Predicción Temprana de Tendencias en Redes Sociales Basada en Características Sociales y Contenido [1]

Autor: Emanuel Juan René Meriles.
Director: Dr. Martín A. Domínguez.
Co-director: Lic. Pablo Gabriel Celayes.
Facultad de Matemática, Astronomía, Física y Computación
Universidad Nacional de Córdoba

Resumen

En estos últimos años, las redes sociales se han hecho cada vez más masivas. En consecuencia, son una fuente fundamental de información y una poderosa herramienta para esparcir ideas y opiniones. Basándose en Twitter, este trabajo estudia el problema de predecir las preferencias de retweeteo de un usuario, dado un tweet, considerando cómo el tweet ha sido compartido por el ambiente de ese usuario; y además el problema más global de predecir si un tweet va a ser popular o no, basado en el comportamiento de retweeteo de usuarios centrales. Para ambos problemas exploramos la evolución de la calidad de la predicción, dependiendo de la cantidad de información disponible en el tiempo desde que un tweet es creado, y elaborar conclusiones sobre el trade-off entre el tiempo transcurrido y la performance de la predicción.

Para el problema de predicción de un usuario, este modelo social de predicción logra, por ejemplo, alrededor de 63.76% en score F_1 usando los primeros 15 minutos de información, 75.2% a las 4 horas, y 86.08% sin considerar ninguna ventana de tiempo. En el caso de la predicción de popularidad, conseguimos scores de 65.67% con 60 minutos de información, 74.4% con 4 horas y 80.73% sin la restricción de ventanas de tiempo, usando el comportamiento del 15% de los usuarios considerados influencers. Todos estos resultados son obtenidos sin considerar el contenido de los tweets. Luego incorporamos features basadas en los word embeddings de fastText para representar el contenido de los tweets. Mientras estos modelos por sí solos obtienen un F_1 de alrededor del 50% para preferencias y predicción de popularidad, cuando se combinan con modelos sociales se mejora la predicción de popularidad, en la mayoría de los casos, en más de un 4%. Para el caso de predicción de preferencia, la incorporación de fastText al modelo social es más útil para pequeñas ventanas de tiempo.

Concluimos que es posible predecir razonablemente la preferencia de retweeteo de un usuario o cuán popular va a ser una publicación, usando sólo la información disponible dentro de los primeros 30-60 minutos.

Palabras Clave: Análisis de Redes Sociales, Aprendizaje Automático, Detección de influenciadores, Detección de comunidades, Modelos de predicción, Twitter

Clasificación (ACM CCS 2012):

- **Applied computing∼Sociology**
- **Computing methodologies∼Natural language processing**
- Computing methodologies∼Support vector machines
- Computing methodologies∼Latent Dirichlet allocation

Agradecimientos

Agradezco en primer lugar a la universidad pública, gratuita y laica. Las oportunidades, logros y desafíos, tanto en la esfera académica, como la personal y social, serían muy diferentes en mi vida sin la posibilidad que tuve de asistir a la Universidad Nacional de Córdoba, enmarcada en un sistema de educación nacional que es consecuente con la idea de entender la educación como un derecho humano.

A mis directores Martín Ariel Domínguez y Pablo Gabriel Celayes, por su precisión y excelente capacidad para transmitir conocimientos; por muchísima paciencia y dedicación; y por brindarme oportunidades, tanto en el ámbito de divulgación científica, como en muchos otros ámbitos.

A toda mi familia, sin ellxs, sin su apoyo y contención, no estaría celebrando este logro.

A lxs que me acompañaron en el camino de cursado.

A todas las personitas hermosas con las que pude alivianar el peso de un trabajo que demandó mucha energía. Un verdadero privilegio tenerlxs en mi vida.

Contents

Chapter 1

Introducción

Las plataformas de medios sociales como Twitter, Facebook e Instagram han ganado una adopción masiva de uso, convirtiéndose en una presencia central en el día a día y en discusiones públicas, con un fuerte impacto en la forma en la que la gente se expresa, informa, e influye la una a la otra. Una de las redes sociales más prominentes es Twitter, una plataforma de microblogging online en tiempo real que permite a sus usuarios publicar, leer y compartir mensajes cortos de hasta 280 caracteres, conocidos como Tweets. Cada vez que un usuario publica un tweet, Twitter le confiere un identificador único y una marca del momento exacto cuando fue publicado (timestamp). Una función frecuentemente usada en ésta red social es el "retweet", que permite a un usuario re-publicar un tweet de otro usuario dentro de su propio flujo de mensajes.

