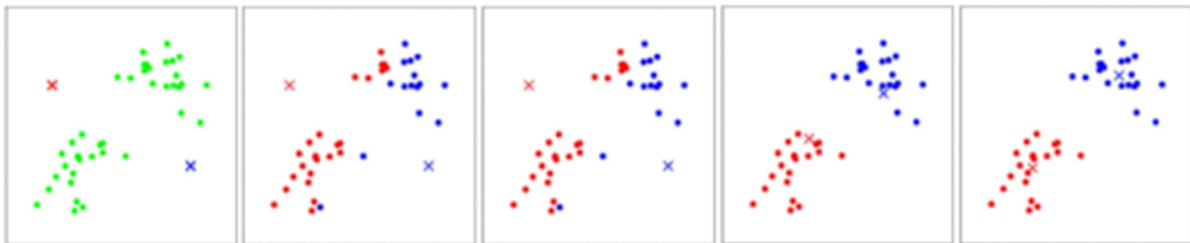


Figura 2-21 Ejemplo de iteraciones de k-means (obtenido de <https://stanford.edu/~cpiech/cs221/handouts/kmeans.html>)

En la Figura 2-21 se observa el proceso, de izquierda a derecha, de clasificación mediante k -medias. En primer lugar, se seleccionan dos centroides, identificados por las marcas en forma de aspa. Seguidamente todas las muestras se asignan al centroide más cercano. En la siguiente iteración, que se corresponde con la cuarta figura comenzando por la izquierda, se recalculan los centroides, y de nuevo, se reclasifican las muestras.

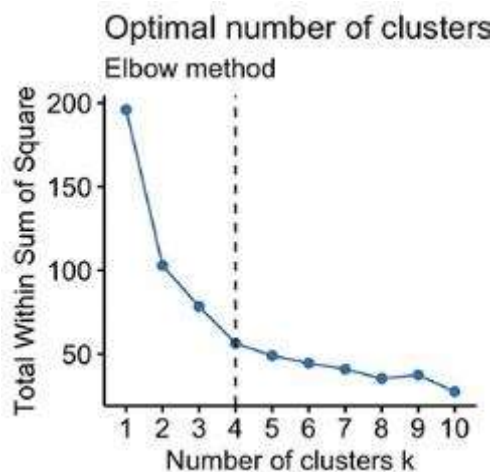


Figura 2-22 Determinación del número óptimo de clases (obtenida de <https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/>)

Es posible optimizar la velocidad de convergencia del algoritmo escogiendo las muestras que actuarán de centroides iniciales con algún criterio. Por ejemplo, escoger una muestra aleatoria inicial y, para elegir la segunda, se escoge aquella que esté lo más alejada de la anterior. Para escoger los sucesivos centroides se elegirán las muestras que maximicen la suma de las distancias de separación a los anteriores centroides.

Otra práctica habitual es ejecutar el algoritmo varias veces, con diversos centroides iniciales, y escoger la clasificación que tenga el menor valor de SCED. De ese modo evitamos una mala selección inicial de los centroides.

Un hiperparámetro que se utiliza comúnmente en este algoritmo es el que establece el número máximo de iteraciones, es decir, cuántas veces actualizamos la posición de los centroides y recalculamos la función SCED. También es posible detenerlo cuando la posición de los centroides, entre una iteración y la siguiente, difiera menos de un valor umbral.

Como se ha mencionado, uno de los principales problemas de los algoritmos no supervisados es la determinación, a priori, de la cantidad de clases en que se clasificarán las muestras, es decir, la determinación del parámetro k . Si ejecutamos el algoritmo sobre un mismo conjunto de muestras y realizamos una gráfica que represente en el eje vertical el valor de SCED obtenido en la última iteración de un algoritmo de k -medias, y el valor del hiperparámetro k en el eje horizontal, obtendremos típicamente una gráfica como la de la Figura 2-22.

Sobre ella es posible aplicar el criterio del “*codo*”, que consiste en escoger el valor de k en el que la pendiente comience a suavizarse. Se observa que, para valores pequeños de k , el valor de SCED disminuye rápidamente, pero, a partir de cierto valor, lo hace de forma más suave. Ese punto, en el que la pendiente de la gráfica pasa de tener un valor grande a otro más reducido es el denominado “*codo*”. La selección de un adecuado valor de k es crítica para realizar una clasificación correcta y, aunque siempre es posible elegir un valor mayor para reducir el SCDE total, corremos el riesgo de caer en el sobreajuste que veíamos en los algoritmos supervisados. Demasiadas clases pueden funcionar bien con un determinado conjunto de muestras de entrenamiento, pero cuando tratemos de clasificar muestras no vistas, la eficacia disminuirá drásticamente.

Otra medida para determinar si el valor de k escogido es bueno es el *coeficiente de silueta* (“*silhouette coefficient*”). El coeficiente de silueta (SC) es posible definirlo del siguiente modo:

$$SC(x) = \frac{b(x) - a(x)}{\max(a(x), b(x))}$$

Donde $a(x)$ es la distancia promedio entre una muestra y el resto de las de su clase y $b(x)$ es la distancia promedio entre dicha muestra y las que conforman la siguiente clase más cercana.

Al dividir la diferencia entre ambos valores por el valor máximo de ambos, el resultado variará entre -1 y 1 . Los mejores agrupamientos estarán muy cohesionados, con un valor bajo de $a(x)$, y separados del resto de agrupamientos, con un valor alto de $b(x)$. De modo que el valor resultante de SC estará próximo a 1 . Cuando se obtienen los valores de SC para cada una de las k clases, es posible compararlo con los obtenidos con otro valor de k , para determinar qué valor es el más adecuado.

Algunas variantes del algoritmo de k -medias realizan una clasificación “*suave*” (“*soft*” o “*fuzzy*”) en la que se asigna una medida de la probabilidad de que una determinada muestra pertenezca a una clase u otra. Esta probabilidad es directamente proporcional a la función de la distancia a su centroide. Con estas variantes se puede conseguir una convergencia más rápida, pero en contrapartida tienen un coste mayor en cada iteración. Un método que utiliza esta aproximación es el FCM (“*Fuzzy-C-means*”)¹.

- Clasificador Bayesiano Ingenuo (“Naive Bayesian Classifier”).

Se trata de un clasificador que se basa en el teorema de Bayes de probabilidad condicionada. Para su aplicación, hay que asumir que las variables que se analizan son independientes entre sí.

¹ El algoritmo FCM fue descrito por Bezdek, J. en *Pattern recognition with fuzzy objective function algorithms*, publicado por Springer en 1981.

Puesto que esto no siempre es estrictamente así, tal asunción se considera ingenua, y es la razón del nombre del clasificador.

La expresión del teorema para dos únicas variables es la siguiente:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Donde $P(X|Y)$ denota la probabilidad de que suceda X si ya ha sucedido Y .

Para explicar el funcionamiento, recurriremos al típico ejemplo del detector de spam. En primer lugar, tendremos que entrenar al clasificador indicándole durante un tiempo qué correos son spam ($P(A)$) y cuáles no. El clasificador puede contabilizar la frecuencia de aparición de determinadas palabras en los correos y obtener su probabilidad ($P(B)$), y también la ocurrencia de ciertas palabras en aquellos correos que son spam, es decir que aparezca una determinada palabra cuando ya se sabe que el correo es spam ($P(B|A)$). A partir de aquí, podemos calcular la probabilidad de que un nuevo correo que contenga una palabra determinada es spam o no ($P(A|B)$).

Puesto que se considera ingenuamente que la aparición de las palabras es independiente, es decir, que el hecho de que aparezca una no incrementa la posibilidad de que aparezca otra concreta, es posible multiplicar las probabilidades. De este modo, podemos averiguar la probabilidad de que un correo sea spam si aparecen ciertas palabras de una lista.

En la Figura 2-23 se ha aplicado este clasificador sobre el mismo conjunto de datos de los algoritmos anteriores.

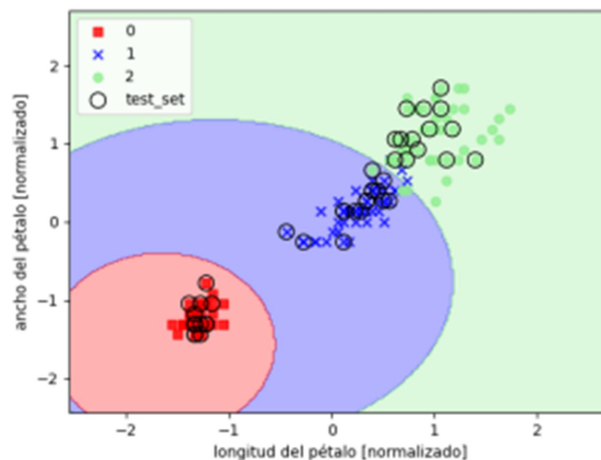


Figura 2-23 Ejemplo de aplicación de Naive Bayes.

Uno de los inconvenientes de este clasificador es que, si en los datos de entrenamiento nunca aparece una determinada palabra, cuando tengamos que predecir si un correo es spam o no, en el que sí aparezca dicha palabra, al considerar que su probabilidad de aparición es nula y multiplicarse por el resto, siempre se obtendrá un valor nulo. Para solucionar este inconveniente se suele introducir un factor de suavizado que evite esos valores nulos. Los más utilizados son *Lindstone smoothing* y *Laplace smoothing*. Más información sobre este clasificador y su aplicación al análisis de textos, se puede encontrar en [27].

- Latent Dirichlet Allocation (LDA).

LDA es un método para la identificación temática (“topic modeling”) automática y no supervisada. El objetivo que se persigue es si ejecutamos LDA con un texto, obtengamos información del tema sobre el que trata. Los temas están definidos por la proporción de ocurrencia de determinadas palabras. Así, si en un texto aparecen con mucha frecuencia las

palabras estrella, galaxia y órbita, se podrá inferir que el tema está relacionado con la astronomía. Los documentos se consideran simplemente contenedores de palabras (“bag of words”) en los que el orden en que aparezcan no es relevante. Al mismo tiempo, los documentos de la colección se caracterizan por una probabilidad de pertenecer a un determinado tema, siendo posible que, en algún caso, esa probabilidad sea similar para más de un tema.

La Figura 2-24 representa el mecanismo conceptual de elaboración de un documento. A la izquierda tenemos un conjunto de temas, identificados por colores y que contienen un listado de palabras con una cierta probabilidad. Lo primero que hay que determinar es qué probabilidad de cada uno de esos temas tendrá el documento. En el ejemplo se ha representado en la parte derecha mediante un histograma. Cuanta más probabilidad tenga un tema, más palabras de su lista aparecerán en el documento. En este proceso, el documento final, estará compuesto únicamente por un conjunto de palabras desordenadas, pero es suficiente para identificarlo como perteneciente a un tema determinado.

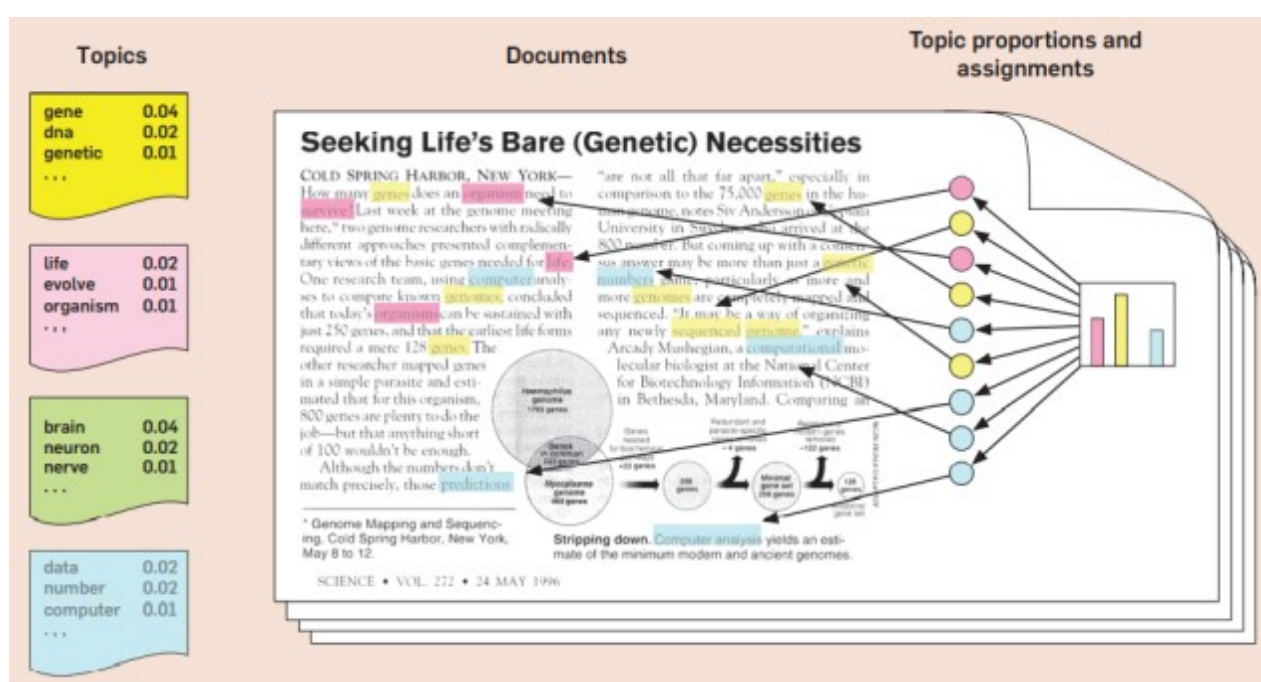


Figura 2-24 Funcionamiento conceptual de LDA (obtenido de [47])

El proceso de generación del documento descrito en el párrafo anterior se aplica a la inversa, mediante LDA, para establecer la pertenencia de un documento a un tema concreto. Para el funcionamiento de LDA, por tanto, se necesitan una colección de documentos y un número de temas prefijado de antemano como hiperparámetro. El algoritmo será capaz de identificar los grupos de palabras más significativos de cada uno de los temas, aún sin tener conciencia de su significado, ni utilizar ningún tipo de diccionario.

La denominación de LDA, hace referencia a la distribución de Dirichlet que es la función de distribución de probabilidad que se asume que siguen, tanto la aparición de palabras dentro de un tema, como los temas con los que se puede relacionar un determinado documento. La palabra “latent” que se podría traducir por subyacente u oculta, recalca el hecho de que el algoritmo deberá obtener las probabilidades a partir de una única variable directamente observable, que es el número de palabras que se pueden contar dentro de los documentos que pertenecen a una colección. Una explicación completa de LDA se puede obtener en [28].

2.4.1.3 Utilización de métodos de análisis de texto en la literatura académica.

Para ejemplificar el uso de algunos de los métodos reseñados anteriormente, podemos recurrir a la literatura académica relacionada.

En [29], los autores utilizan un clasificador bayesiano ingenuo con un conjunto de características que incluye una combinación de LIW's, nombres de ciudades o países, *#hashtags* y *@menciones*. Estas características son recogidas automáticamente de una muestra de más de nueve millones de tweets, generados por más de setecientos mil usuarios.

Los autores consideran que los hashtags y las menciones pueden relacionarse con la ubicación de los usuarios y escogen los diez mil más utilizados, de uno y otro tipo, en su conjunto de muestras para utilizarlos como parte de su vector de características. La idea es que, por ejemplo, el hashtag “*#bemorebulldog*”, puede relacionarse con un equipo de fútbol de la liga australiana ubicado en la ciudad de Victoria, mientras que en “*@7NewMelbourne*” se menciona una cadena de noticias de una ciudad concreta.

En general, los métodos que utilizan el análisis de texto para geolocalizar a un usuario, consideran todos sus tweets como parte de un único documento del que obtener las características, que fundamentalmente son la cuenta de uso de un conjunto, considerablemente grande, de palabras concretas. La dificultad está, por tanto, en la elección de ese conjunto de palabras. En el caso de los hashtags y las menciones, se seleccionarán automáticamente las más utilizadas, pero en el caso de los LIW, mencionados en el punto 2.4.1.1, son mucho más complejos de obtener.

En [30], se utiliza una red neuronal, en lugar de otro tipo de clasificador. En este documento no se ha tratado el funcionamiento de las redes neuronales, pero el lector puede imaginar que una red neuronal es como un conjunto de clasificadores simples, o neuronas, dispuestos en cascada con diversos niveles. Baste mencionar, por ejemplo, que una de las funciones de activación que emplearon en dichas neuronas fue la función sigmoide, la misma que se utiliza en el clasificador basado en regresión logística. Para dar una idea de la complejidad del sistema utilizado, es suficiente indicar que los autores utilizaron diversas redes neuronales con un número de niveles entre 3 y 5, y entre 128 y 4096 neuronas por nivel.

Como conjunto de características, se utilizó un conjunto de palabras (“*bag-of-words*”) con las cincuenta mil más frecuentes, extraídas de los conjuntos de muestras, y seleccionadas por sus valores de TF-IDF.

En [31], los autores filtran los tweets en la etapa de preproceso desechando aquellos que no incluyen ciertas palabras que consideran indicativas de localización. Para seleccionar dichas palabras, utilizan, en el conjunto de entrenamiento, técnicas como la valoración TDF-IDF y PoS. Si en la frase se detectan frases en pasado o futuro, utilizando las marcas generadas por PoS, las posibles ubicaciones que se mencionen no se considerarán puesto que no se refieren al lugar desde el que se emite el tweet. Por ejemplo, en la frase “*I’m going to MoMA*” se utilizará PoS para detectar una frase que indica futuro y, por tanto, la ubicación sugerida por “*MoMA*” (Museo de arte moderno de Nueva York), no se considerará. También se utilizan las probabilidades de Naive Bayes para seleccionar la ubicación del domicilio del usuario basándose en los diversos tweets geolocalizados que ha publicado. En función de dicha probabilidad, introducen un factor de ponderación δ para incluirlo en el conjunto de muestras de entrenamiento con mayor o menor peso. Los clasificadores de árbol de decisión y SVM con kernel RBF se utilizan para la predicción posterior y han sido entrenados a partir de un cierto número de tweets procedentes de localizaciones conocidas y donde los vectores de características incluían la aparición de ciertas palabras.

En [32] se obtienen los temas (“*topics*”) de los tweets de un usuario publicado en un período de tiempo limitado, en el que se estima que su ubicación es próxima a la deducida por una

publicación relacionada con FourSquare. De este modo obtienen vectores de datos con los temas de los tweets de los usuarios que visitaron alguna localización. Esta información se utiliza posteriormente para entrenar clasificadores basados en Naive Bayes, Árboles de Decisión¹ o los k-Vecinos más próximos.

En [33], después de un preprocesado de los textos en los que se lleva a cabo “*tokenization*”, separación de frases, “*lemmatization*” y eliminación de “*stopwords*”, se utiliza una variante de *Latent Dirichlet Allocation*, denominada SLDA. En esta variante es posible el aprendizaje en lotes de datos de entrada que se pueden producir en cualquier momento, de modo que puede actualizar el modelo en períodos intercalados con otros de explotación-predicción. Al comparar la similitud de los “*topic*” en los tweets de entrenamiento con los de las nuevas muestras, SLDA es capaz de estimar la localización.

2.4.2 Métodos basados en el análisis de redes de usuarios

Estos métodos se basan en las relaciones entre usuarios de Twitter. Los usuarios que se suscriben a las publicaciones de otros usuarios recibirán notificaciones cuando estos últimos publiquen algún tweet. Se dice entonces que los primeros siguen a los segundos. Estas relaciones también pueden ser recíprocas, cuando dos usuarios se siguen uno al otro. La representación de estas relaciones se puede modelar mediante un grafo similar al mostrado en la Figura 2-25, donde para distinguir los nodos, o vértices, de las aristas, se han utilizado números de doble dígito para los primeros y de un único dígito para las segundas.

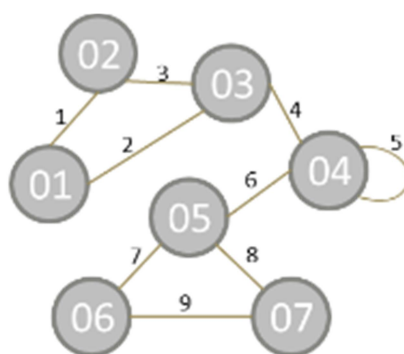


Figura 2-25 Ejemplo de grafo.

La representación gráfica a la que estamos acostumbrados los humanos, debe ser transformada para que pueda ser tratada por las computadoras. Las formas más habituales de representación de la información contenida en los grafos son las que aparecen en la Figura 2-26.

Conjuntos	Matriz de Adyacencia	Matriz de Incidencia	Vector de Grados	Lista de Adyacencia																																																																																																																																																														
$V=\{1,2,3,4,5,6,7\}$ $A=\{\{2,3\}, \{1,3\}, \{1,2,4\}, \{3,4,5\}, \{4,6,7\}, \{5,7\}, \{5,6\}\}$	<table><tr><th></th><th>01</th><th>02</th><th>03</th><th>04</th><th>05</th><th>06</th><th>07</th></tr><tr><th>01</th><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>02</th><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>03</th><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><th>04</th><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr><tr><th>05</th><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr><tr><th>06</th><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><th>07</th><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr></table>		01	02	03	04	05	06	07	01	0	1	1	0	0	0	0	02	1	0	1	0	0	0	0	03	1	1	0	1	0	0	0	04	0	0	1	1	1	0	0	05	0	0	0	1	0	1	1	06	0	0	0	0	1	0	1	07	0	0	0	0	1	1	0	<table><tr><th></th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th></tr><tr><th>01</th><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>02</th><td>1</td><td>0</td><td>3</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>03</th><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>04</th><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><th>05</th><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr><tr><th>06</th><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><th>07</th><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>		1	2	3	4	5	6	7	8	9	01	1	1	0	0	0	0	0	0	0	02	1	0	3	0	0	0	0	0	0	03	0	1	1	1	0	0	0	0	0	04	0	0	0	1	1	1	0	0	0	05	0	0	0	0	0	1	1	1	0	06	0	0	0	0	0	0	1	0	1	07	0	0	0	0	0	0	0	1	1	<table><tr><th>01</th><th>02</th><th>03</th><th>04</th><th>05</th><th>06</th><th>07</th></tr><tr><td>2</td><td>2</td><td>3</td><td>3</td><td>3</td><td>2</td><td>2</td></tr></table>	01	02	03	04	05	06	07	2	2	3	3	3	2	2	$\{\{02,03\}, \{03\}, \{04\}, \{04, 05\}, \{06, 07\}, \{07\}\}$
	01	02	03	04	05	06	07																																																																																																																																																											
01	0	1	1	0	0	0	0																																																																																																																																																											
02	1	0	1	0	0	0	0																																																																																																																																																											
03	1	1	0	1	0	0	0																																																																																																																																																											
04	0	0	1	1	1	0	0																																																																																																																																																											
05	0	0	0	1	0	1	1																																																																																																																																																											
06	0	0	0	0	1	0	1																																																																																																																																																											
07	0	0	0	0	1	1	0																																																																																																																																																											
	1	2	3	4	5	6	7	8	9																																																																																																																																																									
01	1	1	0	0	0	0	0	0	0																																																																																																																																																									
02	1	0	3	0	0	0	0	0	0																																																																																																																																																									
03	0	1	1	1	0	0	0	0	0																																																																																																																																																									
04	0	0	0	1	1	1	0	0	0																																																																																																																																																									
05	0	0	0	0	0	1	1	1	0																																																																																																																																																									
06	0	0	0	0	0	0	1	0	1																																																																																																																																																									
07	0	0	0	0	0	0	0	1	1																																																																																																																																																									
01	02	03	04	05	06	07																																																																																																																																																												
2	2	3	3	3	2	2																																																																																																																																																												

Figura 2-26 Representación de la información de un grafo no dirigido.

¹ Concretamente el denominado “C4.5”, cuya descripción básica se puede encontrar en “C4.5” <https://es.wikipedia.org/wiki/C4.5> [Disponible a 23/12/2021]

- Conjuntos: El grafo está representado por el conjunto de sus vértices V y el de sus aristas A .
- Matriz de Adyacencia: Tiene tantas filas y columnas como nodos tiene el grafo. Es decir, es una matriz cuadrada de dimensión igual al número de nodos. Si un nodo no tiene arista que lo una a otro, en la posición correspondiente a la fila y columna que identifiquen a dicho nodo en la matriz aparecerá un 0 . Si sí que la tiene, aparecerá un 1 . La matriz de adyacencia es simétrica respecto a la diagonal principal.
- Matriz de Incidencia: Tiene tantas filas como nodos y tantas columnas como aristas. Para el nodo especificado en cada fila, se marcará la columna con un 1 , en el caso de que de tal nodo esté conectado con otras mediante la arista correspondiente de dicha columna. Así, por ejemplo, puesto que del nodo “04” salen las aristas 4, 5 y 6, las columnas correspondientes a tales aristas tendrán un valor de 1 .
- Vector de grados: Con tantas columnas como nodos, proporciona información únicamente sobre el número de aristas que salen del nodo correspondiente. Del nodo “03”, salen 3 aristas. A dicho número se denomina grado.
- Lista de Adyacencia: Es una lista que contiene listas de los nodos conectados con el nodo correspondiente. Así, por ejemplo, el primer elemento de la lista es $\{02, 03\}$, que indica que el nodo “01” tiene aristas con esos nodos. Para que la información no sea redundante, se obvia incluir las aristas ya descritas. Por esa razón, el segundo elemento de la lista, que corresponde al nodo “02”, únicamente se incluye una lista que indica que hay una arista con el nodo $\{03\}$. La arista con el nodo “01” no se incluye, puesto que ya se hizo anteriormente.

Los grafos dirigidos, en los que las aristas tienen un sentido, se representan con algunas variaciones.

La existencia de las relaciones de amistad, o seguimiento, entre usuarios de Twitter se puede utilizar, bajo determinados supuestos, para inferir las localizaciones de usuarios relacionados entre sí. Algunos de los métodos utilizados basados en estos grafos de relaciones se describen en los siguientes apartados.

2.4.2.1 Método de cálculo de distancias.

La versión más general de este método la podemos encontrar en [34], en la que los autores analizan las relaciones de amistad en Facebook. A partir de 3.5 millones de usuarios con una media de 10 amigos, obtiene un grafo con 30.6 millones de aristas con localizaciones conocidas. En su estudio tratan de responder a la pregunta de si es más probable que los amigos de un usuario vivan cerca del mismo, obteniendo finalmente la gráfica de la Figura 2-27.

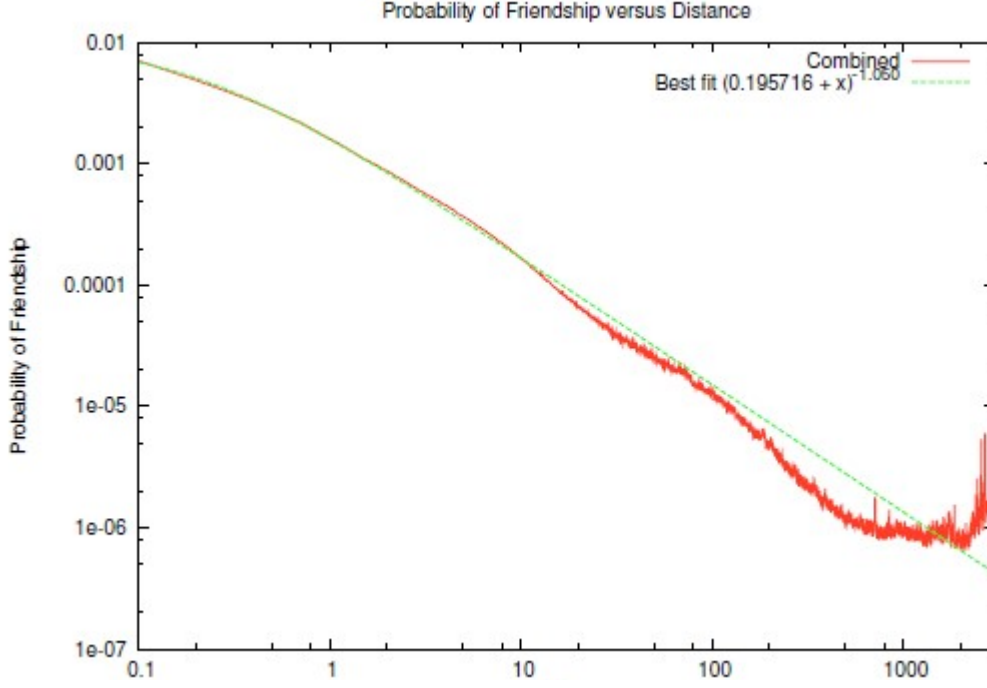


Figura 2-27 Probabilidad de amistad en función de la distancia (obtenido de [34])

La gráfica, muestra que es más probable que dos usuarios sean amigos si viven cerca y, además, permite obtener la ecuación de la función, arriba a la derecha de la figura, revelando que la amistad depende inversamente de la distancia. Analizando los datos con mayor detalle, llegan a observar que la probabilidad de amistad es mayor para regiones menos densamente pobladas, considerando distancias de hasta 25 millas.

A partir de los datos anteriores, los autores pueden predecir la localización de que un usuario u , con una determinada red de amistades, viva en una localización $l_u = (lat, long)$. En primer lugar, para cada amigo v de u , podemos saber que la probabilidad de que vivan a determinada distancia uno del otro, obtenida empíricamente a partir de la gráfica de la Figura 2-27, es:

$$p(l_u - l_v) = 0.0019(|l_u - l_v| + 0.196)^{-1.05}$$

En segundo lugar, podemos dividir los nodos del grafo entre aquellos que son amigos de u , y por tanto tienen una arista que los une, y los nodos que no están enlazados. Al primer grupo lo denominaremos E .

Considerando ahora las distancias con el resto de usuarios del grafo, tanto los que pertenecen a E como los que no pertenecen a dicho conjunto, podemos calcular la probabilidad de una determinada localización l_u como:

$$\prod_{(u,v) \in E} p(l_u - l_v) \prod_{(u,v) \notin E} 1 - p(l_u - l_v)$$

Con el fin de optimizar el cálculo, los autores consideran que el segundo productorio de la expresión se puede calcular como una constante dependiente de la localización, pero independiente de u , por lo que puede ser obtenida previamente, y una única vez, para todas las localizaciones que se consideren (estados, ciudades, etc.). Cada localización, caracterizada por una densidad de población tendrá un valor distinto. En general, las ciudades grandes tendrán un valor de dicho productorio más pequeño que las zonas rurales.

