.UBAfiuba
FACULTAD DE INGENIERÍA

Desarrollo y evaluación de una metodología para la ubicación de usuarios de Twitter

Tesis de grado de
Ingeniería Informática

Tesista: Federico Martín Funes

Director: Dr. Ing. Mariano G. Beiró

Co-director: Dr. Ing. J. Ignacio Alvarez-Hamelin

Noviembre 2021

Índice general

1. Introducción **5**

1.1. Motivación 5

1.2. Estado del arte 6

2. Metodología **11**

2.1. Grafos en redes sociales 11

2.1.1. Redes complejas 15

2.2. Búsqueda de comunidades 16

2.2.1. OSLOM 19

2.2.2. Infomap 20

2.3. Representaciones vectoriales 21

2.3.1. Bag of Words 21

2.3.2. Tf-Idf 22

2.3.3. Word2vec 24

2.3.4. DeepWalk 26

2.3.5. Node2vec 26

2.4. Métodos de selección significativa 29

2.4.1. Distribución de Pearson (χ^2) 29

2.4.2. Información mutua 31

2.5. Métricas y métodos de evaluación 31

2.5.1. Métricas básicas . . . 32
2.5.2. Validación cruzada . . . 36
2.5.3. Validación cruzada anidada . . . 37
2.6. Indexación espacial . . . 39
2.6.1. Árboles KD (KD-Trees) . . . 39
2.7. Clasificadores . . . 41
2.7.1. Naive Bayes . . . 43
2.7.2. Árboles de decisión . . . 45
2.7.3. Regresión logística . . . 45
2.8. Ensamblado de clasificadores . . . 46
2.9. Redes neuronales . . . 47
2.9.1. LSTM - Long short-term memory . . . 50
2.9.2. Transformadores y mecanismos de atención . . . 51
2.9.3. GCN - Graph Convolutional Network . . . 53
2.9.4. GraphSAGE . . . 55
2.9.5. R-GCN . . . 56

3. Análisis exploratorio **57**
3.1. Datos disponibles . . . 57
3.2. Análisis previo . . . 58
3.3. Análisis de perfiles . . . 61
3.4. Análisis de redes . . . 67
3.4.1. Red de menciones . . . 68
3.4.2. Red de seguidores . . . 73
3.5. Análisis de contenido . . . 74
3.5.1. Búsqueda de términos locales . . . 76
3.5.2. Análisis de secuencias . . . 77

4. Desarrollo **81**

4.1. Objetivo a clasificar 81

4.2. Modelo de evaluación 82

4.3. Implementación 84

4.4. Métricas y resultados 89

4.5. Explicabilidad sobre clasificación 89

4.5.1. Explicabilidad sobre redes 91

4.5.2. Explicabilidad sobre contenido 92

4.6. Conjuntos de datos de prueba 96

5. Conclusiones **101**

5.1. Trabajo a futuro 102

6. Bibliografía **105**

Capítulo 1

Introducción

En este capítulo discutiremos la relevancia del problema de predicción de ubicación en redes sociales, y haremos un breve resumen de la literatura existente hasta la fecha.

1.1. Motivación

En los últimos años se ha visto un gran crecimiento de las redes sociales, cada vez más usuarios adoptan una o más de una para comunicarse entre sí o enterarse de las últimas noticias que acontecen a sus familiares, amigos, su país o al mundo en sí. Puede verse a las redes sociales como un mundo de información que se puede aprovechar de diversas formas. Una de ellas sería determinar la ubicación de sus usuarios, información que sería útil para recomendarle contenido personalizado a cada usuario como resúmenes de las noticias más importantes de usuarios cercanos, anuncios regionales o información comercial local al usuario. También puede aprovecharse esta información para el monitoreo de la salud pública (p. ej., monitoreo de síntomas y dolencias generales Paul and Dredze (2011); pronóstico de la incidencia del virus del Zika McGough et al. (2017)) o hasta incluso podría apro-

vecharse para informar sobre catástrofes locales (p. ej., terremotos Sakaki et al. (2010), incendios forestales De Longueville et al. (2009) o inundaciones Jongman et al. (2015)) teniendo en cuenta que estos medios de comunicación pueden llegar a ser más rápidos que los medios tradicionales. Otro uso interesante sería el análisis de comunidades y su comportamiento, algo que sería bastante útil para análisis de la opinión pública.

1.2. Estado del arte

En la presente sección se realizará un breve resumen del estado actual del arte y las diversas metodologías utilizadas para determinar la ubicación de los usuarios. Se hablará también sobre los problemas que conlleva trabajar con datos de redes sociales cuya naturaleza es ruidosa y difícil de procesar, y del modelado de la información disponible. Previamente es necesario explicar la terminología y la información disponible sobre la que se referirá en esta sección.
Un *tweet* es una publicación realizada por un usuario de hasta 140 caracteres dedicada a un conjunto de *seguidores*, que puede contener *hashtags*, imagénes o hasta videos y menciones hacia otros usuarios. En la Figura 1.1 podemos ver un ejemplo de un tweet. Los tweets también poseen *metadata* específica como por ejemplo hora de publicación, ubicación del usuario al realizar el tweet, cantidad de likes, cantidad de *retweets*, etc.

Con respecto a la ubicación del usuario en sus tweets, los usuarios pueden optar por habilitar la geolocalización de sus tweets, sin embargo esta alternativa es muy poco usada hoy en día; Cheng et al. (2010) y Hecht et al. (2011) reportaron que en muestras de más de 1 millón de tweets, menos del 1 % se encuentran geolocalizados, con lo que obtener la ubicación por medio de esta funcionalidad es poco viable y hay que analizar otras alternativas. La otra opción posible para obtener la

Figura 1.1: Ejemplo de un tweet con mención a otros usuarios.

ubicación de los usuarios es verla a través de sus perfiles públicos, pero no existe garantía de que esos datos sean fiables y tan granulares como uno desea. Cheng et al. (2010) descubrió que de una muestra de 1 millón de usuarios, solamente el 26 % de ellos incluían una ciudad como ubicación en su perfil, asimismo resultados similares fueron reportados por Hecht et al. (2011): solamente el 34 % de los usuarios incluían una ubicación real en sus perfiles. Todo esto fue una gran motivación para la creación de diversos métodos y modelos para determinar la ubicación de los usuarios, de sus tweets o incluso reconocer entidades. Por eso, al día de hoy existen tres enfoques distintos para determinar la ubicación de los usuarios de Twitter: (a) determinar la residencia permanente de los usuarios, (b) determinar la ubicación de cada tweet individualmente o (c) detectar e identificar entidades geográficas (p. ej., el Obelisco de Buenos Aires). Para el presente trabajo nos interesa hacer hincapié en los métodos para determinar la ubicación permanente de los usuarios, sin embargo, se discutirán técnicas utilizadas en los trabajos que utilizan otro nivel de predicción para lograr mejores resultados.

Es importante aclarar las métricas y conjuntos de datos utilizados a lo largo de todos estos trabajos. En líneas generales, existen tres conjuntos de datos comunmente utilizados en la literatura: W-NUT, Twitter-World y Twitter-US. En este trabajo presentaremos resultados sobre este último que es un conjunto de usuarios

geolocalizados dentro de los Estados Unidos, y sobre un conjunto de datos que nosotros recopilamos focalizado en Latinoamerica. Con respecto a las métricas reportadas, la exactitud (accuracy) es utilizada mayormente a nivel país pero a nivel ciudad, dada la naturaleza ruidosa y la disparidad en los tamaños de muestra, se opta por utilizar otras métricas más relajadas. Cheng et al. (2010) propuso utilizar la exactitud a 161 kilómetros (*accuracy@161*) que considera que la clasificación es correcta si la ubicación permanente de un usuario se encuentra en un radio de 161 kilómetros de la ubicación en la que se lo clasificó. Con respecto a los métodos, Cheng et al. (2010) propuso utilizar solamente el contenido de los tweets logrando una *accuracy@161* del 51 %; identificando términos que sean locales a cada ciudad en específico, se pueden eliminar palabras ruidosas y conservar aquellas que realmente aportan información con respecto a la ubicación del usuario. Mahmud et al. (2014) y Han et al. (2014) también buscaron identificar términos locales utilizando diferentes modelos: Han et al. (2014) se basó en modelos estadísticos y propuso nuevos métodos basados en la teoría de la información, regresores logísticos basados en pesos y, en particular, una heurística llamada TF-ICF, una variante de TF-IDF en donde el valor de ICF (Alta frecuencia inversa de la ciudad) es mayor si un término es utilizado en pocas ciudades. Más adelante Chi et al. (2016) aprovecharía estos métodos de ganancia de información propuestos por Han et al. (2014) en conjunto con la incorporación del uso de hashtags y menciones para el análisis de contenido, logrando una exactitud del 22 %, aunque no reportan la *accuracy@161*. Mahmud et al. (2014) identificó términos locales basándose en distribuciones estadísticas con heurísticas, también incorporó el uso de hashtags para su análisis, y su enfoque consistió en el ensamblado de múltiples clasificadores a distintos niveles con respecto al contenido (p ej., todos los términos, o solamente aquellos locales) y ubicación (p ej., ciudad, país, estado, zona horaria). Mahmud et al. (2014) también logró resolver la problemática del usuario viajante a través

del filtrado, ya que intentar clasificar usuarios cuya ubicación permanente no se ve reflejada en sus tweets carece de sentido.

Una problemática que surge al clasificar a los usuarios en una ciudad es que dado que hay ciudades con menos cantidad de usuarios y tweets, esto tiende a generar un desbalance en los tamaños de muestra por ciudad. Roller et al. (2012) propuso dividir el espacio en grillas utilizando k-d trees de forma tal que cada grilla contenga una cantidad equitativa de muestras. De esta forma, utilizando los tweets geolocalizados se evita recurrir a métodos que den un sobrepeso a las ciudades con menos muestras. Trabajos como Han et al. (2012) y Han et al. (2014) utilizaron este modelo de división del espacio en conjunto con otro basado directamente en la ciudad más cercana a cada tweet. Ellos plantearon en su trabajo un modelo de apilado de clasificadores (*Stacking Classifier*) en donde clasificaciones individuales son utilizadas para entrenar un meta-clasificador. Basandose solamente en el contenido de los tweets, reportaron que a pesar de que los tweets sean localmente temporales al momento de publicarse, modelos que se basan en el contenido y la zona horaria son insensibles a estos cambios. Lograron una accuracy@161 del 45 % para el conjunto de Twitter-US y en general no tuvieron peores resultados utilizando la división del espacio con k-d trees. Otros trabajos como el de Rahimi et al. (2015) y Rahimi et al. (2017) también utilizan el modelo de división del espacio propuesto por Roller et al. (2012) incorporando, además del contenido de los tweets, la red de menciones que se forma entre los usuarios. Tomando dichas menciones, eliminaron todos aquellos nodos populares y los que no formaban parte del conjunto de entrenamiento, luego generaron aristas entre dichos nodos en caso de que hubiese un camino entre ellos pasando por los nodos eliminados, obteniendo finalmente un grafo denso no dirigido. Por medio de este grafo y el contenido generado por los usuarios, utilizaron un modelo de entrenamiento semi-supervisado para grafos basado en el modelo MAD (*Modified ADsorption*) propuesto por Ta-

lukdar and Crammer (2009) logrando una *accuracy@161* del 60 %. Hasta la fecha, los trabajos de Miura et al. (2017) y Huang and Carley (2019) obtuvieron los mejores resultados sobre Twitter-US rondando una accuracy@161 del 70 %. Estos trabajos se basaron en el uso de redes neuronales que combinan aspectos mediante embeddings del contenido de los tweets, la red de menciones que conforman los usuarios y la información de perfil disponible, sin embargo, ninguno de estos trabajos provee explicabilidad sobre las predicciones realizadas.

En el presente trabajo, se propone nuevos métodos de clasificación de usuarios a nivel ciudad basados en grafos que utilicen no solo el contenido y la red de menciones sino también la red de seguidores. Al ser esta última difícil de capturar, no es utilizada en ninguno de los trabajos mencionados. Por otra parte se analizara también que tan posible es proveer explicabilidad sobre como se clasifican a los usuarios. También se incluirá un método utilizando redes neuronales para comparar los resultados con los del estado del arte.

Capítulo 2

Metodología

En este capítulo se expondrán algunos conceptos teóricos necesarios para el desarrollo de este trabajo. Estos incluyen una introducción a la teoría de grafos, al concepto de búsqueda de comunidades, métodos de aprendizaje automático.

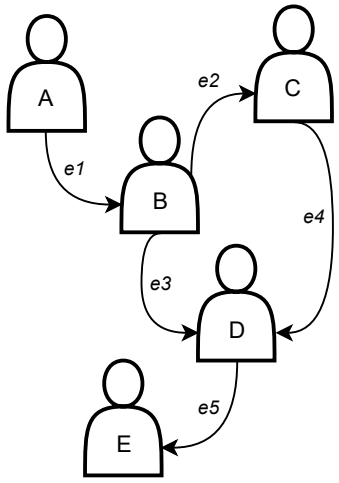
2.1. Grafos en redes sociales

Muchas situaciones del mundo real pueden ser descriptas por medio de un diagrama formado por puntos, en conjunto con líneas que conectan pares de puntos. Por ejemplo, los puntos podrían representar personas, y las líneas podrían indicar si son amigas entre sí. Este es el caso de nuestros usuarios de Twitter que mantienen relaciones seguidor/seguido y de menciones entre ellos. Una abstracción matemática de este tipo de situaciones da lugar al concepto de *grafo*. Utilizaremos los libros de Bondy et al. (1976), Newman (2018) y Coscia (2021) para las definiciones teóricas de los conceptos que trataremos.

Un grafo G es una terna $(V(G), E(G), \psi_G)$ consistiendo de un conjunto $V(G)$ no vacío de vértices, un conjunto $E(G)$ disjunto de $V(G)$ $(E(G) \cap V(G) = \emptyset)$ de aristas, y una función de incidencia ψ_G que asocia a cada arista de G con

un par no ordenado de vértices de G, no necesariamente distintos. Si e es una arista y u, v son vértices tal que $\psi_{\mathrm{G}}(e) = uv$, entonces se dice que e une u y v, y que tanto u como v son incidentes sobre e y sus correspondientes terminales. Un ejemplo de esto puede verse en el grafo G de seguidores de la Figura 2.1, podemos ver que $V(G) = \{A, B, C, D, E\}$ y $E(G) = \{e1, e2, e3, e4, e5\}$, de donde ψ_{G} es definida por $\psi_{\mathrm{e1}} = AB, \psi_{\mathrm{e2}} = BC, \psi_{\mathrm{e3}} = BD, \psi_{\mathrm{e4}} = CD, \psi_{\mathrm{e5}} = DE$. Se dice que dos vértices v_{i} y v_{j} son adyacentes si ambos tienen incidencia sobre una arista en común y, bajo este concepto, vamos a representar a los grafos por medio de sus matrices de adyacencia. La matriz de adyacencia de un grafo G es una matriz de tamaño $v \times v$ (Siendo v el tamaño del conjunto de vértices $V(G)$) definida como $A(G) = [a_{\mathrm{ij}}]$ en donde el elemento a_{ij} es la cantidad de aristas que unen a los vértices v_{i} y v_{j}. Como trabajaremos con una única arista como máximo entre dos vértices adyacentes en nuestros grafos, el elemento a_{ij} representará el peso asignado a la arista que une a los vértices v_{i} y v_{j}; notemos que dicho peso nos permite representar la cantidad de aristas que unen a los vértices v_{i} y v_{j} teniendo en cuenta que podríamos incurrir en pérdida de información en caso de que dichas aristas contemplen un significado distinto, sin embargo, este no es el caso para los métodos que utilizaremos. Representar a los grafos por medio de su matriz de adyacencia nos va a permitir realizar permutaciones y operaciones sobre estos con gran facilidad; una de estas operaciones es la normalización de cada fila de $A(G)$, lo cual da lugar a una matriz estocástica de adyacencia $M(G) = [m_{\mathrm{ij}}]$ en donde el elemento m_{ij} representa la probabilidad de viajar desde el vértice v_{i} a v_{j} para un caminante aleatorio. Esta matriz estocástica de adyacencia nos será útil para métodos que dependan del uso de caminos aleatorios (*random walks*), ya sea métodos para obtener representaciones vectoriales de los vértices, para difundir información o para la detección de comunidades en estos grafos.

Siguiendo con la forma de representar a las relaciones de nuestros usuarios co-



-	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	1	0
C	0	0	0	1	0
D	0	0	0	0	1
E	0	0	0	0	0

Figura 2.1: Grafo G dirigido de seguidores de usuarios de Twitter y su matriz de adyacencia.

mo grafos, debemos definir distintos tipos y características de grafos que usaremos más adelante:

Lazos: Un grafo G presenta un *lazo* si, para alguna arista e y nodo v, $\psi_{\mathrm{e}} = vv$, es decir, la arista conecta al mismo vértice.

Grafo simple: Un grafo G se dice que es simple si no contiene *lazos* y no existe más de una arista que conecte dos mismos vértices.

Camino: Un camino (*walk*) en un grafo G es una secuencia finita no vacía $W = v_0e_1v_1e_2v_2...e_{\mathrm{k}}v_{\mathrm{k}}$ cuyos términos son vértices y aristas alternadas tal que, para $1 \leq i \leq k$, los terminales de e_{i} son $v_{\mathrm{i\text{-}1}}$ y v_{i}. Para grafos simples, la secuencia puede ser determinada únicamente por los vértices $v_0v_1...v_{\mathrm{k}}$.

Grafo dirigido: Un grafo dirigido D es una terna $(V(D), A(D), \psi_{\mathrm{D}})$ consistiendo en un conjunto $V(D)$ no vacío de vértices, un conjunto $A(D)$ disjunto de $V(D)$ $(A(D) \cap V(D) = \emptyset)$ de arcos, y una función de incidencia ψ_{D} que asocia a cada arco de D con un par ordenado de vértices de D, no necesariamente distintos. Si a es un arco y u, v son vértices tal que $\psi_{\mathrm{D}}(a) = (u, v)$, entonces se dice que e une u y v; u corresponde a la *cola* (*tail*) de a y v es su *cabeza* (*head*).

Subgrafo: Un grafo H es un subgrafo de G ($H \subseteq G$) si $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ y ψ_H es la restricción de ψ_G sobre $E(H)$.

Multigrafo: Un grafo G se dice que es un multigrafo si se permiten múltiples aristas entre dos nodos de G. Las aristas y nodos pueden contener etiquetas asociadas, sean cualitativas o cuantitativas.

Grafo multicapa: Un grafo multicapa G con conexiones uno a uno es una cuaterna $(V(G), E(G), \psi_G, L(G))$ donde $V(G)$ es el conjunto de vértices, $E(G)$ el conjunto de aristas, $L(G)$ un conjunto de capas y ψ_G una función de incidencia que asocia cada arista con una terna constituida por dos vértices y una capa $l \in L(G)$. Cada capa representa un grafo simple y las conexiones entre distintas capas son aristas que unen distintas representaciones del mismo nodo en distintas capas.

Grado de un vértice: El grado $d_{\mathrm{G}}(v)$ de un vértice en G es la cantidad de aristas de G incidentes con v. Denotamos $\delta(G)$ y $\Delta(G)$ a los grados mínimo y máximo, respectivamente, de los vértices de G. Para grafos dirigidos se pueden definir dos grados, el grado entrante (*indegree*) $d^{-}{}_{\mathrm{D}}(v)$ de un vértice en D es la cantidad de arcos con v como su cabeza; el grado saliente (*outdegree*) $d^{*}{}_{\mathrm{D}}(v)$ de un vértice en D es la cantidad de arcos con v como su cola. Denotamos como máximo y mínimo grado entrante y saliente a δ^{-}, Δ^{-}, δ^{*}, Δ^{*}, respectivamente.

Camino aleatorio: Un camino aleatorio (*random walk*) es un camino a través de un grafo creado por medio de la toma de pasos aleatorios uniformemente. Se comienza en un vértice inicial específico v_1 y en cada paso del camino se elige uniformemente al azar entre las aristas incidentes a v (arcos con v_1 como su cola para grafos dirigidos) para hacia el otro vértice terminal de la arista. En los caminos aleatorios pueden repetirse las aristas por las que se viaja.

Coeficiente de agrupamiento (clustering coefficient): El coeficiente de agrupamiento mide la probabilidad promedio de que dos nodos de una red sean vecinos, es decir, que exista una arista que conecte ambos nodos. Para calcularlo, debemos

explicar el concepto de triada: Dado un nodo u vecino de v, y a su vez v vecino de otro nodo w, definimos al conjunto uvw como una triada. Entonces, el coeficiente de agrupamiento de un nodo i se calcula como la fracción de pares de nodos conectados de cada triada a la que pertenezca el nodo i con respecto a todos los pares de nodos de cada triada a la que pertenezca el nodo i, matemáticamente hablando:

$$C_{\mathrm{i}} = \frac{número\,de\,pares\,de\,vecinos\,de\,i\,que\,están\,conectados}{número\,de\,pares\,de\,vecinos\,de\,i}$$

Existe también un coeficiente de agrupamiento global, el cual es simplemente el promedio de todos los coeficientes de agrupamiento locales:

$$C = \frac{1}{n}\sum_{i=1}^{n} C_{\mathrm{i}}$$

2.1.1. Redes complejas

Una red compleja es un concepto utilizado para modelar sistemas complejos, sistemas que no pueden ser comprendidos si todo lo que poseemos es una perfecta descripción de sus partes. Dicho de otra forma, las relaciones que conforman las partes de un sistema complejo dejan emerger un conjunto de propiedades globales que no son simplemente la suma de las propiedades locales. Nos interesan particularmente las redes complejas libres de escala (scale-free networks). Dichas redes se suelen caracterizar porque la distribución de grado de sus vértices sigue o aproxima una *ley de potencias*: es decir, existen muchos nodos con bajo grado y muy pocos nodos con grado alto los cuales conectan gran parte de la red y son comúnmente llamados *hubs*. Matemáticamente hablando, para una red libre de escala su distribución de grados p_{k} sigue la ecuación:

$$p_{\mathrm{k}} = Ck^{-\alpha}$$

En dicha ecuación, C es una constante que en líneas generales no es interesante, simplemente existe por normalización para que $\sum_{k=1}^{\infty} p_{\mathrm{k}} = 1$, en cambio α

representa el *exponente* de la ley de potencias y su valor suele rondar en el rango $2 \leq \alpha \leq 3$. Un ejemplo de estas redes libres de escala es el grafo conformado por las menciones entre los usuarios con los que trabajaremos en la sección Análisis de redes. Podemos ver en la Figura 2.2 que dicho grafo sigue una ley de potencias. Existen propiedades muy interesantes en este tipo de redes: La conexión preferencial (*preferential attachment*) es una de ellas y nos dice que es más probable que futuras nuevas aristas sean hacia nodos con grado alto que hacia nodos de grado bajo. Este fenómeno es conocido como "*Los ricos se hacen aún más ricos*", algo que en redes sociales es muy común de ver porque los usuarios siguen a celebridades o a sus ídolos. Otra propiedad muy interesante, conocida como Mundo pequeño (*small-world*) o "*seis grados de separación*" explica que la distancia entre dos nodos cualesquiera de la red suele ser bastante pequeña a causa de los *hubs* que conectan gran parte de las componentes de la red. Otra de las propiedades subyacentes de este tipo de redes es que el coeficiente de agrupamiento de los nodos también sigue una ley de potencias.

2.2. Búsqueda de comunidades

El objetivo de la detección de comunidades es separar la red en grupos de vértices (Comunidades), tales que la densidad de conexión dentro de los grupos es mayor que la densidad de conexiones entre nodos en distintos grupos. La cantidad y tamaño de las comunidades a obtener no son un valor fijo especificado sino que dependen de la la red en sí. Cuando se habla de cantidad y tamaños de comunidades fijos se hace referencia al problema de particionamiento de grafos, tema sobre el cuál no trataremos por diferir del objetivo al que apuntamos: discernir comunidades de tamaño no fijo de usuarios que pertenezcan a la misma ciudad. Debemos notar que las comunidades obtenidas por separar la red podrían estar solapadas, es decir,

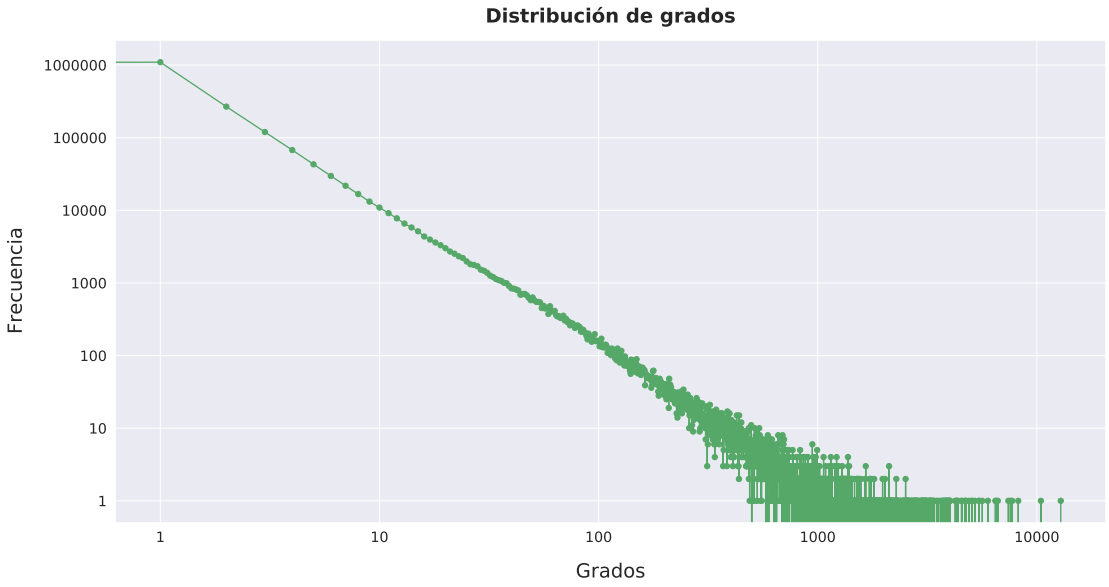


Figura 2.2: Ejemplo de ley de potencias en la distribución de grados entrantes para el grafo de menciones completo en escala logarítmica.

en muchos casos los usuarios pueden pertenecer a más de una comunidad tal como se ilustra en la Figura 2.3.