Una característica de Twitter que la diferencia de otras redes sociales es que la mayoría de su contenido es público por defecto, mientras que su estructura de datos es fácilmente accesible através de la API oficial, que manipula la información referente a la forma en la que los usuarios interactúan entre sí y el contenido que ellas comparten.

La influencia de medios sociales puede ser estudiada tanto a nivel individual como así también a nivel colectivo. En el primer caso, los esuerzos se concentran en entender cómo las preferencias y el comportamiento de un usuario dado son formados por su actividad previa en medios sociales. Por otro lado, los fenómenos colectivos pueden ser estudiados y analizados buscando entender los mecanismos que hacen que las publicaciones se hagan populares sobre grandes audiencias.

En la línea de trabjos previos se ha trabajado en formulaciones de ambos tipos de objetivos como modelos predictivos. En [25], estudiamos el problema de la predicción de preferencias de contenido de un usuario dado basada en las preferencias de sus vecinos. Concretamente, cuánto podemos predecir si ese usuario va a retweetear o no un tweet dado, conociendo la actividad de retweeteo que tuvo el tweet entre sus conexiones sociales. En [26], se llevaron estas ideas a nivel comunidad, prediciendo la popularidad de tweets en vez de preferencias individuales, y reemplazando el conjunto de conexiones sociales de un usuario con un conjunto de influencers que tienen conexiones sociales comunes a toda

la comunidad. Ésto puede ser pensado como una especie de vecindad global o común en el sentido de que todas están, en cierta medida, expuestas a la actividad de estos usuarios muy visibles.

Independientemente de si el objetivo es predecir preferencias individuales o globales, en ambos casos es posible basar las prediciones en información social (quién compartió contenido, y cómo ésto conecta a los usuarios) o en información de contenido (de qué trata el tweet, qué temas e ideas son discutidas en el mismo). Ambos trabajos previos comenzaron haciendo foco en lo que se puede aprender al ver información social pura, para luego extender tales modelos con información sobre el contenido por medio de diversas técnicas de Procesamiento de Lenguaje Natural

Más allá de las interacciones sociales y las características del contenido, el tiempo es la próxima dimensión a explorar cuando se trata de profundizar nuestro conocimiento en la formación de preferencias sociales. *Quién* comparte información y de *qué* trata son importantes, pero el impacto de ambos factores es determinado fuertemente por *cuándo* la actividad ocurre. En el presente trabajo, encaramos las problemáticas de retweeteo y de tendencias usando sólo información disponible en una ventana determinada de tiempo desde la creación de el tweet analizado. Por ejemplo: transcurridos cinco minutos de la creación de un tweet, ¿es posible predecir con efectividad si un usuario va a retweetearlo? O, transcurridos diez minutos, podemos dar una estimación precisa de si un tweet va a ser popular? La información temporal puede además ser incluida entre la características a predecir, considerando no solo si se va a retweetear o no un tweet, sino también cuándo va a suceder. Expandir nuestra investigación en esta línea no solo nos da un entendimiento más profundo de la influencia social, sino que además hace a los resultados mucho más aplicables a entornos online de predicción temprana, al proveernos con una cuantificación de la compensación en performance entre tiempo transcurrido y predictibilidad.

Chapter 2

Fundamentos Teóricos

Para entender el presente trabajo, es necesario dar un contexto teórico, conceptual y terminológico, así como también de trabajos previos realizados sobre la temática por desarrollar.

2.1 Clasificación

Dentro del Aprendizaje Automático Supervisado, el problema de clasificación binaria consiste en, dado un conjunto de datos de entrenamiento, hacer que un modelo infiera reglas que permitan la asignación de clases a nuevos datos. Se considera que el aprendizaje es supervisado si en los datos de entrenamiento se incluye una correcta clasificación, que consiste en la asignación de una clase, de cada uno de los ejemplos para que el modelo deduzca reglas aplicables a los mismos.

Comunmente, los datos de entrada para un modelo son transformados en formatos "entendibles" para el mismo. Este procedimiento se denomina *extracción de features* y se trata de la detección de caracaterísticas relevantes para nuestra tarea de clasificación, y la traducción de las mismas a formas procesables para nuestro modelo (generalmente, formas vectoriales).

En nuestro trabajo, se usarán clasificadores binarios para hacer análisis y predicciones en el entorno de Redes Sociales. Más específicamente, nos centraremos en la predicción temprana de contenido popular en una red social, por un lado; y por otro, en la predicción temprana de contenido interesante o no interesante para un usuario dado.