Por último, y a partir de las expresiones anteriores, se obtienen las probabilidades para las localizaciones de los amigos de u y se selecciona de entre todas ellas, la más alta. Suponiendo que un usuario tiene k amigos, el coste del algoritmo es $O(k^2)$. Por otro lado, los autores

recomiendan obtener los productos de las probabilidades sumando logaritmos en lugar de realizar multiplicaciones directamente para evitar el desbordamiento por abajo en las operaciones (“underflow”).

En [35] los autores utilizan el mismo método que se acaba de describir, aplicándolo a usuarios de Twitter. Destacan, inicialmente, que las particularidades de ambas redes sociales hacen menos probable que los usuarios de Twitter hagan pública su ubicación real. Sin embargo, lo más significativo de este estudio es la incorporación, en las predicciones de la ubicación, de la relevancia de los vínculos de amistad/seguimiento entre los usuarios. Analizando diversos factores determina qué peso debería tener cada una de las aristas en función de la fuerza del vínculo (“*tie strength*”) de amistad/seguimiento.

Analizando los tipos de relaciones y las distancias entre más de 1.75 millones de usuarios de Twitter, llegan a algunas conclusiones. Así, la probabilidad de cercanía es mayor entre usuarios que verifican algunas de las siguientes condiciones.

- Los usuarios tienen una relación de seguimiento/amistad recíproca.
- La probabilidad es casi el doble para usuarios que tienen entre 10 y 99 seguidores, frente a los que tienen entre 1000 y 9999.
- Si el usuario tiene varios amigos con la misma ubicación.
- Si los amigos/seguidores del usuario tienen un valor de LCR¹ elevado.

Por otro lado, también verifican que:

- Los usuarios que simplemente mencionan a otro, de forma no recíproca, no tienen más probabilidad de compartir ubicación.
- Los usuarios que indican en su perfil que viven en ciertas localizaciones, lo hacen con un error menor que los que indican que viven en otras, lo que permite dar más credibilidad, o peso, a esos datos. Es decir, los usuarios con estas ubicaciones, habitualmente más precisas, son más útiles a la hora de predecir las de sus contactos. A las distintas posibles ubicaciones se les asigna un valor que las caracteriza, la mediana del error de localización (MLE). Aquellos usuarios que manifiestan su ubicación, y esta tiene un valor bajo de MLE, viven generalmente cerca de sus contactos.

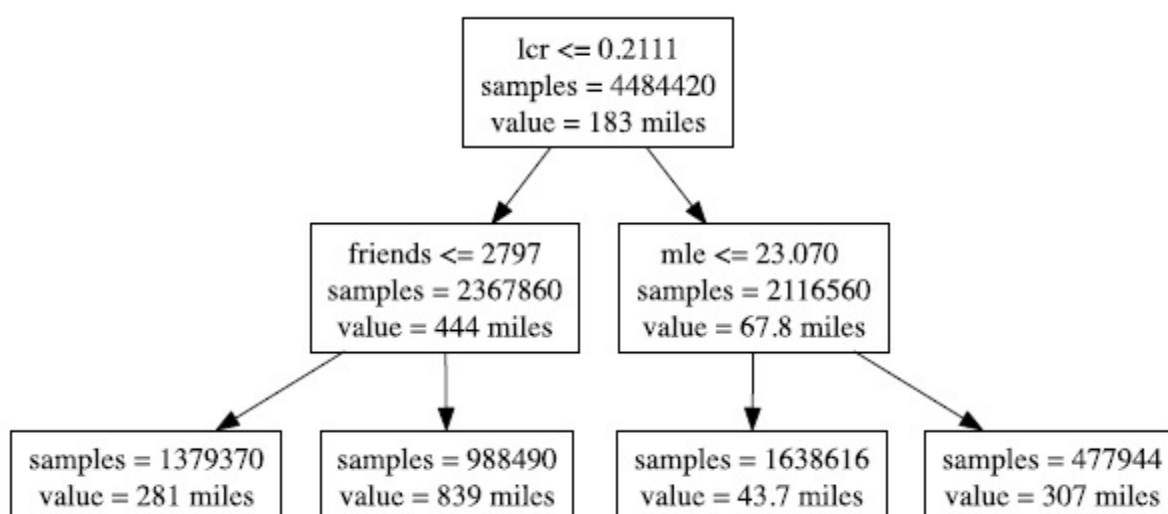


Figura 2-28 Niveles iniciales del árbol de decisión que se utilizará para ponderar los contactos de un usuario en el grafo de relaciones (obtenido de [35]).

LCR o relación de contactos locales (Local Contact Ratio) es el número de amigos/seguidores ubicados a menos de 25 millas, partido por el total de amigos/seguidores. Este índice, se calcula para los amigos/seguidores del usuario a valorar, lo que significa una búsqueda de dos saltos en el grafo (amigos de los amigos de un usuario).

A partir de estos datos, los autores obtienen un árbol de decisión que asignará una precisión en la ubicación que aportan los contactos que se puedan clasificar en una u otra categoría. En la Figura 2-28, se observan los niveles iniciales del árbol obtenido. Así, por ejemplo, un contacto que tenga un LCR de 0.2 y un número de amigos de 3000, sería clasificado en la segunda hoja del árbol comenzando por su izquierda asignándole una distancia de 839 millas, mientras que otro con LCR de 0.3 y una localización con MLE de 22, le correspondería una distancia de 43.7 millas. La arista del grafo correspondiente a este segundo contacto tendría un peso mucho mayor que la del primero.

Con este método de ponderación de los enlaces entre nodos, los autores manifiestan que consiguen una precisión de localización a 25 millas ($\text{acc}@25$) de 63.9% mientras que, con el algoritmo básico, referido anteriormente, se obtiene un valor de 55.7%.

2.4.2.2 Método de propagación de etiquetas.

Supongamos un grafo en el que algunos de sus nodos están marcados con algún tipo de etiqueta que los clasifique y otros no. Basándonos en las relaciones representadas en el grafo, podríamos tratar de calcular la probabilidad de que un nodo no etiquetado adquiriese una determinada etiqueta. Una forma de hacerlo sería contar los distintos caminos que llevan a una etiqueta u otra. La etiqueta a la que conduzcan más caminos, será la que se asigne.

Veamos un ejemplo, en la Figura 2-30 se muestra un grafo en el que existen dos etiquetas, la de color rojo y la de color verde. Existen muchos nodos no etiquetados, de color gris, y queremos saber qué etiqueta le corresponderá al nodo con el número 11. Si obtenemos los caminos que conducen a las distintas etiquetas, que no pasen por otra etiqueta ni dos veces por el mismo nodo, tendremos lo siguiente:

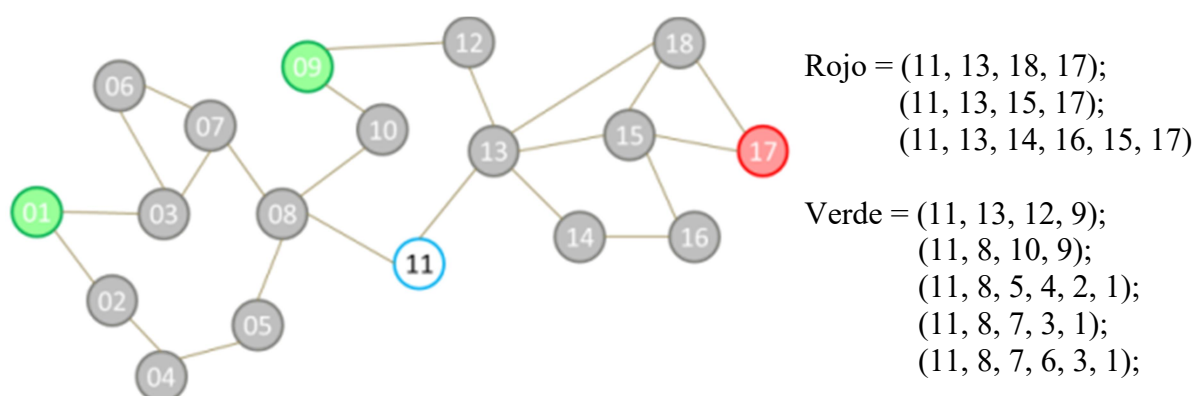


Figura 2-30 Ejemplo de grafo simple no dirigido en el que los nodos rojos y verdes están etiquetados.

Puesto que hay más caminos que conducen a las etiquetas de color verde, el nodo 11 deberá etiquetarse con ese color.

	01	09	17	02	03	04	06	07	...	13	15	18
01	1	0	0	0	0	0	0	0	0	0	0	0
09	0	1	0	0	0	0	0	0	0	0	0	0
17	0	0	1	0	0	0	0	0	0	0	0	0
02	0.5	0	0	0	0	0.5	0	0	0	0	0	0
03	0.333	0	0	0	0	0	0.333	0.333	0	0	0	0
...												
18	0	0	0.333	0	0	0	0	0	0	0.333	0.333	0

Figura 2-29 Extracto de la matriz de probabilidades de encaminamiento de nodos.

Para cuantificar las probabilidades de seguir un camino u otro, se puede definir una matriz T de como la de la Figura 2-29, con tantas filas y columnas como nodos en el grafo.

Vemos que las primeras filas corresponden con los nodos etiquetados. Desde un nodo etiquetado no podemos ir más que al propio nodo, así que le corresponde una probabilidad 1 ; puesto que no puede ir a otros nodos, el resto de columnas tendrá el valor 0 . Desde el nodo 02 podemos ir al 01 y al 04, así que la probabilidad es de un medio para cada caso. Para el nodo 03, las probabilidades son un tercio para encaminarse a los nodos 01, 06 y 07. De igual manera se completa el resto de la matriz.

Hay que reseñar, que también se puede considerar que las uniones entre nodos, las aristas, pueden tener pesos, que afectarían a la probabilidad de seguir un camino u otro.

En esta matriz T , podemos observar que el grupo de filas y columnas correspondientes a los nodos etiquetados, coloreado de azul más claro arriba a la izquierda, es una matriz identidad, con su diagonal a 1 , y el resto a 0 . La parte de coloreada de arriba a la derecha es una matriz nula, con todos sus valores a 0 .

Si elevamos T a una potencia grande, tendiendo a infinito, en las columnas de la porción de la matriz coloreada de rojo obtendremos la probabilidad de que a un nodo concreto le corresponda la etiqueta asociada con su identificador de columna. En la Figura 2-31, en la que se ha elevado la matriz T a su vigésima potencia, se puede observar que para el nodo 02 hay una mayor probabilidad de que le corresponda la etiqueta asociada al nodo 01, es decir, la verde. En cambio, al nodo 18 se le asignará la etiqueta asociada al nodo 17, la de color rojo, por tener más probabilidad.

$$T^{20} =$$

	01	09	17	02	03	04	06	07	...	13	15	18
01	1	0	0	0	0	0	0	0	0	0	0	0
09	0	1	0	0	0	0	0	0	0	0	0	0
17	0	0	1	0	0	0	0	0	0	0	0	0
02	0.827	0.084	0	0.004	0.004	0.001	0.004	0.009		0.003	0.004	0.002
03	0.724	0.123	0.039	0.002	0.009	0.006	0.008	0.011		0.003	0.004	0.003
...												
18	0.032	0.156	0.744	0.002	0.003	0.002	0.003	0.005		0.006	0.007	0.004

Figura 2-31 Extracto de la matriz T elevada a 20.

El resto de los valores situados en la porción de la matriz coloreada de azul más oscuro, tenderán a cero, conforme aumenta la potencia a la que se eleva T .

El hecho de que la matriz T tenga esa estructura compuesta por cuatro sub-matrices bien diferenciadas, permite obtener una expresión de cálculo para asignar las etiquetas más fácilmente computable, tal y como se desarrolla en [36].

Una de las principales utilidades de la propagación de etiquetas es implementar el aprendizaje parcialmente supervisado. Es decir, si tenemos una gran cantidad de muestras de las que únicamente un conjunto reducido está clasificado. Son tan pocas que no podemos utilizarlas como conjunto de entrenamiento de otro tipo de algoritmos clasificadores, puesto que podríamos correr el riesgo de que no fueran representativas. Tampoco queremos aplicar un aprendizaje sin supervisión puro puesto que queremos aprovechar el conocimiento que tenemos sobre ese pequeño conjunto de muestras. Si utilizamos el algoritmo de propagación de etiquetas considerando que los pesos de las aristas son proporcionales a las distancias entre muestras, podremos clasificar todo el conjunto.

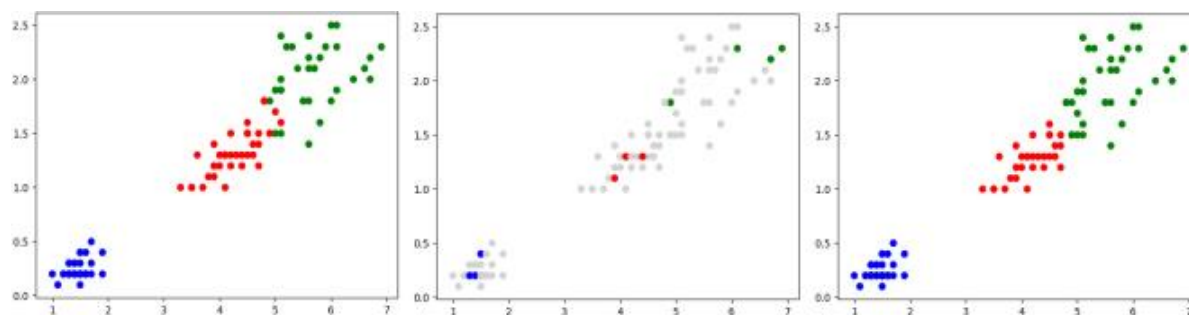


Figura 2-32 Ejemplo de aplicación del algoritmo de Propagación de Etiquetas en aprendizaje semi-supervisado.

Para ejemplificar el uso del algoritmo de Propagación de Etiquetas, podemos observar la Figura 2-32. A la izquierda tenemos las tres clases originales de un conjunto de entrenamiento. En el centro hemos suprimido las etiquetas del 90% de las muestras de cada clase. Las muestras que no tienen etiqueta, se muestran en color gris. A la derecha hemos aplicado la propagación para tratar de determinar las clases, teniendo en cuenta la distancia entre los puntos. Observamos algunas discrepancias en la clasificación de las clases rojas y verdes. En esta aplicación, resulta de suma importancia el tratar de asegurar que el número de etiquetas de cada clase es similar, puesto que en caso contrario el resultado puede ofrecer un resultado bastante erróneo, dependiendo de la posición de las muestras etiquetadas.

En esta aplicación, lo que estamos suponiendo es un grafo completo, en el que todos los nodos están conectados entre sí, con aristas cuyo peso es proporcional a la distancia euclídea entre las muestras. Si el número de muestras es muy elevado, el grafo tiene muchas aristas y resulta computacionalmente costoso. Para evitarlo, es posible considerar únicamente las aristas de los k -nodos más cercanos¹.

2.4.2.3 Método de incrustación de grafos (“network embedding”).

Dentro de la categoría de incrustación de grafos podemos incluir diversos algoritmos como “*Spectral Clustering*”, “*DeepWalk*”, “*Large-scale Information Network Embedding (LINE)*” o “*Node2Vec*”. Estos algoritmos se caracterizan por el aprendizaje de representaciones latentes, de baja dimensión, para los nodos o enlaces de un grafo.

Ya hemos visto anteriormente cómo se puede representar un grafo mediante, por ejemplo, una matriz de adyacencia. Esas representaciones son vectores o matrices con dimensiones elevadas. La matriz de adyacencia de un grafo con 100 nodos es una matriz de 100x100. Estas representaciones son costosas computacionalmente tanto en términos de espacio de memoria como en su tratamiento matemático.

Los algoritmos de incrustación de grafos consiguen reducir la dimensión de la representación, sin perder las características significativas de los grafos que nos permiten, por ejemplo, identificar la homofilia² en la red. Así pues, de cada nodo del grafo se obtendrá una representación vectorial que podrá utilizarse fácilmente con algoritmos de Machine-Learning que clasifiquen las muestras. Además, dicha representación vectorial será eficiente desde el punto de vista computacional.

¹ En la librería de Python `sklearn.semi_supervised.LabelPropagation` se puede especificar el parámetro `n_neighbours` para limitar las aristas del grafo.

² La homofilia de red (“Network homophily”) es una característica de los grafos en los que aquellos nodos con atributos o características similares tienden a estar más próximos entre sí.

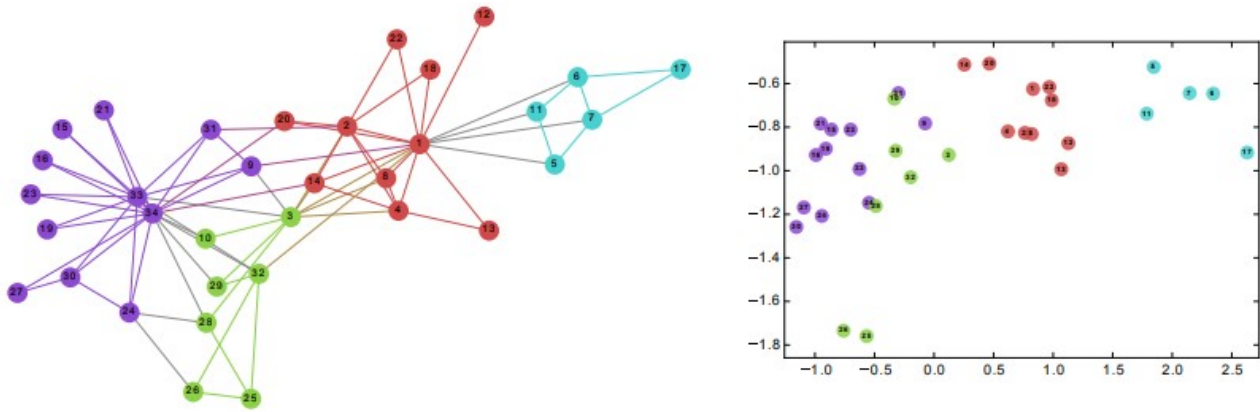


Figura 2-33 Representación de un grafo con dimensión reducida obtenido mediante *DeepWalk*. (Figura obtenida de [37])

En la Figura 2-33 se puede observar la reducción de la dimensión en la representación y, el mantenimiento de las características del grafo. El grafo de la izquierda, tal y como se menciona en [37], procede de un estudio¹ en el que se han empleado hasta ocho contextos diferentes para ponderar los pesos de las relaciones que representan las aristas. A la derecha, la misma información con, únicamente, dos dimensiones obtenida mediante *DeepWalk*.

De los cuatro algoritmos citados, trataremos aquí con detalle únicamente dos: *DeepWalk* y *Node2Vec*.

En *DeepWalk* se utiliza el aprendizaje profundo (“*deep learning*”) para obtener de forma no supervisada las características latentes representadas en un grafo y transformarlas en un conjunto de vectores de dimensión reducida. Para lograrlo, utiliza dos fases diferenciadas. En primer lugar, explora el grafo realizando un cierto número γ de recorridos aleatorios para cada nodo con una profundidad t , o número máximo de nodos en el recorrido, determinada. De este modo se obtiene, para cada nodo, γ vectores de longitud t que representan el contexto de los nodos. La palabra contexto se utiliza en el mismo sentido que en otros procesos de análisis del lenguaje en el que la definición de una palabra es proporcionada por el conjunto de palabras que con mayor probabilidad aparecen junto a ella en una colección de documentos. El modelo *Skip-Gram* trata de predecir en el lenguaje la probabilidad de que, dada una palabra, la suceda otra. Para que un computador aprenda esas probabilidades se le alimenta con diferentes textos de un corpus. Este enfoque ha tenido bastante éxito en el procesamiento del lenguaje natural debido fundamentalmente a la implementación de un algoritmo denominado *Word2Vec*, descrito en [38]. Su mención aquí nos conduce a la segunda fase de *DeepWalk* en la que se utiliza el enfoque de *Skip-Gram* para predecir la probabilidad de que un nodo suceda a otro, del mismo modo que lo hace *Word2Vec*, pero utilizando la secuencia de nodos recorridos aleatoriamente, en lugar de frases y palabras. Ambos, *Word2Vec* y *DeepWalk*, utilizan una función de activación denominada *Softmax*, en la última etapa de la red neuronal, que proporciona los vectores de probabilidad normalizados. El espacio vectorial que se conforma con los vectores de cada nodo constituye la representación equivalente del grafo.

En la Figura 2-34 se observa el proceso completo. En la parte izquierda se seleccionan recorridos aleatorios de un nodo. Para el recorrido $W_{v4} = \{4, 3, 1, 5, \dots\}$ se elige una ventana de $2w+1$ elementos. En este caso se ha seleccionado $w=1$ por lo que la ventana que establece el contexto a considerar u_k tiene tres elementos. Posteriormente, se obtienen mediante *Skip-Gram*

¹ “An information flow model for conflict and fission in small groups”. W. Zachary. 1977. Journal of anthropological research 33(4): 452-473.

los valores que indican la probabilidad de coocurrencia de dos nodos en un mismo contexto. Así pues, en el centro de la figura, observamos que para el nodo v_l se toman en consideración los nodos v_3 y v_5 . De esta forma, podemos obtener para el nodo v_l un vector $\Phi(v_l)$, con una dimensión arbitraria d , que nos indica las probabilidades de co-ocurrencia (en la figura se corresponde con la fila j de la matriz Φ) con otros nodos seleccionados de los recorridos aleatorios. Por último, se utiliza el algoritmo *Softmax* en un árbol jerárquico, que trabaja con la conocida función logística, para calcular la probabilidad de que, dado el nodo v_l , co-ocurran el resto de nodos del contexto. Estas probabilidades constituyen el vector de características que representa v_l . Si se aplica a todos los nodos del grafo obtendremos una matriz con tantas filas como nodos y con d columnas.

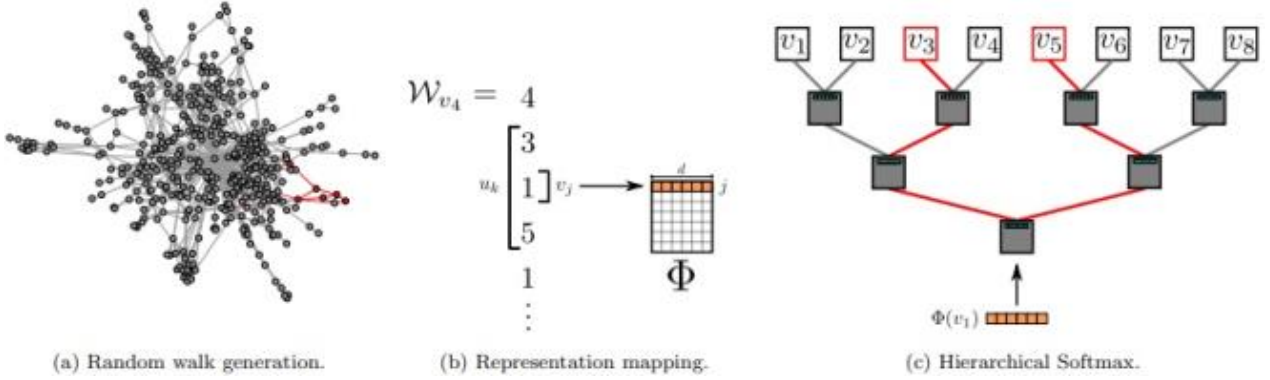


Figura 2-34 Funcionamiento del Algoritmo DeepWalk (obtenida de [37])

Como hemos visto, el algoritmo utiliza diversos hiperparámetros: w , que fija el tamaño de la ventana que determina el alcance del contexto; d , que establece la dimensión del vector de características a obtener; γ , que se corresponde con el número de recorridos aleatorios que se considerarán, y t , que indica la longitud de dichos caminos.

Node2Vec comparte el enfoque de considerar los grafos como textos, donde los nodos equivalen a palabras que se relacionan con otras palabras dentro de un contexto. También utiliza *Skip-Gram* para averiguar el contexto a partir de un nodo y, nuevamente, el resultado es una matriz con tantas filas como nodos y tantas columnas como se decida a partir de un hiperparámetro que establece cuántos nodos hay que incluir en el contexto. *Node2Vec* selecciona los recorridos dentro del grafo de un modo menos aleatorio que *DeepWalk*, puesto que asigna unos pesos que determinan la probabilidad, α , de que se elija un nodo u otro a partir de uno dado. Estos pesos se obtienen a partir de dos hiperparámetros, p y q , que establecen cómo de probable es que se vuelva a recorrer un nodo ya recorrido anteriormente, y cómo de probable es que se seleccione un camino que conduzca fuera del entorno vecino. El objeto es que los caminos aleatorios se escojan entre dos posibilidades extremas determinadas por los métodos de selección del siguiente nodo denominados BFS y DFS.

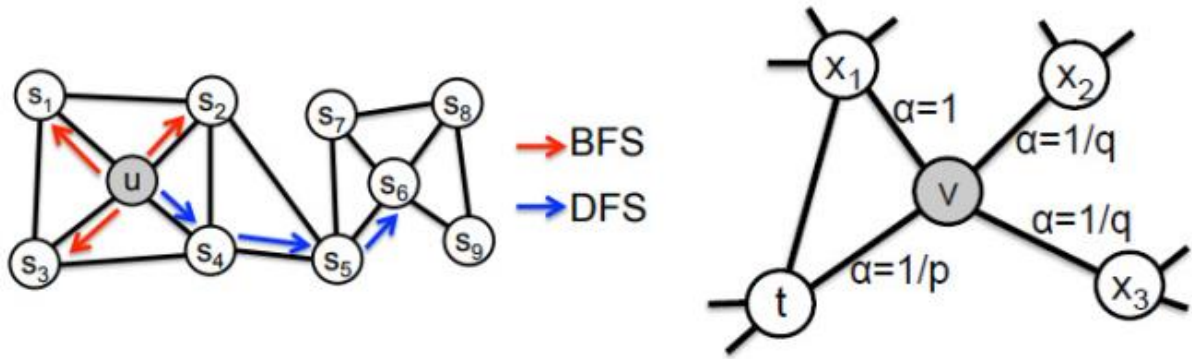


Figura 2-35 Recorridos BFS/DFS y valores de α en los recorridos de Node2Vec (obtenido de [39]).

BFS (*Breadth-first sampling*), o búsqueda primero en anchura, da preferencia a recorrer nodos que se encuentren en la periferia inmediata del nodo origen. Por el contrario, DFS (*Depth-first sampling*), o búsqueda primero en profundidad, prioriza los recorridos de los nodos que se van alejando del origen.

En la figura Figura 2-35, observamos a la izquierda la diferencia entre los recorridos BFS y los DFS. En los BFS, se considerarán antes los nodos que se encuentran enlazados con el origen (s_1, s_2, s_3 , etc.) que se señalan con flechas rojas. En DFS, se considerarán en primer lugar aquellos que se alejen del nodo raíz, señalados con flechas azules. En *Node2Vec* se asigna una probabilidad α , a partir de dos parámetros p y q , para seleccionar los nodos de un modo intermedio entre las opciones que representan BFS y DFS. En la parte derecha de la figura, vemos el valor α que se les asigna a los enlaces del nodo v , después de haber visitado el nodo t . El valor de regresar por el camino ya recorrido, es decir volver del nodo v al t , se controla con el parámetro p . El valor de transitar a un nodo que no esté enlazado con t , lo que se considera ir hacia fuera del entorno vecino, se controla con el parámetro q . Los nodos que sí están enlazados con t tienen un valor de α igual a 1. El recorrido, puramente aleatorio de DeepWalk se conseguiría con $p = q = 1$. El hecho de considerar el nodo anteriormente visitado para determinar el siguiente a visitar, lo denominan recorrido aleatorio de segundo orden (“*2nd order random walk*”). Valores de p muy grandes producen una baja probabilidad de volver a visitar un nodo que ya ha sido visitado, evitando en cierto modo la redundancia. Por otro lado, si $q > 1$ se favorecerá el visitar nodos próximos a t , aproximándose al comportamiento de BFS; mientras que si $q < 1$, se favorece visitar los nodos alejados de t , lo que es propio de DFS.

Otra de las diferencias entre *DeepWalk* y *Node2vec* es que en lugar de utilizar *Softmax* sobre el árbol jerárquico, utiliza otra técnica computacionalmente más efectiva que se denomina muestreo negativo (“*Negative Sampling*”). La idea básica es que, en la fase de aprendizaje, tras obtener el vector de características correspondiente a un nodo, se deberían actualizar los vectores correspondientes a todos los otros nodos, lo cual es computacionalmente muy costoso en grandes grafos, concretamente del orden de $|V|$, es decir, del total de nodos del grafo. En *DeepWalk* se utiliza un árbol jerárquico que reduce el coste al orden de $\log|V|$. Por el contrario, utilizando el muestreo negativo se actualizan los valores de los nodos directamente conectados y de un cierto número prefijado de nodos no conectados. Con ello se reduce todavía más el coste de la actualización de los valores de las características de los vectores.

Comparativamente, puesto que *Node2Vec* es más moderno que los otros métodos, incluso se ha elaborado basándose en algunas particularidades de los mismos, se considera que es más efectivo, tal y como se puede comprobar en las tablas comparativas que se encuentran en [39].

2.4.3 Métodos híbridos

Los métodos que se basan en el análisis del grafo de relaciones entre los usuarios tienen el inconveniente de que son incapaces de predecir la localización de aquellos nodos que se

encuentran aislados, es decir, de aquellos que no tienen ningún contacto más o menos próximo localizado con certeza. Por otro lado, los métodos basados en el análisis del lenguaje ignoran la influencia de las relaciones sociales, que se ha demostrado suficientemente valiosa para ser tomada en consideración. Si combinamos los dos enfoques obtenemos los denominados métodos híbridos que pretenden superar las limitaciones que tienen los anteriores, considerados separadamente.

Según [10], los métodos híbridos utilizan dos enfoques distintos. El primero podríamos denominarlo aproximación por agrupación (“*ensemble approach*”), en el que primero se utiliza uno de los dos métodos y seguidamente el otro, para obtener la localización. El orden puede ser cualquiera, aunque habitualmente es primero el análisis del grafo de relaciones y después el de texto. El segundo enfoque se puede denominar aproximación de aprendizaje conjunto (“*joint learning approach*”). En este caso, utilizamos información tanto del grafo de relaciones como del análisis de textos para construir un modelo basado en un conjunto de características conjunto.