Confiamos en obtener buenos resultados con métodos de búsqueda de comunidades debido a que las personas tienen más probabilidades de tener contacto con aquellos que están más cerca de ellos en una ubicación geográfica que con aquellos que se encuentran lejos. Sin embargo, este no es el único tipo de vínculo que une a las personas. Si nos basamos en el concepto de *homofilia* definido como la fuerte tendencia de las personas a asociarse con otros a los cuáles perciben como similares, las personas pueden relacionarse con otras para, por ejemplo, compartir ideas políticas, algo que en la gran mayoría de los casos tiene una fuerte incidencia local. Sabemos que a pesar de que exista una mayor facilidad de conectarse con gente lejana en las redes sociales, McPherson et al. (2001) aclara que se preservan, aunque más débiles, los patrones de relación con gente cercana geográficamente. Existen varios enfoques para trabajar con detección de comunidades, entre ellos

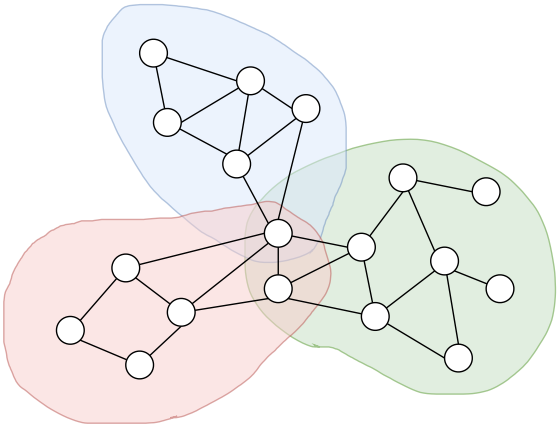


Figura 2.3: Detección de 3 comunidades con solapamiento en un grafo no dirigido.

los más comunes son los basados en maximizar la *modularidad* de las comunidades Newman and Girvan (2004). Definimos la *modularidad* Q como la medida de *homofilia* que representa que tan conectados están los vértices del mismo grupo, y se calcula a partir de la diferencia entre la fracción de aristas entre vértices del mismo grupo y la fracción de dichas aristas esperable si todas las aristas se dispusiesen aleatoriamente entre vértices de cualquier grupo. Su valor es estrictamente menor que 1, tomando valor positivo si hay más aristas entre vértices del mismo tipo en comparación con lo que esperaríamos de aristas al azar y negativo en caso contrario. Específicamente para la detección de comunidades con este enfoque, se considera cada comunidad como un grupo de vértices buscando maximizar la *modularidad*. Sin embargo, debemos considerar que este problema tiene tiempo de ejecución exponencial en el tamaño del grafo, con lo que en la mayoría de casos se hará uso de heurísticas que busquen *modularidad* alta pero no máxima. Otros enfoques posibles para la detección de comunidades son los que consisten en buscar y remover aristas que se encuentren entre comunidades; se suele utilizar la *centralidad de intermediación* (*betweenness centrality*) para la detección de dichas aristas Girvan and Newman (2002). Definimos la *centralidad de intermediación* como la

medida en la que un vértice se encuentra en las trayectorias entre otros vértices, se considera, aunque no necesariamente en todos los casos, que vértices con alta centralidad de intermediación tienen gran influencia en la red debido al control intermedio de mensajes que incurre en ellos. Por último, vamos a mencionar otro enfoque bastante común para la detección de comunidades, basado en descomposiciones jerárquicas de la red en conjuntos de comunidades anidadas. En general, en este caso, se suele utilizar algoritmos de *clustering jerárquico*, en los cuales se comienza por todos los vértices aislados y los vamos agrupando según una medida de similitud a elección (p. ej., distancia coseno o distancia euclídea) Girvan and Newman (2002).

2.2.1. OSLOM

Hoy en día existen muchos métodos de detección de comunidades en grafos, sin embargo, pocos tienen la posibilidad de trabajar con ciertas características de los grafos, entre ellas grafos dirigidos, la detección más de un *cluster* a la que pertenece un vértice de la red o hasta el descubrimiento de varios niveles jerárquicos, algo muy común y presente en las redes. Lancichinetti et al. (2011) propusieron *Order Statistics Local Optimization Method*, abreviado OSLOM, un método de detección de comunidades en grafos que abarca toda esta gama de características y se basa en optimizar localmente la significancia estadística de *clusters* definida con respecto a un modelo nulo global. OSLOM utiliza la significancia como una medida de aptitud para evaluar los *clusters* y se define a la misma como la probabilidad de encontrar dicho *cluster* en un modelo nulo aleatorio. En su trabajo utilizan como modelo nulo al *configuration model* Molloy et al. (2011), diseñado para construir redes aleatorias dada una distribución de grados de los vértices. El modelo parte de un grafo G con N vértices y E aristas, y comienza el proceso de generación de un *cluster*:

1. Se selecciona un nodo i aleatoriamente para agregarlo a un subgrafo $C = \{i\}$.

2. Se agregan los q vértices más significativos del conjunto de nodos vecinos al subgrafo C. El valor q es tomado de una distribución que sigue una ley de potencias.

3. Se realiza un procedimiento de limpieza en dos pasos en donde primero se agregan nodos vecinos externos a C significantes según una tolerancia, luego, se eliminan de C aquellos nodos cuya significancia sea menor a un cierto umbral. Un cluster C se deja de modificar cuando todos los vértices externos son compatibles con el modelo nulo y los internos no lo son. Este proceso se repite N veces dado que los resultados no son determinísticos.

4. El cluster C' resultante es considerado significante si en más del 50 % de las iteraciones su resultado es no vacío.

5. Se vuelve al paso (1) hasta que se sigan encontrando varias veces clusters *similares*.

OSLOM posee algunos parámetros para variar como el nivel de significancia P que juega un papel importante a la hora de determinar el tamaño de los *clusters*, sin embargo, cuando los módulos están bien definidos, los resultados de OSLOM no dependen de una particular elección de parámetros.

2.2.2. Infomap

En 2009 Rosvall et al. (2009) propusieron un método de detección de comunidades basado en el flujo de información usando los principios de dualidad del problema de comprimir información y la detección/extracción de patrones de dicha información. El método consiste en la búsqueda de la mínima descripción

posible (*minimum description length*) de caminos aleatorios que representan flujos en la red por medio de códigos de Huffman. Dicho de otra forma, se busca minimizar la longitud de los códigos de Huffman necesarios para codificar la red completa según las particiones a nivel intra-modular (códigos dentro de una comunidad) y extra-modular (códigos que distinguen comunidades). De esta forma es posible reutilizar códigos y es posible discernir en qué comunidad se encuentra un caminante aleatorio según el código asignado a la comunidad. Este método y el objetivo a minimizar se denomina *map equation* y no necesita del calculo óptimo de los códigos de la red particionada sino que, al basarse en la entropía de Shannon para determinar que tan eficientes son dichos códigos, es posible comparar distintas particiones de la red y determinar la óptima.

Esta técnica fue implementada en Infomap, un método de detección de comunidades desarrollado en C++ por los mismos autores.

2.3. Representaciones vectoriales

En esta sección explicaremos algunos métodos que utilizaremos para obtener representaciones vectoriales de los datos de nuestros usuarios, de forma tal que podamos aplicarlos en métodos que requieren datos de entrada de tipo numérico. Hablaremos sobre métodos para representar tanto textos como nodos de nuestras redes, y de sus correspondientes ventajas y desventajas.

2.3.1. Bag of Words

La forma más básica que se nos ocurre para poder representar documentos es construir vectores del tamaño del vocabulario completo de todos los documentos en donde cada posición del vector representa un término y su valor representa la cantidad de veces que dicho término se encuentra presente en el documento. Este

método de representación de documentos como vectores es comúnmente llamado bolsa de palabras (*bag of words*) y podemos ver un breve resumen de cómo aplicarlo en la Figura 2.4. Debemos notar que este método tiene la ventaja de que es bastante sencillo de construir y los vectores son todos de la misma longitud, pero lleva consigo varias consecuencias. La principal es que la longitud de los vectores puede tornarse bastante grande si consideramos muchos documentos, dado que tendríamos un vocabulario muy extenso, y esto a su vez da lugar a vectores en donde la mayoría de los elementos son cero. Sin embargo, podríamos acotar el tamaño del vocabulario considerando términos que sean utilizados con al menos una mínima frecuencia. Otra de sus principales desventajas es que perdemos la noción de sucesión de términos (también llamado contexto) en los documentos, lo cuál podría ser muy importante si deseamos trabajar con modelos que requieran conocer el orden de la secuencia para, por ejemplo, determinar términos que prosiguen a otros. Podemos aplicar este modelo a las menciones que se producen en los tweets de nuestros usuarios: si consideramos cada usuario mencionado como un término de nuestro vocabulario, podríamos construir vectores en donde cada elemento nos indique la cantidad de menciones realizadas hacia un usuario. De hecho, esto es una forma de construir la matriz de adyacencia de la red de menciones.

2.3.2. Tf-Idf

Una de las consecuencias del modelo de bag of words es que considera que todos los términos son igual de importantes, cuando en muchos casos deseamos que ciertos términos utilizados frecuentemente y que aportan poca información, conocidos como *stop-words*, sean filtrados o considerados como poco relevantes. Para solventar este problema, podemos aplicar el modelo Tf-Idf (Term frequency - Inverse document frequency) el cual consta de dos componentes: la primera componente *Term frequency* (frecuencia de término) es simplemente la cantidad de

1

Documentos	
Documento #1	gran noche en buenos aires
Documento #2	noche en la ciudad de buenos aires

2

Construcción del Vocabulario
Construimos el vocabulario como la unión de los términos de los documentos: Doc #1 U Doc #2
Vocabulario: [gran, noche, en, buenos, aires, la, ciudad, de]

3

Representación vectorial								
	gran	noche	en	buenos	aires	la	ciudad	de
Documento #1	1	1	1	1	1	0	0	0
Documento #2	0	1	1	1	1	1	1	1

Figura 2.4: Ejemplo de construcción de bag of words. En el paso 1 vemos el contenido de dos tweets de ejemplo. En el paso 2 extraemos el vocabulario. En el paso 3 calculamos la frecuencia de aparición de cada palabra en cada documento, y construimos una representación vectorial para cada documento.

veces que un término es utilizado en un documento (similar a Bag of words), y su segunda componente *Inverse document frequency* (frecuencia inversa de documento) nos permite asignar un peso mayor a aquellos términos que no son utilizados con frecuencia, y un peso menor en caso contrario. La *Inverse document frequency* se suele calcular para un término t como:

$$idf_{\mathrm{t}} = log \frac{N}{df_{\mathrm{t}}} \quad ,$$

donde N es la cantidad total de documentos y df_{t} la cantidad de documentos en las que el término t es utilizado. Combinando ambas definiciones de frecuencia de término y frecuencia inversa de documento, podemos calcular el valor tf-idf de un término t para un documento d como:

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t \quad .$$

Este valor será alto cuando el término t sea utilizado varias veces en un conjunto pequeño de documentos y será bajo cuando el término ocurra pocas veces en un documento u ocurra en varios documentos. Existen muchas variantes sobre ambas componentes de tf-idf, en esta sección dimos una vista bastante genérica del método, pero sugerimos al lector el libro de Schütze et al. (2008) para más referencias sobre distintas formas de calcular las componentes del modelo tf-idf.

2.3.3. Word2vec

Los modelos presentados con anterioridad no suelen captar la noción de similitud entre términos sino que tratan cada término como una unidad atómica. Esto se hace por simplicidad, y debido a la cantidad cantidad masiva de información disponible que permite, en líneas generales, obtener buenos resultados. Mikolov et al. (2013) presentaron técnicas basadas en redes neuronales, hoy en día utilizadas por modelos conocidos como *word2vec*, para representar términos en documentos con

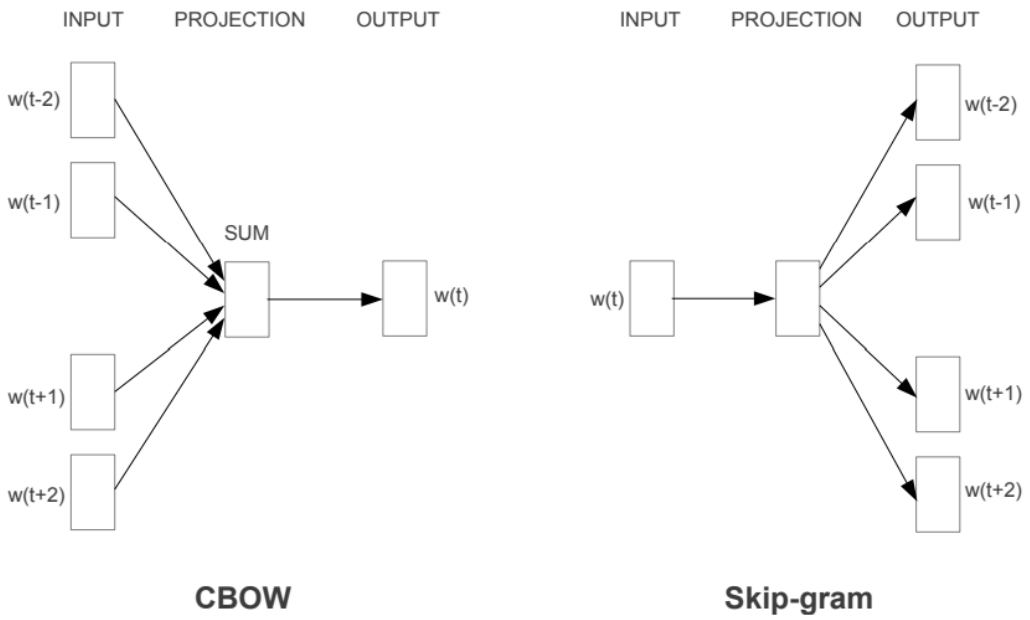


Figura 2.5: Arquitecturas CBOW y Skip-GRAM. Se puede apreciar en CBOW como entrada dos términos pasados y dos términos futuros al término a predecir. Para Skip-GRAM en cambio se utiliza el término para predecir los vecinos cercanos en una ventana de tamaño 5. Imagen extraída de Mikolov et al. (2013)

hasta billones de términos como vectores. Para la construcción de estos términos se basaron en la similitud entre ellos proponiendo una metodología eficiente en cuanto a tiempo de ejecución. En dicho trabajo se propusieron dos arquitecturas distintas conocidas como CBOW (continous bag of words) y Skip-GRAM; ambas constan de una ventana deslizante que genera los conjuntos de entrenamiento. En la arquitectura CBOW se intenta predecir un término según el contexto del mismo considerando términos pasados y futuros según el tamaño de la ventana deslizante. En cambio para Skip-GRAM se busca entrenar un modelo para predecir términos vecinos al término en cuestión. En la Figura 2.5 se muestra un diagrama básico de ambas arquitecturas.

En muchos casos no se busca predecir términos según el contexto sino obtener representaciones vectoriales en bajas dimensiones de cada término (conocidas como *embeddings*). Para eso, se suelen tomar los pesos de las capas ocultas del modelo y utilizarlos como los propios vectores de término. Matemáticamente hablando, se toman las filas como representaciones vectoriales de los términos de una matriz de pesos de $N \times D$, en donde N es la cantidad total de términos y D el espacio de dimensiones del modelo.

2.3.4. DeepWalk

Hasta el momento estuvimos tratando con métodos que, en general, son utilizados para representar documentos como vectores del espacio. Si bien estos métodos son aplicables a redes considerando a los vértices como términos, existen otros métodos enfocados directamente en este tipo de datos. Perozzi et al. (2014) propusieron *DeepWalk*, un método no supervisado para obtener representaciones vectoriales de vértices en grafos, tales donde vértices similares se encuentren próximos en el espacio. El método combina el uso de caminos aleatorios de longitud fija con técnicas comúnmente utilizadas en modelos de procesamiento de lenguaje. En particular consideran la arquitectura de Skip-GRAM, en donde utilizan como vocabulario los vértices de la red y como secuencias o documentos los caminos aleatorios de longitud fija de cada vértice. En la Figura 2.6 se ilustra el objetivo general de este tipo de métodos.

2.3.5. Node2vec

Siguiendo con los métodos para aprender representaciones vectoriales de nodos en grafos, Grover and Leskovec (2016) propusieron *Node2vec* con el objetivo de resolver algunos inconvenientes que poseían los métodos propuestos con anterioridad. Por ejemplo, se modifica la estrategia para la obtención de secuencias por

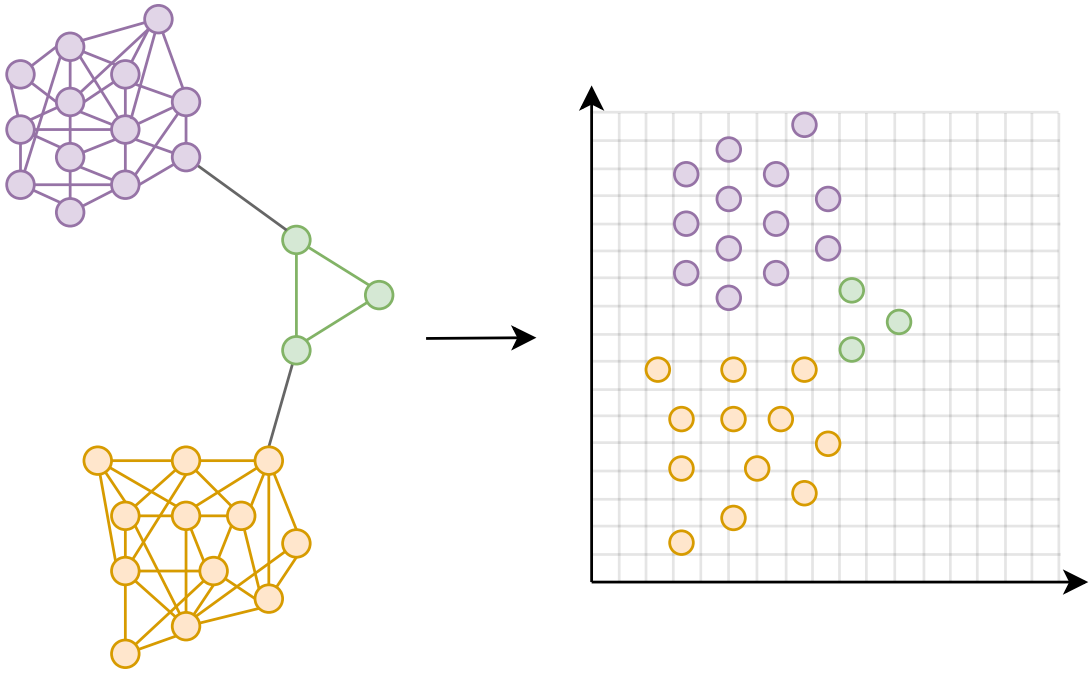


Figura 2.6: Objetivo de métodos de representación vectorial de nodos de un grafo. A modo de visualización se ejemplifica con una proyección a 2 dimensiones.

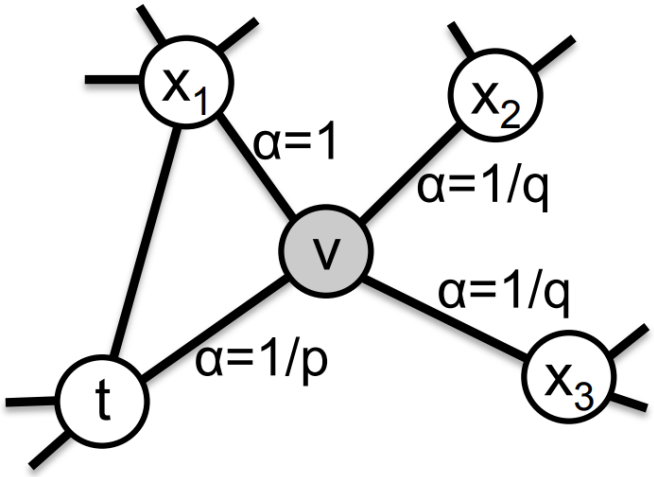


Figura 2.7: Ejemplo de cada paso de un camino aleatorio en *Node2vec*. El caminante recién transiciona del nodo t al nodo v y se encuentra evaluando su próximo paso. α regula las probabilidades no normalizadas de transición según los parámetros p y q. Imagen extraída de Grover and Leskovec (2016).

medio de caminos aleatorios uniformes utilizada en el modelo de *DeepWalk*. En *Node2vec* se propone una estrategia regulable, por medio de parámetros, para el muestreo de nodos vecinos utilizando caminos aleatorios sesgados que combinan estrategias de BFS (Breadth-First Sampling) y DFS (Depth-First Sampling) para determinar qué tanto se explora la vecindad de cada nodo en cada paso. Al combinar ambas estrategias es posible optar por priorizar la representación de nodos cercanos según su equivalencia estructural si se enfocase en una estrategia más cercana a BFS (para, por ejemplo, búsqueda de roles similares dentro de una red) o a DFS (si se desea representar nodos cercanos según la homofilia). En la Figura 2.7 se puede observar la estrategia de muestreo para un caminante aleatorio y la utilidad de los parámetros p y q que regulan las probabilidades de transición no normalizadas en cada paso según el objetivo de representación que se desee.

Es importante notar que la estrategia de muestreo de *DeepWalk* es un caso

particular de *Node2vec* cuando $p = q = 1$ y se diferencian en que *DeepWalk* utiliza *Hierarchical softmax* mientras que *Node2vec* utiliza *Negative sampling*.

2.4. Métodos de selección significativa

Cuando estamos trabajando con muchas características, conocidas como *features*, como es el caso en análisis de textos o nodos en grafos de redes sociales, puede que exista un conjunto de *features* que sean más relevantes que el resto, así como también pueden existir *features* que no aporten información sino ruido, provocando que varios algoritmos clasificadores incurran en un mayor error. Incluso ya de por si, tener muchos *features* generalmente aletarga y aumenta la complejidad de ciertos algoritmos, además de hacerlos incurrir en *overfitting*. Por estas razones, seleccionar los *features* más relevantes es una tarea importante y a la vez complicada por la gran variedad de enfoques que existen para determinarlos. Elegir el que mejor se adapta a un problema en particular puede volverse una tarea extensa. El algoritmo más sencillo para seleccionar *features* significantes consta de calcular la utilidad $A(f, c)$ para un *feature* f y una clase c, y luego seleccionar los k *features* con mayor utilidad descartando el resto. En esta sección presentamos dos métodos para calcular la utilidad de *features*: uno basado en distribuciones estadísticas y otro con un enfoque en la ganancia de información.

2.4.1. Distribución de Pearson (χ^2)

Un método bastante conocido para seleccionar *features* significativos es el uso de pruebas χ^2 Schütze et al. (2008), las cuales constan de probar la hipótesis de que dos eventos, A y B, sean independientes. Es decir, se analiza si $P(AB) = P(A) \cdot P(B)$ o lo que es lo mismo $P(A|B) = P(A)$ y $P(B|A) = P(B)$. Para la selección de *features*, que en nuestro caso suelen ser términos, nos interesa hacer

	c	$\neg c$
t	$O_{\mathrm{t,c}}$	$O_{\mathrm{t},\neg c}$
$\neg t$	$O_{\neg t,\mathrm{c}}$	$O_{\neg t,\neg c}$

Tabla 2.1: Ejemplo de tabla de contingencia para el cálculo de χ^2 para una clase c y un término t.

pruebas sobre la co-ocurrencia de un término en una clase en particular, con lo que ambos eventos A y B serían la ocurrencia del término y la ocurrencia de la clase respectivamente. Luego, calculamos el valor de χ^2 como:

$$\chi^2(t,c) = \sum_{i\in\{t,\neg t\}} \sum_{j\in\{c,\neg c\}} \frac{(O_{\mathrm{i,j}} - E_{\mathrm{i,j}})^2}{E_{\mathrm{i,j}}} \quad ,$$

donde $E_{\mathrm{i,j}} = P(i) \cdot P(j) \cdot M$ es la frecuencia esperada para cada caso, por ejemplo, $E_{\mathrm{t,c}}$ es la frecuencia esperada del término t y la clase c ocurriendo ambos juntos en una observación asumiendo que ambos términos son independientes, $M = \sum_{i\in\{t,\neg t\}} \sum_{j\in\{c,\neg c\}} O_{\mathrm{i,j}}$ es la totalidad de observaciones y $O_{\mathrm{i,j}}$ es la frecuencia observada para cada caso, la cuál puede calcularse fácilmente si construimos una tabla de contingencia como se ve en la Tabla 2.1.

Una vez calculado el valor de χ^2 con un grado de libertad, este nos mide que tanta desviación existe entre la frecuencia esperada E y la frecuencia observada O. Un valor alto de χ^2 indica que se rechaza la hipótesis de independencia entre el término y la clase, con lo que podríamos quedarnos con los k términos que mayor valor χ^2 tengan por clase. Otra alternativa es quedarnos con aquellos términos que estén dentro de un nivel de confianza p que planteemos buscando su valor en la tabla χ^2 para ver si el término es estadísticamente significativo. Por último, debemos notar como desventaja de este método su sensibilidad a las bajas frecuencias de los términos en algunas tablas de contingencia, sin embargo, si aplicamos un

filtro teniendo en cuenta una cantidad mínima de usos, podemos solucionar este problema.

2.4.2. Información mutua

Otro método para la selección de *features* que aplicaremos a términos consiste en el uso de la información mutua Schütze et al. (2008), también conocida como ganancia de información, entre un término y una clase en particular. La información mutua mide qué tanto la presencia o ausencia de un término t contribuye a la correcta clasificación de la clase c. Matemáticamente hablando podemos calcular su valor para un término t y clase c como:

$$I(T,C) = \sum_{i\in\{1,0\}} \sum_{j\in\{1,0\}} P(T=i, C=j) \cdot \log_2 \frac{P(T=i, C=j)}{P(T=i) \cdot P(C=j)} \quad ,$$

donde T es una variable aleatoria que toma el valor 1 cuando la observación/documento contiene al término t y 0 en caso contrario, y C es una variable aleatoria que toma el valor 1 cuando la observación/documento pertenece a la clase c y 0 en caso contrario. Un valor de $I(T,C) = 0$ nos indica que el término ocurre en igual frecuencia para todas las clases. En el otro extremo, $I(T,C) = 1$ nos indica que el término es un perfecto indicador de la clase c, dicho de otra forma, si el término t se encuentra presente en el documento, entonces el documento pertenece a la clase c. Para seleccionar los términos más significativos debemos tomar los k términos con mayor utilidad según la información mutua $I(T,C)$ para cada clase.