2.1.1 Métricas

Entre las formas más populares de evaluar clasificadores supervisados, se encuentran métricas que se basan en contrastar resultados predichos por el clasificador contra los resultados que se tienen anotados de antemano, ya sean de origen fáctico o basados

en análisis de experto en el área de clasificación. En base al conteo y contraste de resultados obtenidos para cada dato predicho, se elaboran varios cocientes, los cuales son representativos de la performance de un clasificador.

Entre las métricas más populares se encuetran:

- **Precisión**: cociente entre cantidad de aciertos de una categoría dada y el total de datos clasificados.
- **Exhaustividad o recall**: cociente entre cantidad de aciertos de una categoría dada y el total de datos de esa categoría en los datos clasificados.
- **F1 score**: mide el desbalance de las dos métricas anteriores. Se calcula como:

$$f1score = 2.\frac{1}{\frac{1}{recall} + \frac{1}{precision}} = 2.\frac{precision.recall}{precision + recall} \tag{2.1}$$

2.2 Grafos

Siguiendo a ambos trabajos anteriores, para representar la información a manejar sobre usuarios y relaciones entre ellos, se optó por el uso de Grafos. Representamos a un usuario con un nodo y a la relación "seguir" (Ver sección 2.6.1) como una arista dirigida con origen en la usuaria seguidora y sentido hacia la usuaria seguida. No se usan pesos en aristas ni en nodos, solo se manipulan de diferentes maneras una red inicial que contiene todas las relaciones al momento de comenzar la recolección de tweets en bruto.

Veremos a continuación algunos conceptos que serán usados a lo largo del trabajo para el análisis de nuestro grafo de usuarios.

2.2.1 Centralidad

La centralidad es una propiedad que se asigna a cada nodo, y representa el nivel de importancia relatiava que el mismo tiene dento de un grafo. Dicha importancia puede estar en correlación con, por ejemplo, tener una gran cantidad de nodos adyacentes o con una ubicación estratégica que permite la interconexión de grandes componentes conexas dentro del grafo. Un nodo con un alto nivel de centralidad será llamado "influenciador", o *influencer*, debido al impacto que puede ocasionar con una acción sobre una gran porción de la red. Algunas de las medidas de centralidad más populares son:

Valencia

También llamado grado de un vértice, la valencia es la cantidad de nodos que tiene un nodo a su alcance en un solo paso de distancia. En un grafo dirigido, se puede discriminar entre aristas entrantes y aristas salientes de un nodo.

Interconectividad (*betweenness*)

La interconectividad[2] es una medida de centralidad basada en las trayectorias más cortas. Para cada par de vértices en un grafo conectado, existe al menos una ruta más corta entre los vértices, de modo que el número de aristas por los que pasa la ruta está minimizado. La interconectividad entonces para un vertice i será la proporción de trayectorias más cortas desde dos vértices distintos a i en las cuales participa este vértice sobre el total de trayectorias máscortas.

Pagerank

Otro algoritmo muy popular en la medición de centralidad es PageRank. Introducido por Google en 1999[3] causó revolución en su popular buscador web. No sólo involucra la información de adyacencia del propio nodo, sino también de sus nodos adyacentes. PageRank es un algoritmo iterativo qué inicia otorgándole un puntaje de $1/N$ a todos los nodos donde N será la cantidad de vértices del grafo. Luego, comenzando por algún nodo seleccionado al azar, inicia su recorrido otorgándole su puntaje dividido la cantidad de sus aristas salientes a todos sus vértices adyacentes. El vértice receptor suma lo enviado por todos sus vecinos a su puntaje asignado. Una vez que todos los vértices han cumplido con la tarea de enviar el puntaje adicional a sus nodos adyacentes, la primer iteración habrá concluido. Luego de varias iteraciones de este procedimiento los puntajes de reelevancia para cada nodo estará definido. El algoritmo cuenta con un factor de amortiguación que modifica ligeramente el resultado anteriormente mencionado quedando como resultante la siguiente fórmula final para el cálculo del PageRank de un nodo j donde $in(j)$ será el conjunto de conexiones entrantes hacia el nodo y $C(j)$ su valor de PR.

$$PR(j) = (1 - d) + d * \sum_{i \in in(j)} \frac{PR(i)}{C(i)} \tag{2.2}$$

Centralidad de Katz

Introducida por Leo Katz en 1953[6] con similitud a Pagerank, la centralidad de Katz es una medida que aporta información de los nodos adyacentes inmediatos como así también de otros nodos que se conecten a través de ellos. Las conexiones hechas con vecinos intermediarios, sin embargo, están penalizadas por un factor de atenuación α el cual es degradado exponencialmente mediante la cantidad de intermediarios que participen.