2.4.3.1 Métodos de aproximación por agrupación (“*ensemble approach*”).

Podemos encontrar esta metodología en [40], donde los autores parten de un grafo construido a partir de la red de menciones de varios conjuntos de datos conocidos (GEOTEXT, TWITTER-US y TWITTER-WORLD). Para la localización utilizan en primera instancia un algoritmo de propagación de etiquetas, como el detallado en 2.4.2.2. Pero tratan de resolver dos problemas de distinta naturaleza.

En primer lugar, la representación del grafo resulta muy costosa computacionalmente debido al número de aristas. Por ello eliminan los nodos que se consideran celebridades. Estos nodos resultan ser sumideros de aristas y, además, aportan poca información sobre la localización del que usuario que los menciona. Los autores utilizan un parámetro umbral T para determinar cuándo se considera un nodo como celebridad. En función de los distintos conjuntos de datos, dicho parámetro lo varían entre 5 y 15. Es decir, los nodos que tengan más de dicho número de menciones son descartados. Esta práctica consigue reducir el número medio de nodos entrelazados por un factor de 4000 en el conjunto de datos TWITTER-WORLD, utilizando un valor de $T = 5$.

El segundo problema es el tratamiento de aquellos nodos que no están conectados a otros con localizaciones conocidas. Aquí es donde se incorpora la información obtenida del análisis de texto. Para ello se incorpora al grafo un nodo adjunto (“*dongle*”) etiquetado con la información obtenida de un modelo de regresión logística.

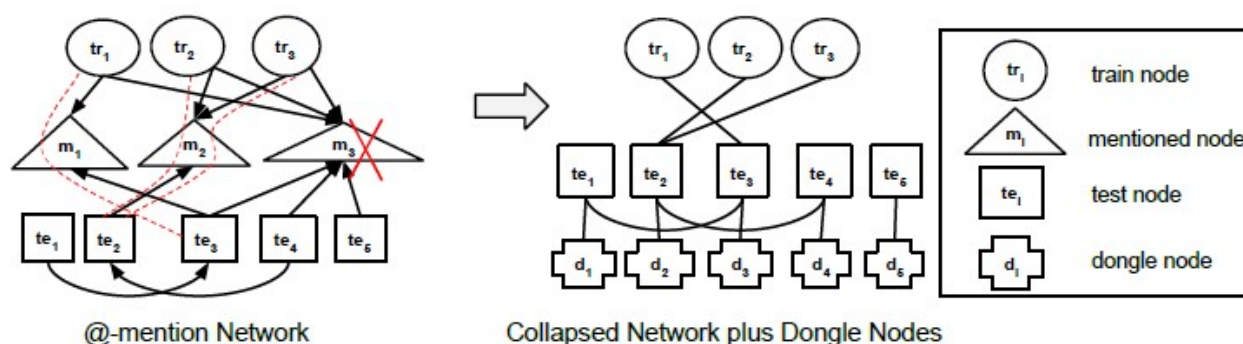


Figura 2-36 Grafo de menciones con supresión de celebridades y adición de los nodos adjuntos (obtenida de [40])

En la Figura 2-36, a la izquierda vemos la forma en que los autores obtienen un grafo colapsado (“*collapsed network*”). Si un nodo A del conjunto de entrenamiento menciona a otro

B , y un nodo del conjunto de test C , también menciona a B , entonces se puede simplificar el grafo extendiendo (líneas rojas punteadas) entre A y C . Además, aquellos nodos que superen un determinado número de menciones se eliminan del grafo. En la figura, se ha eliminado el nodo m_3 . El grafo colapsado no es direccional y relaciona directamente los nodos de test con los de entrenamiento. Además, se utiliza el análisis de textos para añadir a cada nodo de test un nodo adjunto con su correspondiente etiqueta. Estos nodos serán tratados de igual forma que el resto por el algoritmo de propagación de etiquetas.

Con una aproximación similar también podemos encontrar [41]. Aquí, los autores obtienen los mejores resultados cuando aplican sucesivamente distintos métodos utilizando el esquema que denominan “*full cascade*”. Hasta siete métodos distintos ordenados según su capacidad individual de ofrecer resultados de mayor a menor precisión y con exhaustividad creciente. Así, el primer método es muy preciso, pero solo ofrece resultados en un limitado número de ocasiones. Entre los métodos utilizados se alternan los de análisis de texto, que incluyen tanto el uso de diccionarios toponímicos como técnicas de procesamiento de lenguaje natural, con los que recorren el grafo de menciones utilizando la propagación de etiquetas.

2.4.3.2 Métodos de aproximación de aprendizaje conjunto (“*joint learning approach*”).

En este método, se trata de obtener características de diversas fuentes, incluyendo la información de los grafos de relaciones, del texto de los tweets publicados, de los metadatos de los perfiles de usuario, etc., para utilizarlos conjuntamente y obtener el modelo de predicción. Se trata de un enfoque distinto al anterior, en el que se utilizaban los métodos de análisis de texto para complementar a los de análisis del grafo de relaciones, o viceversa, en el sentido de con uno de los métodos se trataba de evitar las flaquezas del otro. En este caso, todas las características obtenidas se utilizan conjuntamente independientemente de la metodología utilizada para obtenerlas.

En [42] encontramos un ejemplo de este método y en la Figura 2-37 podemos ver su esquema de trabajo. En la parte inferior, en cuadros oscuros, encontramos las distintas fuentes de información que se emplean. Observamos que se utilizan tanto los mensajes de los usuarios, como la localización, la descripción y la zona horaria procedente de los metadatos de sus perfiles. Todas estas fuentes, se basan en análisis de textos. Por otro lado, vemos que los autores construyen grafos a partir de las relaciones entre usuarios y de las ciudades donde se localizan. El producto final consiste en una etiqueta que identifica una ciudad, tal y como se representa en el cuadro oscuro situado arriba a la derecha.

El modelo utiliza redes neuronales recurrentes (RNN), implementadas mediante GRU (“*Gated Recurrent Unit*”) y un mecanismo de atención¹ (“*Attention*”). La explicación de estos conceptos excede el alcance de este documento, por lo que no se tratarán en detalle. Únicamente recalcaremos el hecho de que el mecanismo de “*Attention*” aplicado permite aprender cuales son las características más importantes de un conjunto de datos de entrenamiento. De ese modo, el modelo es capaz de conceder más importancia a las características procedentes del análisis de los grafos de relaciones o, por el contrario, a alguna de las características procedentes del análisis de textos o metadatos. Este mecanismo es el que permite considerar las características obtenidas de las diferentes fuentes que, obviamente, tendrán representaciones y escalas distintas.

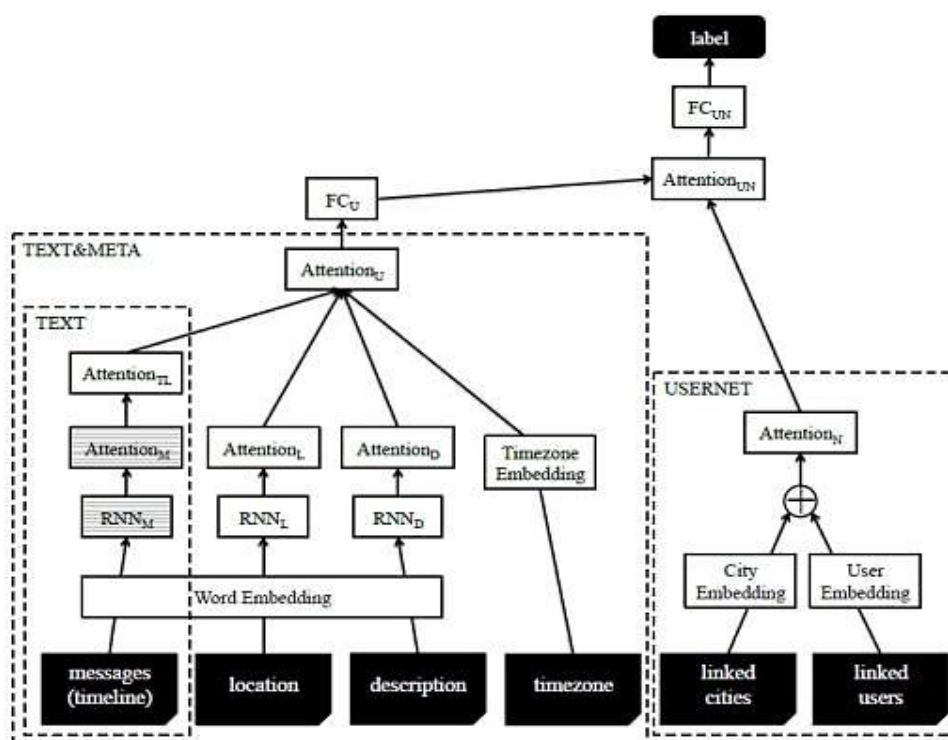


Figura 2-37 Esquema de un modelo de aprendizaje conjunto (obtenido de [42])

2.4.4 Comparación de la eficacia de los diversos métodos

Para poder comparar distintos métodos desarrollados por los académicos, es necesario aplicarlos sobre el mismo conjunto de datos. Afortunadamente, la existencia de algunos conjuntos bien conocidos, como los mencionados en 2.3.1, nos facilita dicha comparación. En [10] se recogen diversas gráficas con las comparativas de la eficacia de la localización obtenida por diversos métodos concretos, alguno de los cuales se han mencionado a lo largo de este documento.

En la Figura 2-38, observamos la comparación entre diversos métodos que utilizan el conjunto de datos denominado GEOTEXT. La eficacia de los mismos se mide mediante la mediana de los errores de localización medidos en kilómetros. En general, se observa que los métodos que hemos denominado híbridos (“*multiview*” en la gráfica) obtienen un mejor rendimiento. Parece lógico que sea así puesto que utilizan tanto la información procedente de los basados en análisis de los grafos de redes de usuarios, como del análisis de textos. El modelo

¹ Se trata de un mecanismo, dentro del contexto de redes neuronales, descrito inicialmente en 2015 por Bahdanau D. et al en su trabajo “Neural machine translation by jointly learning to align and translate”.

denominado *Ra2015hyb* en la gráfica, se corresponde con el descrito en el punto 2.4.3.1, procedente de [40].

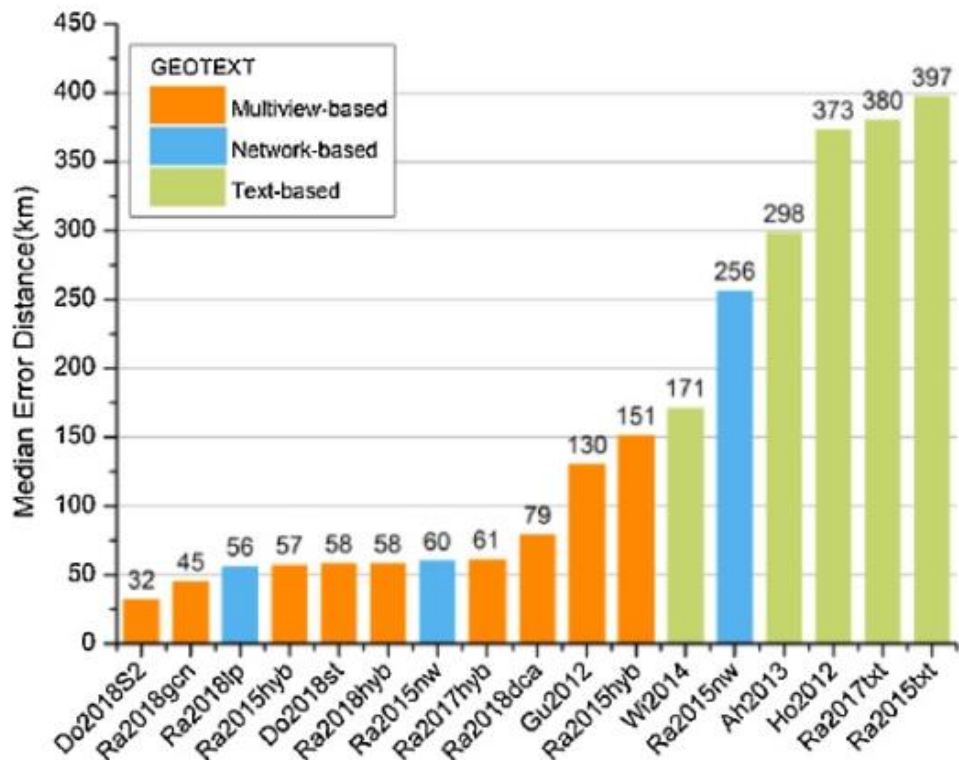


Figura 2-38 Comparación de los métodos de localización (obtenida de [10])

La ventaja en eficacia de los métodos híbridos se manifiesta también cuando se utilizan otros conjuntos de datos, como TWITTER-US o WORLD, para la comparación. Sin embargo, tal y como se señala en [10], algunos de los métodos de análisis de grafos obtienen mejor rendimiento con un conjunto de datos u otro. Por ejemplo, el conjunto WORLD tiene una menor proporción de nodos aislados en las redes de amistad/mención de usuarios, por lo que en dicho conjunto de datos los métodos que explotan esta información obtienen mejor rendimiento que en otros.

En cuanto a la comparación entre los métodos de análisis de textos frente a los de análisis de grafos, en [10] se asegura que los segundos requieren menos hiperparámetros, son más rápidos y más eficientes computacionalmente, por lo que se les considera métodos más potentes que los de análisis de texto. Como contrapartida, la localización de nodos aislados no es posible con este tipo de algoritmos y, puesto que los métodos de análisis de textos son muy sensibles a las técnicas de preprocesado, por lo que, en algunos casos, se pueden obtener mejores resultados que con el análisis de grafos.

3 DESARROLLO DEL TFM

3.1 Metodología

3.1.1 Enfoque

Una vez revisada la literatura académica y expuestos los principales métodos y mecanismos utilizados en la geolocalización, el lector interesado en el diseño e implementación de un sistema que permita la ubicación del usuario que publica un tweet, seguirá percibiendo la existencia de cierta falta de información. Los académicos en sus documentos explican sus métodos de forma general, la extensión de los mismos impide normalmente descender a los detalles necesarios para implementar una solución práctica. La literatura académica está escrita para los propios académicos y dan por supuestas muchas cuestiones que no son, en absoluto, sencillas.

A la hora de exponer los mecanismos de geolocalización, por tanto, no podemos detenernos en la mera recopilación de la información publicada en la literatura académica. Resulta necesario abordar los detalles y los problemas prácticos de una implementación real. Así pues, en este apartado se explican detalladamente todos los pasos para desarrollar el proceso completo, desde la recogida de los datos y su almacenamiento, como el procesado y utilización con los algoritmos de clasificación para obtener la geolocalización. Por tanto, la metodología empleada es la de la descripción de una implementación práctica completa de algunos de los mecanismos utilizados en la bibliografía académica.

Este enfoque puede caer en el exceso de detalle de modo que las soluciones adoptadas en este trabajo no sean todo lo generalistas que pudiera desear alguien que decida abordar el mismo problema utilizando otras herramientas. Sin embargo, tal riesgo sigue resultando mucho menor que el no tener ningún referente de cómo abordar una implementación práctica de los métodos propuestos en la bibliografía.

Tratando de abordar los dos grandes métodos existentes para la geolocalización, el basado en el análisis de texto y el basado en el análisis de las redes de usuarios, hemos codificado una solución independiente para cada uno de ellos de forma que se puedan utilizar, si se cree conveniente, como base para soluciones más elaboradas o, en su defecto, para constatar las dificultades a las que tendrá que enfrentarse aquel que desee implementar una solución al problema.

Para cada una de las soluciones propuestas se han escogido conjuntos de tweets que se han utilizado para el proceso. En la mayor parte de los casos, la localización de los tweets ya se conoce debido al propio proceso de obtención. Con ello conseguimos un conjunto de referencia (“*ground-truth*”) que posteriormente dividimos, para los casos en que se utilicen técnicas de

Machine-Learning, en un subconjunto de entrenamiento y otro de test con el que obtendremos los valores de exactitud. Para el caso del análisis de grafos, no existe el conjunto de entrenamiento, pero igualmente se utiliza el conjunto de referencia para comprobar que los resultados ofrecidos se corresponden, o no, con la realidad. Esta técnica de validación es la misma que se utiliza en la bibliografía académica.

3.1.2 Herramientas utilizadas

Para la implementación de las soluciones que se presentan se han utilizado las herramientas que se relacionan seguidamente.

Desde el punto de vista de hardware se ha empleado un ordenador portátil con procesador *Intel i7* de tercera generación con una tarjeta gráfica NVIDIA GeForce GT 635M con 2 Gb de memoria dedicada. Dicho ordenador contaba con 8Gb de RAM y un HDD SSD de 256Gb. Este equipo tenía en el momento de realización del trabajo más de ocho años de antigüedad, significando que las exigencias en cuanto a capacidad de computación no han sido especialmente elevadas.

En cuanto a conectividad, solo se ha precisado de una conexión a Internet que ha permitido la conexión con las APIs que proporcionaron los tweets que se han analizado, o algún servicio particular como el ofrecido por los diccionarios toponímicos.

En cuanto al software utilizado, el sistema operativo del portátil era un *Windows 10* sobre el que se instaló el programa *PyCharm*¹ como entorno de programación para Python. Para el almacenamiento de los tweets se ha utilizado la base de datos de *MongoDB Community Edition*² que permite el arranque del servicio en modo local. Para la visualización y análisis de los grafos, se ha utilizado en ocasiones *Gephi 0.9.2*, tal y como se describe más adelante, en el apartado correspondiente.

3.2 Obtención de los datos.

3.2.1 La API de Twitter

Twitter ofrece una API que permite a los programadores crear aplicaciones con las mismas funcionalidades que tiene su aplicación, en cuanto a creación de tweets, marcado de tweets como del gusto/disgusto del lector, seguimiento de usuarios, búsqueda de hashtags, etc. La API permite facilita incluso la creación de bots, o la integración de otras aplicaciones con Twitter.

Para que el acceso a las bases de datos de tweets sea posible, Twitter ha establecido unos límites de acceso que varían según la modalidad. Algunas modalidades están orientadas al uso comercial y otras son gratuitas. Todas las modalidades tienen restricciones, como limitación del máximo de solicitudes de datos cada quince minutos o un máximo número de solicitudes mensual³. Para las modalidades que no son gratuitas, los límites son más amplios.

Aun si queremos utilizar la API en su modalidad gratuita, es necesario registrarse y obtener unas credenciales de autenticación que nos permitirán el acceso. Para ello, es necesario rellenar un formulario de solicitud desde el panel de control de nuestra cuenta de Twitter. Finalmente conseguiremos obtener las siguientes claves:

- “Consumer Keys”: “API Key” y “API Secret”.

¹ PyCharm es un IDE de programación que cuenta con una versión “Community” sin coste que se puede obtener en <https://www.jetbrains.com/pycharm/> [Disponible a 23/12/2021]. La versión utilizada ha sido la 2021.1.3x64.

² Las instrucciones de instalación y el enlace de descarga se pueden encontrar en <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/> [Disponible a 23/12/2021]

³ “Rate limits” <https://developer.twitter.com/en/docs/rate-limits> [Disponible a 23/12/2021]

- “Authentication Tokens”: “Access Token” y “Access Token Secret”.

Una vez obtenidas, ya será posible crear aplicaciones con acceso a la API. Para ello, existen diversas opciones. La primera es utilizar directamente la API, mediante el empleo de *curl* desde la línea de comandos generando típicamente consultas GET de HTTP. La otra forma es utilizar alguna librería de algún lenguaje de programación concreto que nos facilite esta tarea, proporcionando, por ejemplo, clases de objetos con sus respectivos métodos. *Tweepy*¹ es una de estas librerías para utilizar con el lenguaje Python.

En la actualidad, Twitter está lanzando una nueva API v2 que permitirá tipos de acceso diferentes a los actuales y otras funcionalidades que, resultarán de gran interés para los analistas. Destaca especialmente el denominado acceso “*Academic Research*”² que amplía algunos de los límites de acceso —el límite mensual asciende a diez millones de tweets— y permite que las consultas de búsqueda accedan al histórico completo de tweets. Prometen acceso completo, hasta marzo de 2006³. Desde finales de 2020, las comunidades que soportan las librerías como *Tweepy* están tratando de proporcionar las nuevas funcionalidades de la API v2. Sin embargo, a fecha de julio de 2021, la última versión estable de *Tweepy* todavía no las ofrece.

A través de la API de Twitter podremos acceder a los tweets publicados por los usuarios, pero no únicamente al texto, sino también a gran cantidad de información adicional, como la fecha y hora de creación del tweet, si el tweet es una respuesta al tweet de otro usuario, si el tweet cita a otro tweet, si es un retweet, si incluye hashtags, urls, menciones, etc⁴. Esta información se proporciona en un diccionario con formato JSON. Probablemente tres de los atributos más importantes que existen en dicho diccionario desde el punto de vista de la localización son los siguientes:

- *Coordinates*: Representa la localización geográfica desde la que se publicó el tweet. Puede tener el valor nulo o contener un objeto de tipo *Point* con la longitud y la latitud, por ese orden. Este atributo reemplaza al atributo *Geo* que se considera obsoleto.
- *Place*: Indica si el tweet está asociado o no a un lugar, aunque no necesariamente es la posición desde la que se originó.
- *User*: Se trata de otro diccionario con formato JSON que contiene la información del perfil del usuario que publicó el tweet. Algunos de los atributos⁵ más interesantes son:
 - *Name*: El nombre del usuario.
 - *Location*: Se trata de una frase de texto libre (string) en la que el usuario puede indicar su localización general con la granularidad que desee. También puede escribir cualquier otra cosa que no sea necesariamente una localización válida.
 - *Followers_count*: El número de seguidores que tiene el usuario.
 - *Friends_count*: El número de usuarios a quien sigue este usuario.
 - *Statuses_count*: El número de tweets emitidos por el usuario.

¹ “Tweepy” <https://www.tweepy.org/> [Disponible a 23/12/2021]

² “Academic Research Access” <https://developer.twitter.com/en/products/twitter-api/academic-research> [Disponible a 23/12/2021]

³ “Search Tweets” <https://developer.twitter.com/en/docs/twitter-api/tweets/search/introduction> [Disponible a 23/12/2021]

⁴ Una relación de todos los atributos del diccionario *Tweet* se puede obtener en “Data dictionary Standard v1.1” <https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/tweet>. [Disponible a 23/12/2021]

⁵ Una relación exhaustiva de los atributos de *User* se encuentra en “Data dictionary Standard v1.1” <https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/user>. [Disponible a 23/12/2021]

También tiene algunos campos obsoletos como pueden ser: *utc_offset*, *time_zone*, *lang*, *geo_enabled*, etc. Estos campos, no deberían utilizarse como fuentes de información en los métodos de localización, puesto que Twitter ya no les da soporte.

3.2.2 Métodos de recogida de datos mediante librerías que emplean la API de Twitter.

Para ejemplificar el uso de la API de Twitter, hemos recogido tweets de dos formas distintas. La primera de ellas utiliza la función *search*¹, en la que se pueden obtener tweets de los últimos siete días que Twitter considera “relevantes”. Este último término es un tanto difuso y en la documentación de la API no se especifica qué condiciones deben tener los tweets para considerarse así. Sin embargo, esta función permite incluir filtros para obtener aquellos que se ajusten a lo especificado por el usuario. Los diversos filtros se pueden ir concatenando para crear una consulta mediante una serie de operadores. También se pueden utilizar unos parámetros adicionales que nos permiten especificar el idioma (“*language*”) o limitar los resultados a un área circular, determinada por su centro y radio (“*geolocalization*”)².

La segunda forma de recoger tweets es obtener los tweets del flujo (“*stream*”) de publicaciones que están realizando los usuarios justo en este momento. De nuevo, es posible especificar valores para ciertos parámetros que nos permitan obtener los tweets de ciertos usuarios (“*follow*”), los relacionados con ciertas palabras (“*track*”), o incluso los geolocalizados en el interior de un rectángulo determinado por coordenadas (“*locations*”)³.

Para acceder a la API de Twitter desde distintos lenguajes de programación, existen diversas librerías que facilitan el uso de las consultas al servicio web, evitando el uso directo de las instrucciones GET/POST del servicio web. La librería que hemos utilizado, perteneciente al Python, es Tweepy. Con Tweepy podemos instanciar un objeto de la clase *API* y utilizar sus métodos para interactuar, indirectamente, con la API de Twitter.



Figura 3-1 Área de obtención de los tweets

¹ Descrita en “Search Tweets” <https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets> [Disponible a 23/12/2021]

² La utilización de los filtros, la construcción de las consultas y el empleo de los parámetros adicionales se puede consultar en “Search Tweets Standard v1.1” <https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/guides/standard-operators>. [Disponible a 23/12/2021]

³ Los parámetros para el filtrado en el “stream” se pueden consultar en “Filter realtime Tweets” <https://developer.twitter.com/en/docs/twitter-api/v1/tweets/filter-realtime/api-reference/post-statuses-filter>. [Disponible a 23/12/2021]

En el Anexo I, se incluye el código que hemos utilizado para la obtención de los tweets con ambos métodos. Para el primero, se ha hecho uso del método *API.search*¹ parametrizado de modo que los tweets recogidos cumplan las siguientes condiciones:

- Los tweets no son re-tweets. De ese modo, evitamos tweets duplicados.
- El lenguaje del tweet es: “es” (español). Simplemente por conveniencia.
- El origen de los tweets está ubicado en el interior de un área circular centrada en unas coordenadas concretas, al sur de la ciudad de Madrid, y con un radio de 700 km. Dicha área incluye toda la Península Ibérica, Ceuta, Melilla las islas Baleares, parte del sur de Francia y del norte de Marruecos. Las islas Canarias no están incluidas. Su representación gráfica se puede ver en la Figura 3-1.

Al utilizar el parámetro *geocode* en la búsqueda, el API de Twitter tratará de devolvernos los tweets con valores de latitud-longitud dentro del área especificada y, en su defecto, devolverá los tweets de los usuarios de cuyo perfil se pueda deducir que pertenecen a una localización cuyas coordenadas se encuentren en el interior de dicha área². Con esta metodología, hemos obtenido diversos conjuntos de tweets, “*tweets_spain*” y “*tweets_spain_ext*” utilizan el radio de 700 km, tal y como se ha descrito. Por otro lado “*tweet_madrid_ext*” y “*tweet_mexico_ext*”, se han obtenido de forma similar, pero utilizando un radio de 25 km para recoger únicamente los tweets procedentes de dichas ciudades.

Para el método de obtención de los tweets procedentes del “*stream*”, Tweepy proporciona una clase denominada *StreamListener*³. Nosotros hemos creado una clase hija que sobrecarga algunos de los métodos. Como parámetro para filtrar los tweets obtenidos únicamente hemos establecido que fueran en idioma español.

3.2.3 Almacenando datos

Los tweets recogidos mediante la API deberán ser almacenados en algún tipo de base de datos para facilitar el “data-mining”. Es posible utilizar bases de datos estructuradas, como MySQL, o no estructuradas, como por ejemplo MongoDB. En nuestro caso hemos optado por utilizar el segundo tipo por diversas razones: En primer lugar, porque los datos que ofrece la API de Twitter, que son bastante similares a un conjunto de colecciones anidadas de python, resultan muy sencillos de convertir en un documento JSON que se puede almacenar directamente en la base de datos no estructurada. Así pues, la primera razón es la sencillez del proceso de almacenamiento. La segunda tiene que ver con el hecho de que la cantidad de datos necesaria para obtener conclusiones del análisis de tweets siempre es elevada. Existen trabajos en los que se han analizado miles de millones de tweets que, obviamente, requieren un espacio de almacenamiento superior al de una estación de trabajo común. Estos datos, obligan a afrontar el proceso de elaboración de métodos prácticos de geolocalización desde la perspectiva de la computación distribuida y, para ese tipo de entorno es para el que han sido diseñadas las bases de datos no relacionales que ofrecen ventajas en cuanto a la escalabilidad. Así pues, la segunda razón, tiene que ver con que MongoDB resultará una mejor opción cuando se trate de manejar un volumen de información que podríamos considerar dentro de los márgenes de lo que se denomina Big Data.

¹ Descrita en “Tweepy API. Twitter API v1.1 Reference” <https://docs.tweepy.org/en/stable/api.html#search-methods>. [Disponible a 23/12/2021]

² Tal y como se especifica en “Search Tweets Standard v1.1” <https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/guides/standard-operators> [Disponible a 23/12/2021]

³ Descrita en “Streaming with Tweepy” https://docs.tweepy.org/en/v3.9.0/streaming_how_to.html [Disponible a 23/12/2021]

Bien es cierto que, en este trabajo, no vamos a alcanzar tales volúmenes de datos, puesto que el objetivo es únicamente mostrar los mecanismos de geolocalización, para lo cual, basta una cantidad de datos bastante más modesta, manejable en una única estación de trabajo.

En nuestro caso, hemos creado una base de datos denominada *Tweets* que contiene varias colecciones. Los tweets obtenidos mediante la función *API.search*, los hemos almacenado en las colecciones denominadas *tweets_spain*, *tweets_spain_ext*, *tweets_madrid_ext* y *tweets_mexico_ext*, mientras que los que hemos obtenido del “stream”, a través de la clase *StreamListener*, los hemos almacenado en la colección denominada *tweets_stream*.

En la Tabla 3-1 se ofrecen algunos de los datos de los tweets de la colección *tweets_spain*. Podemos ver que todos ellos, tienen información de la localización en el campo *place*, o en el campo *user.location*, o en el campo *coordinates*. Esto es así debido a que, como criterio de búsqueda de la función *API.search*, hemos especificado un valor para el campo *geocode*.

Total tweets	14990
Periodo recogida	15:59:51 – 19:53:45 del 26/07/2021
Usuarios distintos	11017
Valor de <i>place</i>	14106 (null); 884 (≠ null)
Valor <i>user.location</i>	189 (= ""); 14801 (≠ ""); 0 (= null)
Valor de <i>coordinates</i>	14949 (null); 41 (≠ null)
(<i>place</i> = null) y (<i>user.location</i> = "") y (<i>coordinates</i> = null)	0

Tabla 3-1 Tweets recogidos en la colección *tweets_spain*

En la Tabla 3-2 se recogen datos similares para los tweets de la colección *tweets_stream*. En este caso, al no incluir ninguna restricción sobre la localización, observamos que el número de tweets geolocalizados es de 122 de un total de 15014. Este valor corresponde a un 0.8%, valor que se ajusta a lo esperado, según lo predicho en la literatura académica.