2.5. Métricas y métodos de evaluación

En esta sección explicaremos varias métricas que se utilizaran para entrenar y evaluar el desempeño de nuestros métodos, para poder compararlos y elegir el más conveniente para el problema de detección de ubicación de usuarios de Twitter. Por

último explicaremos diversas técnicas para la selección de los mejores parámetros en cada algoritmo, y discutiremos formas de trabajo cuando disponemos de pocas muestras y/o las mismas se encuentran desbalanceadas con respecto a las clases.

2.5.1. Métricas básicas

Para evaluar nuestros modelos necesitamos diversas métricas que nos permitan ver, desde distintos puntos de vista, qué tan bien aprendieron y clasifican. Antes de hablar sobre las métricas debemos definir algunos conceptos sobre las clasificaciones que, en nuestro caso, serán con respecto a la ciudad a la que pertenece un usuario. Supongamos inicialmente un problema de clasificación binaria en donde las clases posibles para cada muestra son c y $\neg c$ (luego extenderemos el modelo para N clases). Se dice que c es la clase positiva y $\neg c$ la clase negativa. Definimos entonces:

- Verdadero positivo (TP): El modelo predice correctamente la clase positiva.
- Verdadero negativo (TN): El modelo predice correctamente la clase negativa.
- Falso positivo (FP): El modelo predice incorrectamente como positiva la clase negativa.
- Falso negativo (FN): El modelo predice incorrectamente como negativa la clase positiva.

Una forma de visualizar estos valores es a través de la matriz de confusión, tal como se puede ver en la Figura 2.8.

Definimos la exactitud (*accuracy*) como la cantidad de clasificaciones correctas con respecto a todas las clasificaciones, matemáticamente hablando:

$$Accuracy = \frac{\#Clasificaciones\_Correctas}{\#Clasificaciones\_Totales} = \frac{TP + TN}{TP + TN + FN + FP}$$

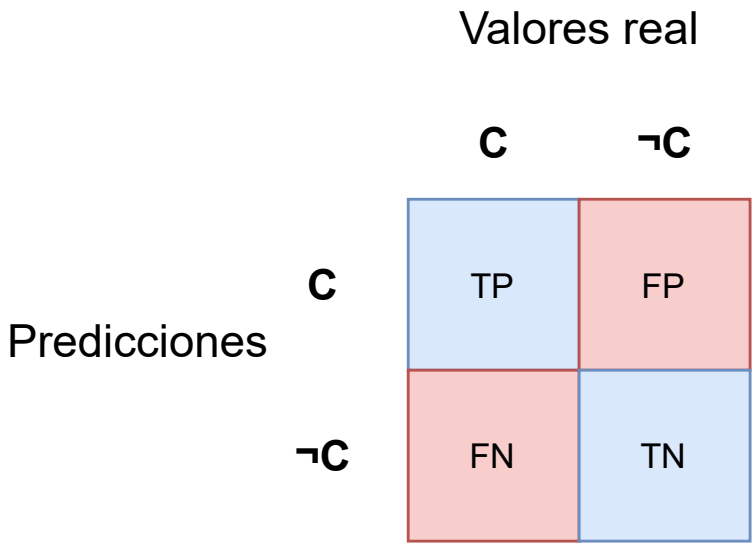


Figura 2.8: Ejemplo de matriz de confusión para la clase positiva c y la clase negativa $\neg c$. La construcción de esta matriz es muy útil para analizar y comparar clasificadores.

Debemos tener especial cuidado con la *accuracy* cuando trabajamos con conjuntos de datos desbalanceados con respecto a las clases, ya que podríamos tener una *accuracy* alta simplemente clasificando todo como la clase mayoritaria, lo cual puede ser contraproducente dependiendo del problema que se esté trabajando. Para comprender mejor estos casos, existen algunas métricas que nos ayudan a analizar las clasificaciones desde otras perspectivas: la precisión (*precision*) nos indica qué fracción de los resultados clasificados positivos son relevantes, es decir, realmente positivos, y la sensibilidad (*recall*) nos indica fracción de los resultados relevantes (realmente positivos) de la clasificación son capturados o clasificados como positivos. Estas se calculan como:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

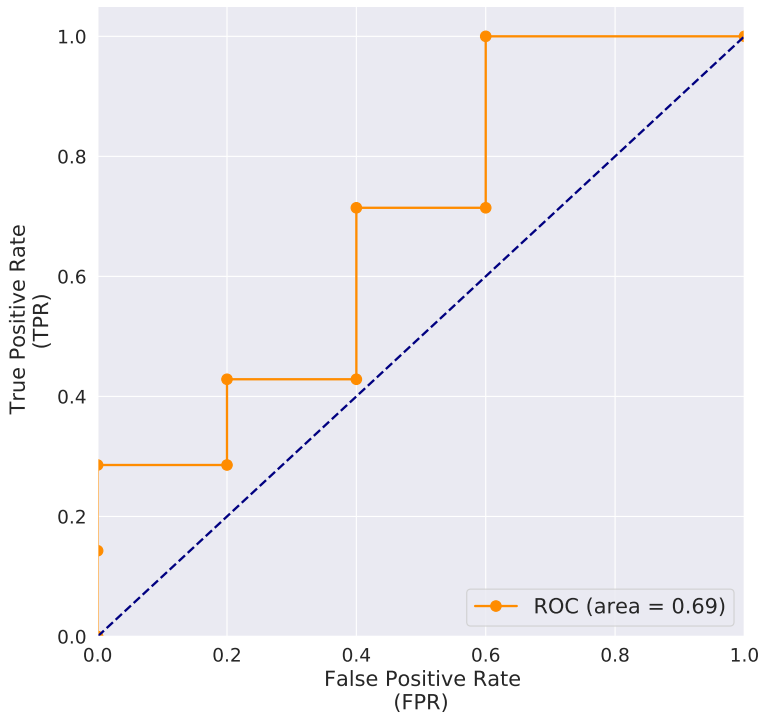
Dependiendo del problema puede que sea necesario darle más relevancia a una que a la otra. Para lidiar con esto, definimos el valor-F$_\beta$ (F_β-*Score*) como una ponderación de la *precision* y el *recall*:

$$F_\beta = (1 + \beta^2)\frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall}$$

Dependiendo del valor de β podemos darle más importancia a una métrica con respecto a la otra: para $\beta < 1$ le asignamos un mayor peso a la *precision*, en caso contrario le estaríamos asignando un mayor peso al *recall*. En la mayoría de los casos se suele utilizar un valor de $\beta = 1$ para darle igual peso a ambos valores, llamándose así F_1-*Score*. Otra métrica que podemos utilizar en conjuntos de datos desbalanceados es la exactitud balanceada (*balanced accuracy*). Se define a la misma como el promedio entre la fracción de resultados realmente positivos clasificados como positivos, es decir el *recall*, y la fracción de resultados realmente negativos clasificados como negativos, llamado especificidad (*specificity*):

$$Balanced\_Accuracy = \frac{Recall + Specifity}{2}$$

La última métrica que usaremos es el área bajo la curva ROC, de ahora en adelante *ROC-Auc*. Se define la curva ROC como la representación de la fracción de verdaderos positivos ($TPR = Recall$) con respecto a la fracción de falsos positivos ($FPR = 1 - Specificity$). Esta curva nos permite analizar las compensaciones relativas entre los verdaderos positivos (TP) y los falsos positivos (FP). En problemas de clasificación binaria la curva ROC se construye por medio de puntajes asignados por los clasificadores (que en nuestro caso serán probabilidades) haciendo variar un umbral que determine cuándo clasificar una muestra como positiva según su puntaje asignado. Un ejemplo de esto puede verse en la Figura 2.9, donde el punto (0, 1) representa una clasificación perfecta. Notemos que un clasificador con un umbral fijo es simplemente un punto en el gráfico de la ROC y



Clase	Puntaje
C	0.90
C	0.80
$\neg C$	0.75
C	0.70
$\neg C$	0.65
C	0.62
C	0.60
$\neg C$	0.50
C	0.40
C	0.38
$\neg C$	0.20
$\neg C$	0.10

Figura 2.9: Ejemplo de curva ROC para clasificación binaria de clases c y $\neg c$. Cada punto del gráfico se construye variando el umbral que considera a una clase como positiva. La línea azul central corresponde a una curva ROC de un clasificador aleatorio.

que, a partir de la ROC, podemos pensar qué tan conservativo es un clasificador, es decir, si solo realiza clasificaciones positivas si tiene una fuerte evidencia (de modo que se producen bajos falsos positivos pero generalmente también bajos verdaderos positivos). Como en los casos anteriores estamos utilizando escalares para poder comparar clasificadores, aquí haremos lo mismo computando el área bajo la curva ROC cuyo valor ronda entre 0 y 1, siendo 0,5 el valor asignado para un clasificador que realiza predicciones al azar. Para más información sobre la curva ROC recomendamos leer el trabajo de Fawcett (2006).

Hasta el momento estuvimos hablando sobre problemas de clasificación binaria pero podemos extender estos modelos a clasificación para más clases como es el caso de este trabajo, en donde poseemos aproximadamente 100 ciudades para clasificar. Existen varias estrategias para el cálculo de cada métrica, pero en líneas generales podemos tomar tres enfoques:

- Enfoque global o *micro*: Considerar los cálculos de métricas en base a TP, TN, FP, FN totales, en esta categoría podemos calcular la *Overall accuracy*.
- Enfoque *macro* o *One-vs-rest*: Considerar para N clases, N clasificaciones binarias donde la clase positiva es la clase en sí y la clase negativa cualquier otra clase. En esta categoría podemos calcular las métricas y ponderarlas, ya sea equitativamente o por peso relativo a la cantidad de muestras por clase. Bajo esta estrategia se suelen calcular los *F-scores*, la *balanced-accuracy* o hasta las *ROC-Auc*.
- *One-vs-One*: Considerar para N clases, $\frac{N(N-1)}{2}$ clasificaciones binarias de todas las clases entre sí. Bajo esta estrategia se suele calcular las *ROC-Auc* y ponderarlas, ya sea equitativamente o por peso relativo a la cantidad de muestras por clase.

2.5.2. Validación cruzada

Como parte del proceso de entrenamiento debemos buscar los mejores parámetros de nuestro modelo, llamados hiper-parámetros. También deberemos evaluar parcialmente nuestros modelos analizando métricas y estimando errores de predicción pero no siempre dispondremos de suficientes muestras para dividir nuestro conjunto de datos en conjuntos de entrenamiento, validación y pruebas. En estos casos podemos aprovechar la técnica de validación cruzada por medio de *K-Folds*, en donde todo el conjunto de datos se divide en K subconjuntos disjuntos del

mismo tamaño, y se evaluan los modelos utilizando $K-1$para entrenamiento y el conjunto restante para validación o pruebas. Esta técnica tiene la ventaja de que todas las muestras pueden formar parte, en alguna de las iteraciones, del conjunto de pruebas y de entrenamiento. Generalmente se suelen tomar divisiones con $K = 5$ o $K = 10$. Un esquema sobre como aplicar la validación cruzada por medio de *K-Folds* puede verse en la Figura 2.10, dando una idea de cómo generamos predicciones para todas las muestras disponiendo de la totalidad del conjunto de datos para verificar que tan bien se adapta nuestro modelo a los mismos. Otro uso interesante es la búsqueda de los mejores hiper-parámetros de un modelo en donde evaluamos diferentes combinaciones de valores de parámetros en distintos conjuntos de entrenamiento para quedarnos con aquellos que, en promedio, den mejores resultados o minimicen el error cometido para los conjuntos de prueba.

2.5.3. Validación cruzada anidada

Si buscamos los mejores hiper-parámetros usando validación cruzada sobre el conjunto de entrenamiento (conjuntos azules de la Figura 2.10) podríamos generar un modelo que se adapta muy bien al mismo pero no logra generalizar bien y por lo tanto clasificar bien al conjunto de pruebas restante. Esto es debido a que para la búsqueda de hiper-parámetros se realiza un entrenamiento y evaluación previa sobre todo el conjunto de entrenamiento. Para resolver este problema debemos de tener un conjunto de validación aparte para validar la búsqueda de mejores hiper-parámetros. Una de las soluciones a este problema es la técnica de la validación cruzada anidada (*nested cross-validation*) en donde agregamos un segundo nivel de validación cruzada sobre cada conjunto de entrenamiento para encontrar los mejores hiper-parámetros del modelo en esta instancia. Un ejemplo de esto puede verse en la Figura 2.11, en donde usamos validación cruzada sobre los conjuntos de entrenamiento para encontrar los mejores hiper-parámetros del modelo en varias

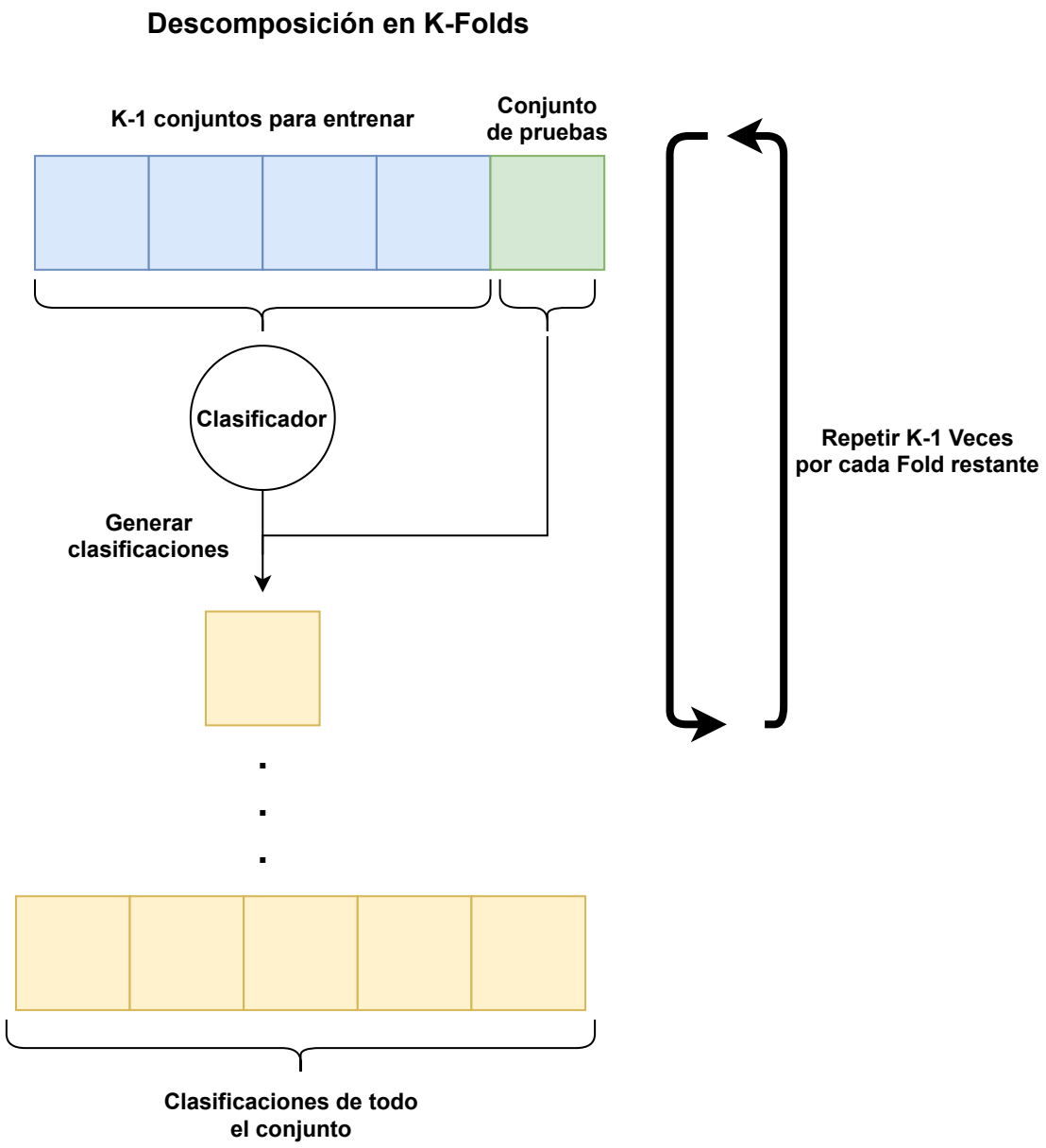


Figura 2.10: Ejemplo de uso de validación cruzada por medio de *K-Folds* para la generación de predicciones de todo el conjunto de datos. Puede verse como una división iterativa del conjunto de datos en donde cada muestra es aprovechada tanto para entrenar como para formar parte del conjunto de pruebas.

combinaciones posibles de entrenamiento y validación en esta instancia. Esta forma de evaluar modelos tiene la ventaja de que no solo aprovechamos todo el conjunto de datos sino que también generamos las clasificaciones para evaluar el modelo en general usando los mejores hiper-parámetros en cada instancia. La principal desventaja de este método de evaluación de modelos es que debemos entrenar $K_1 \cdot K_2$ modelos distintos, en donde K_1 y K_2 son los K_1-Folds internos y K_2-Folds externos respectivamente.

2.6. Indexación espacial

Los índices espaciales son estructuras de datos que permiten acceder a elementos del espacio en forma eficiente. Estos índices son muy comunes en el ámbito de bases de datos para la ejecución de consultas de búsqueda de puntos en el espacio, intersección de polígonos, unión de polígonos y hasta búsqueda de puntos cercanos a otro. En este trabajo vamos a aprovechar estos índices para generar clases para nuestros datos según la distribución en el espacio de los mismos.

2.6.1. Árboles KD (KD-Trees)

Un árbol binario es una estructura de datos definida recursivamente por una colección de nodos los cuales poseen un valor y hasta dos hijos, también conocidos como hijo derecho e hijo izquierdo; el nodo superior del árbol es conocido como raíz y los nodos con mayor profundidad son llamados hojas. Entonces, un árbol KD es un tipo especial de árbol binario que permite organizar puntos en un espacio de dimensión K por medio de particiones en cada dimensión según el nivel de profundidad del árbol. Existen múltiples formas de construir árboles KD: por ejemplo, el Algoritmo 1 nos muestra cómo construir un árbol KD balanceado, es decir, un árbol tal que la distancias entre los nodos hoja y la raíz sean aproximadamente

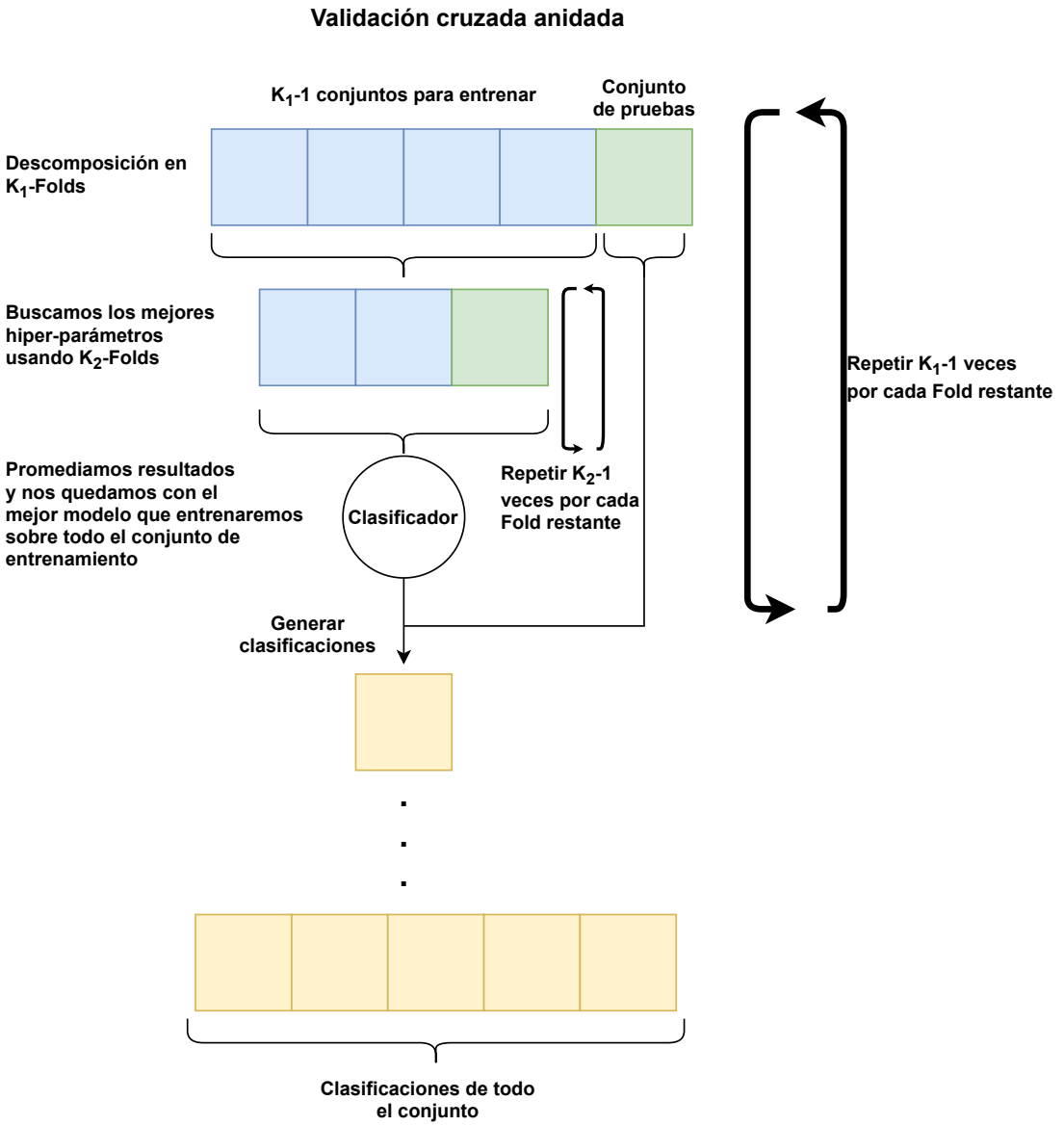


Figura 2.11: Ejemplo de uso de validación cruzada anidada para la generación de predicciones de todo el conjunto de datos usando distintos modelos en cada instancia con sus respectivos mejores hiper-parámetros.

las mismas.

Una de las ventajas de los árboles KD balanceados es que permiten dividir al espacio de dimensión K en regiones con la misma cantidad de puntos para el mismo nivel de profundidad del árbol, con lo que si llevamos esto a la práctica con puntos distribuidos geográficamente por el mundo, podemos construir árboles 2D en donde cada nivel de profundidad nos separa a los puntos en regiones con la misma cantidad de puntos. De hecho, si para un nivel de profundidad N del árbol tenemos M puntos por región entonces en el nivel $N+1$ del árbol tendremos $M/2$ muestras por región; esto ocurrirá siempre y cuando no existan puntos en la misma ubicación. Podemos entonces utilizar KD-Trees para distribuir a nuestros usuarios, según su ubicación como par latitud-longitud, en regiones con una similar cantidad de usuarios. De esta forma podemos evitar el problema de desbalance de muestras por ciudad que tenemos en nuestro conjunto de datos. Podemos ver en la figura 2.12 la separación de nuestros usuarios en regiones por medio de 2D-Trees en donde exigimos un mínimo de 255 muestras por región. La implementación de 2D-Trees utilizada en este trabajo es iterativa y se detiene su ejecución una vez que la generación de un nuevo nivel del árbol no cumpla con el mínimo de muestras que deseamos. Para más información sobre usos de KD-Trees e implementación de sus operaciones recomendamos leer Bentley (1975).

2.7. Clasificadores

En un problema de clasificación existe un conjunto de características o *features* $F = \{f_1, f_2, ..., f_n\}$ y un conjunto fijo de clases $C = \{c_1, c_2, ..., c_m\}$, y el objetivo consiste en clasificar en base a las características de las muestras, sus correspondientes clases asociadas. Para esto, existe una variedad de clasificadores, también llamados algoritmos de aprendizaje, en donde buscamos aprender una función y

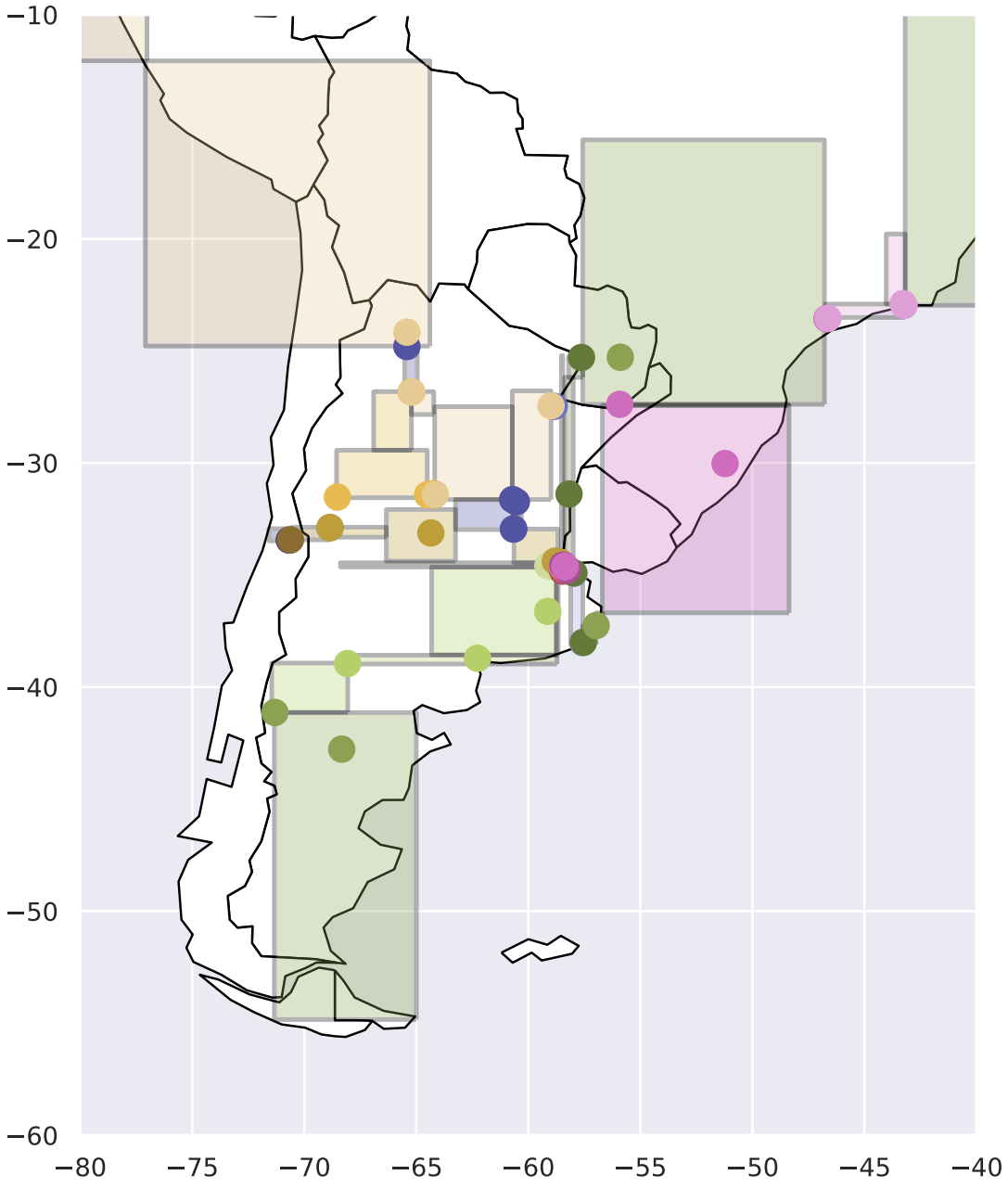


Figura 2.12: Ejemplo de división del espacio de coordenadas para los usuarios de Twitter-ARG-Exact en regiones con un mínimo de 255 muestras usando KD-Tree. Los puntos dentro de cada región representan la mediana de las coordenadas de los usuarios incluidos.