Sea A la matriz de adyacencia y A^k la matriz de conexión entre vértices mediante k intermediarios, n la cantidad de nodos y α el factor de atenuación tenemos que:

$$KATZ(i) = \sum_{k=1}^{\infty} \sum_{j=1}^{n} \alpha^k (A^k)_{ij} \tag{2.3}$$

2.3 Procesamiento de Lenguaje Natural

Procesamiento de Lenguaje Natural (PLN) es un campo de investigación que involucra a disciplinas como la linguistica, ciencias de la computación, ingeniería de la información e inteligencia artificial. Se ocupa del estudio de la interacción entre computadoras y leguaje humano (natural), y en particular cómo se pueden programar computadoras para procesar y analizar grandes volúmenes de datos de lenguaje natural. Existe una muy amplia variedad de tareas de lenguaje natural que involucran distintas áreas del lenguaje como ser: sintaxis, semántica, discurso, habla; y existe una mucho más amplia variedad de metodologías y algoritmos para resolver estas tareas.

Una forma muy popularizada de procesamiento de datos es la creación y manipulación de *embeddings*. Se trata de una forma de modelar datos de entrada vinculándolos a vectores de números reales. Para el caso de procesamiento de texto en particular existe una herramienta llamada fastText.

2.3.1 fastText y *word embeddings*

fastText es una librería desarrollada por Facebook AI Research[19] para la creación de *word embeddings* y para la clasificación de texto. Se puede emplear para aprendizaje automático supervisado y no supervisado.

En particular, para la creación de *word embeddings*, se emplea la parte no supervisada de fastText. Esto es, se crea un modelo capaz de traducir oraciones o documentos (conjuntos de oraciones) a vectores. Este proceso comienza con la creación de vectores por cada palabra en la oración/documento. Estos se obtienen mediante dos formas diferentes: caracterización a nivel palabra y una caracterización a nivel caracter.

La caracterización a nivel palabra puede ser usando la técnica *Continuous Bag-of-Words* (CBOW) o Skip-Gram. Ambos métodos emplean una red neuronal para predecir. El primer método lleva a cabo la tarea de predecir una palabra dado el contexto de la misma; mientras que el segundo, intenta predecir el contexto de una palabra dada. Llamamos contexto al conjunto de palabras que rodean la palabra central, con un tamaño de ventana fijo. A este tipo de tareas de predicción se la denomina tarea de pretexto, ya que no coincide con el objetivo principal que nos propusimos, sino que la mencionada predicción se trata de una tarea intermedia.

De dicha red neuronal, ya entrenada para predecir, se descartan las salidas y entradas dadas de forma que lo que nos sirve para la creación de los *word embeddings* son las capas ocultas de la red. Al hacer nuestra tarea de pretexto, la red neuronal aprendió una forma de caracterizar internamente las palabras (para la posterior predicción de la forma CBOW o Skip-Gram). Es entonces que ahora, dada una palabra, al multiplicarla por la matriz de representación de la red neuronal, se obtendrá la representación de la misma en forma de vector, con tantas dimensiones como capas ocultas tenía la red neuronal.

La caracterización a nivel caracter emplea un método similar de tarea de pretexto: es

decir, dado el dataset de enternamiento del modelo fastText, intenta predecir sobre los datos que él mismo provee. Pero en este caso, en vez de predecir a nivel palabra, lo hace a nivel caracter, empleando *n-gramas*. Un *n-grama* es una subsecuencia de n elementos, dada una secuencia inicial. Los elementos de la secuencia van a ser subpalabras de una palabra elegida, con un tamaño de ventana de cierto tamaño. Es entonces que se va a intentar predecir subsecuencias de las palabras, y se va a seguir luego con el procedimiento de extracción de las capas ocultas de la red, como ya se explicó al hablar de tareas de pretexto.

2.4 Reproducibilidad de experimentos

La reproducibilidad es la capacidad de un ensayo o experimento de ser reproducido y replicado. Se trata de un requerimiento fundamental del método científico. En el entorno de procesos computacionales, existen diversas técnicas y herramientas para lograr que un experimento sea reproducible.

La principal fuente de reproducibilidad es una descripción clara y completa del proceso de un experimento. En computación, esto se traduce generalmente a dos requerimientos generales: el procedimiento detallado llevado a cabo (código, pseudo-código, etcétera) y el conjunto de datos de trabajo. Cuando lo que se comparte es código de programación, se suele incluir el versionado de librerías dependientes que se usó.