Por otro lado, observamos que 8946 tweets tienen algún valor en el campo *user.location*. Eso supone casi un 60% de tweets publicados por usuarios que han indicado algo en el campo de localización de su perfil. Sin embargo, ese algo, puede no ser una localización como por ejemplo las frases (“*ella (17)*” o “*No DM gracias*”).

También se observa que el porcentaje de usuarios que geolocalizan con precisión un tweet especificando las coordenadas del lugar de su publicación son muy pocos.

Total tweets	15014
Periodo recogida	15:27:28 – 16:22:39 del 27/07/21
Usuarios distintos	14324
Valor de <i>place</i> (object)	14892 (null); 122 (≠ null)
Valor <i>user.location</i> (string)	0 (= ""); 15014 (≠ ""); 6068 (= null); 8946 (≠ null)
Valor de <i>coordinates</i>	15007 (null); 7 (≠ null)

Tabla 3-2 Tweets recogidos en la colección *tweets_stream*

Puesto que los tweets de *tweets_spain* solo recogía los tweets con un máximo de 170 caracteres, lo que provocaba que muchos de ellos estuvieran incompletos, se obtuvo, con los mismos parámetros de filtrado otro conjunto de tweets que denominamos *tweets_spain_ext*. En la Tabla 3-3 se muestra información sobre este último conjunto de datos. Se puede observar que los valores son bastante similares a los obtenidos en *tweets_spain* dos meses antes.

Total tweets	15000
Periodo recogida	11:15:32 24/09/21 – 15:52:41 26/09/21
Usuarios distintos	11102
Valor de <i>place</i>	14066 (null); 934 ($\neq$ null)
Valor <i>user.location</i>	222 (= ""); 14778 ($\neq$ ""); 0 (= null)
Valor de <i>coordinates</i>	14951 (null); 49($\neq$ null)
(<i>place</i> = null) y (<i>user.location</i> = "") y (<i>coordinates</i> = null)	0

Tabla 3-3 Tweets recogidos en la colección *tweets_spain_ext*

Los conjuntos “*tweets_madrid_ext*” y “*tweets_mexico_ext*” contienen, cada uno, 5000 tweets recogidos en un radio de 25 km alrededor de las ciudades de Madrid y México.

3.2.4 Datos de referencia (“ground-truth data”)

Los datos recogidos con la función *search* en las colecciones *tweets_spain* y *tweets_spain_ext*, tienen un alto porcentaje de tweets en los que la localización tiene un valor distinto de nulo. Realizando consultas sobre la base de datos podemos obtener aquellos tweets en los que los usuarios manifiesten que su localización es “*Madrid*”, “*Comunidad de Madrid*”, “*Madrid, España*”, o cadenas de texto similares que sugieran que la ubicación del usuario es, al menos, la provincia de Madrid. De una forma similar podemos actuar con la provincia de Barcelona. Los tweets extraídos de la base de datos de esta manera los consideraremos como conjuntos de datos de referencia.

En el caso de los tweets recogidos especificando que el área de obtención es un radio de 25 km en torno a las ciudades de Madrid y México, podemos considerarlos también como conjuntos de referencia.

Es obvio que, en función de las posibilidades de establecer estos conjuntos de referencia, podremos desarrollar algoritmos orientados a obtener localizaciones con mayor o menor granularidad. Si hubiéramos utilizado como datos de referencia los tweets que expresaran coordenadas geográficas concretas, podríamos tratar de clasificar por distritos dentro de una ciudad. Sin embargo, el número de tweets que indican un valor en el campo *coordinates* es demasiado bajo como para trabajar con él. Para tener un número significativo de tweets geolocalizados de tal manera sería necesario recoger un número mucho mayor de tweets.

Los conjuntos de datos de referencia nos servirán para el entrenamiento de los algoritmos de ML, puesto que somos capaces de indicar cuál es el resultado esperado en la clasificación.

3.2.5 Datos de test

Los conjuntos de referencia los dividiremos en dos subconjuntos. Por un lado, el conjunto de datos de entrenamiento cuyas muestras servirán para que los algoritmos ML ajusten sus parámetros con el fin de poder predecir futuras clasificaciones. Por otro lado, obtendremos el conjunto de datos de test que nos servirá para evaluar cómo de bien es capaz de clasificar el algoritmo una vez entrenado.

Resulta imprescindible conocer de antemano cuál es el valor correcto de la clasificación de las muestras de test para poder determinar si el algoritmo ML ha predicho correctamente la clase, o si se ha equivocado. Por ello, los datos de test son un subconjunto de los datos de referencia.

En nuestro caso, seguimos el criterio de que el 70% de los datos del conjunto de referencia constituirán el conjunto de entrenamiento, mientras que el 30% constituirán el conjunto de test. Una cuestión a tener en cuenta es que resulta importante que la proporción entre las muestras de cada clase sea similar tanto en el conjunto de entrenamiento, como en el de test, en relación al total del conjunto de referencia. Si, por ejemplo, el porcentaje de muestras de determinada clase

en el conjunto de referencia fuera del 80%, no sería una buena práctica que el conjunto de entrenamiento, o el de test, tuviera un porcentaje significativamente menor, digamos un 60%. Si consideramos que el conjunto de referencia es representativo de la población a evaluar, los conjuntos de test y entrenamiento también deberían ser representativos, es decir, mantener un porcentaje similar de ocurrencia para cada clase¹.

En nuestro caso, la selección de las muestras de entrenamiento y test, se ha realizado teniendo en cuenta esta consideración, utilizando una función de Python con un parámetro específico, tal y como se describirá más adelante.

3.3 Algoritmos basados en análisis de textos.

3.3.1 Algoritmos basados en diccionarios toponímicos.

Para ejemplificar el uso de este tipo de algoritmos, hemos analizado el texto introducido por los usuarios en el campo localización de su perfil de usuario. Para ellos, se ha elaborado un programa en Python que lee dicha información de los tweets recogidos en la colección *tweets_spain*. Para cada uno de los usuarios, se realiza una consulta al diccionario *GeoNames*, para que nos proporcione las coordenadas de las localizaciones indicadas por los usuarios. El código de dicho programa se puede consultar en el Anexo II.

Para hacer uso de los servicios de *GeoNames* se ha empleado la librería *Geopy*² que facilita también el acceso a otros servicios similares.

El acceso a *GeoNames* se ha realizado desde una cuenta gratuita y tiene una limitación de acceso. En función del tipo de consulta se consumen un cierto número de créditos u otro. Para las consultas que hemos realizado, que consisten en preguntar las coordenadas correspondientes al lugar especificado en una cadena de texto, el límite es de mil consultas por hora y un máximo de veinte mil diarias. Existen otros diccionarios de acceso gratuito pero, tal y como se dijo en el punto 2.4.1.1, *GeoNames* contiene el mayor número de entradas y, por otro lado, no tiene una limitación de acceso mayor que otros.

En la Tabla 3-4 se muestran los resultados obtenidos. En la colección *tweets_spain* existían algunos tweets que tenían el campo *user.location* vacío. Eliminando estos, *GeoNames* proporcionó una localización para el 97.2%. Este dato podría dar a entender que, entre los usuarios que indican una localización en su perfil, la mayor parte se puede utilizar para obtener unas coordenadas. Sin embargo, hay que realizar algunas observaciones que se citan a continuación.

Número de tweets analizados	14490
Tweets con <i>user.location</i> = ""	189
Tweets con localización no resuelta	363
Tweets con localización resuelta	13909
Tweets con localización fuera de límite	529

Tabla 3-4 Resultados obtenidos en colección *tweets_spain*.

Por una parte, hemos detectado que *GeoNames* proporciona coordenadas incluso para cadenas de texto bastante poco significativas. Así, por ejemplo, cuando un usuario especificó que su ubicación era "vlc", que probablemente quería decir la ciudad de Valencia, *GeoNames* ofrece

¹ Una explicación sobre el muestreo estratificado "*stratified sampling*", en el que se pretende que los conjuntos de muestras tengan la misma proporción entre las distintas clases que en la población que representan, se pueden encontrar en <https://www.baeldung.com/cs/ml-stratified-sampling> [Disponible a 14/01/2022]

² La documentación de uso de esta librería se puede encontrar en "Welcome to GeoPy's documentation" <https://geopy.readthedocs.io/en/stable/v> [Disponible a 23/12/2021]

las coordenadas de un volcán ubicado en Samoa y llamado “*Mount Silisili*”. Es decir, parece que *GeoNames*, tiene una tendencia a ofrecer una localización, aun cuando no esté muy claro que se corresponda con la cadena de texto buscada. Esto nos hace pensar que habrá un cierto número de localizaciones que serán erróneas. En definitiva, la libertad que tienen los usuarios para indicar la localización en su perfil de Twitter, junto con el comportamiento de los diccionarios públicos, provoca que las coordenadas ofrecidas no sean absolutamente fiables.

Puesto que sabemos que los tweets recogidos en la colección *tweets_spain* los hemos obtenido de la API de Twitter especificando que su origen debía encontrarse en la zona representada en la Figura 3-1, podemos comprobar que las coordenadas ofrecidas por *GeoNames* están en dicha zona. Para ello, hemos delimitado una caja con las coordenadas de su borde superior derecho y el inferior izquierdo que, aproximadamente, corresponde con dicha zona. La precisión a la hora de equiparar el área del círculo de obtención y la caja de comprobación no es muy significativa, puesto que la mayor parte de la diferencia entre ambas figuras corresponde a las aguas circundantes a la península. Así pues, con dicha comparación, se obtiene son erróneas 529 de las localizaciones ofrecidas por *GeoNames*.

En la Tabla 3-5 podemos observar algunos ejemplos del resultado ofrecido por *GeoNames* ante diversas cadenas de texto. Hay que resaltar que cuando se indica que la latitud y longitud valen 0, realmente *GeoNames* indica que no ha podido obtener localización alguna. Sin embargo, en el algoritmo implementado, hemos optado por asignar unos valores arbitrarios, a tal situación.

Texto <i>user.location</i>	Latitud	Longitud
Granada	12.11667	-61.6667
Daydreaming	0	0
Vlc	-13.6186	-172.487
Silent Hill	18.19152	-77.4592
Málaga – Madrid	0	0
ð — ð —®ð —±ð —¿ð —¶ð —±	0	0
C/ Hontanilla, 1 – Alcazarán	0	0
Alicante Spain	0	0
Entre Granada y Málaga	0	0

Tabla 3-5 Ejemplos de localizaciones ofrecidas por *GeoNames*

Se ha detectado que la ambigüedad de los usuarios al especificar su localización, puede dar lugar a errores en la asignación de coordenadas. El usuario que indicó “*Granada*” se refería a la ciudad española, sin embargo, *GeoNames* nos ofrece la localización del país insular caribeño. Esta situación se producirá igualmente si el usuario indica “*Córdoba*”, puesto que *GeoNames* tiene una lista de posibles ubicaciones y ofrece la que está en primera posición. En este caso, el resultado sería el correspondiente a la ciudad argentina, puesto que la ciudad española está más abajo en dicha lista.

En cuanto a la granularidad con la que los usuarios suelen especificar su posición, se observa que la mayoría indican la ciudad en la que viven y, en menor medida, indican la región o únicamente el País. La forma en que *GeoNames* asocia unas coordenadas a localizaciones genéricas, se desconoce. Por ejemplo, los usuarios que simplemente han indicado “*España*” en su localización, han sido ubicados en las coordenadas, 40° Norte y 4° Oeste, que se corresponden con un punto de la provincia de Toledo cercano al pueblo de Yuncillos.

También vemos que algunos textos, que se presupone que probablemente para el usuario son localizaciones imaginativas, como “*Silent Hill*”, son identificados en ocasiones por *GeoNames* como lugares reales, a los que los usuarios probablemente no querían referirse.

Se observa también que las cadenas que contienen emoticonos o caracteres no alfanuméricos provocan que *GeoNames* no ofrezca resultado.

Adicionalmente se ha elaborado un programa que permite dibujar en un mapa la ubicación de las coordenadas extraídas por *GeoNames* y cuyo código se encuentra en el Anexo III.

En la Figura 3-2, se representan las ubicaciones obtenidas. Se observa que algunas de ellas corresponden a lugares fuera de España. Esto, aunque pueda parecer erróneo, se debe a la forma en la que se ha especificado el filtro de búsqueda a la función *search* de la API de Twitter. El hecho de que se indicara que los tweets estuvieran escritos en español, no ha impedido que algunos de ellos se escribieran desde Portugal o Francia. Por otro lado, algunos puntos que pudieran parecer también erróneos, por situarse en zonas aparentemente sobre el mar, se corresponden con diversas islas, como la isla de Alborán, ubicada en el estrecho, al sur de Málaga.



Figura 3-2 Representación gráfica de las localizaciones ofrecidas por *GeoNames* de *tweets_spain*

Por último, indicar que las cadenas de texto que se han remitido a *GeoNames* no han sido preprocesadas de ninguna manera. La eliminación de los caracteres no alfanuméricos de las cadenas de texto, podría haber producido que *GeoNames* ofreciera coordenadas para algunas de ellas, como por ejemplo “*Alicante | España*”, que, de este modo no ha conseguido. Sin embargo, *GeoNames* es bastante flexible a la hora de procesar las cadenas de texto y no es necesario otro tipo de procesamiento adicional como la *tokenization*. También permite la obtención de coordenadas correctas cuando los lugares indicados son del tipo “*Camp Nou*” o “*Euskal Herria*”.

3.3.2 Algoritmos basados en Machine-Learning.

Para ejemplificar el uso de este tipo de algoritmos, vamos a utilizar inicialmente el conjunto de datos *tweets_spain_ext*. Consultando en la base de datos MongoDB observamos que, del total de tweets que contiene, un porcentaje significativo de los usuarios que los han publicado manifiestan ser de Madrid, o Barcelona. La localización que han indicado los usuarios sólo es suficientemente precisa para suponer que, al menos, viven en las citadas provincias. El conjunto de tweets perteneciente a los usuarios de cada provincia lo dividiremos en dos partes. Una primera parte, con el 75% de los tweets, constituirá el conjunto de entrenamiento con el que serán entrenados los algoritmos de Machine-Learning. La segunda parte, con el 25% de los tweets, la utilizaremos como conjunto de test, para verificar si el algoritmo es capaz de predecir la ubicación correcta.

Si quisiéramos obtener la localización con mayor precisión, necesitaríamos un conjunto de datos de entrenamiento con una suficiente cantidad de usuarios que, en lugar de indicar simplemente su ciudad o provincia, indicasen las coordenadas en las que viven. Con esa información, por otro lado, podríamos plantearnos utilizar algoritmos de regresión, en lugar de clasificación. Sin embargo, el hecho es que el número de tweets que están asociados a unas coordenadas es demasiado bajo. Con porcentajes inferiores al 1%, es necesario recopilar millones de tweets para que los conjuntos de entrenamiento cuenten con un número suficiente de datos, tal y como se mencionó en el punto 3.2.5.

En la Tabla 3-6 se observan el número de tweets que conforman cada conjunto. Los usuarios que indicaron en su perfil la localización “*Madrid*”, o “*Madrid, Comunidad de Madrid*” son los que se utilizarán para representar a dicha provincia. Mientras que, para identificar a Barcelona, se utilizarán los correspondientes a las cadenas de texto “*Barcelona*”, “*Barcelona, España*” y “*Barcelona, Cataluña*”.

Localización del usuario (<i>user.location</i>)	Número de tweets
“Madrid”	1014
“Madrid, Comunidad de Madrid”	871
Total tweets asignados a Madrid	1885
Total usuarios de Madrid	1375
“Barcelona”	466
“Barcelona, España”	629
“Barcelona, Cataluña”	86
Total tweets asignados a Barcelona	1181
Total usuarios de Barcelona	860

Tabla 3-6 Conjuntos de entrenamiento y test para aplicación de algoritmos ML

Únicamente escogemos las dos ciudades de donde proceden el mayor número de tweets para que los algoritmos tengan el mayor número de datos de entrenamiento que nos sea posible. Los algoritmos de clasificación que vamos a utilizar, podrían perfectamente trabajar con un mayor número de clases. Sin embargo, es recomendable que la cantidad de los datos de entrenamiento de cada una de las clases sean de un orden de magnitud similar.

En el Anexo IV se incluye el código que se ha empleado en las distintas fases necesarias para llevar a cabo la localización utilizando diferentes algoritmos de ML.

En primer lugar, ha sido necesario extraer de MongoDB los tweets correspondientes a las localizaciones que se van a utilizar. Se han probado diversos métodos para ellos y, finalmente se ha optado por almacenar los datos, procedentes de una consulta de agregación, en un fichero tipo CSV. El almacenamiento en estructuras JSON, nos facilitaría el almacenamiento de mucha información que podría ser relevante en otros casos, pero que, al mismo tiempo, dificulta su transformación a un vector de características. Los múltiples pares clave-valor y su anidamiento no facilitan la extracción del texto de los tweets, que es lo que simplemente se precisa. Así pues, entre las funciones del código dedicadas a la tarea de extraer los datos, veremos alguna que nos permite el almacenamiento en JSON y otras dos que nos los permiten en CSV. De estas últimas, una utiliza las consultas por agregación¹, mientras que la otra utiliza una llamada a una herramienta externa de MongoDB: *mongoexport*.

¹ La librería de Python *pymongo* nos permite realizar este tipo de consultas en las que se van concatenando las instrucciones de proceso, como búsqueda, ordenado, etc. Para ello se utiliza el método *aggregate* de la clase *collection*. La web con la documentación de esta librería se puede encontrar en “PyMongo 4.0.1 Documentation” <https://pymongo.readthedocs.io/en/stable/> [Disponible a 23/12/2021]

Una vez almacenados los datos que nos interesan en JSON o CSV, ya no es necesario operar con la base de datos, simplemente creamos estructuras *Dataframe* o *Series* procedentes de la librería de Python *pandas*¹. Seguidamente el texto en crudo de los tweets tiene que ser procesado previamente antes de ser empleado por los algoritmos ML. En nuestro caso, hemos implementado las siguientes acciones:

- Convertir todo el texto en minúsculas.
- Eliminar los códigos de nuevas líneas.
- Eliminar todas las menciones, del tipo @usuario.
- Eliminar cualquier palabra que en su interior contenga números.
- Eliminar cualquier palabra con menos de tres letras.
- Eliminar los emojis.
- Eliminar los signos de puntuación.

El objetivo de todas estas medidas es eliminar del texto de los tweets todas aquellas palabras, signos o símbolos poco significativos. No nos interesan las *stopwords* o, lo que es lo mismo, las palabras como “y”, “a”, “si”, “no” puesto que no tienen significado relevante por sí mismas. Tampoco nos interesan las abreviaturas que utiliza el usuario al tratar de acortar el texto escrito, como por ejemplo escribir “q”, en lugar de “que”.

Sería posible incluir en esta etapa otras medidas, como la eliminación de las palabras #hashtags completas, pero nosotros simplemente hemos eliminado el símbolo #, manteniendo la palabra. Es posible que los hashtags más frecuentes en una ubicación sean distintos de los de otra, de modo que lleguen a ser relevantes.

Una vez hemos realizado este proceso en las cadenas de texto de los tweets, procedemos a obtener las palabras que lo constituyen. Para ello utilizamos la clase *CountVectorizer*², de la librería *sklearn.feature_extraction*. Esta función nos permitirá realizar el proceso de *tokenization* y, además, es posible indicarle un conjunto de *stopwords* para que las ignore. Es decir, a partir del método *fit_transform* podemos conseguir una matriz, con tantas filas como documentos distintos (tweets) formen parte del conjunto, o *corpus*, de interés y con tantas columnas como palabras distintas existan en dicho *corpus*. Los datos de dicha matriz indican el número de veces que una palabra concreta aparece en cada uno de los tweets. Las palabras que se encuentren incluidas en el conjunto de las *stopwords*, no se consideran, reduciendo así el número de columnas de la matriz.

En Python, las *stopwords* las hemos obtenido de la librería *nltk.corpus*³. Se trata de un total de 6800 palabras, 313 en español, entre las que encontramos, a modo de ejemplo, “la”, “de”, “habido”, “tuyo”, “algo”, “fue”, “tengan”, etc. Observamos que se incluyen en dicho conjunto artículos, preposiciones, pronombres y, también, muchas formas verbales que habitualmente se utilizan para perífrasis. Las *stopwords* las introducimos en una estructura de datos tipo *lista* y, si lo deseamos, podríamos incluir o eliminar aquellas que nos parezcan convenientes. Por comparar entre idiomas, las *stopwords* para el idioma inglés son 179, lo que parece lógico puesto que en los idiomas latinos se tienen en cuenta las conjugaciones o flexiones verbales, lo cual incrementa notablemente el cómputo.

¹ La web con la documentación de esta librería se puede encontrar en “pandas” <https://pandas.pydata.org/> [Disponible a 23/12/2021]

² La descripción de esta clase se puede encontrar en “sklearn.feature_extraction.text.CountVectorizer” https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html [Disponible a 23/12/2021]

³ Las funcionalidades que nos ofrece esta librería, incluyendo las *stopwords* en diversos idiomas, se pueden encontrar en “Accessing Text Corpora and Lexical Resources” <https://www.nltk.org/book/ch02.html>. [Disponible a 23/12/2021]

Al utilizar la función *fit transform* sobre los textos de los tweets de los usuarios ubicados en Madrid, hemos obtenido una matriz de 1885 filas (tantas como número de tweets analizados, tal y como aparece en la Tabla 3-6), y un total de 7469 columnas que constituyen todas las palabras que no fueron filtradas en la etapa de preprocesamiento. A partir de esta matriz, podemos obtener un vector en el que se muestren de forma las palabras con el número total de apariciones en el *corpus*. Esto nos dará una idea de cuáles son las más utilizadas. En la Tabla 3-7 podemos observar cuáles han sido las palabras más utilizadas y el número de apariciones de las mismas para los usuarios de Madrid y Barcelona.

Curiosamente, una de las palabras más utilizadas en los tweets es el propio nombre de la ciudad en la que se ubican los usuarios. También podemos ver otras palabras que aparecen con gran frecuencia en ambos conjuntos como “*gracias*” o “*ser*”. Estas palabras no nos van a ayudar a identificar si un tweet es de un usuario de Barcelona o de Madrid, puesto que la frecuencia de utilización será similar. Por ello, podríamos incluirlas en el conjunto de *stopwords*. Un estudio detenido nos podría ayudar a determinar cuáles son esas palabras que utilizan los usuarios independientemente de cuál sea su ubicación. Si consiguiéramos determinarlas, sería posible reducir la dimensión de los vectores de características con que, más tarde, trabajarán los algoritmos de ML.

Tweets de usuarios de Madrid				Tweets de usuarios de Barcelona			
Palabra	Total	Palabra	Total	Palabra	Total	Palabra	Total
“gracias”	68	“madrid”	60	“así”	38	“ser”	37
“hoy”	58	“ser”	49	“pues”	35	“bien”	34
“día”	48	“ver”	47	“hace”	31	“barcelona”	30
“hace”	46	“años”	43	“gracias”	30	“solo”	29

Tabla 3-7 Palabras más utilizadas en los tweets de los usuarios de Madrid y Barcelona.

Al igual que hemos obtenido las palabras con más apariciones, también podemos obtener aquellas que se utilizan menos. Si en todo el *corpus* una determinada palabra únicamente se ha utilizado un reducido número de veces, no la podemos considerar significativa, así que podemos eliminar aquellas con una frecuencia de aparición inferior a un cierto valor. En nuestro caso, hemos utilizado un 0.03% que, aproximadamente, suprimirá las palabras que aparezcan menos de 5 veces para el caso de Madrid.

Para obtener la frecuencia de aparición, primero tenemos que contar todas las palabras del corpus. En el caso de los tweets de Madrid, contamos con 15426 palabras, mientras que para Barcelona tenemos 8609. Si eliminamos las palabras que aparecen con una frecuencia inferior al 0.03%, en los tweets de Madrid nos quedamos con 549 palabras, mientras que en los de Barcelona, nos quedamos con 645 palabras. Ahora bien, no queremos eliminar las palabras del conjunto de Madrid si esas mismas palabras sí son significativas en el conjunto de Barcelona, y viceversa. Así pues, teniendo en cuenta esta consideración, las palabras en el conjunto de Madrid se quedan en 790 y en el de Barcelona en 784.

Al eliminar las palabras poco significativas, es probable que algún tweet se haya quedado sin palabras. Al suprimir estos tweets vacíos nos quedan, en el conjunto de Madrid, 1635 tweets y, en el conjunto de Barcelona, 1039 tweets.

La cantidad de filas y columnas resultantes dependerán del umbral de corte seleccionado y afectará significativamente al consumo de memoria y a la velocidad de proceso. Por ejemplo, la

Si representamos mediante un gráfico de barras el número de apariciones de las veinte palabras más utilizadas en la colección de tweets de Barcelona y Madrid, obtenemos la Figura 3-5. Estos gráficos han sido obtenidos con la función `plot.barh` de la clase `Series` de la librería `pandas` de Python.

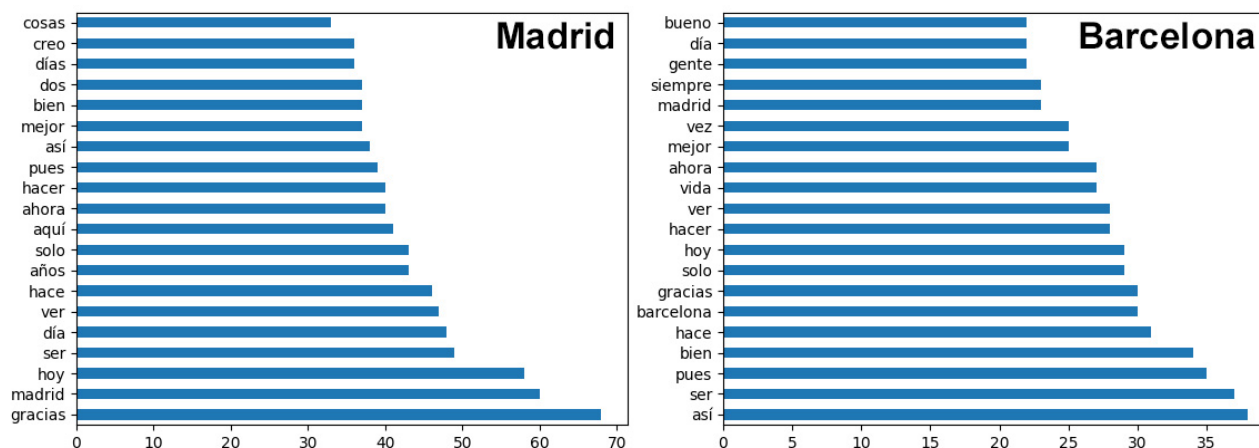


Figura 3-5 Palabras más utilizadas en los conjuntos de tweets

Se observa que algunas palabras aparecen en los dos gráficos, como por ejemplo “gracias”, “día”, “ser”, “ver”, etc. Podríamos plantearnos el eliminar aquellas palabras que tengan una frecuencia similar en ambos grupos por considerar que no son significativas a la hora de clasificar un tweet como de una provincia, o de otra. Con ello conseguiríamos reducir el número de columnas de la matriz de características y facilitar el trabajo al algoritmo ML encargado de la clasificación. Sin embargo, hemos preferido no hacerlo porque las pequeñas variaciones de frecuencia en el uso de las palabras podrían ser significativas. Vemos por ejemplo que la palabra “Madrid” está en las dos listas, aunque con diferentes frecuencias de aparición. No parece muy lógico eliminar esa palabra puesto que, como se aprecia, en el conjunto de las palabras de Madrid, aparece más frecuentemente que en el conjunto de las de Barcelona. Un aspecto que debería ser significativo en la clasificación. De igual modo, este razonamiento podría extenderse también a otras palabras.

Una vez decididas cuáles serán las palabras que vamos a considerar como características, construimos los conjuntos de datos de test y de entrenamiento. Para ello, utilizamos la librería `numpy` de Python y creamos la matriz X que contiene tantas columnas como palabras hemos considerado significativas en los conjuntos de Madrid y Barcelona. De las 790 palabras del conjunto de Madrid y las 784 del conjunto de Barcelona, obtenemos un total de 872 palabras diferentes. Lo que significa que ambos conjuntos tenían un total de 702 palabras en común. El número de filas de la matriz X se corresponde con el total de tweets de los conjuntos de Madrid y Barcelona, una vez eliminados aquellos tweets que no tienen palabras significativas, es decir, $1635+1039=2674$.

El siguiente paso es obtener los conjuntos de entrenamiento y test. Para ello utilizamos la función `train_test_split`¹ de la librería `sklearn.model_selection` de Python, a la cual le indicamos que el tamaño del conjunto de entrenamiento será el 70% de las muestras, mientras que el de conjunto de test incluirá el 30%. En nuestro caso, eso supone 1871 muestras en el conjunto de entrenamiento y 803 en el de test. Uno de los parámetros significativos en esta función es

¹ La descripción de esta función puede encontrarse en “sklearn.model_selection.train_test_split” https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html [Disponible a 23/12/2021]

random_state, cuyo valor podemos fijar para poder reproducir en sucesivas ocasiones el reparto aleatorio de las muestras en los conjuntos de test y entrenamiento.

Para evitar que, de forma aleatoria, el conjunto de test incluya un grupo de muestras con mucho mayor número de clases de un tipo que de otro, incluimos el parámetro *stratify*. De ese modo, conseguiremos que el porcentaje de muestras en el subconjunto de test para la clase Madrid, sea el mismo que en el conjunto de muestras completo. En nuestro caso, el conjunto de test cuenta con 491 muestras que pertenecen a la clase Madrid, lo que supone un 61.1%. Del mismo modo, hay 312 muestras que pertenecen a la clase Barcelona, que suponen un 38.9%. Estos porcentajes son similares a los del total de muestras, por lo que inicialmente asumimos trabajar con un conjunto no balanceado.