Algorithm 1 Construcción de KD-Tree balanceado

function KDTREE(puntos, profundidad=0)

$dim \leftarrow Dimension(puntos)$

$eje \leftarrow profundidad \,\%\, dim$

$mediana\_eje \leftarrow Mediana(puntos, eje)$

$puntos\_izq \leftarrow \{puntos | puntos[eje] \leq mediana\_eje\}$

$puntos\_der \leftarrow \{puntos | puntos[eje] > mediana\_eje\}$

return Nodo(

valor=$mediana\_eje$,

hijo_izquierda=$puntos\_izq, profundidad + 1$),

hijo_derecha=$kdtree(puntos\_der, profundidad + 1)$

)

end function

que mapee *features* F con clases. Este tipo de aprendizaje es el que llamamos aprendizaje supervisado, ya que nosotros definimos el conjunto de clases C fijo de antemano. En esta sección vamos a explicar distintos algoritmos de aprendizaje supervisado que utilizamos a lo largo de este trabajo según las definiciones de Aggarwal (2015).

2.7.1. Naive Bayes

Un clasificador Naive Bayes es conocido por ser el clasificador bayesiano más simple y uno de los más utilizados en el contexto de análisis de textos y diagnóstico médico. Además, es un clasificador que funciona aún cuando la dimensionalidad de los *features* es muy grande. Dada una variable discreta Y cuyo valor es exactamente una de las posibles del conjunto C, y un conjunto de *features* $X_1, ..., X_{\mathrm{d}}$ discretos o continuos, se define como el objetivo de este método entrenar un clasificador que

produzca como resultado la probabilidad *a posteriori* $p(Y|X)$ para los posibles valores de Y basándose en el teorema de Bayes:

$$p(Y = C_{\mathrm{k}}|X = x) = \frac{p(X = x|Y = C_{\mathrm{k}})p(Y = C_{\mathrm{k}})}{p(X = x)}$$

Dado que es muy costoso entrenar clasificadores bayesianos exactos, en la práctica se recurre a un par de simplificaciones para estimar los parámetros. En primer lugar asumimos la condición de que los *features* $X_1, ..., X_{\mathrm{d}}$ son independientes entre sí dado Y, de esta forma el término $p(X = x|Y = C_{\mathrm{k}})$ conocido como verosimilitud o *likelihood* se puede escribir como:

$$p(X = x|Y = C_{\mathrm{k}}) = \prod_{j=1}^{d} p(X_{\mathrm{j}} = x_{\mathrm{j}}|Y = C_{\mathrm{k}})$$

Por asumir que los *features* $X_1, ..., X_{\mathrm{d}}$ son independientes entre sí dado Y es que este método se conoce como Naive Bayes o clasificador bayesiano ingenuo. Otra simplificación que suele hacerse en la práctica es no calcular la evidencia $p(X = x)$ dado que esta no depende de las clases y es constante, con lo que la regla de un clasificador Naive Bayes pasa a ser:

$$p(Y = C_{\mathrm{k}}|X = x) \approx p(Y = C_{\mathrm{k}}) \prod_{j=1}^{d} p(X_{\mathrm{j}} = x_{\mathrm{j}}|Y = C_{\mathrm{k}})$$

Para Naive Bayes, dado un conjunto de *features*, la mejor clase es la realiza el *máximo a posteriori*:

$$\hat{Y} \leftarrow argmax_{\mathrm{C_k}} p(Y = C_{\mathrm{k}}) \prod_{j=1}^{d} p(X_{\mathrm{j}} = x_{\mathrm{j}}|Y = C_{\mathrm{k}})$$

Estos dos parámetros resultantes se suelen estimar por medio de máxima verosimilitud o *maximum likelihood estimation* (MLE): el primer parámetro se estima como $p(Y = C_{\mathrm{k}}) = \frac{N_{\mathrm{k}}}{N}$ donde N es la cantidad de muestras y N_{k} la cantidad de

muestras pertenecientes a la clase C_{k}, y el segundo parámetro *likelihood* se estima como $p(X_{\mathrm{j}} = x_{\mathrm{j}}|Y = C_{\mathrm{k}}) = \frac{N_{\mathrm{k,x}}}{N_{\mathrm{k}}}$ en donde $N_{\mathrm{k,x}}$ es la cantidad de veces que ocurre el *feature* x en muestras de la clase C_{k}. En este trabajo utilizamos clasificadores Multinomial Naive Bayes, que pertenecen a esta familia Naive Bayes y definen que la distribución de la *likelihood* $p(X_{\mathrm{j}} = x_{\mathrm{j}}|Y = C_{\mathrm{k}})$ es la distribución multinomial.

2.7.2. Árboles de decisión

Un árbol de decisión es un método de particionamiento jerárquico de datos utilizado en el contexto de aprendizaje supervisado para relacionar sus respectivas hojas con una clase en específico. El algoritmo comienza con todas las muestras en el nodo raíz y recursivamente divide al árbol utilizando uno o varios *features*, buscando minimizar la suma ponderada de la entropía o el coeficiente de Gini para los dos nodos resultantes de cada división. El algoritmo finaliza cuando todos los nodos hoja contienen muestras de una única clase o cuando que se cumpla algún criterio de finalizado del árbol, como puede ser la longitud máxima o la ganancia de información por división.

2.7.3. Regresión logística

La regresión logística es un método de aprendizaje supervisado bastante popular cuyo objetivo es estimar la probabilidad *a posteriori* $P(Y = i|X)$ basándose en las muestras, en donde X es un vector de *features* discretos o continuos e $Y \in \{0, 1\}$ nos define un problema de clasificación binario. El modelo puede definirse matemáticamente de la siguiente forma:

$$P(Y = 1|X) = g(\theta^T X) = \frac{1}{1 + e^{-\theta^T X}} \quad ,$$

donde $g(z) = \frac{1}{1+e^{-z}}$ es conocida como la función logística o función sigmoidea.

Como la suma de probabilidades siempre debe ser igual a 1, $P(Y = 0|X)$ puede ser estimada como:

$$P(Y = 0|X) = 1 - g(\theta^T X) = \frac{e^{-\theta^T X}}{1 + e^{-\theta^T X}}$$

Como la regresión logística nos predice probabilidades en vez de clases, podemos ajustarla usando la verosimilitud o *likelihood*. Si combinamos ambas ecuaciones presentadas teniendo en cuenta que por cada muestra usada para entrenar el regresor logístico, tenemos un conjunto X y una clase $Y = y$, entonces la probabilidad de dicha clase es $g(\theta^T X)$ si $y = 1$ o $1 - g(\theta^T X)$ si $y = 0$. El modelo por muestra se define en forma compacta como:

$$P(Y = y|X; \theta) = P(Y = 1|X; \theta)^y P(Y = 0|X; \theta)^{1-y}$$

Si asumimos que las muestras fueron generadas de forma independiente, la verosimilitud de los parámetros se puede calcular como:

$$L(\theta) = \prod_{n=1}^{N} (g(\theta^T X^n))^{Y^n} (1 - g(\theta^T X^n))^{1-Y^n} \quad ,$$

donde N es la totalidad de muestras, θ el vector de parámetros a estimar, Y^n es la clase observada para la muestra n y X^n el vector de *features* de la muestra n.

2.8. Ensamblado de clasificadores

El objetivo por el cual entrenamos clasificadores es encontrar un modelo que pueda predecir las clases de futuros datos no vistos. Para esto es necesario que el modelo generalice muy bien estas muestras, llamándose así un clasificador fuerte. En la mayoría de los casos encontrar un clasificador fuerte no es tarea sencilla, pero por medio del ensamblado de varios clasificadores podemos transformar un

conjunto de clasificadores débiles en un clasificador fuerte mediante su combinación. En líneas generales el ensamblado consta en aprender N clasificadores $C = \{c_1(x), c_2(x), ..., c_N(x)\}$ que mapean un conjunto x de *features* a clases, y luego combinar estos resultados en un meta-clasificador $H(x)$ con la esperanza de obtener mejores resultados. Existen diversas técnicas para ensamblar clasificadores, las cuales podemos organizar en un conjunto de categorías, entre ellas las que recaen dentro de este trabajo son:

- *Boosting*: Secuencialmente se determinan los mejores clasificadores para distintas porciones complicadas de generalizar de los datos, y de esta forma combinando los resultados, obtenemos un meta-clasificador que funciona bien en todo el conjunto de datos.

- *Bagging*: Este enfoque consiste en tomar muestras aleatoriamente con reposición para entrenar un conjunto de clasificadores, y luego con los resultados obtenidos entrenar un meta-clasificador.

- *Stacking* o Apilado: El apilado o *Stacking* es una técnica en la cual entrenamos clasificadores en un primer nivel que producen como salida una nueva representación de los *features*, que será utilizada como entrada para un meta-clasificador de segundo nivel.

2.9. Redes neuronales

Una red neuronal es un grafo de unidades conectadas que intenta ser un modelo matemático de las redes neuronales biológicas. Una red neuronal recibe datos de entrada en un conjunto de unidades o neuronas destinadas a la entrada de datos y produce una salida a través de neuronas destinadas a la salida de datos. Una unidad puede actuar tanto como unidad de entrada o como unidad de salida. El

resto de las neuronas son responsables de realizar cálculos lógicos. En lenguaje matemático, una neurona transforma un conjunto de entradas $x = (x_1, x_2, ..., x_d)$ en un escalar o, utilizando la composición de dos funciones:

- Una función ξ que transforma las entradas en un valor de red v por medio de pesos w propios de la unidad: $v = \xi(x, w)$.
- Una función de activación ϕ que transforma el valor de red v en un escalar o: $o = \phi(v)$. A día de hoy existen múltiples funciones de activación y su elección depende exclusivamente de la topología de la red neuronal y el enfoque que se espera de la unidad.

El resultado de la función de activación de la unidad es luego utilizado como entrada de las próximas unidades, conformando así una red de unidades que transfieren señales, como si de neuronas reales se tratase. Una red neuronal puede además poseer varias capas intermedias entre la capa de entrada y capa de salida, conocidas como capas ocultas (*hidden layers*). La red neuronal en su versión más sencilla, que no posee ninguna capa oculta entre sus capas de entrada y salida, es el *perceptrón*. El mismo es utilizado comúnmente para problemas de clasificación en donde los datos son linealmente separables. Cuando un problema de clasificación no es linealmente separable, es necesario el uso de capas ocultas para mejorar las capacidades de la red. Este tipo de redes en donde la información fluye en un único sentido son conocidas como redes neuronales prealimentadas (*feedforward neural networks*) o perceptrones multicapa. Estas redes multicapa definen un mapeo $y = f(x; \theta)$ y su objetivo es aproximar alguna función f^* para un clasificador $y = f^*(x)$ que mapea una entrada x a una clase y aprendiendo los parámetros θ que resulten en la mejor aproximación de dicha función. Podemos pensar que cada capa de una red neuronal prealimentada nos define una función f, por ejemplo, una red con tres capas conectadas en cadena conforma una fun-

ción $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ y esta función $f(x)$ es la que buscamos aproximar, por medio del aprendizaje de parámetros, a la función $f^*(x)$ usando las muestras disponibles para el entrenamiento. Cada muestra x nos define cuál es la salida o clase esperada y que debe generar la capa de salida de la red. La forma en que produce dicho resultado la red no es producto de la muestra sino del algoritmo de aprendizaje que se utilice, por este motivo es que dichas capas son llamadas capas ocultas. Cuantas más capas utilicemos más profundo será nuestro modelo de red, este concepto da lugar al aprendizaje profundo, también conocido como *deep learning*.

Hasta el momento hablamos de redes neuronales sin realimentación o *feedback*. Cuando extendemos la red neuronal para incluir la realimentación de salidas del modelo, las redes pasan a llamarse redes neuronales recurrentes (*recurrent neural networks*). Estas redes, a diferencia de las redes neuronales prealimentadas, presentan ciclos en el grafo que representa la red neuronal y permiten, en entradas secuenciales, utilizar información de pasos anteriores de la secuencia en cada paso subsiguiente de la misma. De esta forma se pueden aprender patrones o dependencias sobre la secuencia en su totalidad.

Un detalle que omitimos hasta el momento es cómo las redes actualizan los pesos de sus capas para ajustar al error obtenido por la función de costo, elegida al momento de entrenar, en cada paso del entrenamiento. Para ajustar los pesos se utiliza el algoritmo de propagación hacia atrás (*backpropagation algorithm*) que permite fluir la información de los costos hacia atrás en la red, de forma tal que se pueda calcular el gradiente, que generalmente corresponde a la función de costo con respecto a los parámetros θ, para ajustar los pesos de cada capa. No debe confundirse este algoritmo con el método de aprendizaje: la propagación hacia atrás se refiere solamente al cálculo del gradiente mientras que otro algoritmo, como por ejemplo el descenso de gradiente estocástico (*stochastic gradient des-*

cent), abreviado SGD, es utilizado para ajustar los pesos o aprender por medio del gradiente. Para más información sobre redes neuronales recomendamos leer como introducción Aggarwal (2015) y si se desea ahondar en redes neuronales profundas recomendamos Goodfellow et al. (2016).

2.9.1. LSTM - Long short-term memory

Como bien mencionamos las RNN son muy útiles para aprender patrones sobre entradas secuenciales como textos de tweets. Sin embargo, dada su naturaleza con ciclos estas tienden a ser redes muy profundas en el tiempo (considerando cada paso en una secuencia como un paso en el tiempo). Las redes muy profundas, en particular las RNN, sufren de un problema conocido como el problema del desvanecimiento de gradiente (*vanishing gradient problem*). Supongamos que en cada paso de una RNN se multiplica por una matriz W. Luego de t pasos esto es equivalente a multiplicar por W^t, y suponiendo además que W tiene una descomposición en valores singulares $W = V.diag(\lambda).V^{-1}$, es inmediato ver que $W^t = V.diag(\lambda)^t.V^{-1}$, con lo que cualquier valor singular λ_i que no esté próximo a 1 va a crecer exponencialmente o va a ser extremadamente bajo. El problema del desvanecimiento del gradiente se refiere al hecho de que el gradiente, al ser escalado por los autovalores de W^t, se va achicando gradualmente a medida que se propaga hacia atrás en la red, provocando que las primeras capas no puedan ajustar sus parámetros para mejorar la minimización de la función de costo. Este problema trae como consecuencia que las RNN no puedan aprender dependencias a largo plazo en una secuencia larga. En 1997 Hochreiter and Schmidhuber (1997) propusieron LSTM (*Long Short-Term Memory*), una RNN capaz de aprender dependencias a largo plazo en entradas secuenciales evitando el problema del desvanecimiento del gradiente. La estructura básica de una LSTM (a día de hoy existen múltiples variantes) puede verse en la Figura 2.13. En dicha estructura po-

demos visualizar dos carriles, uno inferior correspondiente a la entrada de datos y uno superior correspondiente al estado de la celda, es decir, información relevante conservada hasta el paso t. Una idea general de cómo funciona la LSTM puede verse en tres pasos:

- El primer paso consta en eliminar información pasada redundante del estado de la celda C_{t-1} por medio de la primer capa usando una función sigmoidea que determine qué grado de redundancia (recordando que la sigmoide genera valores entre 0 y 1) tiene cada elemento del vector del estado de la celda.

- El segundo paso consiste de actualizar/agregar información al estado de la celda $C_{t-1} \rightarrow C_t$. Se utiliza una capa con una sigmoide para determinar qué elementos del vector actualizar y una capa extra con una tangente hiperbólica para determinar un vector con los nuevos candidatos a agregar al estado de la celda.

- El tercer y último paso consiste en determinar la salida de la celda h_t. Por medio de una capa con una sigmoide se determina qué elementos del vector del estado de la celda se van a utilizar como salida. Luego se filtra el estado de la celda por medio de una tangente hiperbólica para acotar los resultados en un rango entre -1 y 1.

El uso de capas con tangentes hiperbólicas ayuda a evitar el problema del desvanecimiento del gradiente.

2.9.2. Transformadores y mecanismos de atención

Las RNN son un modelo muy útil para problemas de procesamiento del lenguaje natural como lo son la traducción y el resumen de textos. Sin embargo, estos

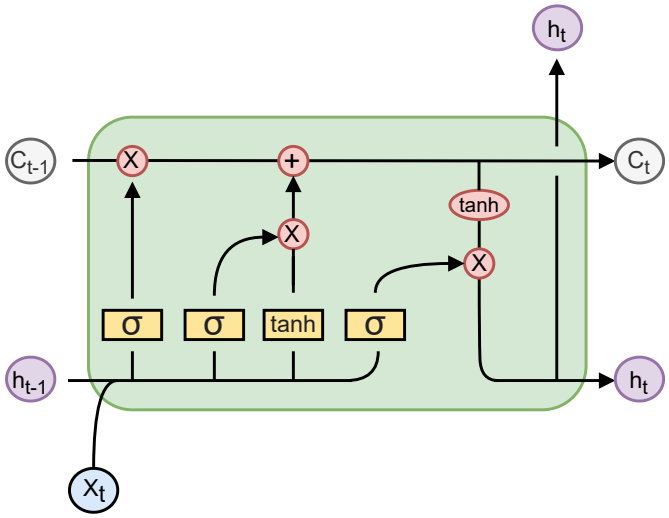


Figura 2.13: Ejemplo de celda de LSTM en el paso t. La unión de flechas representa la concatenación de vectores y su bifurcación, la copia.

modelos pueden ser muy lentos de entrenar debido a que es necesario que procesen toda la secuencia de entrada, que en muchos casos puede ser bastante larga, de forma secuencial. Vaswani et al. (2017) propusieron una arquitectura llamada Transformador (*Transformer*), basada únicamente en mecanismos de atención, capaz de manejar entradas secuenciales al igual que las RNN con la diferencia de que no necesitan procesar la entrada de datos de forma secuencial. Por lo tanto, los transformadores pueden paralelizar gran parte de su trabajo generando un nivel de eficiencia mayor a modelos como las LSTM. La Figura 2.14 muestra la arquitectura propuesta para el Transformador, en donde la sección izquierda del diagrama representa el codificador (*encoder*) capaz de generar las representaciones vectoriales capturando información relevante para el vector de entrada, y la sección derecha del diagrama corresponde a el decodificador (*decoder*), utilizado para construir las secuencias a partir el contexto generado por el codificador. Nos interesa hacer foco en cómo funciona la codificación del Transformador, dado que en este trabajo no vamos a trabajar con modelos que produzcan secuencias, por

lo que la parte de decodificación no será utilizada en nuestros modelos. El transformador, en particular la parte del codificador, utiliza como entrada la suma de dos vectores, uno correspondiente a los *embeddings* de la secuencia de entrada y otro vector único por palabra de la secuencia que ayuda al modelo a determinar su posición (en el trabajo mencionado utilizaron funciones seno y coseno de distintas frecuencias para los vectores de posicionamiento). Cada vector resultante se ingresa a varias capas de atención que determinarán qué otras palabras son relevantes para obtener una buena codificación de la palabra actual. De esta forma el modelo es capaz de aprender dependencias o relaciones para palabras en secuencias largas como ocurre con las LSTM.

2.9.3. GCN - Graph Convolutional Network

En 2017 Kipf and Welling (2017) propusieron las redes convolucionales basadas en grafos (GCN), un método escalable y de bajo costo en cuanto a memoria y procesamiento, para el aprendizaje semi-supervisado en grafos usando conceptos sobre redes neuronales convolucionales. Las GCN toman como entrada una matriz de adyacencia de un grafo con *loops* para cada nodo $(A + I)$ y un vector X de *features* por nodo. El método consta de varias capas en que se propagan los *features* de los nodos buscando aprender una matriz W por medio de una regla basada en aproximaciones de primer orden de convoluciones espectrales en grafos:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in N_i} \frac{1}{c_{ij}} h_j^{(l)} W^{(l)}\right)$$

En cada paso se propaga información, por cada nodo, de sus respectivos vecinos de nivel l, con lo que si utilizásemos una GCN de 3 capas, estaríamos realizando tres pasos de propagación generando que cada nodo conozca la información de nodos ubicados hasta 3 pasos de distancia del mismo.

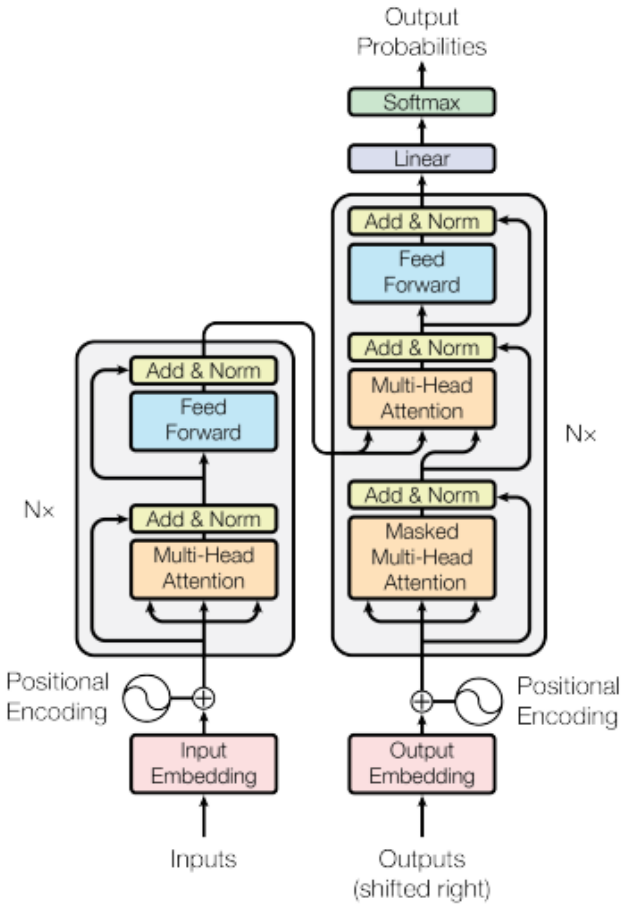


Figura 2.14: Arquitectura del Transformador. Imagen extraída de Vaswani et al. (2017).

Las GCN tienen como ventaja que pueden aplicarse a grafos con una masiva cantidad de vértices, como por ejemplo grafos de redes sociales, debido a que escalan linealmente con la cantidad de aristas que, como sabemos, suelen ser un porcentaje muy bajo con respecto a un grafo completo con dicha cantidad de vértices. Sin embargo, las GCN propuestas en este trabajo no admiten *features* en las aristas y están limitadas a grafos no dirigidos. Por otro lado es posible representar estos casos extendiendo los grafos a grafos bipartitos con nodos adicionales que representen los distintos tipos de aristas. Las GCN tampoco son capaces de asignar un peso a las distintas conexiones durante el entrenamiento, con lo que se considera igual de importante, por ejemplo, una conexión del nodo consigo mismo que con cualquier nodo vecino.

2.9.4. GraphSAGE

La mayoría de los métodos para obtener representaciones vectoriales o *embeddings* de nodos de grafos, como por ejemplo las GCN, Node2vec y DeepWalk, necesitan del grafo de la red en su totalidad para generar las representaciones de un único nodo. Esto provoca que si se quisiera generar un *embedding* de un nodo nuevo en la red, se deba re-entrenar el método con el nuevo grafo en su totalidad. Para redes que van creciendo con el tiempo, como ocurre con las redes sociales, generar *embeddings* para nuevos nodos se convierte en una tarea costosa. Este tipo de algoritmos que requieren de toda la información del grafo para generar los *embeddings* son conocidos como algoritmos transductivos. Hamilton et al. (2017) propusieron GraphSAGE, un entorno de trabajo capaz de generar *embeddings* de nodos en grafos de forma inductiva, es decir, a partir de nodos no vistos durante la etapa de entrenamiento. Al igual que las GCN, GraphSAGE utiliza los *features* para construir los *embeddings*, pero en vez de calcularlos por nodo lo hace por vecindario. El algoritmo entrena un conjunto de funciones agregadoras (*aggregator*

functions) que reciben como entrada la vecindad de un nodo (nodos adyacentes o próximos en K pasos) y cada una agrega información (*features* y características topológicas como el grado) de cada nodo en la vecindad. De esta forma se construyen *embeddings* de vecindad para cada nodo que se concatenan con los *embeddings* de los nodos (que inicialmente son sus *features*) para finalmente entrenarlo por medio de una red neuronal y actualizar los *embeddings* de ser necesario.

2.9.5. R-GCN

Las redes convolucionales basadas en grafos relacionales (R-GCN) propuestas por Schlichtkrull et al. (2018) son una adaptación y extensión de las GCN que resuelven varios de los inconvenientes que mencionamos en la Sección 2.9.3. Las R-GCN son capaces de trabajar con multigrafos en donde los nodos, también llamados entidades, pueden conectarse entre sí por medio de aristas de distinto tipo, también llamadas relaciones. De esta forma es posible generar representaciones vectoriales o *embeddings* de nodos en grafos, dirigidos o no dirigidos, dando distinto valor a las aristas de los nodos según el tipo de relación que representen. Una ventaja inmediata de este modelo respecto a las GCN, es que los *loops* necesarios en los nodos conforman un tipo especial de relación distinto al resto. Las R-GCN pueden ser muy útiles para aplicar en grafos de redes sociales en donde existan distintos tipos de entidades y relaciones entre ellas. Por ejemplo en Twitter, las entidades pueden ser los usuarios y las relaciones que los unen pueden ser de dos tipos distintos, relaciones seguidor-seguido y relaciones de menciones por medio de tweets.