Por otro lado, existen algoritmos complejos que requieren de una cierta aleatoriedad interna para su funcionamiento. Es el caso por ejemplo de mezclas aleatorias de elementos; necesidad de hacer steps (pasos) de longitud aleatoria para ir aproximando a un mínimo local; entre otras. En estos casos, las librerías que precisan de un procedimiento con esta características disponen de la posibilidad de fijar una *semilla*. Una semilla es un valor usado para inicializar un generador de números pseudo aleatorio. Se llama "pseudo aleatorio", ya que se trata de un algoritmo determinístico que genera una secuencia de números, cuyas propiedades aproximan las propiedades de secuencias de números aleatorios.

El conjunto de datos de trabajo hace también a la reproducibilidad de un experimento. En efecto, existen siempre un conjunto de características que son, o se presume que sean, decisivos para la obtención de resultados similares entre experimentos. Estas características aveces pueden ser mesurables y comparables, pueden implicar cualidades cualitativas o cuantitativas, se pueden sintetizar en valores o coeficientes, etcétera. Muchas veces los conjuntos de datos usados para un experimentos son publicables pero no es siempre posible. En estos casos, una forma muy popular de compartir un conjunto de datos es dar un procedimiento preciso para la recolección de un conjunto de datos nuevo que, al seguir sus pasos, asegure ciertas condiciones y propiedades sobre el conjunto de datos de un experimento.

2.5 Clasificadores

El núcleo de la tarea de clasificación consiste en el algoritmo de clasificación elegido. En efecto, dependiendo del caso de aplicación, se recomienda una amplísima variedad de ellos. Algunos de los factores que pueden influir a la hora de elegir un algoritmo son: dimensionalidad de vectores, variabilidad de datos, cantidad de datos de entrenamiento, capacidad disponible de procesamiento y memoria, precisión exigida o deseada, disponibilidad de tiempo de desarrollo del modelo o de predicción del mismo.

El presente trabajo, basándose en dos trabajos previos, elige como algoritmo de clasificación a las *Support Vector Machines*.

2.5.1 Support Vector Machines (SVM)

Se trata de un tipo de clasificador lineal. Esto significa que realiza la tarea de clasificación creando hiperplanos que separen el universo de datos en clases diferentes. En particular, las SVM tratan de optimizar las distancias entre los puntos observados de diferentes clases y el hiperplano aprendido, maximizando la distancia entre los datos y la frontera de decisión.

De lo dicho, surge entonces el requisito básico de un universo linealmente separable. Esto no necesariamente viene dado o es conseguible dado un problema. Sin embargo, existen técnicas que intentan paliar esta situación. Una de las técnicas más populares consiste en transformar los datos iniciales de forma que el nuevo espacio sea linealmente separable, manteniendo al mismo tiempola caracterización inicial de cada dato de entrada.

Esta técnica es conocida con el nombre de *kernel trick*, que propone un conjunto de funciones de *kernel* (función de mapeo a altas dimensionalidades), como transformadoras de datos. Se intenta así que, luego de aplicar alguna de dichas transformaciones, los datos sean linealmente separables.

Búsqueda del mejor hiperplano

La búsqueda de cualquier hiperplano que distinga los datos no es suficiente, ya que esto podría determinar una mala generalización de la clasificación y decaer en un sesgo que involucre subajuste. Por lo tanto, es importante prestar atención a la forma de buscar el mejor hiperplano.

Dados datos de entrenamiento $\{x_i, y_i\}$ con i=1,..j. Sea $x_i \in \mathbb{R}^\kappa$ el vector numérico proveniente de cada observación y cada $y_i \in$ -1, 1 la representación de la clase perteneciente. Diremos que los datos son linealmente separables si existe un hiperplano separador qué los distinga, de la forma:

$$xw + b = 0 \ (x \in \mathbb{R}^\kappa, \ b \in \mathbb{R})$$

El márgen del hiperplano se define como la suma de las distancias d_+ y d_- donde cada una se define como la distancia fronteriza de la muestra más cercana de cada clase.