Puesto que para muchos de los algoritmos de ML es necesaria la normalización de los valores en la matriz de características, utilizaremos la clase *StandardScaler*¹ de la librería *sklearn.preprocessing*. Con el método *fit* obtenemos la media y la desviación típica para cada una de las características. Con el método *transform* normalizamos cada uno de los valores de la matriz al restarlos de la media y dividir el resultado por la desviación típica. En este punto, es importante notar que la media y la desviación típica se obtienen del conjunto de entrenamiento, sin incluir las muestras del conjunto de test. Con estos valores normalizaremos ambos conjuntos, tanto el de entrenamiento, como el de test. La idea es que las muestras de test hay que tratarlas como si fueran muestras nuevas de las que nunca antes hubiéramos tenido noticia, por lo que no podríamos incluirlas en el cálculo de la media y desviación típica que se aplican a los conjuntos de entrenamiento.

Finalmente podemos aplicar distintos algoritmos de ML. Seguidamente, tratamos algunas de las peculiaridades observadas en cada uno de ellos.

3.3.2.1 Regresión Logística.

Para aplicar este algoritmo utilizamos la clase *LogisticRegression*² de la librería *sklearn.linear_model* de Python. Uno de los parámetros a ajustar es el valor de *C*. Con valores altos de este hiperparámetro conseguiremos un modelo que se aproxime más a las muestras de entrenamiento. Esto requerirá, para valores extremos, aumentar el número máximo de iteraciones (*max_iter*) que realizará el algoritmo. Sin embargo, sabemos que un modelo demasiado ajustado a las muestras de entrenamiento no responderá bien ante las muestras de test. Este fenómeno se denomina sobreajuste (*overfitting*), tal y como mostramos en la Figura 2-12. Por otro lado, unos valores demasiado bajos de *C* proporcionarán un modelo muy general, que no ofrecerá una alta precisión, ni en la clasificación de las muestras de entrenamiento, ni en las de test. A este fenómeno, contrario al del sobreajuste se le denomina, en inglés *underfitting*.

En nuestro caso, hemos ejecutado el algoritmo para diversos valores de *C* midiendo la precisión de la clasificación de los conjuntos de test y de entrenamiento para poder observar los efectos anteriormente citados.

En la Figura 3-6 Influencia del hiperparámetro *C* en la exactitud, podemos ver que para valores de *C* superiores a 1 obtendremos un modelo que clasifica correctamente más del 90% de las muestras de entrenamiento, pero que únicamente clasifica correctamente el 60% de las muestras de test. Si elegimos un valor más bajo de *C*, obtendremos una precisión en la clasificación de las muestras de entrenamiento menor, pero, para ciertos valores, aumentamos la precisión de la clasificación de las muestras en el conjunto de test. Podemos decir que para valores de *C* mayores de 0.01 entramos en la zona de sobreajuste, mientras que, para valores de

¹ La descripción de esta clase se puede encontrar en “*sklearn.preprocessing.StandardScaler*” <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html> [Disponible a 23/12/2021]

² La descripción de esta clase se puede encontrarse en “*sklearn.linear_model.LogisticRegression*” https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html [Disponible a 23/12/2021]

C menores de 0.001, la caída de ambas curvas nos indica que estamos entrenando escasamente al modelo (“underfitting”).

Por otro lado, para descartar que diferencia de exactitud entre los resultados de test y entrenamiento, pudiera tratarse de un hecho accidental relacionado con una selección particular de las muestras que conforman cada conjunto, podemos dividir las muestras especificando otro valor de *random_state* en la función *train_test_split*. Sin embargo, variando dicho valor y ejecutando el algoritmo con distintos conjuntos de test y de entrenamiento, seguimos sin sobrepasar un 67% de éxito en la precisión de la clasificación del conjunto de test.

Selección del valor de C

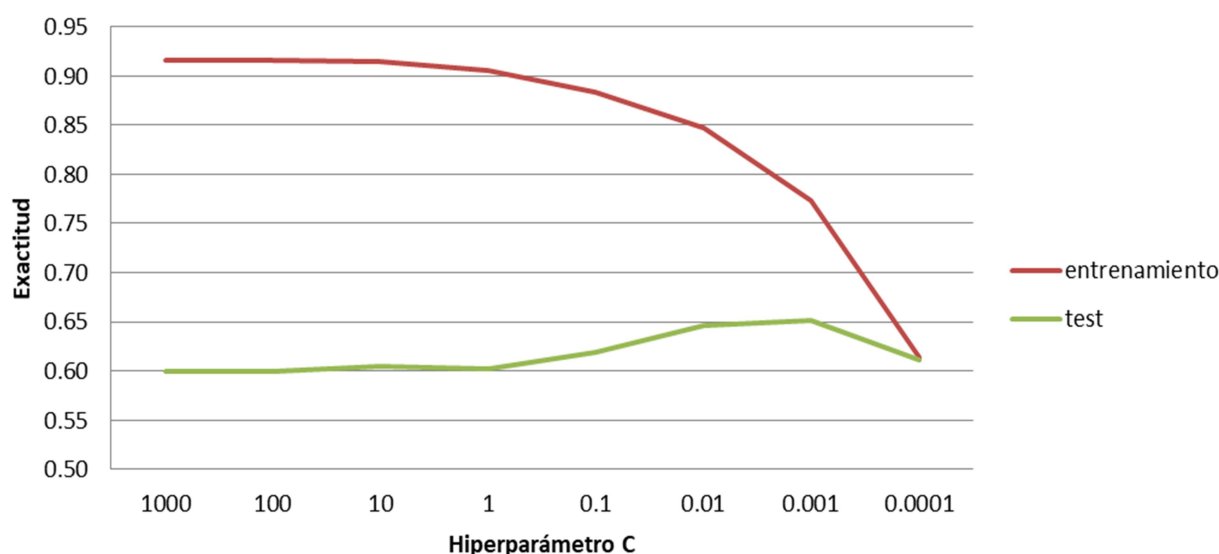


Figura 3-6 Influencia del hiperparámetro C en la exactitud

En cualquier caso, resulta significativo el hecho de que el algoritmo clasifica con bastante éxito las muestras de entrenamiento, obteniendo exactitudes del 85%. Su escaso desempeño cuando se enfrenta a las muestras de test, podría indicar que el modelo debe perfeccionarse, puesto que las muestras de entrenamiento no resultan plenamente representativas de la realidad. Para ello se podría tratar de aumentar el número de muestras, o bien modificar el vector de características, es decir, el número de palabras del vocabulario. Aún así, es posible que las diferencias de vocabulario entre los usuarios de ambas localizaciones no sean suficientemente relevantes como para que podamos clasificar las muestras con eficacia.

Otras de las funciones útiles de la librería *sklearn.metrics* de Python son *classification_report*¹ y *confusion_matrix*². La primera nos ofrece como resultado la Tabla 3-8, donde se nos ofrecen diversos valores de interés como la precisión, la exhaustividad, el valor F1 y la exactitud³. La precisión responde a la pregunta ¿qué proporción de las predicciones fue realizada correctamente? En nuestro caso, la precisión obtenida es de un 66% para las muestras de Madrid y 69% para las de Barcelona. El resultado es el cociente del valor de los positivos

¹ La descripción de esta función se puede encontrar en “sklearn.metrics.classification_report” https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html [Disponible a 23/12/2021]

² La descripción de esta función se puede encontrar en “sklearn.metrics.confusion_matrix” https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html [Disponible a 23/12/2021]

³ Precisión (“precision”), exhaustividad (“recall”) y exactitud (“accuracy”) son las traducciones que utilizamos en este documento. Una explicación de estos conceptos se puede encontrar en J. Martínez Heras “Precision, Recall, F1, Accuracy en clasificación” <https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/> [Disponible a 23/12/2021]

verdaderos (TP) entre la suma de todos los positivos, tanto los positivos verdaderos como los positivos falsos (FP).

$$\text{Precisión} = \frac{TP}{TP + FP}$$

	Precision	Recall	F1	Support
0 (Madrid)	0.66	0.93	0.77	491
1 (Barcelona)	0.69	0.25	0.37	312
Accuracy			0.67	803

Tabla 3-8 Informe de clasificación con Regresión Logística en Madrid-Barcelona.

Por otro lado, también tenemos datos de la exhaustividad (“*recall*”), que sirve para responder a la pregunta ¿Qué proporción de las muestras de una determinada clase fue identificada correctamente? El valor que se ofrece es el cociente del valor de los positivos verdaderos (TP) entre la suma de los positivos verdaderos (TP) y los negativos falsos (FN).

$$\text{Exhaustividad} = \frac{TP}{TP + FN}$$

En nuestro caso, vemos que el 93% de las muestras de Madrid se identificaron correctamente, pero únicamente se consiguió con el 25% de las de Barcelona.

La consideración de ambos valores es importante para la evaluación de un modelo. Habitualmente los esfuerzos en incrementar la precisión producirán una disminución en la exhaustividad. Una de las métricas que trata de combinar la información de estas dos medidas es el valor de la columna *F1*, que nos ofrece la media armónica de las dos columnas anteriores.

Otra de las medidas que nos ofrece la función *classification_report* es la medida de exactitud (“*accuracy*”), cuyo valor se calcula mediante el cociente entre las predicciones correctas y el total de predicciones, según la expresión:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Donde *TN* son los negativos verdaderos y *FP* son los positivos falsos.

La función *confusion_matrix* nos ofrece precisamente los valores TP, TN, FP y FN. En nuestro caso hemos obtenido los valores de la Tabla 3-9.

		Clases predichas	
		0	1
Clases reales	0 (Madrid)	(TN) 456	(FP) 35
	1 (Barcelona)	(FN) 234	(TP) 78

Tabla 3-9 Matriz de confusión con Regresión Logística en Madrid-Barcelona.

Podemos observar que, cuando el modelo predice que un tweet pertenece a Barcelona, hay un 75% de probabilidad de que, realmente, sea de Madrid.

Sin embargo, si trabajamos con un conjunto balanceado de muestras los resultados son diferentes. Si en lugar de trabajar con los 1885 tweets del conjunto de Madrid, eliminamos unos

cuantos para tener exactamente los 1181 que tenía el conjunto de Barcelona conseguimos balancear las muestras iniciales exactamente al 50%. Una vez se aplica el preprocesamiento nos quedamos con 1043 tweets de Madrid y 1048 de Barcelona, que sigue manteniendo el equilibrio. Seguidamente establecemos el conjunto de entrenamiento con 733 de Barcelona y 730 de Madrid. Y el de test con 315 y 313 tweets. Los resultados obtenidos en estas condiciones son los que se muestran en la Tabla 3-10 y en la Tabla 3-11.

	Precision	Recall	F1	Support
0 (Barcelona)	0.68	0.69	0.68	315
1 (Madrid)	0.68	0.67	0.68	313
Accuracy			0.68	628

Tabla 3-10 Informe de clasificación con Regresión Logística y balanceo de muestras.

		Clases predichas	
		0	1
Clases reales	0 (Barcelona)	(TN) 218	(FP) 97
	1 (Madrid)	(FN) 104	(TP) 209

Tabla 3-11 Matriz de confusión con Regresión Logística y balanceo de muestras.

Aunque la exactitud no parece haber aumentado significativamente, ahora tenemos un 68% y anteriormente un 67%, hay que tener en cuenta la referencia (“*baseline*”) sobre la que se calcula. Inicialmente teníamos un conjunto de muestras en las que el 61% procedían de Madrid y el algoritmo lograba una exactitud del 67%. Ahora conseguimos un 68% pero sobre un conjunto en el que hay un 50% de muestras de cada tipo.

Realizando el mismo proceso pero utilizando esta vez los tweets de los conjuntos “*tweets_madrid_ext*” y “*tweets_mexico_ext*” obtenemos resultados similares, tal y como se puede ver en la Tabla 3-12 y Tabla 3-13.

	Precision	Recall	F1	Support
0 (Madrid)	0.66	0.72	0.69	1254
1 (México)	0.70	0.65	0.67	1290
Accuracy			0.68	2544

Tabla 3-12 Informe de clasificación con Regresión Logística en Madrid-México.

		Clases predichas	
		0	1
Clases reales	0 (Madrid)	(TN) 902	(FP) 352
	1 (México)	(FN) 456	(TP) 834

Tabla 3-13 Matriz de confusión con Regresión Logística en Madrid-México.

3.3.2.2 Máquinas de Vectores Soporte (SVM).

Para aplicar este algoritmo utilizamos la clase *SVC*¹ de la librería *sklearn.svm* de Python. Esta función tiene unos cuantos parámetros de interés. El primero de ellos es “*kernel*”, que nos permite definir el tipo de hiperplanos que separarán las clases. En nuestro caso utilizaremos el valor “*rbf*” que nos ayudará a delimitar las clases en aquellos casos en que la utilización de planos lineales o polinómicos no sea la opción más adecuada. De este modo utilizaremos planos diferentes de los utilizados en la regresión logística, que utiliza planos lineales.

Al igual que para el algoritmo de regresión logística, obtenemos el informe de clasificación y la matriz de confusión para la clasificación de los conjuntos balanceados de Madrid y Barcelona. Los resultados los podemos ver respectivamente en la Tabla 3-14 y la Tabla 3-15.

	Precision	Recall	F1	Support
0 (Barcelona)	0.64	0.70	0.67	315
1 (Madrid)	0.66	0.60	0.63	313
Accuracy			0.65	803

Tabla 3-14 Informe de clasificación con SVM para Madrid-Barcelona.

		Clases predichas	
		0	1
Clases reales	0 (Barcelona)	(TN) 220	(FP) 95
	1 (Madrid)	(FN) 125	(TP) 188

Tabla 3-15 Matriz de confusión con SVM para Madrid-Barcelona.

También realizamos el mismo proceso, pero utilizando esta vez los tweets de los conjuntos “*tweets_madrid_ext*” y “*tweets_mexico_ext*”. Los nuevos resultados se recogen en la Tabla 3-16 y en la Tabla 3-17.

¹ La descripción de esta clase se puede encontrar en “*sklearn.svm.SVC*” <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html?highlight=svc#sklearn.svm.SVC> [Disponible a 23/12/2021]

	Precision	Recall	F1	Support
0 (Madrid)	0.65	0.78	0.75	1254
1 (México)	0.74	0.59	0.66	1290
Accuracy			0.69	2544

Tabla 3-16 Informe de clasificación con SVM para Madrid/México.

		Clases predichas	
		0	1
Clases reales	0 (Madrid)	(TN) 981	(FP) 273
	1 (México)	(FN) 525	(TP) 765

Tabla 3-17 Matriz de confusión para SVM para Madrid/México.

En cuanto a los hiperparámetros relevantes en el algoritmo SVM, tenemos que destacar dos. El primero de ellos, se denomina “*Gamma*” y sirve para determinar cuántos puntos se van a considerar para calcular las distancias y ubicar los hiperplanos. Un valor alto de gamma indica que solo los puntos más cercanos serán tenidos en cuenta. Recordemos que SVM pretende ubicar los hiperplanos lo más lejos posible de las muestras. Con “*Gamma*” especificamos qué muestras vamos a considerar para calcular la ubicación del hiperplano.

El segundo es “*C*”, que es el parámetro de penalización y que cuantifica el peso de los fallos en la clasificación. Indica cuántos fallos de clasificación se toleran en las fronteras del modelo. Sirve para controlar el equilibrio entre, por un lado, una frontera de decisión suavizada que permite lograr un modelo más generalista, y, por otro, una mayor precisión en la clasificación de los datos de entrenamiento. Cuando *C* es alto, clasificará mejor las muestras de entrenamiento, pero con el riesgo de caer en el sobreajuste.

Siempre es posible ajustar estos parámetros en nuestro modelo probando manualmente las distintas combinaciones, pero existen otras opciones. En nuestro caso, hemos utilizado la función *GridSearchCV* de la librería *sklearn.model_selection* de Python. Con ella podemos especificar los diversos valores que queremos asignar a los diferentes parámetros para que pruebe todas las combinaciones posibles y nos ofrezca finalmente cuál es la combinación que nos otorga el mejor valor de exactitud (“*accuracy*”).

La Figura 3-7 nos muestra una representación gráfica del valor de la exactitud (“*accuracy*”) obtenida con el algoritmo SVM sobre el conjunto de datos de Madrid y México. Se ha utilizado la función *GridSearchCV* para ejecutar el algoritmo con diversas combinaciones de valores de “*C*” y de “*Gamma*”. Hemos elegido colores fríos para los valores de exactitud más bajos y colores cálidos para los más altos. De este modo, podemos decidir cuáles son los parámetros más adecuados para nuestro modelo.

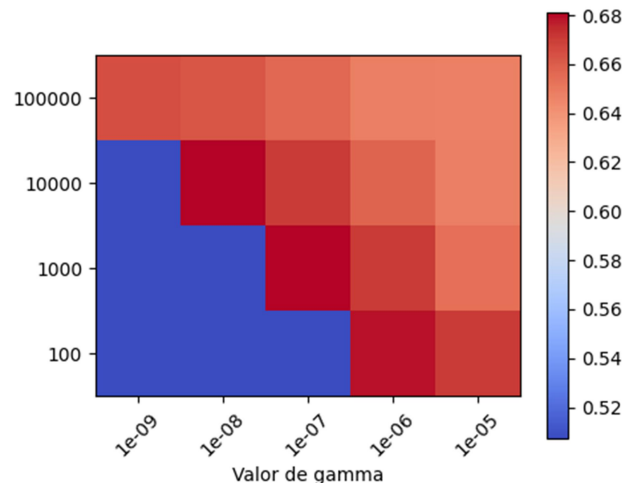


Figura 3-7 Representación gráfica de la exactitud en función de C y de Γ .

En cuanto a la selección de los hiperparámetros en este caso, podemos observar en la Figura 3-7, que la mejor combinación se obtiene para los valores de $\Gamma = 1e-07$ y $C = 10000$.

3.3.2.3 Bosque aleatorio.

Continuando con la clasificación del conjunto balanceado de Madrid-Barcelona, utilizamos ahora la clase *RandomForestClassifier*¹ de la librería *sklearn.ensemble* de Python. Los hiperparámetros de mayor interés son “*n_estimators*” que establece el número de árboles de decisión que se van a utilizar. Cada uno de estos árboles trabajará sobre un subconjunto de las muestras. El otro parámetro es “*max_depth*” que indica la máxima profundidad de ramificación de los árboles. Para determinar cuál es la tupla de hiperparámetros más adecuada, volveremos a utilizar, como se hizo en SVM, la función *GridSearchCV*.

Los resultados obtenidos para el informe de clasificación y la matriz de confusión, se pueden observar en la Tabla 3-18 y la Tabla 3-19, respectivamente.

	Precision	Recall	F1	Support
0 (Barcelona)	0.58	0.67	0.62	315
1 (Madrid)	0.60	0.50	0.55	313
Accuracy			0.59	628

Tabla 3-18 Informe de clasificación con Bosque Aleatorio para Madrid/Barcelona.

¹ La descripción de esta clase se puede encontrar en “*sklearn.ensemble.RandomForestClassifier*” <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html> [Disponible a 23/12/2021]

		Clases predichas	
		0	1
Clases reales	0 (Barcelona)	(TN) 212	(FP) 103
	1 (Madrid)	(FN) 156	(TP) 157

Tabla 3-19 Matriz de confusión con Bosque Aleatorio para Madrid/Barcelona.

Para el caso de los conjuntos de tweets de Madrid y México, los resultados obtenidos se muestran en la Tabla 3-20 y la Tabla 3-21.

	Precision	Recall	F1	Support
0 (Madrid)	0.64	0.72	0.68	1254
1 (México)	0.69	0.61	0.65	1290
Accuracy			0.66	2544

Tabla 3-20 Informe de clasificación con Bosque Aleatorio para Madrid/México.

		Clases predichas	
		0	1
Clases reales	0 (Madrid)	(TN) 897	(FP) 357
	1 (México)	(FN) 499	(TP) 791

Tabla 3-21 Matriz de confusión con Bosque Aleatorio para Madrid/México.

En la Figura 3-8 se muestra la representación obtenida para seleccionar los valores adecuados de los hiperparámetros $n_estimators = 3$ y $max_depth = 250$.

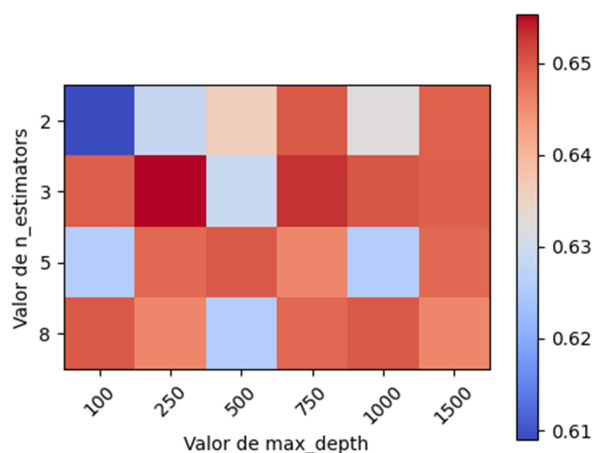


Figura 3-8 Representación gráfica de la exactitud en función de $n_estimators$ y max_depth

Por otra parte, hemos identificado las palabras (características) que, una vez finalizado el trabajo del clasificador, tienen coeficientes más grandes y, por tanto, han sido consideradas como

más significativas en la clasificación. El efecto de estos coeficientes es que, si un tweet contiene alguna de estas palabras, se le asignará una clase u otra con mayor probabilidad.

En la Figura 3-9 se muestran las 8 palabras más significativas y el número de apariciones de cada una en el conjunto de tweets de Madrid y de Ciudad de México. Se observa que algunas de las palabras, como “*españa*” y “*madrid*” son más frecuentes en los tweets procedentes de Madrid, mientras que “*méxico*” o “*pesos*” son más habituales en los de Ciudad de México. Estas palabras no son sorprendentes, sin embargo, otras sí lo son. Así por ejemplo “*cdmx*” parece ser la abreviatura que se utiliza para referirse a Ciudad de México en dicha ciudad y, obviamente no se utiliza en Madrid. Por otro lado, “*salariorosa*”, hace referencia a una medida política puesta en marcha por el gobierno mexicano por la cual se abona cierta cantidad de dinero a las amas de casa mexicanas. Esta medida fue muy comentada en Twitter en la fecha en la que se recogieron los datos y se identifica con el hashtag #*SalarioRosa*. Otras palabras como “*saludos*” o “*apoyo*” quizás sean modismos más frecuentes en una ciudad que en otra.

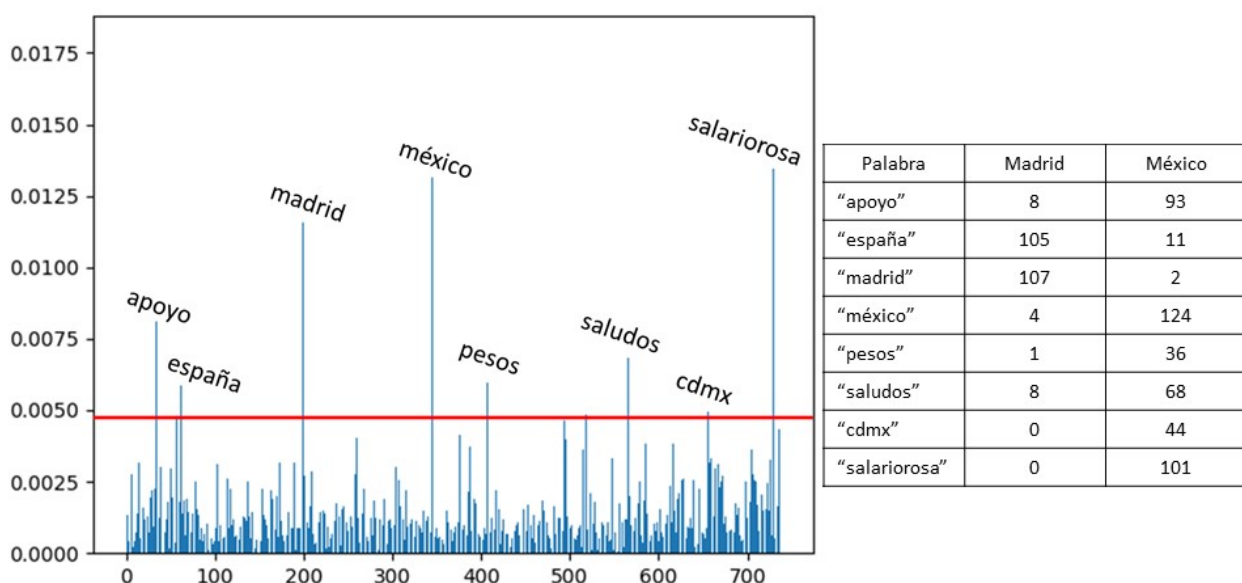


Figura 3-9 Coeficientes de las palabras más significativas en la clasificación Madrid-México.

3.4 Algoritmos basados en análisis de redes de usuarios.

3.4.1 Construcción de las redes de usuarios

La información recogida en los tweets no incluye información suficiente para construir las redes de usuarios. Los datos relativos al usuario que emitió el tweet únicamente dan información del número de seguidores de dicho usuario (“*followers_count*”), el número de personas a las que sigue dicho usuario (“*friends_count*”). No se proporciona la información que identifique a los seguidores o amigos.

Para conseguir las listas de usuarios seguidores o amigos de un usuario concreto es necesario realizar consultas adicionales al API de Twitter. Algunas de las funciones relacionadas con estas consultas son las siguientes:

- GET followers/ids¹, que nos devolverá los ID’s de los usuarios que siguen a otro usuario dado. El acceso a estos datos está limitado a 15 consultas cada 15 minutos.

¹ Esta función y sus limitaciones de acceso se describen en: “Follow, search and Get users” <https://developer.twitter.com/en/docs/twitter-api/v1/accounts-and-users/follow-search-get-users/api-reference/get-followers-ids> [Disponible a 23/12/2021]

- GET Friends/ids¹, similar a la anterior pero que nos devuelve los usuarios a quienes sigue un usuario dado. También tiene la limitación de 15 consultas cada 15 minutos.

Además, cada consulta nos podrá proporcionar una lista de, como máximo, 5000 ID's de usuarios, ya sean seguidores o amigos.

Estas restricciones dificultan en gran medida la formación de las redes de usuarios. Por ejemplo, obtener la red de usuarios seguidores de la colección “*tweets_madrid_ext*” tiene 5000 tweets y 3091 usuarios distintos. Si queremos construir un grafo que incluya todos esos usuarios y sus enlaces con sus amigos o seguidores, tendríamos que realizar, al menos, una consulta por usuario. Pero para aquellos usuarios que tuvieran más de 5000 usuarios tendríamos que realizar más de una. Si suponemos 3091 consultas, supondría realizar 207 tandas de consultas. Puesto que cada tanda conlleva un tiempo de 15 minutos, resulta imposible obtener la información necesaria antes de 52 horas. En definitiva, se trata de un proceso muy costoso.

Con las funciones anteriormente citadas, obtendremos simplemente una lista de ID's. Si queremos más información, como por ejemplo el nombre con el que el usuario se identifica en Twitter (“*screen_name*”), tendremos que realizar consultas adicionales utilizando la función GET users/lookup² que, en este caso tiene una limitación de 900 consultas cada 15 minutos.

Otras redes de usuarios de Twitter se pueden obtener mediante la aplicación *Gephi*³. Este software tiene un plugin denominado *TwitterStreamingImporter*⁴ que permite visualizar en tiempo real la creación de un grafo de usuarios basado en diversos aspectos, como por ejemplo las menciones o las citas que un usuario hace de otro. Este plugin hace uso del acceso del streaming de la API de Twitter, es decir, de los tweets que se están publicando en tiempo real, pero no permite de momento la utilización de los datos históricos ni la importación de datos externos. Sin embargo, permite el filtrado de los tweets según diversos campos, para acotar la búsqueda. Así por ejemplo nos permite indicar un grupo de palabras o hashtags, un grupo de usuarios o un idioma. El grafo se construirá en base a los tweets que cumplan alguno de estos filtros. Se trata por tanto de un filtro OR, no AND.

Desde el punto de vista de la geolocalización, puede ser interesante el último filtro disponible en este plugin de *Gephi*, en el cual se nos permite especificar las coordenadas de las esquinas noreste y suroeste de unas cajas, de modo que los tweets que se obtienen proceden de ubicaciones en su interior. Esto podría ayudarnos a constituir un conjunto de usuarios localizados a partir del cual construir un grafo y poder aplicar las técnicas de geolocalización presentadas en el párrafo 2.4.2.

La conformación de los grafos y la extracción de sus parámetros característicos se pueden realizar en Python con las librerías “*Networkx*”⁵, o “*igraph*”⁶, con ligeras variaciones entre ambas. Con cualquiera de ellas podemos obtener datos simples como el número de nodos y

¹ Esta función y sus limitaciones de acceso se describen en: “Follow, search and Get users” <https://developer.twitter.com/en/docs/twitter-api/v1/accounts-and-users/follow-search-get-users/api-reference/get-friends-ids> [Disponible a 23/12/2021]

² Esta función y sus limitaciones de acceso se describen en: “Follow, search and Get users” <https://developer.twitter.com/en/docs/twitter-api/v1/accounts-and-users/follow-search-get-users/api-reference/get-users-lookup> [Disponible a 23/12/2021]

³ Gephi es un programa de código abierto y gratuito que se puede utilizar para la representación y el análisis de grafos. Su página oficial es “Gephi makes graphs handy” <https://gephi.org/> [Disponible a 23/12/2021]

⁴ La información sobre este plugin y su funcionamiento se puede encontrar en “Twitter Streaming Importer” <https://github.com/seinecele/gephi-tutorials/blob/master/src/main/asciidoc/en/plugins/twitter-streaming-importer-en.adoc> [Disponible a 23/12/2021]

⁵ La descripción de esta librería se puede encontrar en “NetworkX Reference” https://networkx.org/documentation/stable/_downloads/networkx_reference.pdf [Disponible a 23/12/2021]

⁶ La descripción de esta librería se puede encontrar en “Get started with igraph in Python” <https://igraph.org/python/> [Disponible a 23/12/2021]

aristas de los grafos, así como otras métricas más complejas como el diámetro del grafo o la centralidad del vector propio. Sin embargo, resulta más intuitivo importar el grafo en *Gephi* y visualizar el grafo interactivamente en lugar de hacerlo mediante ejecuciones de código. *Gephi*, además, también nos proporciona los valores de muchos de los parámetros que caracterizan el grafo.