Capítulo 3

Análisis exploratorio

En este capítulo describiremos los conjuntos de datos utilizados y exploraremos algunas de sus características. Haremos un análisis sobre los perfiles de los usuarios con el objetivo de verificar qué tan fiable puede ser dicha información para determinar la residencia. Explicaremos cómo conformamos las redes que modelan las relaciones entre nuestros usuarios y sus respectivas ventajas y desventajas. Por último, detallaremos nuestro procedimiento para la detección de términos locales, y la conformación de secuencias para métodos basados en redes neuronales.

3.1. Datos disponibles

Para este trabajo se utiliza un conjunto de datos de tweets y usuarios capturados durante la campaña electoral del 2019 en Argentina descripto en Reyero et al. (2021). En total disponemos de aproximadamente 900 millones de tweets de los cuales 1 millón se encuentran geolocalizados con coordenadas específicas y 14 millones poseen un *bounding box* indicando una ciudad especificada por los mismos usuarios al momento de realizarse el tweet. Estos últimos tweets no garantizan ser tan certeros como los geolocalizados, ya que los usuarios pueden indicar la ciudad

que ellos deseen desde la propia interfaz de usuario de Twitter, pero decidimos utilizarlos para tener una mayor base de usuarios para poder trabajar y porque, al día de hoy, no existen trabajos que aprovechen este tipo de tweets que son más abundantes. Si bien la ubicación de estos puede hacer referencia a una ubicación distinta a la de permanencia del usuario, nos proponemos analizar y verificar qué tan fiable puede llegar a ser esta información para que, a futuro y si es posible, más trabajos puedan utilizar estos datos sin tener que recurrir a la escasa cantidad de tweets geolocalizados. Además, como fue especificado en la Sección 1.2, también hemos obtenido el conjunto de datos Twitter-US (Roller et al. (2012)) sobre el que se han realizado diversas pruebas en otros trabajos de la literatura. También aplicaremos nuestro método sobre este conjunto de datos para comparar los resultados con el estado del arte.

3.2. Análisis previo

Comenzando por los 14 millones de tweets que contienen una ciudad especificada por los usuarios y determinada por medio de un *bounding box*, procedimos a validar y unificar dichas ciudades utilizando el *gazetteer* GeoNames[1]. Por medio de GeoNames, podemos no solo validar los nombres de las ciudades sino también unificarlas bajo un par latitud/longitud más certero. Adicionalmente GeoNames nos aporta información extra como el tamaño de la población, la cual nos servirá más adelante para entender los problemas de distribución de los tweets. En total, de los 14 millones de tweets, 12 millones poseen ciudades con nombres válidos según GeoNames, con lo que de ahora en adelante utilizaremos estos tweets cuando nos refiramos a tweets con ubicación especificada por los usuarios. Con respecto a los tweets con geolocalización exacta, se realizaron pruebas usando las mismas ciuda-

[1] https://www.geonames.org/

des determinadas por Twitter sin validar con GeoNames, dado que esto elimina bastantes muestras que de por sí son escasas. También se probaron otras formas de distribuir las muestras basándose en los pares latitud/longitud de los tweets.
Disponemos de aproximadamente 2 millones de usuarios, de los cuales 1 millón especificaron una ubicación, sea verdadera o falsa, en su perfil. Para escoger sus respectivas ciudades de residencia con mayor fiabilidad, decidimos quedarnos con la ciudad en la que los usuarios publicaron la mayoría de sus tweets, dado que elegir la ubicación de sus primeros tweets puede ser confuso por el simple hecho de que solo poseemos tweets realizados durante el 2019, desconociendo todo lo ocurrido con anterioridad. Este enfoque también nos permitiría, viendo la distribución de sus tweets y de ser necesario, filtrar aquellos usuarios que podrían considerarse viajantes y que introduzcan ruido al entrenamiento del modelo que utilizaremos. Luego de determinar la ciudad de residencia de los usuarios, obtuvimos dos conjuntos de datos sobre los que avanzaremos: uno basado en tweets con geolocalización exacta, el cual llamaremos Twitter-ARG-Exact y otro con restricciones más relajadas, de ahora en adelante, Twitter-ARG-BBox, el cual consiste en tweets con ubicación especificada por los usuarios. En la Tabla 3.1 podemos ver un resumen de ambos conjuntos y sus respectivos tamaños. Puede notarse una tercera columna de tweets totales no mencionada hasta el momento; dichos tweets consisten en todos los realizados por los usuarios pertenecientes a cada conjunto, estén o no geolocalizados. Decidimos tener estos últimos en cuenta para ampliar la cantidad de información disponible por usuario.

Respecto a las ciudades abarcadas en nuestros conjuntos de datos y a la cantidad de muestras disponibles en cada una, disponemos de 229 ciudades con al menos 100 muestras distribuidas a lo largo de 22 países en Twitter-ARG-BBox, siendo en su mayoría usuarios residentes en Argentina. Podemos ver la distribución de usuarios por país en la Figura 3.1.

Conjuntos	#Usuarios	#Tweets con ubicación	#Tweets totales
Twitter-ARG-Exact	37.146	625.180	27.574.343
Twitter-ARG-Bbox	141.209	9.298.954	124.192.146

Tabla 3.1: Conjuntos de datos generados durante este trabajo.

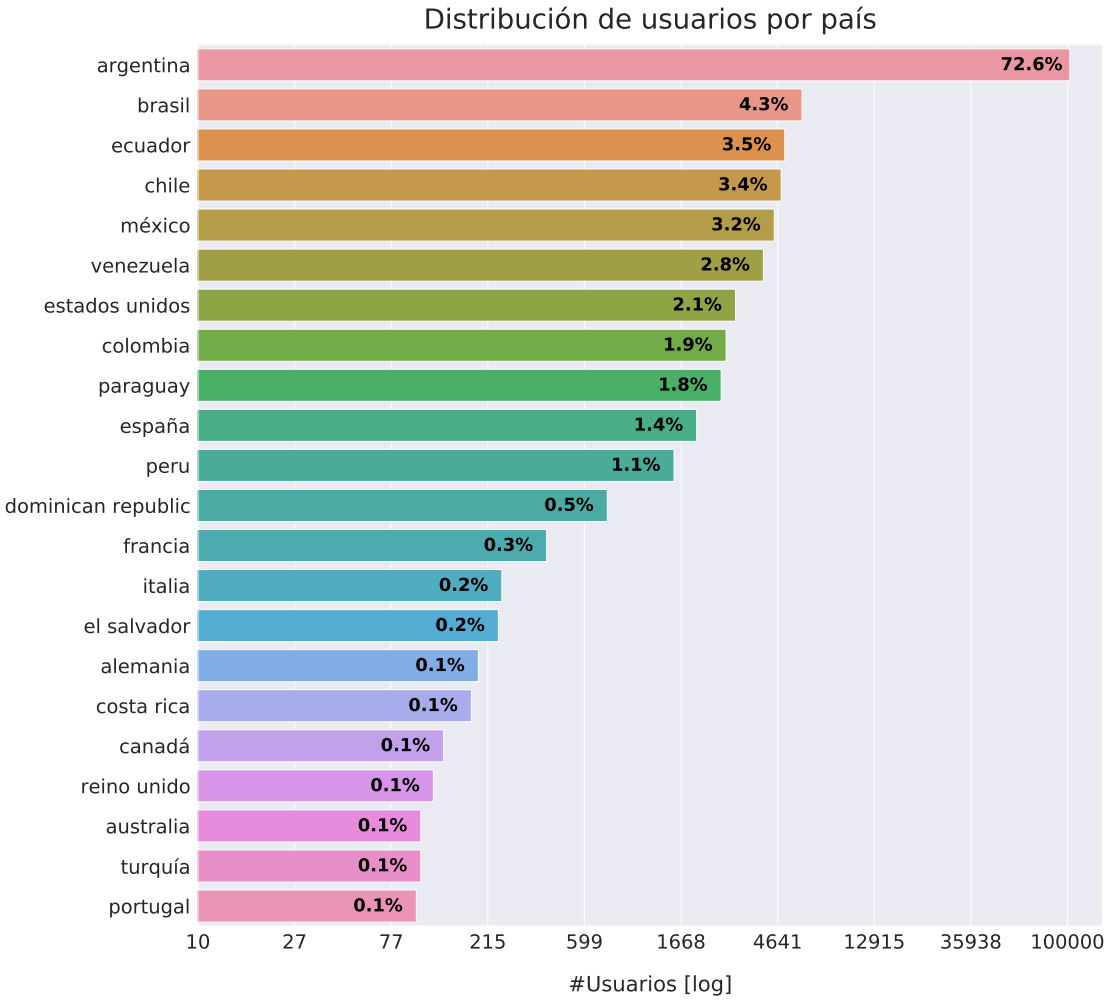


Figura 3.1: Distribución de usuarios en los países del conjunto Twitter-ARG-BBox.

En cambio, para el conjunto Twitter-ARG-Exact, disponemos de 95 ciudades con al menos 100 muestras distribuidas a lo largo de 14 países, tal como se ve puede ver en la Figura 3.2. Además podemos observar en la Figura 3.3 cómo se encuentran distribuidas las muestras a través del mundo.

Además de tener un gran desbalance con respecto a las muestras por país en nuestros conjunto de datos, esto mismo ocurre a nivel ciudad por cada país. Por ejemplo para Argentina la mayoría de las muestras corresponden a la Ciudad Autónoma de Buenos Aires, como se puede apreciar en la Figura 3.4.

Debemos aclarar que durante el resto del capítulo trabajaremos exclusivamente con el conjunto de datos Twitter-ARG-Exact, siendo el conjunto Twitter-ARG-BBox utilizado únicamente para realizar comparaciones en cuanto a los resultados y ver si es posible verificar que dicho conjunto es válido con respecto a las ubicaciones provistas por los usuarios.

Respetando la política de privacidad de Twitter con respecto a la información de los usuarios, publicamos únicamente los identificadores de los tweets en el repositorio de github: `https://github.com/fedefunes96/twitter-location-data`

3.3. Análisis de perfiles

Como bien mencionamos, gran parte de los usuarios deciden incluir su ubicación en sus perfiles. Si bien en la Sección 1.2 vimos que trabajos como el de Cheng et al. (2010) y Hecht et al. (2011) informan que un bajo porcentaje de usuarios incluían una ubicación real, nos interesa verificar qué tan verídica puede ser esta información para aquellos usuarios que disponen de una ubicación real en sus perfiles. Para ello, vamos a procesar cada una de las ubicaciones de los perfiles de nuestros usuarios para obtener una ubicación sobre la que hagan referencia a nivel ciudad y/o país. La Tabla 3.2 nos muestra ejemplos de perfiles de nuestros usuarios

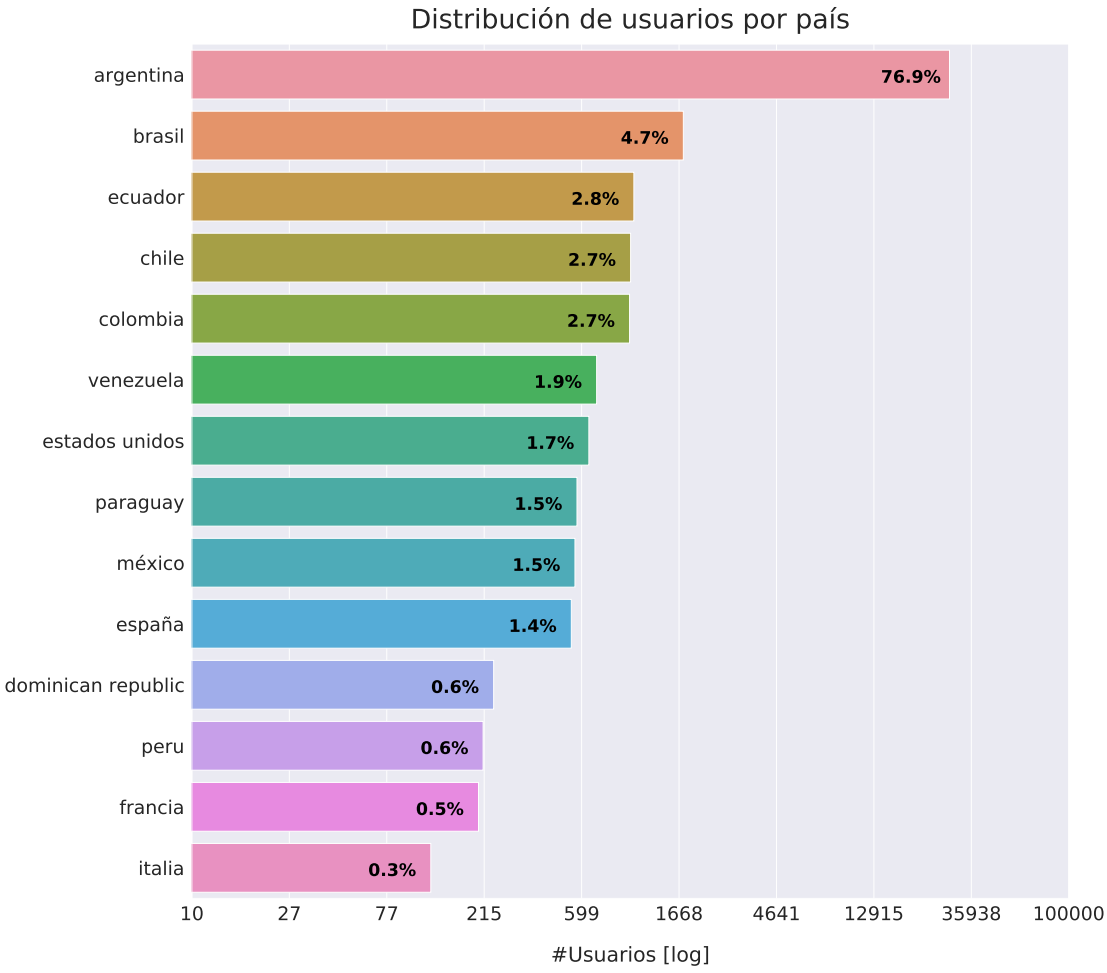


Figura 3.2: Distribución de usuarios en los países del conjunto Twitter-ARG-Exact.

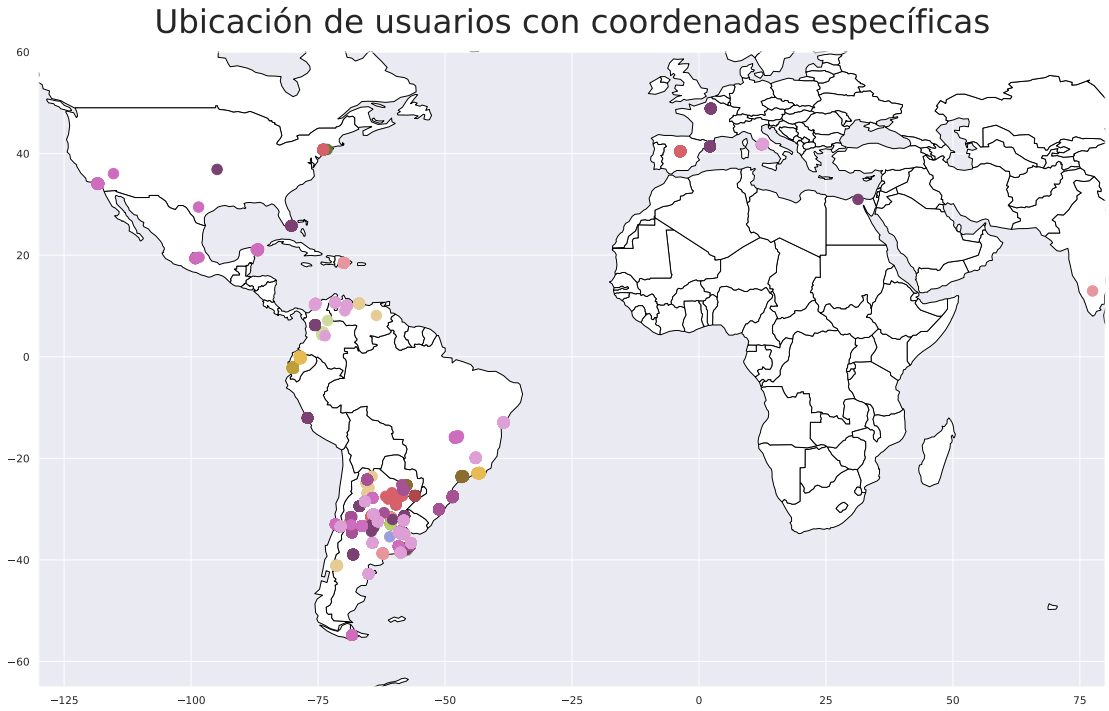


Figura 3.3: Distribución de usuarios a lo largo del mundo para el conjunto Twitter-ARG-Exact.

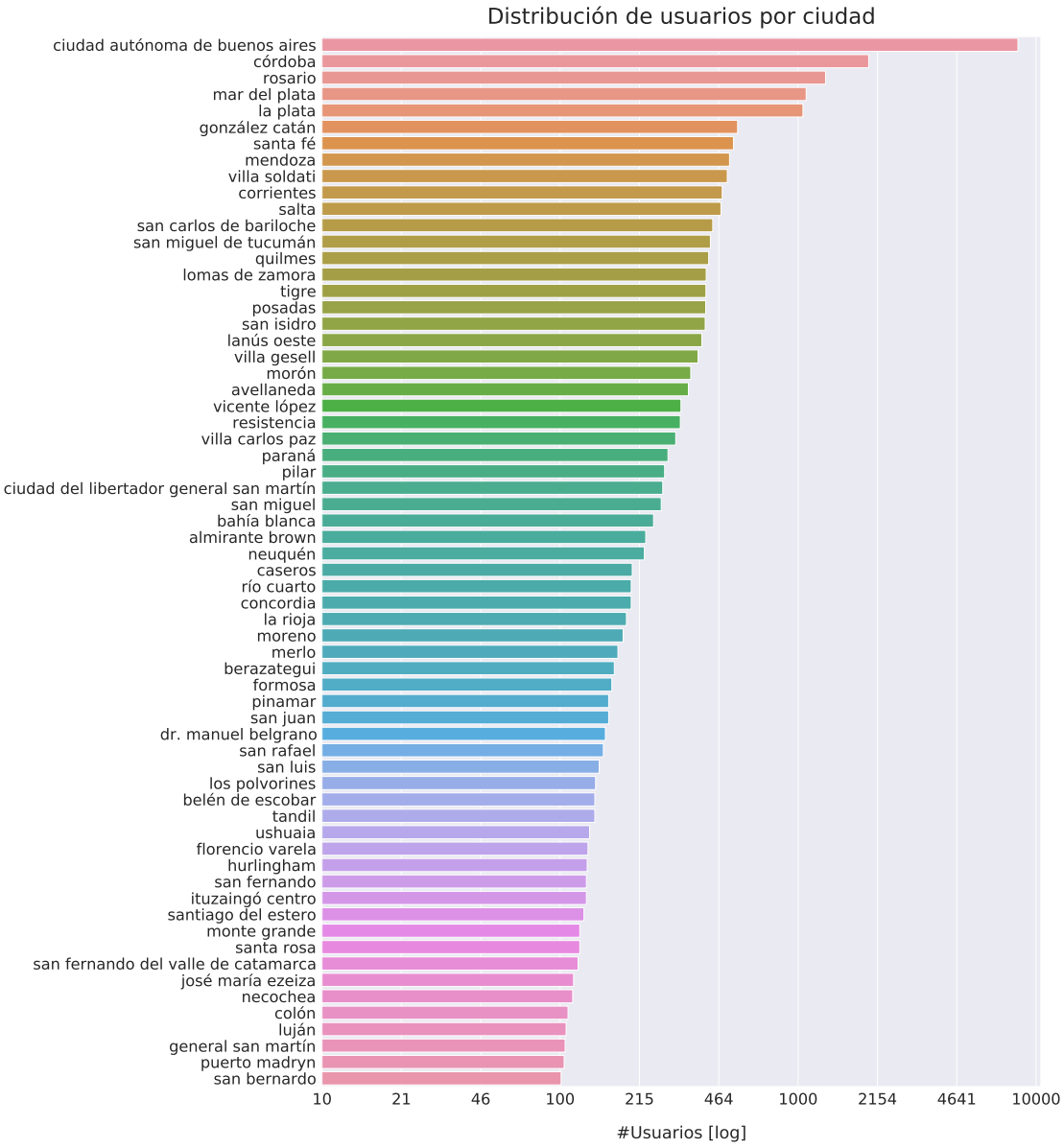


Figura 3.4: Distribución de usuarios en las ciudades de Argentina para el conjunto Twitter-ARG-Exact

Ubicación perfil	Ubicación real
Txalapa, veracruz	xalapa,méxico
ushuaia	ushuaia, argentina
buenos aires	mar del plata,argentina
santiago, chile	san carlos de bariloche,argentina
perez - santa fe - argentina	perez,argentina
buenos aires, argentina	ciudad autónoma de buenos aires,argentina
palermo hollywood	ciudad autónoma de buenos aires,argentina

Tabla 3.2: Ejemplos de perfiles de usuarios para Twitter-ARG-Exact.

y la ambigüedad con la que se manejan los nombres de las ubicaciones, así como también la dificultad al utilizar distintos niveles de granularidad de ubicación.

Nuestro proceso consiste en el pasaje a minúsculas de los perfiles y la separación en palabras por medio de caracteres especiales. Esto en el mejor caso nos dará como resultado dos palabras indicando la ciudad y el país respectivamente. En otros casos, hay que analizar cada palabra y determinar por medio de un *gazetteer*, en este caso, GeoNames, si con la totalidad de palabras disponibles se hace referencia a una única ciudad, a varias ciudades o simplemente a un país. La Figura 3.5 resume los resultados obtenidos para los usuarios con perfil disponible, donde se puede apreciar que en hasta el 64.5 % de los casos es posible obtener una ciudad a partir del perfil de estos usuarios.

Si analizamos la veracidad de los perfiles con ubicación exacta, en el 51 % la ciudad del perfil coincide con la de la mayoría de los tweets del usuario. De aquí podemos deducir que aproximadamente el 21 % de nuestros usuarios con perfil disponible tienen coincidencia entre la ciudad de sus perfiles y la seleccionada en la mayoría de sus tweets. ¿Qué ocurre con el 79 % restante? Podemos analizar la

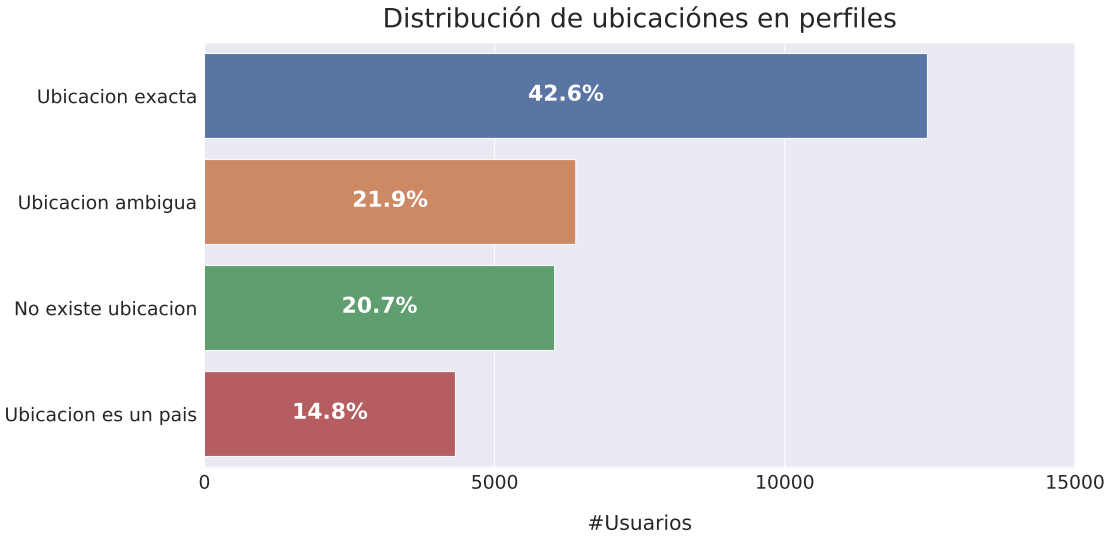


Figura 3.5: Distribución de tipos de ubicación de perfil de los usuarios de acuerdo a la ubicación indicada en la mayoría de sus tweets para Twitter-ARG-Exact. Considerando aquellas ciudades presentes en GeoNames.

distribución de distancias entre las ciudades nombradas en los perfiles y las obtenida sde la mayoría de los tweets para verificar que los porcentajes obtenidos sean similares, de forma tal que si fuese un problema de granularidad con respecto a la ubicación, por ejemplo, barrios contenidos en ciudades, debería darnos diferencias en distancia por debajo de los 10km[2]. Los resultados obtenidos se muestran en la Figura 3.6, de donde podemos apreciar que el 69 % de nuestros usuarios con ubicación exacta se encuentran, según sus perfiles, a menos de 10km de la ciudad en la que mayormente publican tweets. De esto podemos concluir que, para la totalidad de nuestros usuarios con perfil disponible, el 29 % puede ser localizado por medio de sus perfiles con un error de hasta 10km. ¿Y qué ocurre con los usuarios con ubicación ambigüa? De ellos pudimos determinar que el 45 % se encuentran a menos de 10km. Combinando estos resultados podríamos ubicar a aproximadamente el

[2]Consideramos que 10km puede llegar a ser una restricción muy relajada pero al disponer de muchas ciudades, los resultados no variarán mucho.