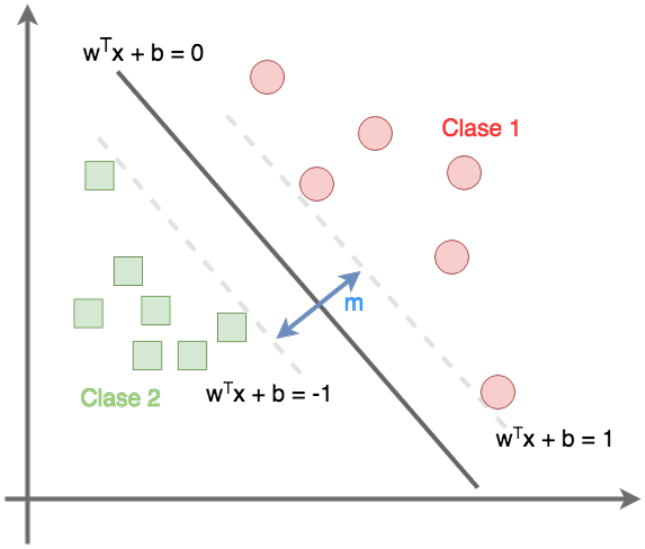

Figure 2.1: SVM y la mejor frontera de decisión, tan lejos de los datos cómo sea posible.

Quedan entonces caracterizados los hiperplanos tangentes por las siguientes ecuaciones.

$$\begin{aligned} \mathbf{x}_i\mathbf{w} + b \geq +1, \quad & y_i = +1 \\ \mathbf{x}_i\mathbf{w} + b \leq -1, \quad & y_i = -1 \\ \equiv & \\ y_i(\mathbf{x}_i\mathbf{w} + b) - 1 \geq 0, \quad & \forall i \end{aligned} \tag{2.4}$$

$$m = d_+ + d_- = \frac{|(-b+1) - (-b)|}{\|\mathbf{w}\|} + \frac{|-b-(-b-1)|}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|} \tag{2.5}$$

Dado que en los puntos más cercanos a la frontera de cada clase vale la igualdad, la distancia geométrica entre estos dos planos será $\frac{2}{\|w\|}$. Por lo tanto maximizar el márgen entre las fronteras quedará reducido a minimizar la norma euclidea $\|w\|$

Truco de Kernel

Como se mencionó antes, aplicar una transformación a nuestros datos mediante una función puede darnos otra perspectiva de los datos que permita lograr encontrar un hiperplano que los clasifique. *Kernel* es el nombre que se le da a este mapeo de cada punto a un nuevo universo de mayor dimensión. Entre los kernel más populares encontramos funciones lineales, polinomios como también funciones de base radial (RBF). Más allá de las funciones genéricas, también es posible aplicar cualquier tipo de función personalizada

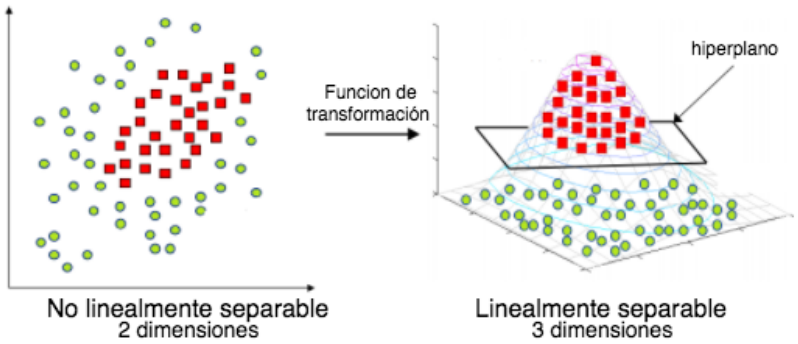

Figure 2.2: Ejemplo de Kernel-Trick de 2D a 3D

para la resolución en particular de un problema de datos no separables. Vemos en la Figura 2.2 cómo es aplicada una función cuadrática como kernel sobre un plano de 2 dimensiones.

Parametrización

La gran variedad de aplicaciones en las cuales se utilizan las SVM's como clasificador hace que deban ser genéricas pero sin perder rendimiento. Para ello cuentan con parámetros de ajuste que cambiarán sutilmente el algoritmo según la necesidad. La búsqueda de los parámetros puede llegar a ser una tarea compleja y poco intuitiva aunque existe la posibilidad de conseguir los mejores valores para cada uno de forma automática. *Gridsearch* es un algoritmo que se encarga justamente de realizar esta búsqueda. Mediante una lista de posibles asignaciones para cada parámetro, implementa todas las combinaciones posibles y, aplicando la técnica de validación cruzada, dará cómo resultado el mejor modelo y sus parámetros de construcción.

La validación cruzada o cross-validation [24] es una técnica utilizada para evaluar los resultados de un modelo estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. Como podemos ver en la Figura 2.3, la misma consiste en un procedimiento iterativo, en el cual se calculan las métricas de evaluación sobre diferentes particiones, todas provenientes del mismo dataset. Los resultados de cada iteración son promediados en un resultado final y ésta será la métrica considerada para el modelo en cuestión.