En nuestro caso hemos obtenido el grafo de usuarios del conjunto de tweets *tweets_madrid_ext*, que como se ha mencionado anteriormente consta de 5000 tweets procedentes de la ciudad de Madrid. En el grafo se representan los usuarios de este conjunto que siguen a otros usuarios pertenecientes también a dicho conjunto. Es decir, hemos construido el grafo basándonos en la relación de “usuarios que se siguen a otros usuarios” (“*followers*”). En *tweets_madrid_ext* hay 3191 usuarios distintos, de los que 437 tienen más de 5000 seguidores. Este subgrupo lo hemos excluido del grafo puesto que, tal y como se lleva a cabo en otros mecanismos descritos en la literatura académica relacionada, se les puede considerar como celebridades, dado su alto número de seguidores. La idea subyacente es que inferir la geolocalización de un usuario basándose en que sigue a una celebridad es muy poco fiable. Entre los usuarios filtrados de este modo, se encuentran “*elpais*”, “*diarioas*”, “*EFEnoticias*”, “*Cosmopolitan_es*” u “*hola*”, entre otros.

Número de nodos	1865
Número de aristas	3856
Grado medio	4.135
Diámetro de la red	15
Longitud media del camino	5.704
Modularidad	0.693
Componentes conexos	54

Tabla 3-22 Características del grafo *tweet_madrid_ext* obtenidas con Gephi

Tras filtrar las celebridades, nos quedan 2754 usuarios en el conjunto, algunos de los cuales no tienen seguidores dentro del propio conjunto, por lo que finalmente el número de nodos en el grafo se reduce a 1865. A pesar de ello, se constata que, utilizando únicamente 5000 tweets circunscritos a una determinada ciudad, el grafo resultante es bastante grande y denso. En la Tabla 3-22 se muestran algunos de los parámetros que definen el grafo obtenido. Los valores se han calculado con el programa *Gephi* y se describen seguidamente.

- El *Grado medio* es el número medio de relaciones entre nodos. Cada uno de los 1865 usuarios tiene una media de 4.135 aristas que los unen a otros nodos.
- El *Diámetro de la red* es el número máximo de nodos que hay que atravesar para unir los dos nodos más alejados, que estén conectados entre sí, por el camino más corto.
- La *Longitud media del camino* es el número de nodos que hay que atravesar, por término medio, para unir dos nodos del grafo.
- La *modularidad*¹ es un valor que expresa la cohesión entre los nodos. Una modularidad alta indica que existen comunidades (“clústers”) con más relaciones entre sus nodos de las que se esperaría en una red aleatoria.
- Los *componentes conexos* son el número de grupos de usuarios conectados entre sí. Un valor muy alto implica la existencia de muchos grupos aislados.

¹ El valor de modularidad obtenido por Gephi sigue el algoritmo de cálculo descrito por V.D. Blondel, J.L. Guillaume, R. Lambiotte y E. Lefebvre en “Fast unfolding of communities in large networks” Publicado en el Journal of Statistical Mechanics: Theory and Experiment (2008)

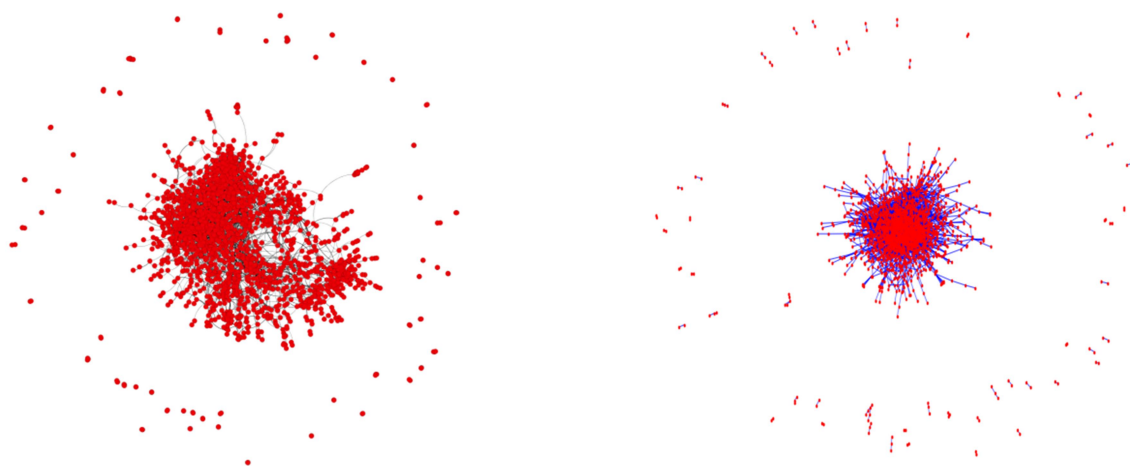


Figura 3-10 Representaciones gráficas del grafo extraído de *tweets_mad_ext*.

En la parte izquierda de la Figura 3-10 se muestra el grafo correspondiente a la relación de seguimiento entre usuarios del conjunto *tweet_madrid_ext*. La distribución de los nodos se ha realizado con el algoritmo “*ForceAtlas 2*”¹ en el que los nodos, o grupos de nodos, enlazados permanecen juntos, mientras que los no enlazados se dispersan a la periferia.

En el Anexo V, se incluye el código de Python para la generación del grafo a partir de los tweets almacenados en una colección de MongoDB, así como la representación gráfica del mismo que se puede observar en la parte derecha de la Figura 3-10. Para la creación del grafo se ha utilizado la librería “*Networkx*” y “*matplotlib*”. En este caso, para la distribución de los nodos se ha optado por el método “spring” que utiliza el algoritmo Fruchterman-Reingold² y que produce un resultado similar al utilizado en *Gephi*. Hay que resaltar que la mayoría de estos métodos de representación no son deterministas y que, aun trabajando con el mismo grafo, el gráfico puede variar de una ejecución a otra.

Adicionalmente hemos obtenido dos conjuntos de 2500 tweets procedentes de las ciudades de Madrid y Barcelona. Una vez se han suprimido los usuarios con más de 5000 seguidores, nos quedan 1389 usuarios en el conjunto de Barcelona y 1425 en el de Madrid. Al construir el grafo combinando los dos grupos de usuarios obtenemos un total de 1794 nodos en total y 3420 aristas. A este conjunto lo denominamos *tweets_mad_bar*.

En la Figura 3-11 se observa la representación del grafo, donde se han coloreado los nodos correspondientes a la ciudad de Madrid de rojo y los de Barcelona de verde. En este caso, el grado medio es de 3.813 y la modularidad de 0.773. Se trata, por tanto, de un grafo menos denso que el obtenido únicamente con tweets de la ciudad de Madrid, lo cual corrobora el hecho de que los usuarios tienden a tener más seguidores y/o amigos en las ubicaciones próximas. Al tratarse de dos ciudades separadas, como Barcelona y Madrid, la densidad de aristas disminuye, puesto que no es tan común que haya tantos seguidores entre los usuarios de ambas ciudades.

¹ El algoritmo ForceAtlas 2 fue desarrollado por Jacomy M, Venturini T, Heymann S y Bastian M. Se describe en “ForceAtlas2 a continuous graph layout algorithm for handy network visualization designed for the Gephi software” PLoS One. 2014 Jun 10.

² La referencia de este algoritmo se puede encontrar en “The Fruchterman-Reingold layout algorithm” https://igraph.org/r/doc/layout_with_fr.html [Disponible a 23/12/2021]

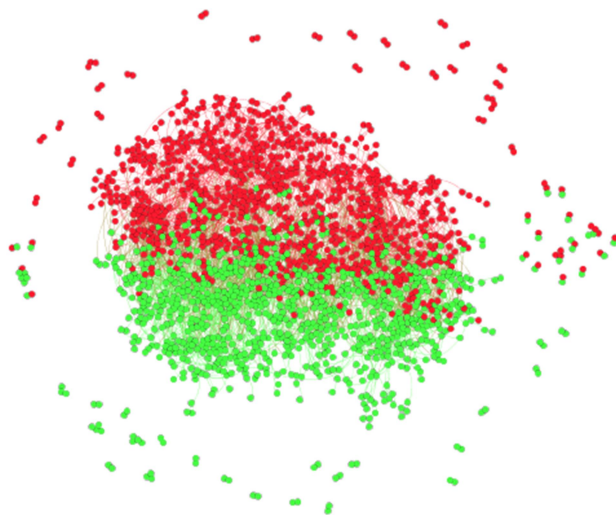


Figura 3-11 Grafo del conjunto *tweets_mad_bar* obtenido con Gephi

Aprovechando que en este grafo conocemos la ubicación real de los usuarios, podremos aplicar el algoritmo de propagación de etiquetas para predecir las localizaciones de un conjunto de usuarios de test y así, obtener las medidas de exactitud.

3.4.2 Algoritmos basados en la propagación de etiquetas

La propagación de etiquetas, tal y como se mencionó en 2.4.2.2, es uno de los algoritmos frecuentemente utilizados en la literatura académica para la geolocalización de usuarios. Para comprobar su funcionamiento, lo hemos aplicado, en primer lugar, al conjunto de datos *tweet_madrid_ext*, descrito en el párrafo anterior.

Los algoritmos de propagación de etiquetas que podemos encontrar en las librerías de Python “*Networkx*” e “*igraph*” pueden obtener las comunidades existentes en un grafo sin necesidad de proporcionarles más información que la del propio grafo. Esto lo consiguen asignando inicialmente una etiqueta a cada nodo, seleccionando aleatoriamente algún nodo para iniciar el cómputo, calculando las probabilidades de los diferentes caminos, considerando el peso de las aristas, si se proporciona y etiquetando nuevamente aquellos nodos que lo requieran. El proceso no es determinista y, sucesivas ejecuciones del mismo provocarán que un determinado nodo acabe etiquetado en diversas comunidades. Sin embargo, el cómputo es rápido y se podría llevar a cabo repetidamente para, finalmente, clasificar un nodo como de una comunidad u otra en función de todos los valores obtenidos en las diferentes ejecuciones.

Aplicando el algoritmo a nuestro grafo obtenido del conjunto *tweet_madrid_ext* obtenemos 203 comunidades distintas. En la parte izquierda de la Figura 3-12 vemos todas esas comunidades y hemos indicado el tamaño de las cinco que tienen más de 40 nodos.

En la parte derecha, en lugar de colorear todas las comunidades, únicamente hemos coloreado las que tienen más de 40 nodos. Los nodos no coloreados pertenecen mayoritariamente a comunidades con menos nodos. Así pues, el algoritmo de propagación de etiquetas es capaz de obtener una estimación de las distintas comunidades existentes en un grafo en el que no se haya etiquetado ninguno de sus nodos. Este comportamiento es el que podemos encontrar en la librería “*Networkx*” con su función “*label_propagation_communities*”¹.

¹ Esta función está descrita en “*networkx.algorithms.community.label_propagation.label_propagation_communities*”

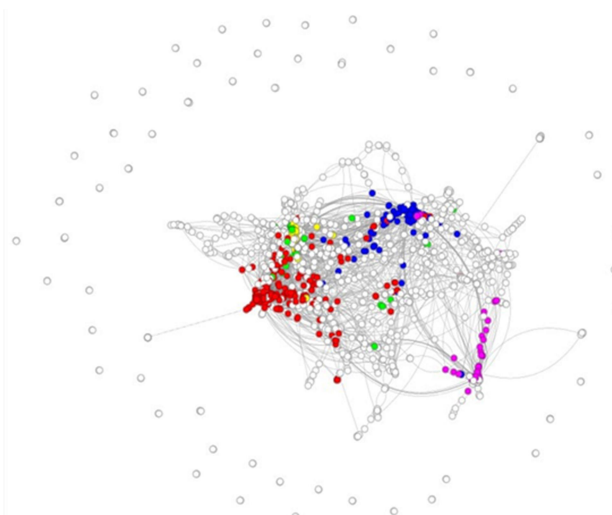
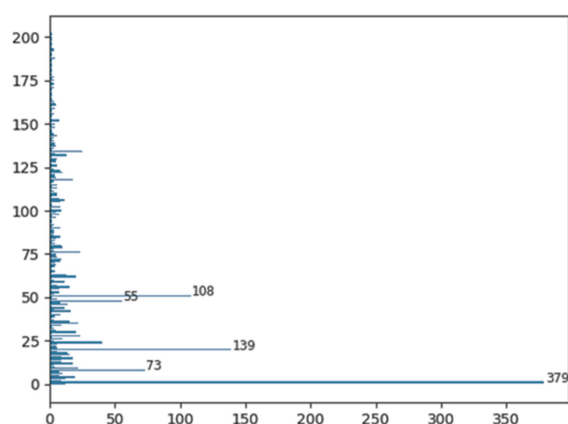


Figura 3-12 Comunidades en *tweet_madrid_ext* obtenidas con propagación de etiquetas.

Por su parte, la librería de Python “*igraph*” sí que nos permite la asignación previa de etiquetas a aquellos nodos que consideremos y mantener como no etiquetados el resto. Además, podemos indicar que dichas etiquetas previamente asignadas son inamovibles, para que no las modifique como resultado de la ejecución del algoritmo. Las etiquetas definidas se propagarán entre los nodos que estén conectados entre sí progresivamente hasta alcanzar un estado estable. Esta capacidad de “*igraph*”, ausente en “*Networkx*”, nos permite su utilización de forma más ajustada al planteamiento de nuestro problema, en el cual, conocemos las localizaciones de un pequeño número de usuarios y queremos inferir las del resto. Así pues, en nuestra implementación hemos utilizado dicha librería y la función “*community_label_propagation*” que es la que implementa la propagación de etiquetas.

En nuestro caso, sabemos que todos los tweets del conjunto *tweets_madrid_ext* proceden de la ciudad de Madrid. Lo que inicialmente podemos tratar de averiguar es cómo de bien se propagan las etiquetas. Así pues, asignamos de forma aleatoria una única etiqueta, que correspondería con la ubicación de Madrid, a un cierto porcentaje de usuarios del grafo y dejamos sin etiquetar el resto. La pregunta que queremos contestar es cuántos nodos quedarán sin etiquetar en el grafo.

En *tweets_madrid_ext* hay 1865 usuarios distintos, si etiquetamos un 30% de los nodos del grafo, conseguimos que las etiquetas se expandan hasta conseguir etiquetar 1813 nodos, lo que supone un 97%. Si únicamente etiquetamos el 10%, las etiquetas se propagan a 1781 nodos, lo que constituye el 95%. Podemos concluir que la propagación se produce convenientemente. Las etiquetas no se propagan a todos los nodos porque no todos están conectados entre sí. Veámos en la Tabla 3-22 que había 54 componentes conectados. Si en alguno de dichos componentes no hay ninguno etiquetado, el algoritmo nunca les asignará una etiqueta. Así que el porcentaje de nodos etiquetados depende mucho de la densidad del grafo y de la existencia de nodos o comunidades aisladas. Resulta obvio que cuantos más usuarios analicemos más probabilidad habrá de que existan interconexiones entre los nodos, pero acabamos de comprobar que escogiendo al azar menos de 2000 usuarios en una ciudad que tiene más de tres millones de habitantes, la densidad del grafo es suficiente para poder utilizar este método de geolocalización con éxito.

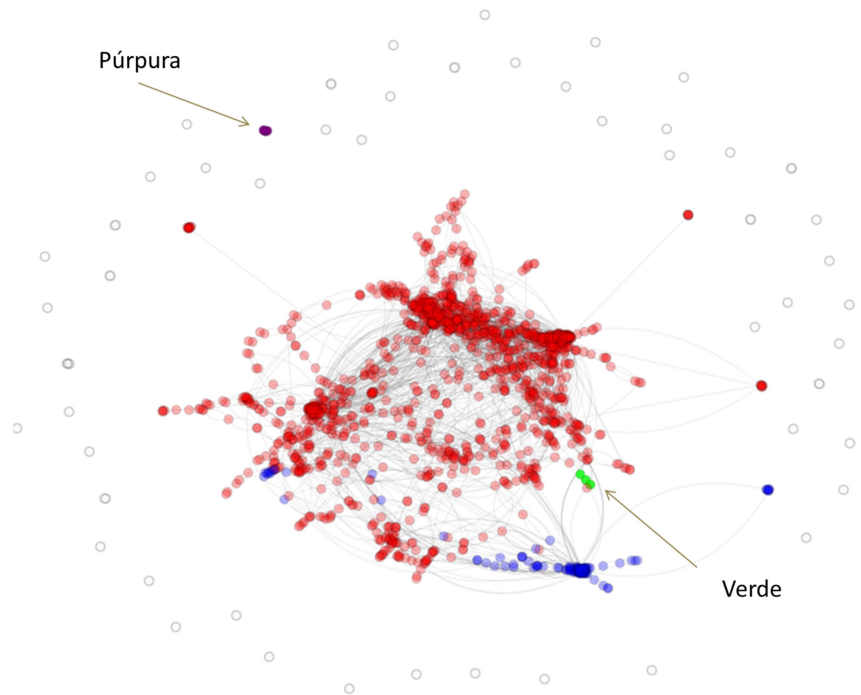


Figura 3-14 Propagación de etiquetas a partir de cuatro nodos iniciales.

En los grafos en los que alguna de las comunidades está fuertemente conectada y que además tiene un “*componente gigante*”¹ que incluye numerosos nodos, es habitual que alguna de las etiquetas predomine sobre las demás. Para visualizar correctamente estos grafos conviene que dichas etiquetas tengan, al representarlas, un cierto nivel de transparencia, para evitar que se superpongan al resto y no permitan identificar a las comunidades minoritarias.

En este ejemplo, de los 1865 nodos, las etiquetas se han propagado a un total de 1750 de ellos, partiendo únicamente de los cuatro nodos etiquetados inicialmente. Es decir, que hemos logrado etiquetar el 93.83% del total.

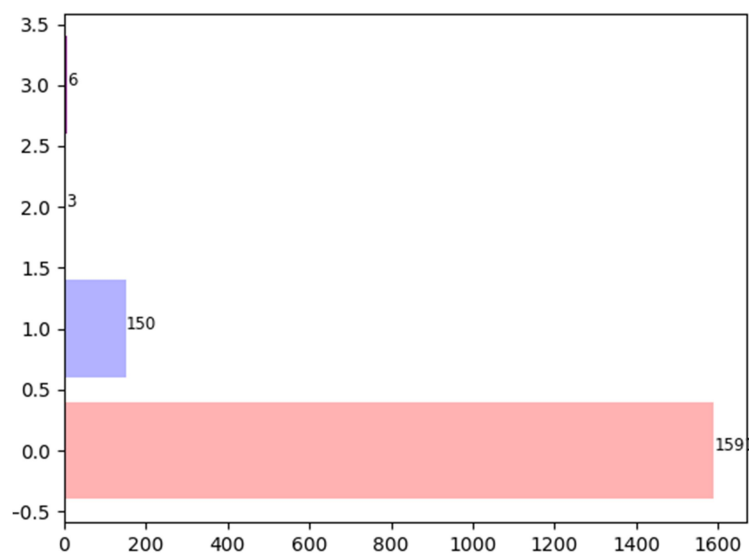


Figura 3-15 Nodos en cada clase tras la propagación de etiquetas

¹ El concepto de componente gigante se puede encontrar definido en “Giant component” https://en.wikipedia.org/wiki/Giant_component [Disponible a 23/12/2021]

Centrándonos ahora en el conjunto de datos que hemos denominado *tweets_mad_bar* con 1774 nodos, tenemos 1553 de los que conocemos su localización real, Madrid o Barcelona. Etiquetamos un porcentaje inicial de esos nodos y dejamos sin etiquetar el resto. Aplicamos el algoritmo de propagación de etiquetas y conseguimos que dichas etiquetas se expandan por el resto de nodos, alcanzando un porcentaje que varía entre el 94% y el 96% según la prueba realizada. Posteriormente comprobamos si las etiquetas se han expandido correctamente, es decir si los nodos que realmente son de Madrid o Barcelona han sido etiquetados por el algoritmo como tales. Obviamente, cuantos más nodos etiquetemos inicialmente con la etiqueta correcta, mejores resultados obtendremos. Hemos hecho las mediciones para porcentajes de etiquetado inicial del 10%, 20% y 30%. Puesto que el algoritmo de propagación de etiquetas no es determinista, hemos realizado las pruebas cinco veces para cada porcentaje de etiquetado inicial y hemos recopilado la media de los resultados obtenidos.

La información resultante se puede observar en la Tabla 3-23, donde se recogen los valores de precisión, exhaustividad y exactitud a la hora de identificar correctamente la localización real de cada nodo. Además, también se ofrecen los datos de la matriz de confusión con el total de los nodos identificados correctamente (“*true*”) y erróneamente (“*false*”) para cada etiqueta.

El código de Python utilizado para obtener estos resultados se incluye en el Anexo V.

Nodos	1553	Madrid	803	51.7%				
		Barcelona	750	48.3%				
Etiquetas Inicio	Expansión		PRE	REC	F1	ACC	TRUE	FALSE
10%	94%	Madrid	0.656	0.646	0.648	0.638	490.4	259.2
		Barcelona	0.624	0.648	0.624		444.8	269.6
20%	95%	Madrid	0.772	0.678	0.722	0.728	520.2	247.8
		Barcelona	0.690	0.782	0.732		552.8	155.2
30%	96%	Madrid	0.814	0.744	0.774	0.776	580.0	200.0
		Barcelona	0.746	0.814	0.778		584.8	133.2

Tabla 3-23 Resultados de propagación de etiquetas en *tweets_mad_bar*.

4 RESULTADOS

4.1 Sobre la obtención de datos.

Para la obtención de los tweets que constituyen los datos de los diversos métodos de geolocalización se precisa, como se ha mencionado, acceso a la API de Twitter. Los lenguajes de programación suelen implementar librerías que facilitan la integración de dicha API con las estructuras de datos propias de cada uno. Es el caso de Python y la librería Tweepy tal y como hemos visto.

El acceso a los datos de la API de Twitter tiene limitaciones, en función de la modalidad, que evitan que los usuarios puedan llegar a colapsar el servicio. La modalidad gratuita que se implementa en la versión 1.1 de la misma permite obtener 2500 tweets por hora. Eso significa que si se desea recopilar un número considerable de tweets resulta necesario planificar el proceso de obtención, y dedicar recursos durante un tiempo que se puede prolongar durante varios meses.

El almacenamiento de los tweets, puede llegar a ser una cuestión a tener en cuenta por su volumen. En nuestro caso, puesto que hemos trabajado con volúmenes relativamente reducidos, las bases de datos locales existente en un único ordenador aislado han sido suficientes.

Para los distintos métodos de geolocalización utilizados se han utilizado diversos conjuntos almacenados en colecciones de MongoDB. Durante el proceso de obtención se han utilizado consultas a la API de Twitter que han permitido que los tweets obtenidos fueran procedentes de alguna localización concreta para, de ese modo, construir los conjuntos de entrenamiento y de datos de test que requieren el conocimiento a priori del resultado de la geolocalización. La obtención de los datos de tal manera provoca que no sean completamente aleatorios lo que afecta, sin lugar a dudas, al porcentaje de ellos que contienen información relevante para la geolocalización.

Si nos fijamos en el conjunto *tweets_stream*, que se recogió únicamente con la limitación de que el idioma fuera español, se observa que menos del 0.05 % incluyen coordenadas en los metadatos de los tweets. Esto significa que si queremos obtener un conjunto de tweets que sirva de referencia y que nos permita entrenar algoritmos de Machine-Learning para obtener la geolocalización con una precisión suficiente para distinguir entre barrios, distritos o calles, será necesario obtener un volumen de datos varios órdenes de magnitud mayor que si simplemente nos basta con geolocalizar a nivel ciudad.

4.2 Sobre la aplicación de algoritmos basados en análisis de textos.

4.2.1 Método basado en diccionarios toponímicos.

Gracias a las APIs que proporcionan el servicio de geolocalización a partir de una cadena de texto, como es el caso de *GeoNames* es posible obtener las coordenadas de una gran cantidad de palabras o grupos de palabras que identifiquen un lugar. Es significativo que la búsqueda no se limita a unigramas. Con estos diccionarios toponímicos es trivial obtener coordenadas, por lo que podemos trabajar en los algoritmos de clasificación con cadenas de caracteres descriptivas de los lugares, sabiendo que siempre podremos transformarlas en coordenadas si fuera preciso para, por ejemplo, calcular distancias o medidas de error. Sin embargo, hemos visto que, en ocasiones, *GeoNames* no es capaz de ofrecer coordenadas ante cadenas de texto que un humano sí podría identificar y que, en otros casos, ofrece unas coordenadas de un lugar, que aparentemente tiene poco que ver con el significado de la cadena de texto. A pesar de ello, *GeoNames* consiguió ofrecer unas coordenadas ante el 97% de las cadenas de texto que se le presentaron procedentes del campo *location* del perfil de usuario del conjunto de tweets *tweets_spain*. La comprobación de que todas las coordenadas ofrecidas son correctas nos obligaría a revisar manualmente los 15.000 tweets. Aunque tenemos constancia de que, en algunos casos, dichas coordenadas son erróneas, desconocemos el porcentaje exacto. En [43] podemos encontrar un estudio detallado de la exactitud de *GeoNames* elaborado en 2013, pero obviamente, no ofrece un dato que nos sirva para obtener la precisión a la hora de obtener coordenadas a partir de las cadenas de texto que indican los usuarios de Twitter.

El conjunto de datos “*tweets_stream*”, que ha sido obtenido sin restricciones de localización, contiene un 59.5% de tweets en los que el campo “*location*” no está vacío. Sobre ese porcentaje podemos asumir que *GeoNames* será capaz de ofrecernos unas coordenadas en el 97% de los casos. Eso supone una cobertura del 57.8%, pero, por desgracia, somos incapaces de obtener un valor de exactitud (“*accuracy*”).

Un factor a tener en cuenta cuando se utilizan las APIs de diccionarios toponímicos es su limitación de uso. En el caso de *GeoNames* un usuario con acceso básico tenía 20.000 créditos diarios y podía consumir un máximo de 1000 por hora. Cada tipo de petición consumía un número u otro de créditos.

4.2.2 Método basado en análisis del texto de los tweets.

Inicialmente, al analizar los tweets del conjunto “*tweets_spain_ext*” y tratar de clasificar a los usuarios de Madrid y Barcelona a partir del conjunto de palabras que utilizan en sus tweets mediante el algoritmo de regresión logística, obtuvimos unos resultados bastante malos. Aunque la exactitud era del 63%, hay que considerar que el conjunto de muestras estaba desequilibrado y las muestras de Madrid constituían un 61%. Un algoritmo que simplemente clasificara todas las muestras como de Madrid, alcanzaría una precisión cercana al 61%. Este fenómeno se denomina paradoja de la exactitud¹. Posteriormente sacrificamos algunas de las muestras del conjunto de Madrid, que era el más numeroso, para conseguir la misma cantidad de muestras que en el de Barcelona. Al aplicar ahora los algoritmos se obtuvieron unos resultados razonables. En particular, para el de regresión logística se obtuvo una precisión del 68%.

Teniendo en cuenta que las muestras en este segundo caso eran un subconjunto de las anteriores parece claro que, para el buen funcionamiento de los algoritmos utilizados, el trabajar con conjuntos de muestras balanceados resulta fundamental.

¹ En “Accuracy Paradox” <https://towardsdatascience.com/accuracy-paradox-897a69e2dd9b> [Disponible a 23/12/2021] se puede encontrar una descripción ilustrativa de este fenómeno.

Al aplicar el mismo método de clasificación sobre los tweets obtenidos en las ciudades de Madrid y México conseguimos con el algoritmo SVM un 69% de exactitud. En este caso, el porcentaje de muestras fue del 49.3% y del 50.7% para Madrid y México respectivamente. Un reparto muy cercano al 50%.

Al representar gráficamente cuáles han sido las palabras que, tras los algoritmos de Machine-Learning obtenían unos coeficientes más significativos, y por tanto un mayor peso en la clasificación, se observa que efectivamente hay palabras que se espera sean más frecuentes en un lugar u otro (“méxico”, “madrid”, “españa”, “cdmx”, “salariorosa”, “pesos”).

4.3 Sobre la aplicación de algoritmos basados en análisis de redes de usuarios.

Con el fin de observar el funcionamiento de un algoritmo de propagación de etiquetas, obtuvimos el grafo de seguidores (“*followers*”) del conjunto *tweets_madrid_ext*. De los 5000 tweets que lo componían se extrajeron los usuarios diferentes y, considerando como “*celebrities*” aquellos que tenían más de 5000 seguidores, se suprimieron todos los que tenían dicho número o superior.

Para construir las relaciones de un usuario, únicamente se consideraron los seguidores que pertenecieran al propio conjunto de *tweets_madrid_ext*, es decir, si un usuario no tenía seguidores que perteneciesen al propio conjunto, se mostraría como un usuario aislado. Teniendo en cuenta esa restricción, se podría pensar que el grafo resultante sería poco denso. Sin embargo, no fue así y se constató que los grafos de amistad de las redes sociales son capaces de vincular a una gran mayoría de usuarios.

Las pruebas que se realizaron sobre este conjunto nos mostraron que con, únicamente el 10% de los nodos etiquetados aleatoriamente, se conseguía propagar las etiquetas a un 95% de ellos. Esto, simplemente nos da una idea de lo densamente que estaba conectado el grafo.

Posteriormente elegimos cuatro nodos aleatorios del grafo y los etiquetamos de forma diferente. Aunque todos los usuarios procedían de la ciudad de Madrid, simulamos que conocíamos que algunos de ellos vivían en un barrio o distrito diferenciado. En tal caso, al propagar las etiquetas, se observó que alguna de ellas se volvía predominante sobre el resto.