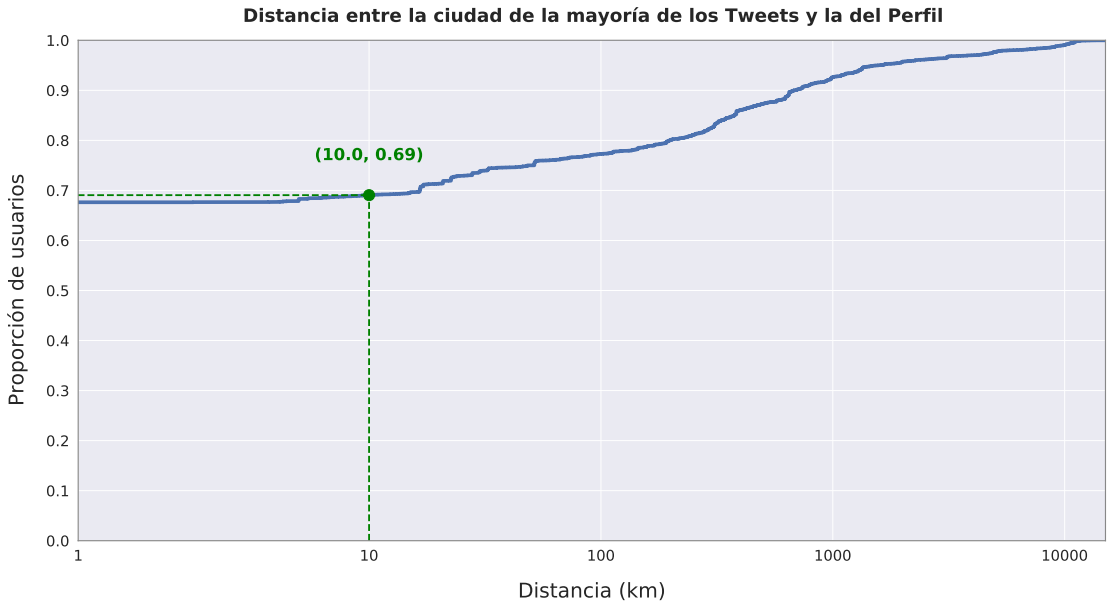


Figura 3.6: Distribución acumulada de distancias entre la ciudad nombrada en el perfil y la elegida por medio de la mayoría de los tweets de los usuarios con ubicación exacta.

39 % de nuestros usuarios con perfil disponible a menos de 10km. Estos resultados son consistentes con lo observado por Cuevas et al. (2014).

3.4. Análisis de redes

En esta sección trataremos sobre las redes conformadas por las relaciones entre nuestros usuarios, haciendo hincapié en la teoría de grafos. Caracterizaremos las redes y explicaremos cómo transformarlas en valores aptos para nuestros modelos de entrenamiento en conjunto con las dificultades que incurren al trabajar con estos datos. Al final de la sección daremos un breve resumen de los conjuntos de datos conformados que se utilizarán más adelante. Para la manipulación de las redes tanto de seguidores como de menciones utilizamos NetworkX de Hagberg

et al. (2008), una biblioteca de Python para la manipulación de grafos.

3.4.1. Red de menciones

En Twitter los usuarios pueden realizar tweets nombrando a otros usuarios para que a estos les llegue una notificación y puedan estar atentos al contenido del publicador, esta característica es llamada mención de usuarios y se pueden identificar en los tweets como un arroba seguido del nombre del usuario (@usuario). Poseemos aproximadamente 30 millones de menciones realizadas por nuestros usuarios sobre las cuales estaremos trabajando en esta sección.

Existen diversas formas de trabajar con las menciones. Por ejemplo, en forma matricial, llamando A a dicha matriz de menciones de tamaño $m \times m$ ($m = \#Usuarios\ totales$) en donde cada elemento $A_{ij} = v$ representa que el usuario i menciona v veces al usuario j. Sobre esta matriz A podemos realizar varios procedimientos: Uno de los procedimientos de nuestro interés es analizar qué usuarios deben formar parte de dicha matriz de menciones. Por ejemplo, si consideramos a la matriz de menciones como un grafo, sabemos que la distribución de los grados de sus vértices conforman una ley de potencias tal como se mostraba en la Figura 2.2. Es decir, existen muchos usuarios con pocas menciones incidentes y pocos usuarios con muchas menciones incidentes. La pregunta entonces es qué usuarios son los que más influyen para determinar la ubicación de los usuarios de nuestro conjunto de datos. Si tomamos aquellos usuarios cercanos con pocas menciones incidentes seguramente nos den información más específica que si tomásemos usuarios populares. En otras palabras, el presidente de Argentina tiene muchas menciones incidentes pero estas poco nos dicen sobre la ubicación de los usuarios que lo mencionan (bien podrían ser de cualquier ciudad de Argentina). Es difícil considerar hasta qué punto un usuario es popular: por ejemplo, las radios locales de una ciudad pueden ser muy influyentes sobre la ubicación de un usuario a pesar de tener

muchas menciones, con lo que se debe ser cautelosos con los filtros de la red a emplear. ¿Y por qué no mantener a todos los usuarios de la red? Si bien es posible mantenerlos, puede llegar a ser costoso por el hecho de que justamente por ser populares, agregan muchísimas aristas a nuestro grafo y enlentecen nuestros métodos (también, métodos que propondremos más adelante requieren el filtro de estos usuarios populares). Otro procedimiento posible es trabajar con la matriz de menciones completar y aplicar algoritmos de regresión lineal o que utilicen árboles de decisión. Estos últimos funcionarían bien para usuarios medianamente populares pero no captarían usuarios poco populares. Por otra parte estos métodos pueden ser bastante lentos por la cantidad de *features* con los que estamos trabajando. Entonces, deberemos trabajar con métodos de detección de comunidades (que son más eficientes y están preparados para este tipo de información) o métodos que nos permitan reducir dimensiones reteniendo la mayor cantidad de información posible (o al menos una parte significativa) sobre la red.

Hasta el momento estuvimos hablando de utilizar la matriz de menciones en cualquiera de sus posibles variantes, ya sea dirigida, no dirigida, pesada o no pesada entre otras. Sin embargo, debemos notar que utilizar la matriz de menciones directamente no es lo ideal, porque los usuarios de la red mencionan usuarios por fuera del conjunto de datos que poseemos, lo cuál agrega muchos nodos externos sobre los que no poseemos información más allá de las menciones. Entonces, una opción posible es compactar nuestra red de menciones de forma de solo trabajar con usuarios de nuestro conjunto de datos, tratando de preservar la mayor cantidad de información posible. Una forma de utilizar únicamente los nodos de nuestra red es cortar la red de menciones eliminando todos aquellos nodos por fuera de nuestro conjunto de datos. Si bien esto es factible, perderíamos muchísima información, ya que gran parte de las menciones de los usuarios son hacia usuarios externos. Entonces, lo que proponemos es construir una red de *comenciones*, representa-

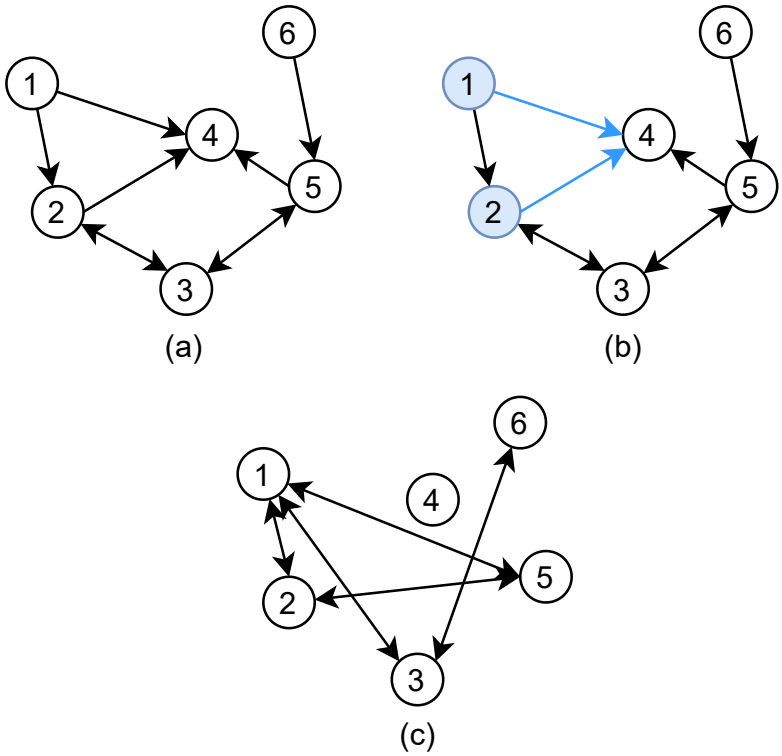


Figura 3.7: Obtención de la red de *comenciones* por medio de la red de menciones. (a) representa un grafo de menciones dirigida; en (b) podemos ver que los nodos 1 y 2 poseen una mención en común (*comención*); en (c) ya vemos la red de *comenciones*, sobre la cual podemos notar que todas las aristas son bidireccionales, con lo que de ahora en adelante consideraremos la matriz como no dirigida.

da por medio de la matriz de *comenciones* C, cuyo elemento $C_{ij} = v$ nos indica que existen v menciones en común entre los nodos i y j. Un ejemplo sobre cómo conformar esta red puede verse en la Figura 3.7.

Dado que los nodos externos a nuestro conjunto poseen únicamente aristas entrantes, la red de *comenciones* va a tener a dichos nodos aislados y va a conectar a nuestros nodos solo si poseen una mención en común, independientemente de si mencionan usuarios externos o de nuestro conjunto. Sin embargo, debemos notar dos inconvenientes: el primero es que alguno de nuestros nodos podría incurrir

en un camino más largo hacia un nodo cercano del que poseía con anterioridad (ver nodos 5 y 6 en la Figura 3.7, que dejan de tener una conexión directa en la red de *comenciones*); el segundo inconveniente está relacionado con la cantidad de aristas que existen en la red de *comenciones*. Consideremos que tenemos un nodo i de grado entrante n (esto quiere decir que n nodos mencionan al nodo i), entonces, cuando generemos la red de *comenciones*, dichos n nodos tendrán todos una arista en común entre ellos, siendo $\frac{(n-1)^2+(n-1)}{2}$ aristas en total. Para resolver el primer inconveniente poseemos dos soluciones: una consiste en generar lazos para cada nodo, con lo cual cualquier mención directa entre dos nodos se conservaría, mientras que otra solución sería combinar ambos enfoques, conservando la red de menciones para los usuarios de nuestra red y generando *comenciones* entre ellos utilizando únicamente usuarios externos a nuestro conjunto. Dicho de otra forma, uniendo a nuestros nodos entre sí si existe un camino entre ellos por medio de un nodo externo. A esta red la llamaremos "*red de menciones extendida*". Para reducir los efectos del segundo inconveniente, podemos eliminar a los nodos populares como bien mencionamos antes, ya que estos son los que mayor cantidad de aristas generarían y menos información sobre la ubicación del usuario aportarían.

Luego de presentar estos nuevos grafos de *comenciones* y de *menciones extendido*, daremos una breve indicación de como construirlos a partir del grafo de menciones. Para generar el grafo de *comenciones* podemos, partiendo de la matriz de adyacencia del grafo dirigido de menciones (sin lazos) M ordenada de tal forma que las primeras N filas y columnas correspondan a los usuarios de nuestro conjunto, sumar una unidad en la diagonal de forma de agregar lazos a cada nodo para mantener conexiones directas, y luego multiplicar la matriz resultante por su transpuesta. En resumen, calculando $C = (M + I) \cdot (M + I)^{\mathrm{T}}$ y quedándonos con las primeras N filas y columnas, resulta el grafo de *comenciones* de los usuarios de nuestro conjunto. En cambio, para generar el grafo de *menciones extendido*

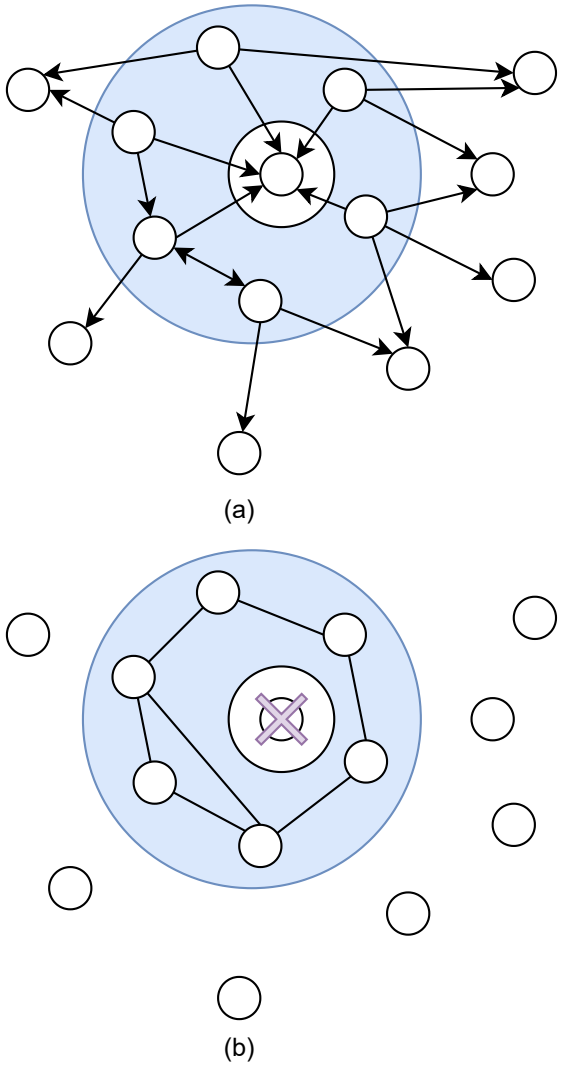


Figura 3.8: Ejemplo de red de menciones donde el conjunto azul representa usuarios pertenecientes a nuestro conjunto. En (a) observamos la red de menciones dirigida; en (b) la red de *comenciones* resultante al eliminar nodos populares

Grafo	#Nodos	#Aristas	Filtro nodos populares
Menciones completo	1.670.106	9.107.669	-
Menciones cortado	37.146	206.608	-
Comenciones	37.146	4.597.803	$>$ 20 menciones únicas
Menciones extendido	37.146	4.524.003	$>$ 20 menciones únicas

Tabla 3.3: Distintos tipos de grafos sobre las menciones y sus tamaños para Twitter-ARG-Exact.

partimos de la misma matriz M y una matriz correspondiente a un grafo bipartito $X_M \in \mathbf{1}^{n\times(m-n)}$ que representa las menciones realizadas por los usuarios de nuestro conjunto hacia usuarios externos. Conformamos la matriz de menciones extendida $F = (M + M^T) + (X_M \cdot X_M^T - diag(X_M \cdot X_M^T))$, en donde el término $(M + M^T)$ corresponde a un grafo de menciones no dirigido entre los usuarios de nuestro conjunto, y el término $(X_M \cdot X_M^T - diag(X_M \cdot X_M^T))$ a un grafo de *comenciones* con respecto a usuarios externos. La Tabla 3.3 resume los tamaños de cada uno de los grafos obtenidos para trabajar con las menciones.

3.4.2. Red de seguidores

A diferencia de las menciones, obtener el conjunto de seguidos (*followees*) y seguidores (*followers*) de los usuarios es más costoso por los tiempos entre peticiones a la API de Twitter. Debido a esto, existen muy pocos trabajos que aprovechen esta información, que creemos puede ser un determinante clave para detectar la ubicación de los usuarios. Debemos notar que para esta red, nuestros usuarios ahora poseen aristas entrantes de nodos por fuera de nuestro conjunto de datos, y, como en el caso de la red de menciones, debemos acotar la red para poder trabajar con diversos métodos. Por esto seguiremos los mismos conceptos para

Grafo	#Nodos	#Aristas	Filtro nodos populares
Seguidores completo	686.694	5.903.440	-
Seguidores cortado	37.146	164.376	-
Coseguidores	37.146	18.882.006	$>$ 20 seguidores únicos
Seguidores extendido	37.146	1.441.746	$>$ 20 seguidores únicos

Tabla 3.4: Distintos tipos de grafos sobre los seguidores y sus tamaños para Twitter-ARG-Exact.

reducir la cantidad de nodos de nuestra red de seguidores, es decir, conformar redes de *coseguidores* y de *seguidores extendido*. Debemos tener especial cuidado sobre cómo conformar dichas redes a partir del grafo de seguidores: para el grafo de *coseguidores*, hay que tener en cuenta el hecho de que manejar tanto seguidos como seguidores nos indica que poseemos muchas más aristas entre usuarios de nuestro conjunto. Entonces, al conformar el grafo de *coseguidores*, incurriríamos en una masiva cantidad de aristas entre nuestros usuarios, por lo que aún poniendo un filtro más estricto sobre la cantidad máxima de seguidores únicos por nodo el problema persistiría. La Tabla 3.4 resume los grafos obtenidos para los seguidores y seguidos de nuestros usuarios; hacemos notar que trabajaremos con el grafo de *coseguidores* dada su inmensa cantidad de aristas.

3.5. Análisis de contenido

Analizar el contenido de los tweets de nuestros usuarios es una tarea fundamental para determinar sus correspondientes ciudades de residencia. Este es un trabajo que recae dentro del campo conocido como Procesamiento del Lenguaje Natural (NLP), el cual es bastante amplio y complejo, y que nos brinda muchísi-

Tweet original	Tweet procesado	Hashtags
At work #trabajo #toys #juguetes #jugueteria #buenosaires en Ciudad Autónoma de Buenos Aires https://t.co/gud6uoILmQ	at work en ciudad autónoma de buenos aires	trabajo, toys, juguetes, jugueteria, buenosaires
Feliz Aniversario ♡ 23 años #bodasdeagua que sigamos navegando juntos en Cartagena, Colombia https://t.co/8TWfnP3gj4	feliz aniversario 23 años que sigamos navegando juntos en cartagena colombia	bodasdeagua

Tabla 3.5: Ejemplos de tweets procesados y hashtags extraídos.

mas herramientas para trabajar con estos datos. En esta sección haremos hincapié en aquellos métodos que utilizamos, justificando debidamente el por qué, y sus correspondientes ventajas y desventajas.

Antes de poder analizar el contenido de los tweets de nuestros usuarios, debemos procesarlos de forma tal que podamos eliminar términos redundantes, unificar términos similares y reducir la cantidad de información a procesar. Nuestro enfoque consiste en primero pasar todos los tweets a minúsculas, eliminando las menciones (ya que estas son analizadas por medio de otros enfoques desarrollados secciones anteriores), extraer los *hashtags* utilizados por al menos 10 usuarios distintos, que se identifican por medio del carácter # , y finalmente eliminar símbolos, emojis, saltos de línea y direcciones a páginas web. De esta forma obtenemos un conjunto de tweets filtrados y hashtags por separado tal como se ve en la Tabla 3.5.

3.5.1. Búsqueda de términos locales

Hoy en día las personas suelen publicar contenido en redes sociales utilizando modismos, términos populares del momento o hasta abreviaciones de lugares. Sabemos por trabajos como los de Cheng et al. (2010), Han et al. (2012), Han et al. (2012) y Mahmud et al. (2014) que dentro de estos términos podemos encontrar algunos que se encuentren asociados a una zona geográfica en específico. Por ejemplo, es muy probable que la gente de Almirante Brown hable del Boulevard Shopping siendo que se encuentra dentro de la zonal Llamaremos a estos términos *Local Indicative Words* (LIW) tal como se hizo en los trabajos previamente mencionados. Encontrar LIW puede ser una tarea complicada y costosa por la enorme cantidad de términos que se manejan, sin embargo, hoy en día existe una gran variedad de métodos para identificarlos. En esta sección haremos hincapié en algunos métodos que utilizamos, con sus correspondientes ventajas y desventajas. Para nuestros ensayos, consideramos un término como cualquier unigrama, bigrama o trigrama con al menos 10 usos por distintos usuarios y un máximo de uso de hasta el 20 % de los usuarios, de forma de filtrar palabras comúnmente usadas y que poco aporten a la búsqueda de LIW. También consideramos que los *hashtags* podrían ser un fuerte indicativo de localidad, con lo que los tendremos en cuenta a la hora de seleccionar los más significativos.

Nuestro primer enfoque consiste en el uso de métodos estadísticos para realizar pruebas y determinar, con un cierto nivel de confianza, si ciertos eventos ocurren de forma no aleatoria. En particular, vamos a usar la distribución de Pearson χ^2 tal como se hizo en trabajos como el de Han et al. (2014). Como primer paso debemos definir las hipótesis a verificar y a nuestros eventos: los eventos con los que trataremos consisten en si un término es local a una ciudad, definiendo como hipótesis nula la independencia entre el término y la ciudad, y la dependencia como la hipótesis alternativa. Como segundo paso, construimos una tabla de contingencia

tal como se ve en la Tabla 3.6. Como tercer paso calculamos la frecuencia esperada para cada caso como $E_{i,j} = P(i) \cdot P(j) \cdot M$, siendo $M = \#Observaciones$. Finalmente calculamos $\chi^2 = \sum_{i,j}^{n} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}$ con un solo grado de libertad en nuestro caso y buscamos su valor en la tabla de valores de χ^2 según el nivel de confianza que deseamos tener. A partir de este momento podemos tomar varios enfoques para trabajar con los resultados obtenidos de χ^2 para cada término y ciudad, por ejemplo, quedarnos con los N términos con mayor valor para cada ciudad, o quedarnos con aquellos términos que para una ciudad en específico, estén dentro del nivel de confianza que planteemos. Si decidimos quedarnos con los K términos más significativos podemos ver cuáles serían esos términos para cada ciudad. En la tabla 3.7 podemos ver resumen de términos más significativos encontrados para algunas ciudades, en donde observamos que la mayoría de dichos términos son el nombre de la ciudad en sí. Nuestro segundo enfoque para encontrar LIW consiste en usar la información mutua para determinar si la presencia o ausencia de un término ayuda a clasificar correctamente a una ciudad; una vez calculado el valor para cada término y ciudad, nos quedamos con los K términos más significativos por ciudad igual que como hicimos con las pruebas χ^2. Este tipo de métodos de búsqueda de términos locales suelen tener varias desventajas: la primera es que son incapaces de captar relaciones entre términos, con lo que gran parte de la información sobre cómo hablan las personas en distintas ciudades no puede ser capturada, y la segunda desventaja es que si consideramos el uso de unigramas, bigramas y trigramas, la cantidad de términos a evaluar se vuelve excesivamente grande a medida que incorporamos más muestras.

3.5.2. Análisis de secuencias

En vez de enfocarnos en la búsqueda de términos locales podríamos analizar los tweets de los usuarios como una secuencia y entrenar un algoritmo que, además de

	Ciudad	Otra ciudad
Término	$O_{\mathrm{t,c}}$	$O_{\mathrm{t,oc}}$
Otro término	$O_{\mathrm{ot,c}}$	$O_{\mathrm{ot,oc}}$

Tabla 3.6: Ejemplo de tabla de contingencia para el cálculo de χ^2 para un término. La totalidad de muestras $M = \sum_{i,j}^{n} O_{\mathrm{i,j}}$.

determinar términos que puedan ser locales a una ciudad, sea capaz de determinar relaciones entre términos. De esta forma se tendría en cuenta el cómo escriben los usuarios para determinar dónde viven. Dado que los métodos para entrenar sobre secuencias de texto, como las LSTM o los Transformadores, requieren que las secuencias de entrada sean de longitud fija, debemos de acotar el contenido de los tweets de los usuarios a una longitud razonable para captar la mayor cantidad de información posible, manteniendo a su vez un tiempo de entrenamiento no tan elevado (recordemos que estos métodos tienen mayor complejidad que un clasificador Naive Bayes). Analizando la distribución de la longitud de los tweets concatenados de nuestros usuarios en la Figura 3.9, y sabiendo que gran parte de los usuarios utilizan muchas palabras repetidas veces, consideramos tomar una longitud que se encuentre por encima de la mediana y sea menor a 500 palabras. Finalmente representamos los tweets concatenados de los usuarios como una secuencia fija de las primeras N palabras utilizadas representada cada una con un número único, y para aquellas secuencias con menos de N palabras, completamos las mismas agregando un simbolo especial en su comienzo.

Ciudad	Términos significativos
Almirante Brown, Argentina	'burzaco', 'burzaco en', 'foto en almirante', 'almirante', 'en claypole', 'en jose marmol', 'rafael calzada'
Villa Soldati, Argentina	'villa soldati', 'bidegain nuevo gasometro', 'gasometro club', 'en villa soldati', 'villa soldati distrito', 'polideportivo roberto', 'champagne', 'pedro bidegain'
Villa Gesell, Argentina	'en gesell', 'villa gesell con', 'villa gesell 2019', 'en carilo', 'villa gesell hoy', 'dixit', 'pueblo limite', 'le brique oficial', 'foto en villa'
Vicente Lopez, Argentina	'tecnopolis buenos aires', 'en tecnopolis', 'vte lopez', 'gral manuel belgrano', 'en florida vicente', 'ensaladita', 'vicente lopez buenos', 'vicente lopez'
Santa Fé, Argentina	'santa fe mi', 'en esparanza santa', 'estanislao', 'norte salta', 'estanislao lopez', 'estadio brigadier general', 'santa fe no', 'santa fe acaba', 'en santa fe'

Tabla 3.7: Ejemplo de términos más significativos según pruebas χ^2 para algunas ciudades para el conjunto Twitter-ARG-Exact.

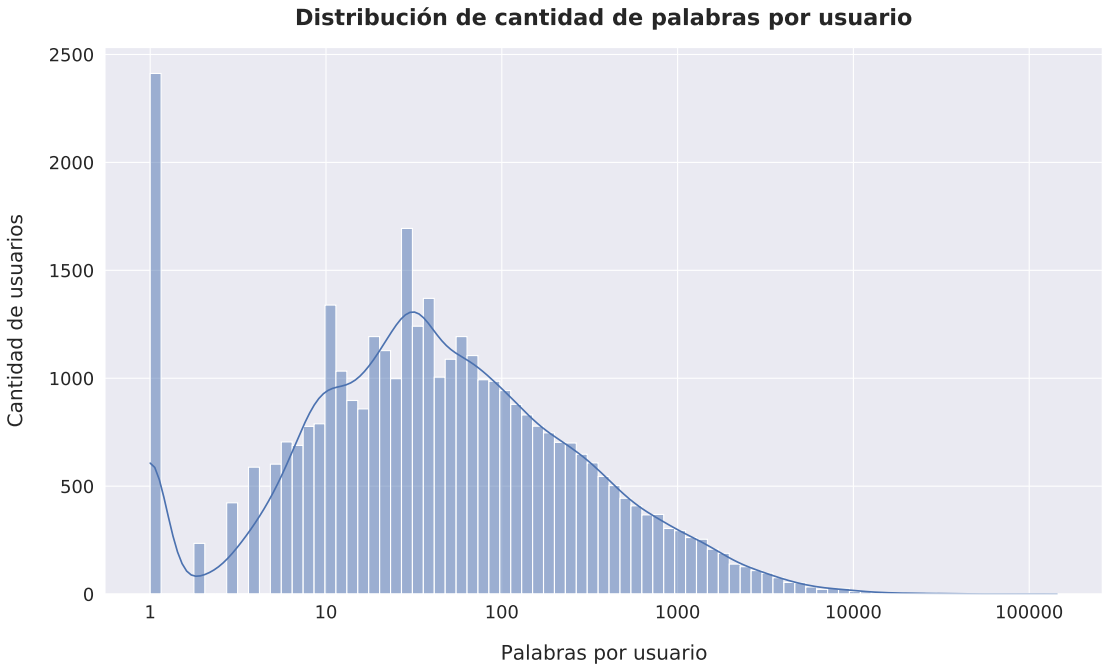


Figura 3.9: Distribución de cantidad de palabras utilizadas por los usuarios del conjunto Twitter-ARG-Exact.