Entre los parámetros que gridsearch puede ajustar sobre este tipo de clasificadores de manera automática se encuentra el denominado Parámetro de Margen Débil representado como C en los argumentos. Como intuitivamente su nombre lo indica, este indicador, dejará la posibilidad de que la frontera de decisión no sea tan estricta, tomando algunas

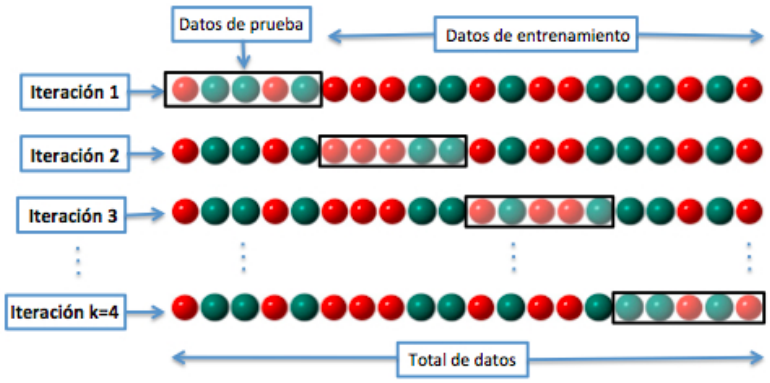

Figure 2.3: Procedimiento de validación cruzada con k iteraciones

observaciones etiquetadas con una clase, en el territorio de la otra.

2.6 Trabajos Previos

El presente trabajo es el resultado de la profundización y ampliación de dos trabajos previos[25][26]. Ambos tratan la predicción de tendencias en redes sociales. Un primer trabajo desde una perspectiva individual centrada en un usuario, mientras que en un segundo trabajo se hace foco en contenido, clasificándolo como popular o no. En esta sección repasaremos procesos e ideas fundamentales presentes en los trabajos anteriores, comenzando con una breve reseña de los conceptos y funcionalidades básicas a propósito de la red social Twitter, la escogida para llevar a cabo los experimentos.

2.6.1 Uso de Twitter

Twitter es una red social en la que sus usuarios pueden interactuar a través de *tweets*. Un tweet es una unidad de contenido que puede consistir en texto, imagen, video, ciertas alternancias entre éstas, o algunas otras formas de contenido multimedia. Dado un tweet un usuario puede darle un "me gusta", *retweetear* un tweet (es decir, re-publicar un tweet en el flujo de mensajes propio), retweetear agregándole un comentario, o simplemente responder a un tweet con otro tweet. Se crean así "hilos" de tweets al hacerse cadenas de respuestas. Cuando un una persona comparte un tweet, éste aparece en el *timeline* o cronología de usuario.

Los usuarios, por su parte, interactúan siguiendose los unos a los otros. El seguir a un usuario significa que los tweets (incluyendo retweets, ya que un retweet es también

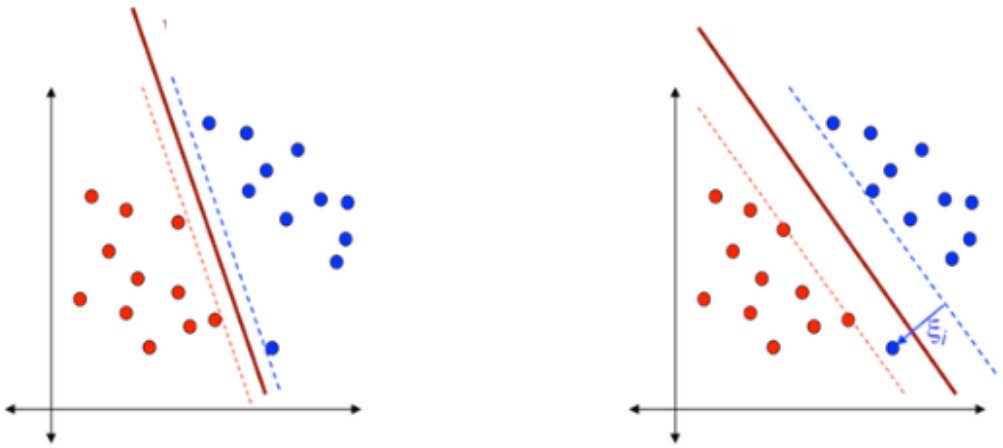

Figure 2.4: Muestra de cómo debilitando el margen se puede encontrar una frontera más amplia.

un tweet) de la seguida aparecerán en el *timeline* o cronología de inicio de el usuario que sigue. El *timeline* de inicio de un usuario es su vista inicial y la forma principal de interacción con la red social: en él aparece la actividad de los usuarios que sigue.