Por último, se realizó un experimento completo obteniendo usuarios procedentes de las ciudades de Barcelona y Madrid. Se eliminaron las celebridades y se creó el grafo de seguidores. El resultado fue un grafo menos denso que el que se obtuvo para los usuarios procedentes de *tweets_madrid_ext*, pero, sorprendentemente, también existían muchas relaciones entre los usuarios de ambas ciudades. Los usuarios de Madrid constituían el 51.7% del total, frente al 48.3% de los de Barcelona. Una vez construido el grafo se etiquetaron un 10% de los usuarios de Madrid y un 10% de los usuarios de Barcelona y se aplicó el algoritmo de propagación de etiquetas. El resultado fue que se consiguieron etiquetar el 94% de los nodos y, se consiguió una exactitud del 64%. En el caso de que se incrementara el número de etiquetas iniciales al 20% o al 30%, se conseguía un 72.8% o un 77.6% de exactitud respectivamente.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

A lo largo de este trabajo se han descrito detalladamente los métodos existentes en la literatura académica para la geolocalización de los usuarios de Twitter, así como las técnicas y herramientas que emplean. A todo lo cual, conjuntamente, hemos denominado mecanismos. Sin embargo, lo que nos otorga una visión detallada del funcionamiento de dichos algoritmos es la implementación que hemos realizado. Se han conseguido programas funcionales capaces de realizar las tareas de geolocalización utilizando el análisis de textos y el análisis de redes de usuarios.

Las implementaciones realizadas no pretenden competir en exactitud ni eficiencia con los sistemas presentados por los autores citados en la bibliografía. Lo que se ha pretendido es llenar el vacío existente entre la descripción de los métodos de geolocalización y la obtención del código que finalmente lo implementa. Durante el transcurso de ese proceso, hemos obtenido enseñanzas que podemos presentar como conclusiones a continuación.

La utilización de Python como lenguaje de programación nos ha facilitado el acceso a ciertas librerías específicas que facilitan la implementación. Desde la utilización de *Tweepy*, como intermediario con la API de Twitter, hasta librerías que pusieron a nuestro alcance los algoritmos de Machine-Learning y análisis de grafos.

MongoDB ha sido una herramienta de uso sencillo para almacenar con eficiencia los miles de tweets recogidos como muestras para aplicar los métodos de geolocalización. El conocimiento de su funcionamiento en una instalación local, facilitará sin duda su empleo en la nube.

La API de Twitter ha permitido el acceso a los tweets, pero hay que resaltar que, si se opta por el modo de acceso que no supone coste para el usuario, las limitaciones en el número de consultas que se le pueden hacer por unidad de tiempo son un inconveniente. La obtención de los tweets está limitada a 2500 por hora y para obtener la lista de seguidores de un usuario, únicamente es posible realizar 15 consultas cada cuarto de hora. Twitter ya ha lanzado nueva versión de su API que permite un acceso orientado para académicos y que, una vez implementado en las librerías de los lenguajes de programación, proporcionará el acceso a la información con menores limitaciones.

Las herramientas gráficas mencionadas a lo largo del Capítulo 3 nos permiten visualizar el comportamiento de los algoritmos implementados y tomar decisiones adecuadas. Los gráficos de nubes de palabras o los que sirven para identificar los mejores valores de los hiperparámetros se pueden obtener gracias al uso de ciertas librerías de Python y su empleo ha sido incluido en el

código de los Anexos. Por otro lado, el uso de la aplicación *Gephi* ha resultado fundamental para ilustrar los grafos que representan las relaciones entre usuarios.

En lo referente a los métodos que utilizan análisis de texto a través de los diccionarios toponímicos podemos resaltar lo siguiente:

- Resulta complicado comprobar la exactitud con la que el diccionario toponímico ofrece las coordenadas relacionadas con las cadenas de texto que identifican lugares. Sin embargo, son capaces de ofrecer resultados para un alto porcentaje de entradas.
- Dado que actualmente menos del 1% de los usuarios proporciona las coordenadas desde donde publicó el tweet, resulta imprescindible tener que recurrir a los diccionarios toponímicos para convertir textos que indiquen una ubicación en coordenadas.
- Se ha observado que cerca del 50% de los usuarios proporcionan información en su perfil que podría facilitar su localización mediante los diccionarios toponímicos.

Sobre los métodos que analizan el lenguaje utilizado por los usuarios para inferir su localización mediante técnicas de Machine-Learning podemos resaltar que:

- Al aplicar los algoritmos sobre los conjuntos de Madrid y Barcelona en los que ambas clases tenían un número de muestras desequilibrado, los resultados obtenidos de exactitud empeoraron considerablemente. Resultó preferible, sacrificar muestras de la clase más numerosa para obtener un conjunto equilibrado para ambas clases.
- El número de tweets recogido en este trabajo ha sido reducido y limitado en el tiempo. Se ha observado que algunas de las palabras que han servido en mayor medida para clasificar los tweets como procedentes de una ciudad u otra dependen, en ocasiones, del tema de conversación que durante el periodo de recogida de tweets sea de actualidad. Si se recogen tweets en un lugar donde en un determinado momento gran cantidad de tweets se refieren a un cierto tema, los algoritmos de Machine-Learning obtendrán fácilmente la relación entre ese asunto (“*topic*”) y la ubicación. Pero, si cierto tiempo después ya no se habla de ese asunto, ya no podremos inferir la ubicación. En definitiva, este tipo de métodos parecen muy sensibles temporalmente. El entrenamiento del algoritmo que hagamos con los tweets de hoy, puede que nos ofrezca buenos resultados hoy, pero muy malos cuando tratemos de clasificar tweets dentro de unos meses.

Sobre los métodos que analizan las redes de usuarios representadas mediante grafos, se resaltan las siguientes conclusiones:

- Las relaciones de seguimiento y amistad entre usuarios de Twitter, al ser más duraderas en el tiempo resultarán más fiables a la hora de ubicar a los usuarios en base a datos obtenidos con anterioridad.
- La obtención de las redes de usuarios, debido a la limitación de acceso a la API de Twitter, tiene un alto coste en términos temporales.
- En el preprocesado de los datos, resulta imprescindible identificar y filtrar a las celebridades, puesto que no aportan información útil para la localización y incrementan muchísimo el tiempo de obtención de la lista de seguidores.

Desde un punto de vista general, podemos decir que:

- Para aplicaciones que requieran precisar la ubicación con mayor precisión que el de ciudad, resultará necesaria la obtención de, al menos, varios cientos de miles de tweets para conseguir unos conjuntos de datos de referencia mínimos, bien para etiquetar a los usuarios de los grafos o para entrenar a los algoritmos de Machine-Learning.

- El procedimiento de recogida de tweets cuando se va a emplear el método de análisis de textos debe planificarse y extenderse en el tiempo para evitar que esté determinado por la utilización de palabras relacionadas con tópicos concretos, utilizados únicamente durante un periodo concreto (elecciones, catástrofes, etc). Desde ese punto de vista, las técnicas basadas en análisis de redes de usuarios resultan más robustas.

5.2 Líneas futuras

El propio propósito de este trabajo es servir de base para aquellos que, sin tener conocimientos previos sobre la materia, quieran profundizar en los métodos de geolocalización. Por tanto, las posibilidades de ampliación del mismo no tienen límite. Sin embargo, tratando de agruparlas conceptualmente, identificamos dos líneas de actuación.

La primera de ellas está relacionada con la transición del procesamiento local, en un único ordenador, a un conjunto de servidores. El diseño e implementación de un sistema de geolocalización en un clúster de computación es un salto cualitativo que requeriría un estudio detallado. Desde la recogida y almacenamiento de los tweets en una base de datos en la nube, con varios servidores, hasta la preparación de los datos y la paralelización de los algoritmos de clasificación.

La segunda de ellas incide en la mejora de la exactitud en el proceso de geolocalización. En este campo se abren multitud de posibilidades. En términos generales, la aplicación de métodos híbridos que utilicen, tanto técnicas propias del análisis de textos, como del análisis de usuarios conjuntamente parece que es la mejor manera de lograrlo. En este sentido, en la literatura académica hemos mencionado la utilización de redes neuronales para combinar distintos métodos.

Si nos inclinamos por los métodos de análisis de textos, resulta necesario profundizar en las técnicas de obtención y preprocesado de los datos. De modo que podamos obtener un modelo entrenado útil durante un período de tiempo lo más largo posible, inmune a los cambios temáticos de las conversaciones. Si, por el contrario, preferimos utilizar el análisis de redes de usuarios, la selección de características que nos permitan la asignación de pesos a las aristas sería interesante. Aunque habría que asumir el alto coste computacional que supone la realización de las consultas necesarias a la API para obtener la información necesaria para la creación de estos grafos. También serían interesantes las técnicas de reducción de la dimensión en la representación de las redes de usuarios aplicando *DeepWalk*, *Node2Vec* u otros procedimientos similares.

6 BIBLIOGRAFÍA

- [1] Tahir M. Nisar, Man Yeung, «Twitter as a tool for forecasting stock market movements: A short window event study,» *Journal of Finance and Data Science*, vol. 4, nº 2, 2018.
- [2] Kumar D. Ukkusuri S., «Utilizing geo-tagged tweets to understand evacuation dynamics during emergencies: a case study of hurricane Sandy,» de *Companion proceedings of the web conference*, 2018.
- [3] Li C., Sun A., «Extracting fine-grained location with temporal awareness in tweets: a two-stage approach,» *Journal of the association for information science and technology*, vol. 68, pp. 1652-1670, 2017.
- [4] Schosser S., Toninelli D., Cameletti M., «Comparing methods to collect and geolocate tweets in Great Britain,» *Journal of Open Innovation: Technology, Market and Complexity*, 2021.
- [5] Ajao O., Hong J., Liu W., «A survey of location inference techniques on Twitter,» *Journal of Information Science* 1-10, 2015.
- [6] Chong W-H., Lim E-P., «Fine-grained geolocation of tweets in temporal proximity,» *ACM Transactions of Inf. Syst.*, 2019.
- [7] Drezde M., Osborne M., «Geolocation for Twitter: Timing Matters,» de *Proceedings of NAACL-HLT. pp:1064-1069*, San Diego, CA, 2016.
- [8] Do T.H., Nguyen D.M., Tsiligiani E., Cornelis B., Deligiannis N., «Multiview deep learning for predicting twitter user's location,» 2017.
- [9] Lin J., Cromley R.G., «Inferring the home locations of Twitter users based on the spatiotemporal clustering of Twitter data,» *Transactions in GIS*, vol. 22, pp. 82-97, 2017.
- [10] Luo X., Qiao Y., Li C., Ma J., Liu Y., «An overview of microblog user geolocation methods,» *Information Processing and Management*, vol. 57, p. 102375, 2020.
- [11] Chen J., Liu Y., Zou M., «Home location profiling for users in social media,» *Information & Management*, vol. 53, 2015.
- [12] Mahmud J., Nichols J., Drews C., «Home location identification of Twitter users,» *ACM Transactions on Intelligent Systems and Technology*, vol. 5, nº 3, pp. 1-21, 2014.

- [13] Sloan L., Morgan J., «Who tweets with their location? Understanding the relationship between demographic characteristics and use of geoservices and geotagging on Twitter,» PLOS ONE, 2015.
- [14] Gerlitz C., Rieder B., «Mining one percent of Twitter: Collections, Baselines, Sampling,» M/C Journal. Vol. 16, No. 2, 2013.
- [15] Eisenstein J., O'Connor B., Smith N. A., Xing E. P., «A latent variable model for geographic lexical variation,» de *Empirical methods in Natural Language Processing*, Cambridge, MA., 2015.
- [16] Roller S., Speriosu M., Rallapalli S., Wing B., Baldrige J., «Supervised text-based geolocation using language models on an adaptive grid,» de *Joint conference on empirical methods in Natural Language Processing and Computational Natural Language Learning*, pp 1500-1510, Jeju island (Korea), 2012.
- [17] Han B., Cook P., Baldwin T., «Geolocation prediction in social media data by finding location indicative words,» de *Proceedings of COLING 2012: Technical Papers*. pp 1045-1062, Bombay, 2012.
- [18] Ramachandran D., Parvathi R., «Analysis of Twitter specific preprocessing technique for tweets,» de *International conference of recent trends in advanced computing*, 2019.
- [19] O'Connor B.T., Krieger M., Ahn D., «TweetMotif: Exploratory search and topic summarization for twitter,» de *Proceedings of the fourth international AAAI conference on weblogs and social media*, 2010.
- [20] Kotzias D., Lappas T., Gunopulos D., «Home is where your friends are: Utilizing the social graph to locate twitter users in a city,» *Information Systems*. No. 57; Pp 77-87, 2015.
- [21] Compton R., Jurgens D., Allen D., «Geotagging one hundred million twitter accounts with total variation minimization,» 2015.
- [22] Ozdikis O., Ramampiaro H., Nørvåg K., «Locality-adapted kernel densities of term co-occurrences for location prediction of tweets,» *Information Processing and Manangement*, vol. 56, pp. 1280-1299, 2019.
- [23] Jurgens D., Finety T., McCorrison J., Xu Y-T., Ruths D., «Geolocation prediction in Twitter using social networks: a critical analysis and review of current practice,» de *Proceedings of the ninth international AAAI Conference on web and Social Media*, 2015.
- [24] Han B., Cook P., Baldwin T., «Text-based Twitter user geolocation prediction,» de *Journal of Artificial Intelligence Research*. vol 49 pp 451-500, 2014.
- [25] Zubiaga A., Voss A., Procter R., Liakata M., Wang B., Tsakalidis A., «Towards real-time, country-level location classification of worldwide tweets,» de *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 9, pp. 2053-2066, 2017.
- [26] Alonso J., Costa E., Fernández M., «Language independent Big-Data system for the prediction of user location on Twitter,» de *IEEE International Conference on Big Data*, 2016.
- [27] S. Raschka, «Naive Bayes and Text Classification I - Introduction and Theory,» 2014. [En línea]. Available: <https://arxiv.org/pdf/1410.5329v3.pdf>.
- [28] Blei D.M., Ng Y.A., Jordan M.I., «Latent Dirichlet Allocation,» *Journal of Machine Learning Research*, vol. 3, pp. 993-1022, 2003.

- [29] Chi L., Lim K.H., Alam N., Butler C.J., «Geolocation prediction in twitter using location indicative words and textual features,» de *Proceedings of the 2nd Workshop on Noise user-generated Text*, Osaka, Japan, 2016.
- [30] Lourentzou I., Morales A., Zhai C., «Text-based geolocation prediction of social media users with neural networks,» de *IEEE International Conference on Big Data*, 2017.
- [31] Lee K., Ganti R.K., Srivatsa M., Liu L., «When twitter meets foursquare: tweet location prediction using foursquare,» de *MOBIQUITOUS '14: Proceedings of the 11th international conference on mobile and ubiquitous systems.*, 2014.
- [32] Galal A., El-Korany A., «Enabling semantic user context to enhance Twitter location prediction,» *Proceedings of the 8th international conference on agents and artificial intelligence*, vol. 1, p. 223.230, 2016.
- [33] Lozano M. G., Scheiber J., Brynielsson J., «Tracking geographical locations using a geo-aware topic model for analyzing social media data,» *Elsevier. Decision Support Systems*, vol. 99, pp. 18-29, 2017.
- [34] Backstrom L., Sun E., Marlow C., «Find me if you can: Improving geographical prediction with social and spatial proximity,» de *Proceedings of the 19th international conference on WWW*, Raleigh, 2010.
- [35] McGee J., Caverlee J., Cheng Z., «Location prediction in social media based on tie strength,» de *22nd Conference on information and knowledge management*, San Francisco, 2013.
- [36] Zhu X., Ghahramani Z., «Learning from labeled and unlabeled data with label propagation,» 2002.
- [37] Perozzi B., Al-Rfou R., Skiena S., «Deepwalk: online learning of social representation.,» de *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*, New York, 2014.
- [38] Mikolov T., Corrado G., Chen K., Dean J., «Efficient estimation of word representation in vector space,» *Proceedings of the international conference on learning representations*, pp. 1-12, 2013.
- [39] Grover A., Leskovec J., «node2vec: Scalable Feature Learning for Networks,» de *Proceedings of the 22nd ACM SIGKDD International conference on knowledge discovery and data mining*, 2016.
- [40] Rahimi A., Cohn T., Baldwin T., «Twitter user geolocation using a unified text and network preidction model,» de *Proceedings of the 53rd annual meeting ot the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, Beijing, 2015.
- [41] Jayasinghe G., Jin B., McHugh J., Robinson B., Wan S., «CSIRO Data 61 at the WNUT geo shared task,» de *Proceedings of the 2nd workshop on noisy user-generated text. pages 218-226.*, Osaka, 2016.
- [42] Miura Y., Taniguchi M., Taniguchi T., Ohkuma T., «Unifying text, metadata and user network representations with a neural network for geolocation prediction,» de *Proceedings of the 55th annual meeting of the association for computational linguistics*, Vancouver, 2017.
- [43] D. Ahlers, «Assessment of the Accuracy of GeoNames Gazetteer Data,» de *Proceedings of*

the 7th Workshop on Geographic Information Retrieval. Pages 74-81., Orlando, Florida, Nov-2013.

- [44] Acheson E., De Sabbata S., Purves R.S., «A quantitative analysis of global gazetteers: Patterns of coverage for common feature types,» *Computers, Environment and Urban Systems*, vol. 64, pp.309-320, 2017.
- [45] Raschka S., Mirjalili V., *Python Machine Learning*, Birmingham: Packt Publishing Ltd., 2017 segunda edición.
- [46] Lee Y.J., Yeh Y.R., Pao H.K., «Support Vector Machines and their applications in bankruptcy prognosis,» *Handbook of Computational Finance*, 2012.
- [47] B. D., «Probabilistic topic models,» *Communications of the ACM*, vol. 55, nº 4, pp. 77-84, 2012.
- [48] Belice L., Büchi M., De Gaetano C., Madlberger L., Mitchell P., Skarpelis A., Stähle H., Van Damme K., Van der Vlist F., «One Percent of Twitter, Part II: Geotags, Text Analysis, and Event Profiling,» *Digital methods initiative*, 2014. [En línea]. Available: <https://wiki.digitalmethods.net/Dmi/DmiSummer2014GeotagsTextAnalysisAndEventProfiling>.
- [49] H. Marius, «Multiclass Classification with Support Vector Machines (SVM), Dual Problem and Kernel Functions,» 9 junio 2020. [En línea]. Available: <https://towardsdatascience.com/multiclass-classification-with-support-vector-machines-svm-kernel-trick-kernel-functions-f9d5377d6f02>. [Último acceso: 23 December 2021].

ANEXO I: CÓDIGO OBTENCIÓN DATOS

- **Código para la obtención de datos de la API de TWITTER y almacenaje en MongoDB.**

FICHERO main.py

```
import tweepy
import json
# twitter_keys es un fichero que incluye las claves de acceso a la API
import twitter_keys as tk
# la clase siguiente se codifica a continuación en fichero a parte
from MyStreamListener import MyStreamListener
from pymongo import MongoClient

# Abrimos conexión con servidor MongoDB local.
client = MongoClient('localhost')
# Seleccionamos la base de datos existente para guardar tweets
db = client['Tweets']
# Autenticación utilizando constantes para acceso API de Twitter
auth = tweepy.OAuthHandler(tk.API_KEY, tk.API_SECRET_KEY)
auth.set_access_token(tk.ACCESS_TOKEN, tk.ACCESS_SECRET_TOKEN)

# Instanciamos API de Tweepy. Parámetros para gestionar rate-limit.
api = tweepy.API(auth, wait_on_rate_limit=True,
wait_on_rate_limit_notify=True)

# ***** Extraer tweets del archivo de Twitter con filtros *****
# los tweets se localizan en el interior de un área circular
# se utiliza la DB['tweets_spain']
for tweet in tweepy.Cursor(api.search, q=" -filter:retweets",
                           geocode="40.307284,-3.685610,700km",
lang="es").items(2500):
    # print(json.dumps(tweet._json, indent=4))
    db['tweets_spain'].insert_one(tweet._json)

# ***** Extraer tweets en español en tiempo real *****
# MyStreamListener es una clase que sobrecarga la clase StreamListener de
# Tweepy. Ver fichero MyStreamListener.py
listener = MyStreamListener()
stream = tweepy.Stream(auth=api.auth, listener=listener)
try:
    stream.sample(languages=['es'])
    """ Bloque para filtrar si se quiere mediante un box de coordenadas
    stream.filter(
        locations=[-9.6290377742, 35.9868750206, 4.4774075383,
                    43.9638685079],
        languages=['es']
    )
    """
except KeyboardInterrupt:
    print("*** Tecla pulsada para interrumpir. ***")
finally:
    print("*** Finalizado. ***")
```

```
stream.disconnect()
```

FICHERO MyStreamListener.py

```
import tweepy
import time
from pymongo import MongoClient

# overriding tweepy.StreamListener
class MyStreamListener(tweepy.StreamListener):
    def __init__(self):
        tweepy.StreamListener.__init__(self)
        # atributo tiempo de espera para rate-limit (segundos)
        self.t_ = 30
        # abrimos conexión con MONGODB, se utiliza la DB['Tweets']
        client = MongoClient('localhost')
        self.db = client['Tweets']

    def on_connect(self):
        print("*** C O N E C T A D O ***")

    def on_status(self, status):
        # muestra el tweet completo en pantalla
        # print(json.dumps(status._json, ensure_ascii=False, indent=4))
        # esta línea imprime solo el texto del tweet
        # print(status.text) # esta línea imprime solo el texto del tweet
        # guardamos información en MONGODB
        self.db['tweets_stream'].insert_one(status._json)

    def on_error(self, status_code):
        print(status_code)
        if status_code == 429:
            # esperamos exponencialmente un tiempo en segundos
            zzz = 2*self.t_
            time.sleep(zzz)
            print("App is being rate-limited, waiting %d seconds...." %zzz)
        else:
            self.t_ = 30
        # devolvemos True para reconexión
        return True

    def __del__(self):
        self.db.close()
```

ANEXO II: CÓDIGO DE LOCALIZACIÓN MEDIANTE DICCIONARIO TOPONÍMICO

- Código para la obtención de las coordenadas de las localizaciones especificadas por los usuarios en su perfil mediante el servicio web proporcionado por *GeoNames*.

FICHERO main.py

```
from pymongo import MongoClient
from geopy.geocoders import GeoNames
from geopy.exc import (GeocoderTimedOut,
                        GeocoderRateLimited,
                        GeocoderQuotaExceeded)

import pandas as pd
# import time

# Lista donde almacenaremos localizaciones
locations = []
# Abrimos acceso a GeoNames con cuenta existente.
# Fijamos el timeout en el máximo que estamos dispuestos a esperar
# respuesta del servicio,
# si se alcanza y no hay respuesta, se generará una excepción
geo = GeoNames(username='ben73and', timeout=100)
# geo = Nominatim(user_agent="MyTFM")

# Abrimos conexión con servidor MongoDB local
client = MongoClient('localhost')
# Seleccionamos la base de datos existente para leer los tweets
db = client['Tweets']
colec = db['tweets_spain']

# Utilizamos un sistema de paginación de los documentos de la BD para
# trabajar con ellos en tandas. Cada ejecución sucesiva requiere aumentar
# el page_number.
page_number = 1
page_size = 500
skips = page_size * (page_number - 1)
# Los documentos están ordenados por _id.
# Gracias a eso, funciona la paginación.
cursor = colec.find({}, {"_id": 0, "user.id_str": 1, "user.location":
1}).skip(skips).limit(page_size)
i = 1
for x in cursor:
    if x is not None:
        s = x['user']['location']
        # print(s)
        lat = lon = 0.0
        if s != "":
            # mostramos el número de veces que hemos llamado
            # al servicio de localización
            print(i)
            i += 1
            try:
                loc = geo.geocode(s)
```

```
except GeocoderTimedOut:
    print("El servicio de geocoding no responde. Time-out
          alcanzado.")
    break
except GeocoderRateLimited as e:
    print("Velocidad de solicitud limitada.")
    if e.retry_after is not None:
        print("Hay que esperar %s segundos" % e.retry_after)
    break
except GeocoderQuotaExceeded:
    print("Se ha alcanzado la cuota límite de acceso del
          usuario.")
    break
else:
    if loc is not None:
        print(loc)
        lat = loc.latitude
        lon = loc.longitude
        # Nominatim está limitado a una petición por segundo.
        # activar siguiente línea
        # time.sleep(1)
    locations.append([x['user']['id_str'], s, lat, lon])
df = pd.DataFrame(data=locations, columns=['user_id', 'loc', 'lat', 'lon'])
# almacenamos las localizaciones en un fichero csv (a = append)
df.to_csv("loc01.csv", index=False, mode='a')
```

ANEXO III: REPRESENTACIÓN LOCALIZACIONES

- **Código para la representación gráfica, en un mapa, de las coordenadas obtenidas por algún método de localización.**

FICHERO main.py

```
# Se hace uso de la librería Basemap descrita en:
# https://basemaptutorial.readthedocs.io/en/latest/

from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import pandas as pd

# Un tamaño arbitrario para dibujar el mapa
fig = plt.figure(figsize=(8, 8))
# Un punto central y unas medidas de ancho y alto en metros
# (valores para mapa en torno a la Península Ibérica)
m = Basemap(projection='lcc', resolution='l',
            width=14E5, height=14E5,
            lat_0=40.3, lon_0=-3.68,)
m.drawcoastlines(color='gray')
m.drawcountries(color='gray')
# leemos las localizaciones de un fichero csv
locations = pd.read_csv('data\location.csv')
lats = locations['lat']
lons = locations['lon']
# dibujamos los puntos correspondientes a los vectores
m.scatter(lons, lats, latlon=True, s=2, color='r')
# mostramos el resultado
plt.show()
```

ANEXO IV: CÓDIGO LOCALIZACIÓN MEDIANTE ALGORITMOS DE MACHINE-LEARNING

- Código para la extracción de una base de datos MongoDB de los tweets procedentes de localizaciones especificadas en el perfil de los usuarios.

FICHERO myfuncs.py

```
from pymongo import MongoClient
import os
from bson.json_util import dumps
import pandas as pd

def tweets2json():
    # Guarda todos los tweets de una colección MongoDB en json
    # Accedemos a la base de datos
    client = MongoClient('localhost')
    # Especificar db y colec a conveniencia
    db = client['Tweets']
    colec = db['tweets_spain_ext']
    # Hacemos una consulta mediante agregación
    # para filtrar tweets de Madrid.
    cursor = colec.aggregate([
        {
            "$match": {
                "$or": [
                    {"user.location": "Madrid"},
                    {"user.location": "Madrid, Comunidad de Madrid"}
                ]
            },
        },
        {
            "$project": {
                "_id": 0,
                "full_text": 1
            }
        }
    ])
    '''
    # El código siguiente es similar, pero permite almacenar más
    # información sobre los tweets.
    cursor = colec.aggregate([
        {
            "$match": {
                "$or": [
                    {"user.location": "Madrid"},
                    {"user.location": "Madrid, Comunidad de Madrid"}
                ]
            },
        },
        {
            "$group": {
```

```

        "_id": "$user.id",
        "tweets": {
            "$addToSet": "$full_text"
        },
        "num_tweets": {
            "$count": {}
        }
    }
}

])
'''

# guardamos los tweets en un fichero json
lista = list(cursor)
print("Total elementos en tweets_mad:", len(lista))
json_data = dumps(lista, indent=2)
with open('data/tweets_mad.json', 'w') as file:
    file.write(json_data)
# Ahora filtramos los tweets de Barcelona
cursor = colec.aggregate([
    {
        "$match": {
            "$or": [
                {"user.location": "Barcelona"},
                {"user.location": "Barcelona, España"},
                {"user.location": "Barcelona, Cataluña"}
            ]
        },
    },
    {
        "$project": {
            "_id": 0,
            "full_text": 1
        }
    }
])
# Podríamos utilizar un código similar al "comentado" anteriormente
# para obtener un JSON con más información.
# Guardamos los tweets en un fichero json
lista = list(cursor)
print("Total elementos en tweets_bar:", len(lista))
json_data = dumps(lista, indent=2)
with open('data/tweets_bar.json', 'w') as file:
    file.write(json_data)

def tweets2csv():
    # Guarda todos los tweets de una colección MongoDB en CSV
    # Accedemos a la base de datos
    client = MongoClient('localhost')
    # Especificar db y colec a conveniencia
    db = client['Tweets']
    colec = db['tweets_spain_ext']
    # Hacemos una consulta mediante agregación
    # para filtrar tweets de Madrid
    cursor = colec.aggregate([
        {

```

```
        "$match": {
            "$or": [
                {"user.location": "Madrid"},
                {"user.location": "Madrid, Comunidad de Madrid"}
            ]
        },
        {
            "$project": {
                "_id": 0,
                "full_text": 1
            }
        }
    ])
# Guardamos los tweets en un fichero csv
# Pasamos antes por una lista para evitar incluir en el csv los pares
# clave valor y otra información innecesaria
l_cursor = []
for i in cursor:
    l_cursor.append(*i.values())
s = pd.Series(l_cursor)
print("Total elementos en tweets_mad:", s.count())
s.to_csv('data/tweets_mad.csv', header=False)
# Para filtrar tweets de Barcelona
cursor = colec.aggregate([
    {
        "$match": {
            "$or": [
                {"user.location": "Barcelona"},
                {"user.location": "Barcelona, España"},
                {"user.location": "Barcelona, Cataluña"}
            ]
        },
        {
            "$project": {
                "_id": 0,
                "full_text": 1
            }
        }
    ])
l_cursor = []
for i in cursor:
    l_cursor.append(*i.values())
s = pd.Series(l_cursor)
print("Total elementos en tweets_bar:", s.count())
s.to_csv('data/tweets_bar.csv', header=False)
```

```
def alltweets2csv():
    # Guarda todos los tweets de una colección MongoDB en csv
    # Requiere la instalación en local de las Tools de Mongo que se pueden
    # obtener en https://www.mongodb.com/try/download/database-tools
    # utilizamos la ruta por defecto, cambiar según instalación.
```

```

str_mongoexport = '"C:\\Program
Files\\MongoDB\\Tools\\100\\bin\\mongoexport"'
str_database = " -d Tweets"
str_collection = " -c tweets_spain_ext"
# si los tweets son de hasta 280 caracteres
str_fields = " --fields full_text"
# si los tweets son de 140 caracteres
# str_fields ="text"
# resulta difícil incluir --query por el hueco de "Program Files" y el
uso de "/"
str_export_to = " -o data\\tweets.csv"
cmd_str = str_mongoexport + str_database + str_collection \
    + str_fields + " --type csv --noHeaderLine " + str_export_to
print("Ejecutando exportación... ", cmd_str)
os.system(cmd_str)