Capítulo 4

Desarrollo

En el presente capítulo se mostrarán los resultados obtenidos con los métodos y las métricas que utilizamos sobre los conjuntos Twitter-ARG-Exact y Twitter-ARG-BBox. También se mostrarán los resultados en el conjunto de datos Twitter-US para comparar con el estado del arte.

4.1. Objetivo a clasificar

Uno de los principales inconvenientes ya mencionado con anterioridad es el desbalance de muestras a nivel ciudad. Afortunadamente existen múltiples formas de minimizar esta problemática, y en este trabajo vamos a nombrar los métodos utilizados y cuál nos dió los mejores resultados. Uno de los enfoques que utilizamos es balancear el conjunto de datos generando y eliminando muestras de forma aleatoria a través de *SMOTE* Chawla et al. (2002) de forma que mantengamos un balance a nivel ciudad según cada país. Otro enfoque comúnmente utilizado es dividir al espacio por medio de *KD-Trees* Roller et al. (2012), generando regiones en donde la cantidad de muestras es similar, esto trae como ventaja que ciudades con muy baja cantidad de muestras pueden juntarse dentro de una misma región

mejorando los resultados a simple vista. Sin embargo, presenta dos inconvenientes: el primero es que ciudades con una gran cantidad de muestras con respecto al resto se dividirán en regiones, provocando que los resultados probablemente empeoren por ir de un nivel ciudad a uno similar a barrios; y el segundo es que la API de Twitter nos ofrece coordenadas para los tweets geolocalizados con una precisión de decimales tal que puede ocurrir (y ocurre) que varios usuarios se encuentren localizados bajo las mismas coordenadas. En estas situaciones el uso de *KD-Trees*, según el valor mínimo de muestras que se tome por región, puede no generar regiones balanceadas. En este trabajo consideramos utilizar regiones de como mínimo 255, 500 o hasta 1255 muestras para generar distintas resoluciones de regiones posibles, sin embargo, las regiones no se encontraron balanceadas para los dos primeros casos. El último enfoque que consideramos es ajustar pesos en la función de costo de algunos métodos que utilizamos, para darles más valor a las ciudades con menor cantidad de muestras. De todos los enfoques para lidiar con el desbalance de las ciudades en nuestras muestras, el que mejor resultados nos garantizó es el ajuste de pesos por ciudad.

4.2. Modelo de evaluación

En este trabajo definimos un modelo de evaluación para comparar varios métodos que planteamos, aprovechando todas las muestras de usuarios que tenemos que, como bien sabemos, no son tantas como las de otros conjuntos de datos utilizados en trabajos de esta índole. Nuestro modelo puede ser descripto de la siguiente forma:

1. Comenzamos con N clasificadores $c_1, c_2, ..., c_{\mathrm{N}}$ que utilizan un subconjunto de los *features* de los usuarios para entrenar. Por ejemplo, un clasificador puede intentar determinar la ubicación de los usuarios utilizando únicamente la red

de menciones.

2. Usando validación cruzada anidada por medio de *K-Folds*, en donde cada *Fold* mantenga la proporción de muestras por ciudad (esto se conoce como *Stratified K-Folds*), primero generamos 5-Folds externos de nuestro conjunto de datos: cuatro de estos conjuntos se utilizarán para entrenar y sobre el restante se generarán probabilidades de pertenencia a cada clase por cada clasificador.

3. Los cuatro conjuntos para entrenar son divididos a su vez en 3-Folds internos de forma de encontrar los mejores hiper-parámetros para cada clasificador en cada una de estas instancias, y con el mejor clasificador obtenido, se entrena sobre los cuatro conjuntos de partida.

4. Se repiten los pasos (1) a (3) hasta iterar sobre todos los 5-Folds externos. Al final de este paso obtenemos probabilidades de pertenencia a cada clase según cada clasificador.

5. Utilizamos las probabilidades obtenidas por cada clasificador como una nueva representación de los *features* para entrenar un meta-clasificador, también por medio de validación cruzada anidada, que genere las clasificaciones finales.

En la Figura 4.1 podemos ver un resumen de cómo funciona nuestro modelo de evaluación. Debemos notar que este modelo es una combinación de validación cruzada anidada con un apilado de clasificadores (*Stacking-Classifier*) que tiene como ventaja que nos permite asegurar una cota mínima de resultados, según las métricas, para cada método planteado. Como desventaja debemos notar que este método requiere varios entrenamientos por clasificador (15 para cada clasificador/meta-clasificador considerando la búsqueda de hiper-parámetros) y esto puede llegar a

ser costoso en cuanto a tiempo de ejecución. Sin embargo, como no poseemos muchos datos y utilizamos métodos que en general son bastante rápidos como Naive Bayes, no resulta un inconveniente. La otra desventaja es que la combinación de los *features* se hace por medio de ajuste de pesos a través de un meta-clasificador y esto no nos permite captar la relación entre los distintos *features* de manera no lineal. Sin embargo, para solventar esto podemos aplicar clasificadores que operen sobre todo el conjunto de *features*, como haremos al utilizar redes neuronales.

4.3. Implementación

Presentaremos a continuación distintos clasificadores que serán utilizados para un conjunto de *features* en particular siguiendo nuestro modelo de evaluación. Comenzando por los clasificadores que utilizan solamente el contenido de los usuarios, proponemos los siguientes modelos:

- `MNB-BOW-TEXT`: Filtramos para quedarnos con los términos utilizados al menos 10 veces, considerando unigramas, bigramas y trigramas, y como máximo por el 20 % de los usuarios de forma de eliminar *stop-words* y términos poco recurrentes. Luego los representamos por medio de *bag of words* y entrenamos utilizando Multinomial Naive Bayes.

- `MNB-BOW-`χ^2`-TEXT`: Mismo filtro y representación que `MNB-BOX-TEXT` pero quedándonos con los top K términos más representativos de cada ciudad según pruebas de χ^2, y entrenando con Multinomial Naive Bayes sobre estos términos.

- `LR-BOW-`χ^2`-TEXT`: Similar a `MNB-BOW-`χ^2`-TEXT` con la diferencia que entrenamos con un regresor logístico.

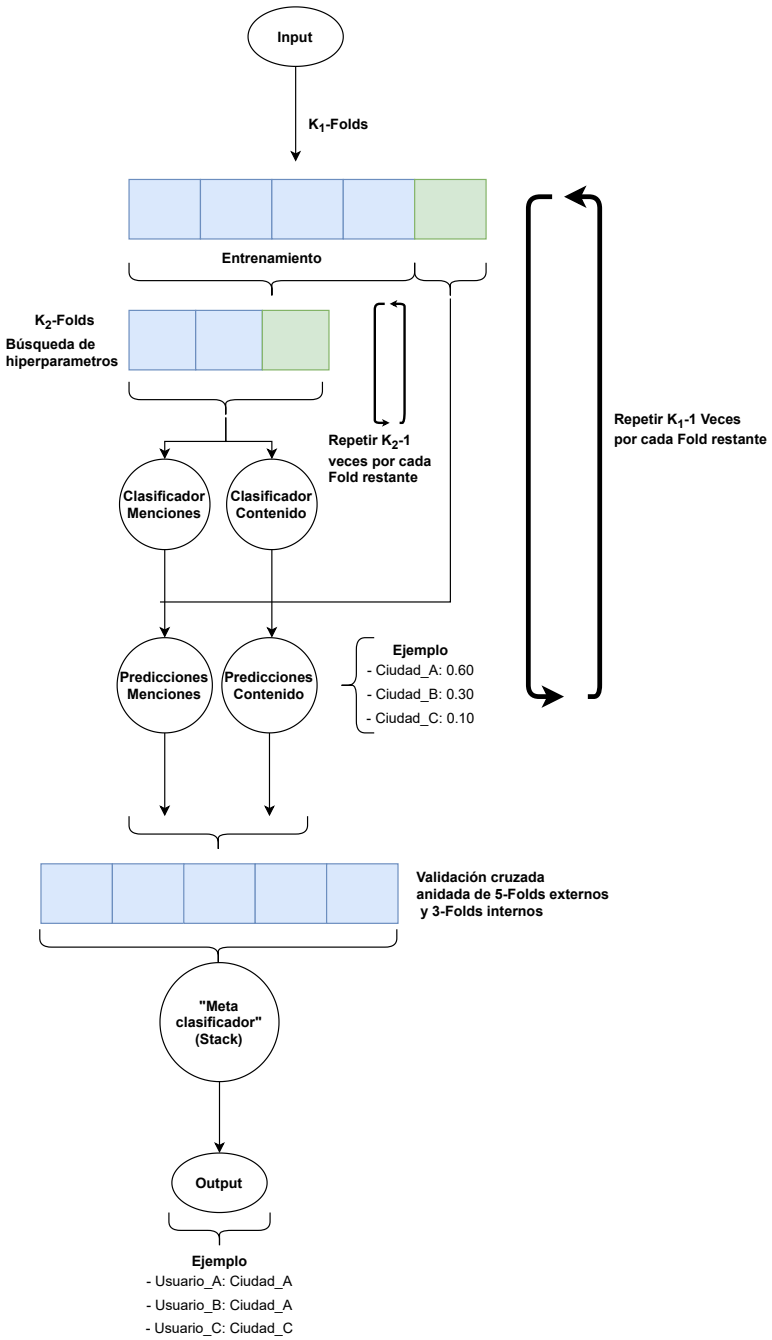


Figura 4.1: Ejemplo de nuestro modelo de evaluación para dos clasificadores apilados (stack) que operan con el contenido de los tweets y la red de menciones respectivamente.

- `MNB-BOW-MI-TEXT`: Igual que `MNB-BOX-`χ^2 pero seleccionando los mejores términos según la información mutua.

- `BiLSTM-TEXT`: Conformamos una red neuronal con una primera capa de representación vectorial (*embedding*) en bajas dimensiones para las 10.000 palabras más utilizadas, luego utilizamos una capa LSTM bidireccional, y finalmente una capa densa para determinar la probabilidad de pertenencia a cada ciudad. Por cada usuario consideramos las primeras 256 palabras utilizadas dentro de las 10.000 palabras más utilizadas del conjunto de datos.

- `BiLSTM-W2V-TEXT`: Igual que `BiLSTM-TEXT` con la diferencia de que pre-entrenamos *embeddings* de términos (considerando unigramas, bigramas y trigramas) usando Word2vec.

- `TF-W2V-TEXT`: Conformamos una red neuronal con un codificador de Transformador entrenando tanto *embeddings* posicionales como *embeddings* de las 10.000 palabras más utilizadas. Aplicamos *embeddings* pre-entrenados por medio de Word2vec como pesos iniciales. Consideramos utilizar una dimensión de 32 tanto para las capas de *embeddings* como para la FFN (red neuronal prealimentada).

Adicionalmente proponemos un conjunto de métodos que operan solamente con las redes conformadas por los usuarios por medio de menciones o seguidores:

- `MNB-NET`: Entrenamos un clasificador Multinomial Naive Bayes sobre la matriz de adyacencia del grafo de menciones cortado o el grafo de seguidores cortado (3.3).

- `Infomap-NET`: Buscamos comunidades sobre el grafo multicapa de menciones extendido con seguidores extendido usando Infomap. Luego entrenamos un clasificador Multinomial Naive Bayes sobre las comunidades encontradas.

- `OSLOM-NET`: Buscamos comunidades sobre el grafo de menciones extendido o el grafo de seguidores extendido usando OSLOM. Luego entrenamos un clasificador Multinomial Naive Bayes sobre las comunidades encontradas.

- `N2V-LR-NET`: Obtenemos representaciones vectoriales en bajas dimensiones de los nodos del grafo de menciones extendido o seguidores extendido. Luego entrenamos con regresión logística sobre los vectores resultantes. Utilizamos PecanPy de Liu and Krishnan (2020) como implementación de Node2vec que nos permite trabajar sobre grafos pesados.

Finalmente proponemos métodos que operan tanto con las redes conformadas por las relaciones de nuestros usuarios como con el contenido de sus tweets:

- `GCN-EXT-Preds`: A diferencia de `GCN-EXT-`χ^2, utilizamos como *features* de los usuarios las probabilidades de pertenencia a cada ciudad según el método que mejor resultado nos dio para análisis de contenido, que resultó ser `LR-BOW-`χ^2`-TEXT`.

- `GraphSAGE-EXT-Preds`: Igual a `GCN-EXT-Preds` con la diferencia de que usamos GraphSage como modelo en vez una GCN.

- `RGCN-EXT-Preds`: Aplicamos una R-GCN sobre el grafo de menciones extendido y seguidores extendido, usando como *features* las probabilidades de pertenencia a cada ciudad según el mejor método de análisis de contenido, igual que en `GCN-EXT-Preds`. La Figura 4.2 nos ilustra la arquitectura de este modelo.

- `RGCN-ML-Preds`: Igual que `RGCN-EXT-Preds` pero utilizamos variantes de grafos multicapa: grafo multicapa de menciones locales y co-menciones externas, grafo multicapa de menciones y seguidores.

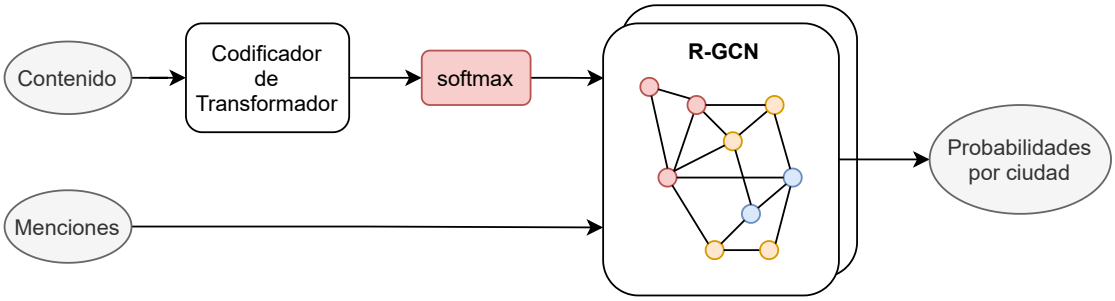


Figura 4.2: Arquitectura de un modelo `RGCN-EXT-Preds`.

- `GCN-EXT-`χ^2: Igual que `RGCN-EXT-Preds` pero consideramos utilizar como *features* de los usuarios los top k términos por ciudad según pruebas χ^2.

- `META-LR-BEST`: Usando el modelo de evaluación (4.1), entrenamos clasificadores para las redes y contenido usando los métodos que mejor resultados dieron respectivamente. Para contenido utilizamos el modelo `LR-BOW-`χ^2`-TEXT`, y para las redes `N2V-LR-NET`. Finalmente utilizamos un regresor logístico como meta-clasificador.

Con respecto a algunas herramientas adicionales, utilizamos la biblioteca Scikit-Learn de Pedregosa et al. (2011) para implementaciones de métodos de selección significativa y clasificadores no basados en redes neuronales. Para el uso de redes neuronales utilizamos la biblioteca Keras de Chollet et al. (2015) que nos ofrece facilidad para la implementación de las redes que utilizamos. Por último, para la implementación de redes neuronales en grafos utilizamos StellarGraph de Data61 (2018) que nos ofrece un ambiente de trabajo muy similar al de Keras.

Todos los modelos presentes se encuentran presentes en el repositorio de github: `https://github.com/fedefunes96/twitter-location`.

4.4. Métricas y resultados

En esta sección presentamos los resultados obtenidos para cada uno de nuestros modelos según métricas comúnmente utilizadas en otros trabajos de la literatura. Las Tablas 4.1 y 4.2 resumen los resultados de cada método que aplicamos y qué conjunto de *features* se utilizó para cada uno; en negrita podemos ver los mejores resultados en cada caso. Para Twitter-ARG-Exact debemos considerar que cada ciudad tiene asociado un par latitud/longitud que determinamos como el promedio de las coordenadas de todos los usuarios pertenecientes a dicha ciudad, y sobre esas coordenadas se calcularán la *acc@161* y los errores de cada método. A diferencia de Twitter-ARG-Exact, en Twitter-ARG-BBox nos quedamos con aquellas ciudades descriptas por Twitter que pertenezcan al *gazetteer* GeoNames, con lo que las coordenadas que GeoNames especifica por ciudad serán las utilizadas para calcular las métricas correspondientes.

4.5. Explicabilidad sobre clasificación

En la aplicación de modelos predictivos en las Ciencias Sociales Computacionales es importante ser capaces de explicar los motivos de una predicción puntual. En muchos casos es tan importante comprender el por qué de una clasificación como el obtener una exactitud elevada. Incluso en muchos casos, modelos que poseen una alta exactitud pueden comportarse en forma injusta o sesgada, discriminando a distintos sectores o grupos de una población. Esto es especialmente importante en aplicaciones sociales de la tecnología, y ha generado un área del aprendizaje automático vinculada con la equidad (fairness), sesgo (bias) y explicabilidad de las predicciones. En esta sección intentaremos aplicar algunas de estas herramientas para entender el funcionamiento de nuestros modelos y explicar predicciones puntuales.

Método	Acc	Acc@161	Error prom. (Km)	Error mediana (Km)
Contenido				
MNB-BOW-TEXT	59.2 %	71.0 %	716.8	5.4
MNB-BOW-χ^2-TEXT	63.1 %	74.0 %	638.6	4.9
LR-BOW-χ^2-TEXT	**67.2 %**	**76.9 %**	**524.8**	**4.7**
MNB-BOW-MI-TEXT	59.3 %	71.2 %	702.2	5.4
BiLSTM-TEXT	60.5 %	72.1 %	663.4	5.3
BiLSTM-W2V-TEXT	60.4 %	75.0 %	547.5	5.2
TF-W2V-TEXT	65.8 %	76.4 %	522.5	4.9
Menciones				
MNB-NET	38.7 %	58.5 %	925.5	24.5
OSLOM-NET	42.6 %	62.0 %	811.2	18.2
N2V-LR-NET	**46.0 %**	**66.0 %**	**711.2**	**13.5**
Seguidores				
MNB-NET	36.8 %	57.6 %	904.3	25.4
OSLOM-NET	26.1 %	43.0 %	1417.5	386.5
N2V-LR-NET	**39.2 %**	**59.9 %**	**865.5**	**21.7**
Redes (Seguidores + menciones)				
Infomap-NET	37.5 %	58.8 %	768.9	25.1
Contenido + Menciones				
GCN-EXT-Preds	49.1 %	68.7 %	661.8	10.8
GraphSAGE-EXT-Preds	72.7 %	82.3 %	369.1	3.9
RGCN-EXT-Preds	**73.1 %**	**82.9 %**	**367.5**	**3.8**
RGCN-EXT-χ^2	70.0 %	81.0 %	411.2	4.1
RGCN-ML-Preds	71.1 %	81.8 %	398.1	4.0
META-LR-BEST	72.2 %	82.7 %	374.9	3.9
Contenido + Redes				
RGCN-ML-Preds	71.7 %	82.3 %	380.2	4.0
META-LR-BEST	**72.7 %**	**83.2 %**	**371.4**	**3.9**

Tabla 4.1: Tabla resumida de resultados de métodos aplicados en Twitter-ARG-Exact.

Método	Acc	Acc@161	Error prom. (Km)	Error mediana (Km)
Contenido				
MNB-BOW-χ^2-TEXT	39.6 %	51.8 %	1082.7	66.8
TF-W2V-TEXT	**39.7 %**	**53.2 %**	**914.7**	**51.6**
Menciones				
N2V-LR-NET	35.7 %	51.8 %	1032.6	66.8
Seguidores				
N2V-LR-NET	36.9 %	53.2 %	1023.2	51.6
Contenido + Redes				
META-LR-BEST	**51.5 %**	**65.3 %**	**719.23**	**0.0**
RGCN-EXT-Preds	48.6 %	64.4 %	690.6	6.4

Tabla 4.2: Tabla resumida de resultados de métodos aplicados en Twitter-ARG-BBox.

Vamos a tratar de explicar como funciona nuestro modelo con mejores resultados basándonos en los términos más importantes para la clasificación de los usuarios y sus relaciones con otros usuarios cercanos. Recordemos que como tenemos un modelo de evaluación sobre todo el conjunto de datos, podemos tomar cualquier conjunto de los *K-Folds* externos para determinar que parámetros consideran los clasificadores entrenados en esa instancia para clasificar a los usuarios en el conjunto restante.

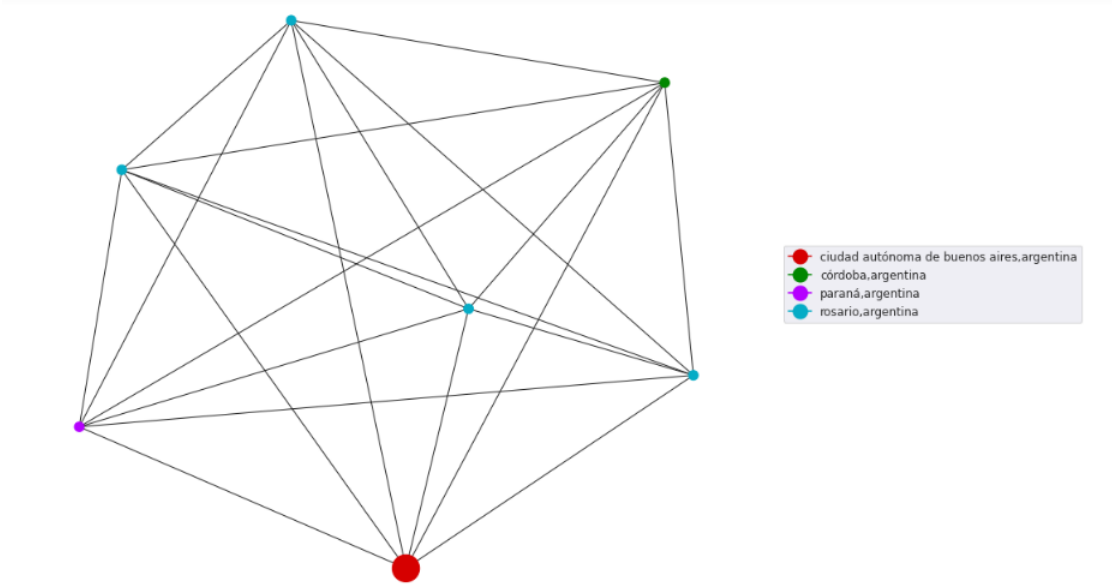
4.5.1. Explicabilidad sobre redes

Comenzando por las menciones de los usuarios, el método que mejor resultados nos dio fue el uso de Node2vec en conjunto con un regresor logístico para separar linealmente los nodos representados vectorialmente en el espacio. En sí Node2vec es un método no determínistico, con lo que para comprender su funcionamiento

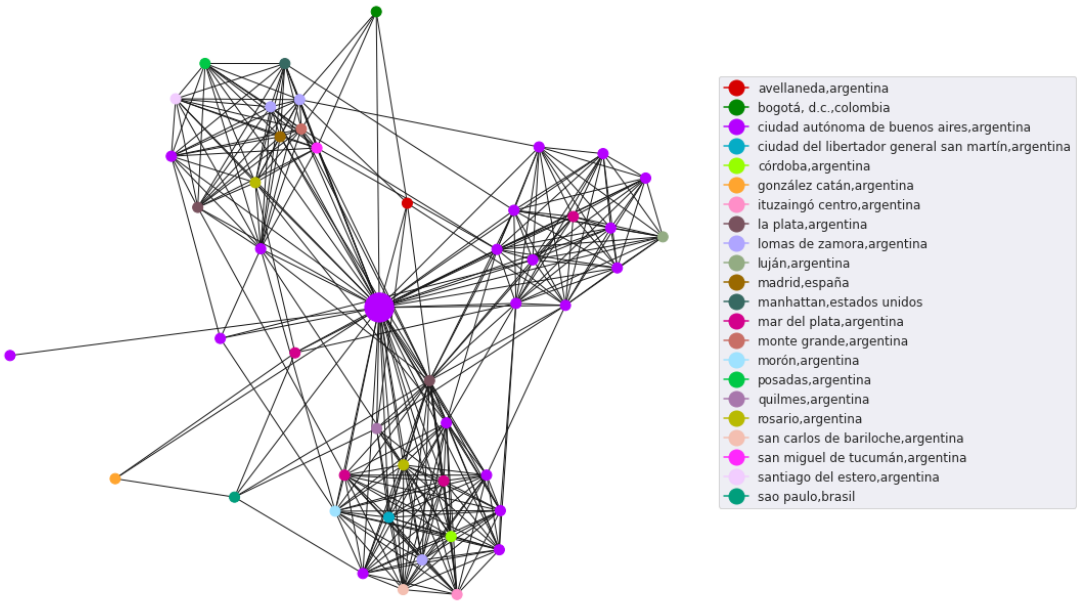
podemos analizar la EGO-Net de un usuario: la EGO-Net es un subgrafo que corresponde al conjunto de nodos a distancia K del usuario, considerando las aristas que unen a dichos nodos entre si y con el usuario. Dos ejemplos de la EGO-Net con profundidad 1 para dos usuarios puede verse en la Figura 4.3, en donde podemos apreciar que para el primer caso (4.3a) la mayoría de las conexiones de dicho usuario pertenecen a la ciudad de Rosario en Argentina. En cambio para el segundo usuario (4.3b), la mayoría de las conexiones son hacia usuarios de la Ciudad Autónoma de Buenos Aires, Argentina. Siendo que Node2vec genera *embeddings* de nodos explorando las vecindades de los nodos a un nivel de profundidad que depende de sus parámetros, esta representación de EGO-Nets nos puede ayudar a comprender cómo se distribuirán algunos puntos en el espacio. Por ejemplo, en la Figura 4.4 podemos ver una proyección en 2 dimensiones usando TSNE de los *embeddings* de Node2vec para los nodos de las EGO-Nets. Finalmente en la Figura 4.5 podemos ver la representación en 2 dimensiones de las clasificaciones de los *embeddings* de los nodos usando un regresor logístico en donde ambos usuarios pertenecientes a la ciudad autónoma de Buenos Aires son finalmente clasificados como un usuario de Rosario y uno de la Ciudad Autónoma de Buenos Aires respectivamente. El análisis en cuanto a la red de seguidores extendida es análogo, dado que el método con mejor resultado fue el mismo.