```

- **Código de preprocesado del texto de los tweets.**

FICHERO mytextfuncs.py

```

import re
import emoji

def clean_text(text):
    # convertir a minúsculas, eliminar signos de puntuación, etc.
    text = str(text).lower()
    # eliminar nuevas líneas
    text = re.sub("\n", " ", text)
    # eliminar urls
    text = re.sub(r"\S*https?:\S*", " ", text)
    # eliminar menciones (@fulano)
    text = re.sub(r"@S+", " ", text)
    # eliminar cadenas con números
    text = re.sub(r"\w*\d\w*", " ", text)
    # eliminar palabras con menos de 3 caracteres
    text = re.sub(r'\b\w{1,2}\b', '', text)
    # eliminar emojis
    text = re.sub(emoji.get_emoji_regexp(), r"", text)
    # punctuators tiene 33 caracteres
    punctuators = '!"#$%&\'()*+,-./:;<=>¿@[\\]^_`{|}~'
    # vacio tiene que tener los mismos caracteres que punctuators
    vacio = ' '
    # la siguiente línea elimina signos de puntuación
    # return text.translate(str.maketrans('', '', punctuators))
    # la siguiente línea sustituye un signo de puntuación por un espacio
    text = text.translate(str.maketrans(punctuators, vacio))
    return text

```

- **Código de principal de clasificación de los tweets procedentes de Madrid y México mediante algoritmos de Machine-Learning.**

FICHERO main.py

```
import myfuncs
import gc
import sys
import mytextfuncs
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# PASO 1: De MongoDB a fichero csv
myfuncs.alltweets2csv("tweets_madrid_ext")
myfuncs.alltweets2csv("tweets_mexico_ext")

# PASO 2: De fichero csv a pandas-series
s_mad = pd.read_csv('data/tweets_madrid_ext.csv', header=None,
names=['tweets'], index_col=False)
s_mex = pd.read_csv('data/tweets_mexico_ext.csv', header=None,
names=['tweets'], index_col=False)

# PASO 3: Preproceso de datos
# limpiamos tweets con función clean_text
s_mad_clean = pd.Series(s_mad.tweets.apply(lambda x:
mytextfuncs.clean_text(x)))
s_mex_clean = pd.Series(s_mex.tweets.apply(lambda x:
mytextfuncs.clean_text(x)))

# PASO 4: Obtención de características
# tokenizamos y eliminamos las stopwords
stopwords_sp = list(stopwords.words('spanish'))
cv = CountVectorizer(stop_words=stopwords_sp)
cv_result_mad = cv.fit_transform(s_mad_clean)
cv_result_mad = cv_result_mad.toarray()
mad_counts = pd.DataFrame(cv_result_mad, columns=cv.get_feature_names_out())
cv_result_mex = cv.fit_transform(s_mex_clean)
cv_result_mex = cv_result_mex.toarray()
mex_counts = pd.DataFrame(cv_result_mex, columns=cv.get_feature_names_out())

print("Información sobre el conjunto Madrid antes del preproceso: ")
print(mad_counts.info(verbose=False))
print("Información sobre el conjunto Mexico antes del preproceso: ")
print(mex_counts.info(verbose=False))
```

```

# Eliminamos palabras con una frecuencia por debajo de un umbral
# 1. Obtenemos total de palabras
mad_sum_words = pd.DataFrame(index=mad_counts.index)
mad_counts['sum'] = mad_counts.sum(axis=1)
tot_mad_words = mad_counts['sum'].sum()
mex_counts['sum'] = mex_counts.sum(axis=1)
tot_mex_words = mex_counts['sum'].sum()
mad_counts.drop(['sum'], axis=1, inplace=True)
mex_counts.drop(['sum'], axis=1, inplace=True)
print("Número total de palabras en conjunto Madrid: ", tot_mad_words)
print("Número total de palabras en conjunto México: ", tot_mex_words)

# 2. Eliminamos columnas con palabras poco frecuentes (0.3 por mil)
# Calculamos cuántas palabras supone una frecuencia del 0.3 por mil
umbral = 0.0003
mad_min_words = round(tot_mad_words * umbral)
mex_min_words = round(tot_mex_words * umbral)
print("El número mínimo de palabras para el 0.03% en el conjunto Madrid: ",
mad_min_words)
print("El número mínimo de palabras para el 0.03% en el conjunto Mexico: ",
mex_min_words)
mad_droplist = [i for i in mad_counts.columns if mad_counts[i].sum() <
mad_min_words]
mex_droplist = [i for i in mex_counts.columns if mex_counts[i].sum() <
mex_min_words]

# Las palabras que tengan pocas apariciones en un grupo no se eliminan si
# tienen suficientes apariciones en el otro
for i in mad_droplist:
    if i not in mex_droplist and i in mex_counts.columns:
        mad_droplist.remove(i)
for i in mex_droplist:
    if i not in mad_droplist and i in mad_counts.columns:
        mex_droplist.remove(i)
mad_counts.drop(mad_droplist, axis=1, inplace=True)
mex_counts.drop(mex_droplist, axis=1, inplace=True)
print("Columnas del conjunto Madrid eliminadas: ", len(mad_droplist))
print("Columnas del conjunto Mexico eliminadas: ", len(mex_droplist))

# ADICIONAL: Representación gráfica
# Nube de Palabras con tweets de Madrid
# incluimos las palabras eliminadas junto con las stopwords originales
stopwords_mad = stopwords_sp + mad_droplist
print("Número de stopwords en Madrid", len(stopwords_mad))
tweets_text = " ".join(twt for twt in s_mad_clean)
wc = WordCloud(stopwords=stopwords_mad, background_color='white',
               max_words=100, relative_scaling=1).generate(tweets_text)
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.show()

# Nube de palabras con tweets de México
stopwords_mex = stopwords_sp + mex_droplist
print("Número de stopwords en Mexico", len(stopwords_mex))

```

```
tweets_text = " ".join(twt for twt in s_mex_clean)
wc = WordCloud(stopwords=stopwords_mex, background_color='white',
               max_words=100, relative_scaling=1).generate(tweets_text)
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.show()

# Barras horizontales con tweets de Madrid
mad_tot = pd.Series([mad_counts[i].sum() for i in mad_counts.columns],
                    index=mad_counts.columns, name='total')
mad_tot.sort_values(ascending=False).head(20).plot.barh()
plt.show()

# Barras horizontales con tweets de Mexico
mex_tot = pd.Series([mex_counts[i].sum() for i in mex_counts.columns],
                    index=mex_counts.columns, name='total')
mex_tot.sort_values(ascending=False).head(20).plot.barh()
plt.show()

# 3. eliminamos filas (tweets) que no tengan ninguna palabra frecuente
mad_counts['sum'] = mad_counts.sum(axis=1)
mex_counts['sum'] = mex_counts.sum(axis=1)
mad_droplist = [i for i in mad_counts.index if mad_counts.loc[i, 'sum'] == 0]
mex_droplist = [i for i in mex_counts.index if mex_counts.loc[i, 'sum'] == 0]

...

# imprimir tweets eliminados para supervisión
pd.options.display.max_colwidth = 280
print("Tweets eliminados del conjunto Madrid:")
for i in mad_droplist:
    print(s_mad.loc[i, 'tweets'])
print("Tweets eliminados del conjunto Barcelona:")
for i in bar_droplist:
    print(s_bar.loc[i, 'tweets'])
...

mad_counts.drop(['sum'], axis=1, inplace=True)
mex_counts.drop(['sum'], axis=1, inplace=True)
mad_counts.drop(mad_droplist, axis=0, inplace=True)
mex_counts.drop(mex_droplist, axis=0, inplace=True)
print("Tweets del conjunto Madrid sin ninguna palabra significativa: ",
      len(mad_droplist))
print("Tweets del conjunto Mexico sin ninguna palabra significativa: ",
      len(mex_droplist))
print("Información sobre el conjunto Madrid después del preproceso: ")
print(mad_counts.info(verbose=False))
print("Información sobre el conjunto Mexico después del preproceso: ")
print(mex_counts.info(verbose=False))

# guardamos la suma de palabras en un dataframe
mad_sum_words = pd.DataFrame(columns=mad_counts.columns)
mad_sum_words.loc['mad_sum_words', :] = mad_counts.sum(axis=0)
mex_sum_words = pd.DataFrame(columns=mex_counts.columns)
mex_sum_words.loc['mex_sum_words', :] = mex_counts.sum(axis=0)
```

```

sum_words = pd.concat([mad_sum_words, mex_sum_words], join='outer',
).fillna(0)
print()

# CREAMOS los conjuntos de test, entrenamiento y resultados
# 1. Creamos el conjunto de resultados
# La categoría Madrid tendrá el valor numérico 0
y = np.full(len(mad_counts.index), 0)
# Añadimos los valores de Mexico = 1
y = np.pad(y, (0, len(mex_counts.index)), constant_values=1)
print("Matriz de y: ", y.shape)

# 2. Agrupamos los datos de Madrid con los de Mexico
X = pd.concat([mad_counts, mex_counts], join="outer",
ignore_index=True).fillna(0).to_numpy()

print("Matriz de X: ", X.shape)
print("suma en X", X[:, 0].sum())
print("suma en sum_words", sum_words.iloc[:, 0].sum())
print("palabra: ", sum_words.columns[0])

# liberamos memoria....
# Las líneas siguientes son para ver el uso de la memoria
# print('*****MEMORIA*****')
# for name, size in sorted(((name, sys.getsizeof(value)) for name, value in
locals().items()),
#                               key=lambda x: -x[1])[:10]:
#     print("{:>20}: {:>12}".format(name, size))
# print('*****')
del cv_result_mad
del cv_result_mex
del s_mad
del s_mex
del s_mad_clean
del s_mex_clean
del mad_counts
del mex_counts
del stopwords_sp
del mad_droplist
del mex_droplist
gc.collect()

# 3. Repartimos las muestras: 70% para entrenamiento y 30% para test
# consultar https://realpython.com/train-test-split-python-data/
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=1, stratify=y)
print("Matriz de X_train: ", X_train.shape)
print("Matriz de X_test: ", X_test.shape)
print("Matriz de y_train: ", y_train.shape)
print("Matriz de y_test: ", y_test.shape)
# con stratify mantenemos la proporción de etiquetas de clase en los
conjuntos de test y training
print('Número de etiquetas de clase en y:', np.bincount(y))
print('Número de etiquetas de clase en y_train:', np.bincount(y_train))

```

```
print('Número de etiquetas de clase en y_test', np.bincount(y_test))

# 4. Estandarizamos las muestras. Reseñar que la media y la devstd se
calculan
# a partir de X_train y se aplican tanto a X_train como a X_test.
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

# PASO 5: Utilizamos algoritmos de ML
# Utilizamos LOGISTIC REGRESSION
lr = LogisticRegression(C=0.001, random_state=1, max_iter=10000)
lr.fit(X_train_std, y_train)
y_pred = lr.predict(X_test_std)
print("Score test: ", lr.score(X_test_std, y_test))
print("Score train: ", lr.score(X_train_std, y_train))
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

# Utilizamos SVM
# Con GridSearchCV analizamos los hiperparámetros
param = {'C': [100000, 10000, 1000, 100],
         'gamma': [1e-09, 1e-08, 1e-07, 1e-06, 1e-5],
         'kernel': ['rbf']}
gs = GridSearchCV(SVC(), param, n_jobs=-1, refit=True, cv=3, verbose=2)
gs.fit(X_train_std, y_train)
print(gs.best_estimator_)
y_pred = gs.predict(X_test_std)
# alternatively podemos utilizar una llamada directa a la función SVC
# svm = SVC(kernel='rbf', random_state=1, C=1, gamma=0.001)
# svm.fit(X_train_std, y_train)
# y_pred = svm.predict(X_test_std)
print("Score test: ", gs.score(X_test_std, y_test))
print("Score train: ", gs.score(X_train_std, y_train))
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

# Representación gráfica de la variación de la exactitud en función de gamma
y C
scores = gs.cv_results_['mean_test_score']
scores = np.array(scores).reshape(len(param['C']), len(param['gamma']))
print(scores)
plt.figure(figsize=(5, 4))
plt.subplots_adjust(left=0.15, right=0.95, bottom=0.15, top=0.95)
plt.imshow(scores, interpolation='nearest', cmap='coolwarm')
plt.xlabel('Valor de gamma')
plt.ylabel('Valor de C')
plt.colorbar()
plt.xticks(np.arange(len(param['gamma'])), param['gamma'], rotation=45)
plt.yticks(np.arange(len(param['C'])), param['C'])
plt.show()
```

```

# Utilizamos RANDOM FOREST
# param para Madrid-México
param = {'n_estimators': [2, 3, 5, 8],
        'max_features': ['auto'],
        'max_depth': [100, 250, 500, 750, 1000, 1500],
        'criterion': ['gini']}
gs = GridSearchCV(RandomForestClassifier(random_state=1), param, n_jobs=-1,
refit=True, cv=3, verbose=2)
gs.fit(X_train_std, y_train)
print(gs.best_estimator_)
y_pred = gs.predict(X_test_std)
print("Score test: ", gs.score(X_test_std, y_test))
print("Score train: ", gs.score(X_train_std, y_train))
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

# Representación gráfica de la variación de la exactitud en función de
n_estimators y max_depth
scores = gs.cv_results_['mean_test_score']
scores = np.array(scores).reshape(len(param['n_estimators']),
len(param['max_depth']))
plt.figure(figsize=(5, 4))
plt.subplots_adjust(left=0.15, right=0.95, bottom=0.15, top=0.95)
plt.imshow(scores, interpolation='nearest', cmap='coolwarm')
plt.xlabel('Valor de max_depth')
plt.ylabel('Valor de n_estimators')
plt.colorbar()
plt.xticks(np.arange(len(param['max_depth'])), param['max_depth'],
rotation=45)
plt.yticks(np.arange(len(param['n_estimators'])), param['n_estimators'])
plt.show()

# representación gráfica de las características más importantes (Random
Forest)
features = gs.best_estimator_.feature_importances_
plt.bar([x for x in range(len(features))], features)
plt.show()
# imprimimos las palabras más significativas y el número de apariciones en
los conjuntos de Madrid y Barcelona/México
min_important_features = np.amin([x for index, x in
enumerate(sorted(features, reverse=True)) if index < 8])
indexes = [index for index, x in enumerate(features) if x >=
min_important_features]
for i in range(len(indexes)):
    w = sum_words.columns[indexes[i]]
    # print(f"palabra significativa: {w} ")
    #     f"(Madrid: {sum_words.loc['mad_sum_words', w]}) "
    #     f"(Barcelona: {sum_words.loc['bar_sum_words', w]})")
    print(f"palabra significativa: {w} "
          f"(Madrid: {sum_words.loc['mad_sum_words', w]}) "
          f"(México: {sum_words.loc['mex_sum_words', w]})")

```

ANEXO V: CÓDIGO LOCALIZACIÓN MEDIANTE ALGORITMOS DE ANÁLISIS DE REDES DE USUARIOS

- **Código para la extracción de usuarios de una base de datos MongoDB, conformación del grafo de “followers” mediante consultas a la API de Twitter y almacenamiento, para posterior análisis.**

```
from pymongo import MongoClient
from tweepy import API, Cursor, OAuthHandler, TweepyException
import twitter_keys as tk # fichero local con las claves de Twitter
import pandas as pd
import json
# requiere incluir scipy en el venv
import networkx as nx
import matplotlib.pyplot as plt

# 1. ACCESO A LA API DE TWITTER
# Autenticación utilizando constantes para utilizar API de Twitter
auth = OAuthHandler(tk.API_KEY, tk.API_SECRET_KEY)
auth.set_access_token(tk.ACCESS_TOKEN, tk.ACCESS_SECRET_TOKEN)

# Instanciamos API de Tweepy.
# Parámetro para gestionar rate-limit. Versión 4.4 ya no utiliza
wait_on_rate_limit_notify
api = API(auth, wait_on_rate_limit=True)

# 2. ACCESO A LISTA DE USARIOS EN MONGODB
# Abrimos conexión con servidor MongoDB local
client = MongoClient('localhost')
# Seleccionamos la base de datos existente para guardar Tweets
db = client['Tweets']
colec = db['tweets_mad2']
# Extraemos los usuarios que han escrito los tweets y los guardamos en una
lista
# limitamos a 5000 usuarios, por conveniencia en acceso a API y por efecto
celebrities
user_list = list(colec.distinct("user.id", {"user.followers_count": {"$lte":
5000}}))
# ordenamos la lista para poder reiniciar si el proceso se interrumpe
user_list.sort()
total_users = len(user_list)
print("Total usuarios distintos en tweets_mad2 con menos de 5000 followers",
total_users)

...

# 2. (ALTERNATIVO) RELLENAMOS LA LISTA DE USUARIOS DESDE UN JSON
# Útil si se exportan los datos desde la BD cuando la consulta es compleja.
f = open('data\\usuarios_mad_bar.json')
data = json.load(f)
user_list = [user["user"]["id"] for user in data]
user_list.sort()
```


- **Código para la lectura de un grafo almacenado en csv, etiquetado de un cierto número de nodos, ejecución del algoritmo de propagación de etiquetas y visualización.**

```

import igraph as ig
import pandas as pd
import matplotlib.pyplot as plt
import random
import numpy as np

def draw_graph(graph, color_list):
    # fig, ax = plt.subplots(figsize=(10, 8)) # solo para matplotlib
    # ax.axis('off') # solo para matplotlib
    visual_style = dict()
    visual_style["layout"] = G.layout("drl")
    visual_style["bbox"] = (1000, 800)
    visual_style["vertex_size"] = 10
    # visual_style["vertex_label"] = g.vs["name"] # si hay muchos nodos no
    usar
    visual_style["vertex_color"] = color_list
    visual_style["vertex_frame_color"] = "#00000030" # solo para pycairo
    visual_style["vertex_frame_width"] = 1
    # visual_style["mark_groups"] = True # poco útil con muchos nodos y
    clústers
    # solo con VertexCollection
    visual_style["edge_width"] = 0.1
    # visual_style["target"] = ax # solo para matplotlib
    ig.plot(graph, **visual_style) # Dibujar el grafo completo
    # ig.plot(labelProp, **visual_style) # Dibujar VertexCollection
    plt.show()

# ***** MAIN *****

# INICIALIZACIÓN DEL GRAFO
temp = pd.read_csv("data\\net_follow_mad.csv") # leemos el csv con los datos
del grafo.
G = ig.Graph.DataFrame(temp, directed=False) # convertimos de pandas a
igraph
num_nodes = G.vcount()
print(f"El número de nodos es: {num_nodes}")
if num_nodes < 2:
    print("El grafo tiene menos de dos nodos. Finalizando.")
    exit(0)
# print(g.vs["name"]) # aquí están los nombres de los nodos (números de
usuario de Twitter)

# SELECCIÓN DE NODOS INICIALES
initial = np.full(num_nodes, -1) # Todos los nodos sin etiqueta
fixed = np.full(num_nodes, False) # Ninguna etiqueta fija
random.seed(6)
'''

# a) Etiquetamos un porcentaje fijo de nodos iniciales
proportion_labels = 0.1 # porcentaje de nodos a etiquetar

```

```
num_labeled_nodes = int(proportion_labels * num_nodes) # número de nodos a
etiquetar
num_clusters = 1 # número de comunidades distintas
print(f"Se etiquetan el {proportion_labels*100}% de los nodos. Total =
{num_labeled_nodes}")
labeled_nodes = random.sample(G.vs.indices, num_labeled_nodes)
initial[labeled_nodes] = 1 # Etiquetamos algunos nodos
fixed[labeled_nodes] = True # Los nodos etiquetados no pueden cambiar

# coloreamos nodos etiquetados para representación
color = ig.color_name_to_rgba("rgba(100%, 100%, 100%, 0%)") # inicialmente
todos los nodos transparentes.
colorlist = [color] * num_nodes
for n in labeled_nodes:
    colorlist[n] = "#FF0000" # color rojo
draw_graph(colorlist)
'''

# b) Alternativamente etiquetamos únicamente num_labeled_nodes nodos con
diferentes etiquetas
num_labeled_nodes_per_cluster = 1 # nodos etiquetados en cada comunidad
num_clusters = 4 # número de comunidades distintas
num_labeled_nodes = num_labeled_nodes_per_cluster * num_clusters # número
total de nodos etiquetados

labeled_nodes = random.sample(G.vs.indices, num_labeled_nodes)
i = 0
for n in labeled_nodes:
    initial[n] = i % num_clusters
    print(f"Se ha etiquetado el nodo {n}, con la etiqueta {i %
num_clusters}")
    fixed[n] = True
    i += 1

# coloreamos nodos etiquetados para representación
# colors = ["red", "blue", "green", "purple", "orange"]
colors = [ # colores con cierta transparencia para mayor visualización
    (1.0, 0.0, 0.0, 0.3), # red
    (0.0, 0.0, 1.0, 0.3), # blue
    (0.0, 1.0, 0.0, 0.8), # green
    (0.5, 0.0, 0.5, 0.8), # purple
    (1.0, 0.5, 0.0, 0.8) # naranja
]
color = ig.color_name_to_rgba("rgba(100%, 100%, 100%, 0%)") # inicialmente
todos los nodos transparentes.
colorlist = [color] * num_nodes

for n in labeled_nodes:
    c = colors[initial[n] % len(colors)]
    colorlist[n] = c
    print(f"El nodo {n}, ha sido coloreado con el color {c} y tiene la
etiqueta {initial[n]}")
# draw_graph(G, colorlist) # visualizar los nodos etiquetados inicialmente
# Esta visualización no resulta relevante puesto que la representación no es
determinista:
```

el grafo dibujado ahora, no será igual al que se dibuje tras la propagación de etiquetas.

PROPAGACIÓN DE ETIQUETAS

```
labelProp = G.community_label_propagation(initial=initial, fixed=fixed)
# labelProp = G.community_label_propagation() # si no utilizamos etiquetado
# inicial
num_clusters = len(labelProp)
print(f"PROPAGACIÓN DE ETIQUETAS. El número de comunidades es:
{num_clusters}")
```

```
print("*****")
for n in labeled_nodes:
    for c in range(num_clusters):
        if n in labelProp[c]:
            print(f"El nodo {n} tiene la etiqueta {c}")
print("*****")
```

GRÁFICA DE TAMAÑO DE CLUSTERS

```
if num_clusters > 1:
    eje_x = list(range(num_clusters))
    eje_y = [len(labelProp[x]) for x in range(num_clusters)]
    plt.barh(eje_x, eje_y, color=colors)
    for i, v in enumerate(eje_y):
        plt.text(v + 1, i, str(v), fontsize="small", color="darkgrey")
    plt.show()
else:
    print(f"El número de nodos en la única comunidad es:
{len(labelProp[0])}")
```

REPRESENTACIÓN DEL GRAFO

```
'''
# a) Coloreamos solo los nodos que pertenezcan a comunidades con más nodos
que cierto umbral
min_nodes_in_cluster = 20
big_clusters = []
for c in range(num_clusters):
    if len(labelProp[c]) > min_nodes_in_cluster:
        big_clusters.append(labelProp[c])
print(f"Número de clústers con más de {min_nodes_in_cluster}:
{len(big_clusters)}")

palette = ig.ClusterColoringPalette(num_clusters)
# palette = ig.RainbowPalette(n=len(clusters))
color = ig.color_name_to_rgb("rgba(100%, 100%, 100%, 0%)") # inicialmente
todos los nodos transparentes.
colorlist = [color] * num_nodes
for c in range(len(big_clusters)):
    for n in big_clusters[c]:
        colorlist[n] = palette.get(c)
draw_graph(colorlist) # dibujamos el grafo
'''
```

```
# b) Alternativamente coloreamos todos los nodos, si hay muchas comunidades,
es conveniente utilizar una paleta
# palette = ig.ClusterColoringPalette(num_clusters)
colored_nodes = 0
for c in range(num_clusters):
    col = colors[c % len(colors)]
    print("-----")
    print(f"La comunidad {c} tiene el color {col}")
    colored_nodes += len(labelProp[c])
    print(f"La comunidad {c} tiene {len(labelProp[c])} nodos:
{labelProp[c]}")
    for n in labelProp[c]:
        colorlist[n] = col
        # colorlist[n] = palette.get(c)
print(f"El total de nodos coloreados es: {colored_nodes}
({colored_nodes/num_nodes*100:5.2f}%)")
draw_graph(G, colorlist)
```

- **Extracto del contenido del fichero graph2_mad_bar.csv**

```
760839,415511675
760839,4779511
760839,409869924
760839,199033833
760839,836206736059076608
[...]
sigue hasta 5878 líneas
...]
```

- **Código para la generación de un grafo a partir de datos csv con datos de usuarios procedentes de Madrid y Barcelona. Aplicación del algoritmo de propagación de etiquetas y obtención del informe de clasificación y matriz de confusión.**

```
import igraph as ig
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# LEEMOS EL GRAFO DEL CSV A UN DATAFRAME
temp = pd.read_csv("data\\graph2_mad_bar.csv") # leemos el csv con los datos
del grafo.
temp.columns = ["source", "target", "location"]
print("El número de filas de temp es: ", len(temp))
users_df = temp.copy()
users_df.drop("target", axis=1, inplace=True)
users_df.drop_duplicates(subset=["source"], ignore_index=True, inplace=True)
print("El número de filas de users_df es: ", len(users_df))
mad_users = users_df[users_df["location"] == "Madrid"]
num_mad_users = len(mad_users)
print(f"En número de usuarios de Madrid es: {num_mad_users}")
bar_users = users_df[users_df["location"] == "Barcelona"]
```

```

num_bar_users = len(bar_users)
print(f"En número de usuarios de Barcelona es: {num_bar_users}")

# INICIALIZACIÓN DEL GRAFO
G = ig.Graph.DataFrame(temp, directed=False) # convertimos de pandas a
igraph
num_nodes = G.vcount()
print(f"El número de nodos es: {num_nodes}")

# SELECCIÓN DE NODOS INICIALES
initial = np.full(num_nodes, -1) # Todos los nodos sin etiqueta
fixed = np.full(num_nodes, False) # Ninguna etiqueta fija
prop_labels = 0.3 # porcentaje de nodos a etiquetar
num_mad_lab_nodes = int(prop_labels * num_mad_users) # número de nodos a
etiquetar de Madrid
num_bar_lab_nodes = int(prop_labels * num_bar_users) # número de nodos a
etiquetar de Barcelona
num_clusters = 2 # número de comunidades distintas
print(f"Se etiquetan el {prop_labels*100}% de los nodos de Madrid. Total =
{num_mad_lab_nodes}")
print(f"Se etiquetan el {prop_labels*100}% de los nodos de Barcelona. Total =
{num_bar_lab_nodes}")
print("INFORMACIÓN DEPENDIENTE DE VARIABLE ALEATORIA")
# Los nodos escogidos en cada conjunto son siempre los mismos, mientras los
random_state no varíen, aunque ejecutemos el programa sucesivas veces,
mad_lab_nodes y bar_lab_nodes serán idénticos.
mad_lab_nodes = mad_users.sample(n=num_mad_lab_nodes, random_state=6) #
nodos aleatorios en Madrid
bar_lab_nodes = bar_users.sample(n=num_bar_lab_nodes, random_state=3) #
nodos aleatorios en Barcelona
# Inicializamos nodos de Madrid
list_labels = [1, 2] # por si se utilizan más etiquetas
for index, row in mad_lab_nodes.iterrows():
    ind = G.vs.find(name=row["source"]).index
    initial[ind] = list_labels[0]
    fixed[ind] = True
# Inicializamos nodos de Barcelona
for index, row in bar_lab_nodes.iterrows():
    ind = G.vs.find(name=row["source"]).index
    initial[ind] = list_labels[1]
    fixed[ind] = True
# PROPAGACIÓN DE ETIQUETAS
# Cada ejecución del labelProp es diferente, aunque los nodos etiquetados
sean los mismos.
labelProp = G.community_label_propagation(initial=initial, fixed=fixed)
num_clusters = len(labelProp)
print(f"PROPAGACIÓN DE ETIQUETAS. El número de comunidades es:
{num_clusters}")
total_labeled = 0
for i in range(num_clusters):
    n = len(labelProp[i])
    print(f"La comunidad con etiqueta {i}, tiene {n} elementos.")
    total_labeled += n
# Aquí se están contando todos los nodos del grafo, no solo los procedentes
de "source"

```

```
print(f"Se han etiquetado {total_labeled} nodos de un total de {num_nodos}")
y_true = np.array([1 if users_df.loc[x]["location"] == "Madrid" else 2 for x
in range(len(users_df))])
y_pred = np.full(len(users_df), 0) # nodos sin etiqueta
for i in range(num_clusters):
    print(f"tamaño de labelProp[{i}] = {len(labelProp[i])}")
print("_____")
for c in range(len(labelProp)):
    for n in labelProp[c]:
        ind = users_df.index[users_df["source"] == G.vs["name"][n]].tolist()
        if len(ind) == 0:
            # Puesto que en el grafo hay nodos sumidero, es posible que no
los
            encontremos en users_df
            # print("no se ha encontrado el valor ", G.vs["name"][n])
            pass
        else:
            # print("El índice es: ", ind[0]) # ind solo debe contener un
valor
            # print("La etiqueta del grafo es: ", G.vs["name"][n])
            # print("La etiqueta del dataframe es: ", users_df.loc[ind[0],
"source"])
            y_pred[ind] = list_labels[c]

# Hay que eliminar las filas de los nodos no etiquetados para obtener el
classification_report
rows_not_labeled = [i for i in range(len(y_pred)) if y_pred[i] == 0]
y_true2 = np.delete(y_true, rows_not_labeled, axis=0)
y_pred2 = np.delete(y_pred, rows_not_labeled, axis=0)

print(f"Número de nodos <source> no etiquetados: {len(rows_not_labeled)} de
un total de {len(users_df)} "
      f"({len(rows_not_labeled)*100/len(users_df):.2f}%")
print(f"La precisión obtenida es del {accuracy_score(y_true2,
y_pred2)*100:.2f}%")
print(classification_report(y_true2, y_pred2, target_names=["Madrid",
"Barcelona"]))
print("Matriz de confusión:")
print(confusion_matrix(y_true2, y_pred2))
```

- **Extracto del contenido del fichero graph2_mad_bar.csv**

```
2199211,2679741,Madrid
2199211,202984975,Madrid
2199211,335560498,Madrid
2199211,574157240,Madrid
2199211,209216680,Madrid
2679741,2199211,Barcelona
3117141,586025465,Madrid
[...]
sigue hasta 5196 líneas
...]
```