4.5.2. Explicabilidad sobre contenido

Siguiendo ahora con el contenido de los tweets de los usuarios, podemos analizar nuestro clasificador de contenido, sea basado o no en redes neuronales, usando SHAP ((SHapley Additive exPlanations) de Lundberg and Lee (2017), una herramienta para explicar las salidas de un clasificador usando enfoques basados en la teoría de juegos. Para nuestro clasificador LGR-BOW-χ^2, podemos analizar en la Figura 4.6 qué *features* generan mayor impacto para clasificar a usuarios de la

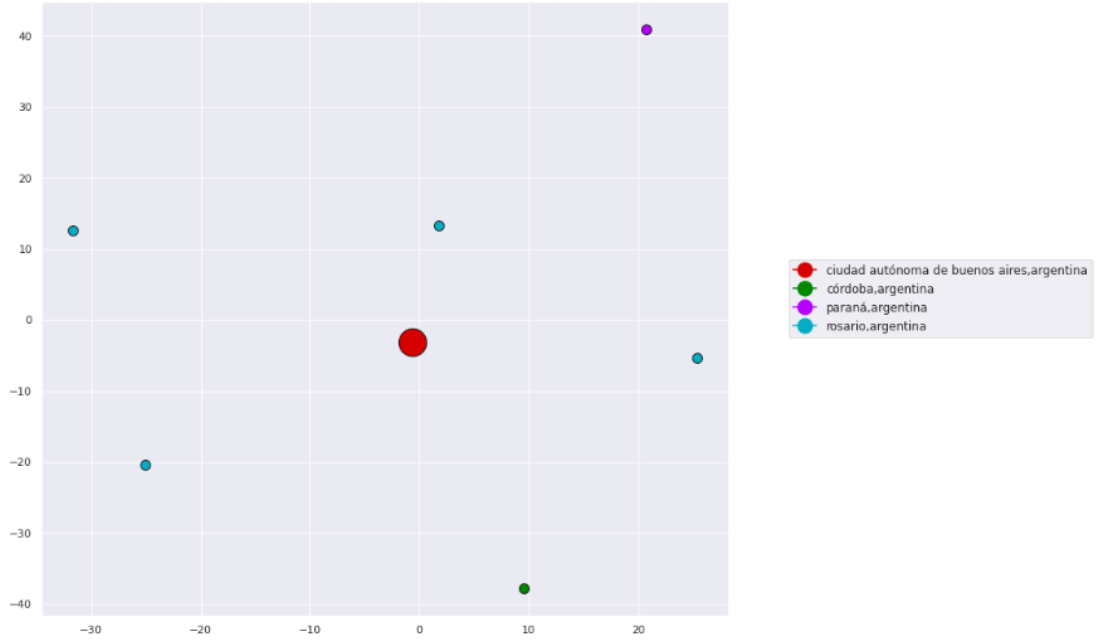


(a)



(b)

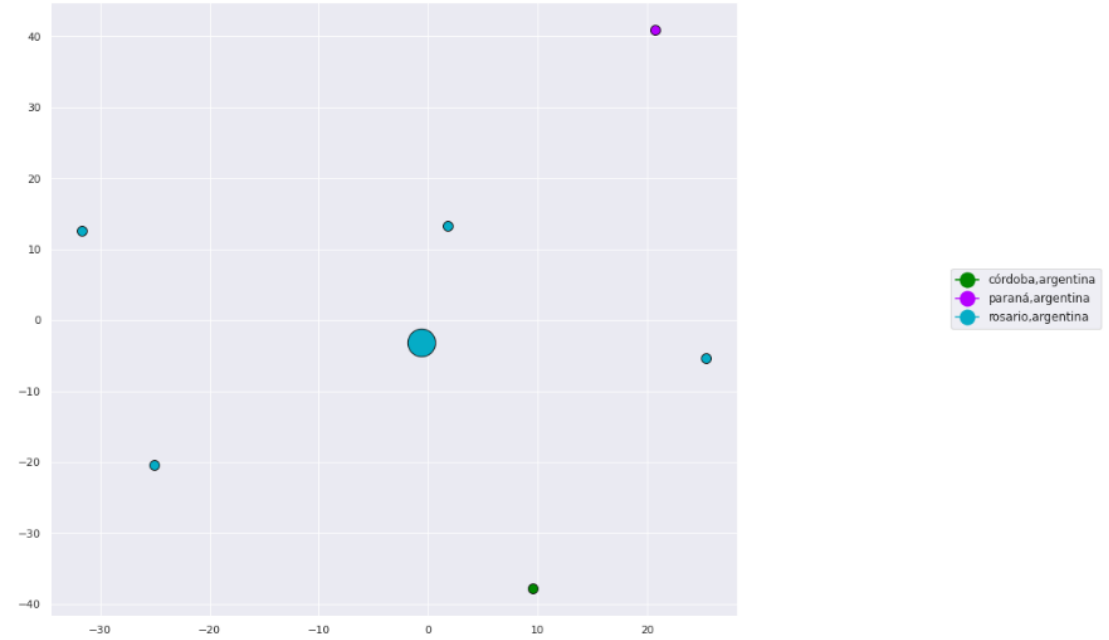
Figura 4.3: EGO-Nets con profundidad 1 de dos usuarios de Twitter-ARG-Exact pertenecientes ambos a la ciudad autónoma de Buenos Aires, Argentina. Por cuestiones de visibilidad, se agrandaron los nodos de ambos usuarios en sus respectivas redes.



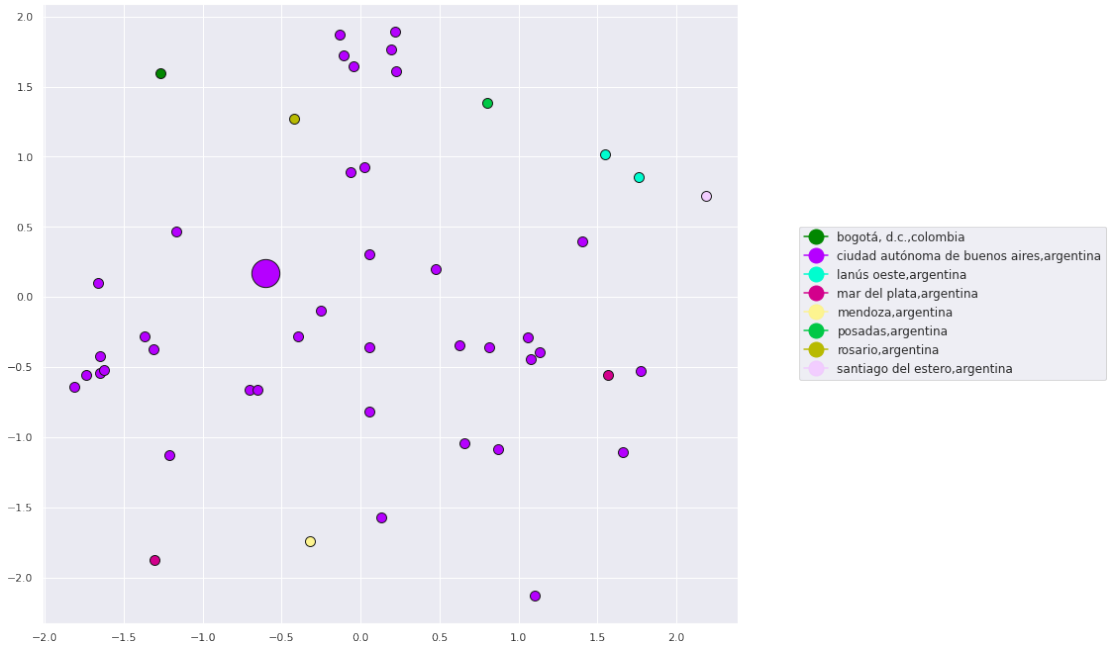
(a)

(b)

Figura 4.4: Representación en 2 dimensiones de los *embeddings* de los nodos de las EGO-Nets presentadas en la Figura 4.3. Para la representación en 2 dimensiones utilizamos OpenTSNE de Polivar et al. (2019).



(a)



(b)

Figura 4.5: Representación en 2 dimensiones con OpenTSNE de las clasificaciones de un regresor logístico sobre los *embeddings* de los nodos de las EGO-Nets presentadas en la figura 4.4

.

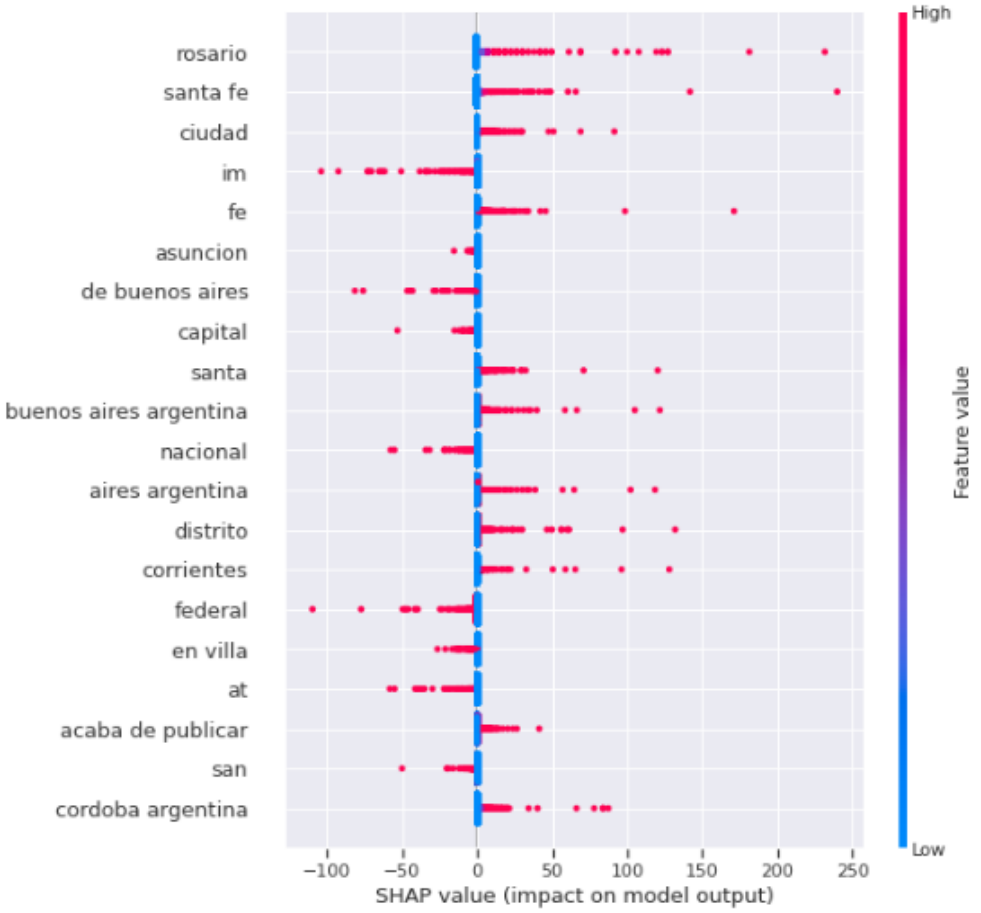


Figura 4.6: Gráfico SHAP mostrando los *features*, en este caso términos, que más influyen en la clasificación de LGR-BOW-χ^2 para usuarios de Rosario, Argentina.

ciudad de Rosario, Argentina.

4.6. Conjuntos de datos de prueba

Nos interesa saber que tan bien funcionan nuestros métodos en comparación con métodos propuestos por otros trabajos. Para eso, vamos a utilizar un conjunto de datos analizado en muchos trabajos de la literatura, conocido como Twitter-US Roller et al. (2012). Este conjunto de datos solo dispone del contenido generado por

los usuarios, con lo que debemos de preprocesarlo de la misma forma que hicimos con nuestros conjuntos, además de obtener los *hashtags* y menciones. Decidimos quedarnos con aquellos *hashtags* utilizados por al menos 10 usuarios distintos y con aquellos usuarios que fueron mencionados por menos de 20 usuarios distintos. Uno de los inconvenientes que encontramos al comparar nuestro modelo con otros trabajos es que estos suelen utilizar distintos modelos de clases (*labels*) para entrenar, con lo que la comparación podría no ser justa. Este es un problema que hemos detectado, estudiando algunos trabajos de la literatura. Por ejemplo, Huang and Carley (2019) comparan su modelo con el conjunto de clases propuestos por Han et al. (2012) y Rahimi et al. (2015) con el conjunto de clases propuesto por Roller et al. (2012). Para evitar este problema, decidimos generar dos tablas: una con métodos aplicados sobre regiones generadas por KD-Trees con como máximo 2400 usuarios (siendo en total 256 regiones siguiendo los lineamientos de Roller et al. (2012) y Rahimi et al. (2015) que pueden verse en la Figura 4.7), y la otra con clases propuestas por Han et al. (2012), en donde se colapsan ciudades bajo una misma región, considerando aquellas ciudades resultantes con más de 100.000 habitantes (en total se consideran 378 ciudades), luego asignamos a cada usuario a la ciudad resultante más próxima. En particular, para el conjunto de clases propuestas por Han et al. (2012), obtuvimos 403 clases empleando su método. Las Tablas 4.3 y 4.4 resumen los resultados de otros trabajos además de incluir los nuestros a modo de comparación. Decidimos, dado que estamos trabajando con tweets en inglés, utilizar *embeddings* pre-entrenados para el contenido utilizando datos disponibles del método GloVe Pennington et al. (2014).

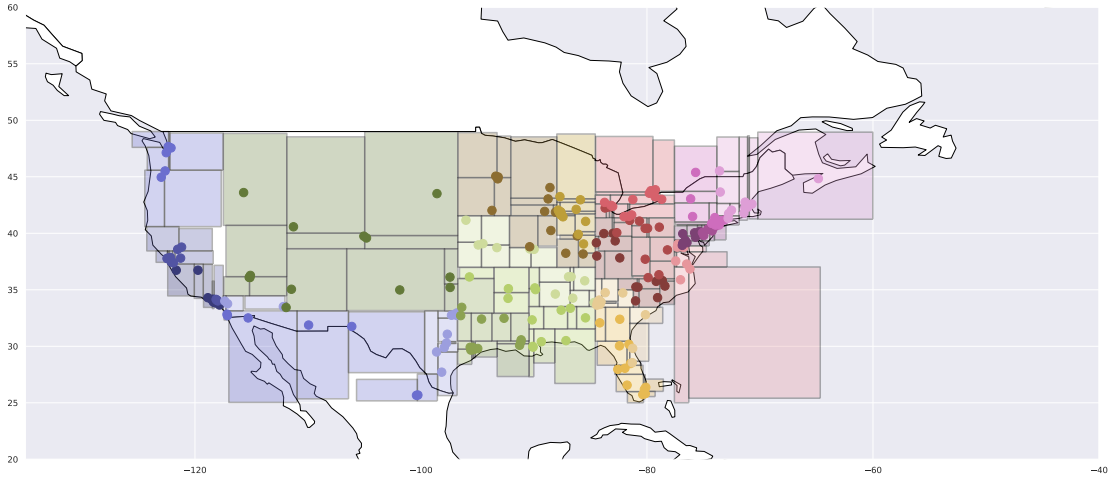


Figura 4.7: División del espacio de coordenadas de los usuarios de Twitter-US en regiones con un mínimo de 2400 muestras usando KD-Tree. Los puntos dentro de cada región representan la mediana de las coordenadas de los usuarios incluidos.

Método	Acc@161	Error prom. (Km)	Error mediana (Km)
Miura et al. (2017)	60.1 %	582.8	66.5
Rahimi et al. (2017)	61.0 %	515.0	77.0
Rahimi et al. (2018)	66.0 %	420.0	56.0
RGCN-EXT-Preds	67.0 %	384.0	57.2
META-LR-BEST	67.0 %	394.8	57.0

Tabla 4.3: Tabla resumida de resultados de métodos aplicados en Twitter-US considerando 256 clases obtenidas por medio de un KD-tree siguiendo los lineamientos de Roller et al. (2012) y Rahimi et al. (2015).

Método	Acc@161	Error prom. (Km)	Error mediana (Km)
Miura et al. (2017)	61.5 %	481.5	65.0
Huang and Carley (2019)	70.8 %	361.5	31.6
RGCN-EXT-Preds	66.6 %	408.7	43.3
META-LR-BEST	69.0 %	421.0	35.3

Tabla 4.4: Tabla resumida de resultados de métodos aplicados en Twitter-US considerando el modelo de ciudades propuesto por Han et al. (2012).

Capítulo 5

Conclusiones

En el presente trabajo construimos dos conjuntos de datos de Twitter focalizados primordialmente en Argentina, uno basado en coordenadas exactas, Twitter-ARG-Exact, como se hizo en trabajos de la misma índole, y otro basado en ubicaciones seleccionadas por los usuarios al momento de realizar los tweets, Twitter-ARG-BBox.

Presentamos varios métodos para la detección de la ubicación a nivel ciudad de los usuarios en conjuntos de datos de la red social Twitter y, en particular, propusimos un modelo de evaluación que combina el uso de varios clasificadores que, además de generar resultados similares a los del estado del arte, logra ser bastante sencillo de interpretar. También utilizamos métodos basados en redes neuronales usando conceptos modernos como los transformadores y redes neuronales en grafos. En particular, propusimos el uso de las R-GCN que pueden aplicarse sobre grafos multicapa pesados, sean dirigidos o no dirigidos, dando muy buenos resultados. También propusimos el uso de GraphSAGE, un método inductivo que evita la necesidad de re-entrenar el conjunto si deseáramos agregar un nuevo nodo. Incorporamos además el uso de la red de seguidores en nuestro análisis para evaluar que tanta mejoría provoca en los métodos el disponer de esta información que

suele ser muy difícil de obtener. Propusimos el uso de algoritmos de detección de comunidades como OSLOM e Infomap sobre distintos grafos generados por las relaciones de nuestros usuarios. Estos métodos resultan ser bastante eficientes, tanto en tiempo de computo como en uso de memoria, y representan una alternativa, poco utilizada, al uso de métodos para obtener representaciones vectoriales de nodos en grafos dado que sus resultados suelen ser similares. Propusimos también una nueva forma de construir los grafos de relaciones para los usuarios: basándose en la red de relaciones interna en combinación con relaciones en común contra usuarios externos, denominamos a este tipo de redes como *extendidas*.

Demostramos que el conjunto de datos basado en ubicaciones seleccionadas por los usuarios a través de la API de Twitter, Twitter-ARG-BBox, tiene un comportamiento similar con respecto al conjunto de datos basado en coordenadas exactas, Twitter-ARG-Exact, y que por lo tanto esta información podría utilizarse en trabajos donde haga falta mayor cantidad de muestras, dado que este tipo de datos es más sencillo de obtener.

Por último, propusimos un análisis más profundo sobre los perfiles de los usuarios demostrando que es posible ubicar un gran porcentaje de usuarios a menos de 10km de la ciudad a la que detectamos que pertenecen.

5.1. Trabajo a futuro

En este trabajo se exploraron diversas formas y metodologías para determinar la ubicación de usuarios de la red social Twitter. Sin embargo, varias metodologías y filtros quedaron por fuera del alcance de este trabajo. A futuro sería interesante trabajar sobre los siguientes puntos:

- Para nuestros métodos basados en el contenido utilizamos solamente los tweets con ubicación. Con lo que sería interesante aprovechar el contenido

de la totalidad de tweets.

- Para Twitter-ARG-BBox se utilizaron los tweets con ubicación para el entrenamiento dejando de lado el resto de tweets por falta de recursos para trabajar con tanta información. A futuro se espera poder integrar toda la información disponible.

- Un inconveniente que puede afectar la efectividad de nuestros métodos es el problema del viajante de usuarios en redes sociales. Eliminar aquellos usuarios que no tengan una ubicación permanente sería provechoso. Una posible implementación sería, ya que determinamos como ubicación de un usuario la ciudad donde mayormente publican tweets, filtrar aquellos usuarios cuya proporción de tweets publicados con respecto a la ciudad donde mayormente publican sea mayor a un parámetro α a determinar.

- En este trabajo utilizamos mecanismos de atención en nuestras redes neuronales, con lo que sería interesante extender este modelo a redes neuronales basadas en grafos. A futuro se espera que se trabaje sobre una implementación de las GAT (Graph ATtention Networks) Veličković et al. (2018) como un método para la clasificación de usuarios.

Capítulo 6

Bibliografía

Aggarwal, C. C. (2015). Data classification. In *Data Mining*, pages 285–344. Springer.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Bondy, J. A., Murty, U. S. R., et al. (1976). *Graph theory with applications*, volume 290. Macmillan London.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

Cheng, Z., Caverlee, J., and Lee, K. (2010). You are where you tweet: a content-based approach to geo-locating twitter users. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 759–768.

Chi, L., Lim, K. H., Alam, N., and Butler, C. J. (2016). Geolocation prediction in

twitter using location indicative words and textual features. In *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*, pages 227–234.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Coscia, M. (2021). *The Atlas for the Aspiring Network Scientist*.

Cuevas, R., Gonzalez, R., Cuevas, A., and Guerrero, C. (2014). Understanding the locality effect in twitter: measurement and analysis. *Personal and Ubiquitous Computing*, 18(2):397–411.

Data61, C. (2018). Stellargraph machine learning library. `https://github.com/stellargraph/stellargraph`.

De Longueville, B., Smith, R. S., and Luraschi, G. (2009). .ºmg, from here, i can see the flames!.ª use case of mining location based social networks to acquire spatio-temporal data on forest fires. In *Proceedings of the 2009 international workshop on location based social networks*, pages 73–80.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.

Girvan, M. and Newman, M. E. (2002). Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

Hagberg, A., Swart, P., and S Chult, D. (2008). Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035.

Han, B., Cook, P., and Baldwin, T. (2012). Geolocation prediction in social media data by finding location indicative words. In *Proceedings of COLING 2012*, pages 1045–1062.

Han, B., Cook, P., and Baldwin, T. (2014). Text-based twitter user geolocation prediction. *Journal of Artificial Intelligence Research*, 49:451–500.

Hecht, B., Hong, L., Suh, B., and Chi, E. H. (2011). Tweets from justin bieber's heart: the dynamics of the location field in user profiles. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 237–246.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Huang, B. and Carley, K. (2019). A hierarchical location prediction neural network for Twitter user geolocation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4732–4742, Hong Kong, China. Association for Computational Linguistics.

Jongman, B., Wagemaker, J., Romero, B. R., and De Perez, E. C. (2015). Early flood detection for rapid humanitarian response: harnessing near real-time satellite and twitter signals. *ISPRS International Journal of Geo-Information*, 4(4):2246–2266.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

Lancichinetti, A., Radicchi, F., Ramasco, J. J., and Fortunato, S. (2011). Finding statistically significant communities in networks. *PLOS ONE*, 6(4):1–18.

Liu, R. and Krishnan, A. (2020). Pecanpy: a fast, efficient, and parallelized python implementation of node2vec. *bioRxiv*.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc.

Mahmud, J., Nichols, J., and Drews, C. (2014). Home location identification of twitter users. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3):1–21.

McGough, S. F., Brownstein, J. S., Hawkins, J. B., and Santillana, M. (2017). Forecasting zika incidence in the 2016 latin america outbreak combining traditional disease surveillance with search, social media, and news report data. *PLoS neglected tropical diseases*, 11(1):e0005295.

McPherson, M., Smith-Lovin, L., and Cook, J. M. (2001). Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444.

Mikolov, T., Corrado, G., Chen, K., and Dean, J. (2013). Efficient estimation of word representations in vector space. pages 1–12.

Miura, Y., Taniguchi, M., Taniguchi, T., and Ohkuma, T. (2017). Unifying text, metadata, and user network representations with a neural network for geoloca-

tion prediction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1260–1272.

Molloy, M., Reed, B., Newman, M., Barabási, A.-L., and Watts, D. J. (2011). A critical point for random graphs with a given degree sequence. In *The Structure and Dynamics of Networks*, pages 240–258. Princeton University Press.

Newman, M. (2018). *Networks*. Oxford university press.

Newman, M. E. and Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113.

Paul, M. J. and Dredze, M. (2011). You are what you tweet: Analyzing twitter for public health. In *Fifth international AAAI conference on weblogs and social media*.

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

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710.

Polivar, P. G., Stravar, M., and Zupan, B. (2019). opentsne: a modular python library for t-sne dimensionality reduction and embedding. *bioRxiv*.

Rahimi, A., Cohn, T., and Baldwin, T. (2015). Twitter user geolocation using a unified text and network prediction model. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 630–636, Beijing, China. Association for Computational Linguistics.

Rahimi, A., Cohn, T., and Baldwin, T. (2017). A neural model for user geolocation and lexical dialectology. In *Proceedings of ACL-2017 (short papers) preprint*, Vancouver, Canada. Association for Computational Linguistics.

Rahimi, A., Cohn, T., and Baldwin, T. (2018). Semi-supervised user geolocation via graph convolutional networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2009–2019. Association for Computational Linguistics.

Reyero, T. M., Beiró, M. G., Alvarez-Hamelin, J. I., Hernández, L., and Kotzinos, D. (2021). Evolution of the political opinion landscape during electoral periods. *EPJ Data Science*, 10(1):31.

Roller, S., Speriosu, M., Rallapalli, S., Wing, B., and Baldridge, J. (2012). Supervised text-based geolocation using language models on an adaptive grid. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1500–1510.

Rosvall, M., Axelsson, D., and Bergstrom, C. T. (2009). The map equation. *The European Physical Journal Special Topics*, 178(1):13–23.

Sakaki, T., Okazaki, M., and Matsuo, Y. (2010). Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer.

Schütze, H., Manning, C. D., and Raghavan, P. (2008). *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge.

Talukdar, P. P. and Crammer, K. (2009). New regularized algorithms for transductive learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 442–457. Springer.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph Attention Networks. *International Conference on Learning Representations*. accepted as poster.