En este trabajo, siguiendo a los dos trabajos anteriores, consideraremos reelevantes como actividad en la red social:

- Las conexiones entre usuarios (relación seguidor-seguido). Dato sobre el cual se arman grafos dirigidos.

- La actividad de tweeteo y retweeteo de usuarios como forma de interacción entre los mismos.

2.6.2 Prediction of User Retweets Based on Social Neighborhood Information and Topic Modelling. [25]

Este trabajo propone la predicción de retweets para un usuario dado. Mediante el análisis del comportamiento de los usuarios allegados al objetivo, determina si éste hará o no *retweet* de un tweet dado.

Para cada usuario central sobre el que se trata de hacer la predicción, se elige un conjunto de usuarios reelevantes para tal usuario. Es entonces que, dado un tweet, las *features* para ese tweet y para ese usuario central, estarán dadas en base a si ese tweet fue retweeteado o no por cada uno de los usuarios en el conjunto de usuarios reelevantes. Éstas serán también llamadas *features sociales*, pues se trata de información del contexto social del usuario central (Ver sección 5.1.4).

Cabe aclarar que el trabajo explora mejoras empleando PLN y evalúa la performance de diferentes algoritmos de clasificación.

Siguiendo esta idea, en el trabajo se realizó el experimento para un conjunto de usuarios seleccionados. Se obtuvo un puntaje *F1* promedio de $0,84$ para un conjunto de usuarios bajo estudio.

2.6.3 Analyzing the Retweeting Behavior of Influencers to Predict Popular Tweets, with and without Considering their Content [26]

Este trabajo, al igual que el presentado antes (Ver 2.6.2), intenta encarar el problema de la predicción de tendencias en redes sociales y en particular en Twitter. Sin embargo, éste se diferencia en que no trata de hacer la predicción centrada en un usuario, sino que enfoca el problema en una comunidad.

Lo que se hace en este trabajo es, primero, encontrar una forma de discriminar tweets populares o no dentro de un universo reducido de usuarios, durante un período limitado. El objetivo es intentar predecir si un tweet va a ser popular o no en dicho universo. Para ello se caracterizan los tweets, en un primer momento, en base a features puramente sociales.

Dado el universo de usuarios, lo que va a caracterizar un tweet es si fue aceptado (retweeteado) o no por un subconjunto de usuarios. Se pretende, entonces que éste subconjunto (denominado *influencers*) sea determinante a la hora de evaluar si un tweet va a ser popular o no en este universo.

Cabe aclarar que el trabajo explora además mejoras incorporando análisis de Procesamiento de Lenguaje Natural, diferentes algoritmos de selección de usuarios influencers, entre otras.

Siguiendo esta idea de trabajo se obtuvo un puntaje *F1* que va desde los $0,6$ hasta los $0,87$, que crece a medida que crece la cantidad de influencers que se tienen en cuenta para los features.

Chapter 3

Herramientas

En este capítulo describimos brevemente las tecnologías empleadas en el desarrollo de este trabajo, para recolección y almacenamiento de datos, manipulación y análisis de grafos, análisis de datos en general, Aprendizaje Automático, Procesamiento de Lenguaje Natural y visualización.

3.1 Almacenaje y recolección de datos

3.1.1 Tweepy

Esta librería permite comunicarse desde Python con Twitter y usar su API (*Application Interface*). Provee acceso autenticado y permite acceder a todos los métodos que ofrece la API oficial de Twitter, facilitando extraer programáticamente información sobre usuarios, sus conexiones y su actividad. Se extendió esta librería con una funcionalidad de rotación de credenciales para facilitar la descarga de nuestros datos.

3.1.2 MongoDB

MongoDB es un sistema de manejo de base de datos. Es orientado a objetos, no relacional, dinámico y escalable. Los datos son guardados en colecciones en forma de unidades llamadas documentos. Estos documentos son objetos que disponen de variables cantidades de atributos, almacenados como clave-valor, y además pueden tener más objetos anidados dentro de sus atributos.

3.2 Procesamiento de grafos

3.2.1 NetworkX

NetworkX es una librería de Python para la creación, manipulación y estudio de la estructura, dinámica y funcionamiento de redes complejas (grafos). Permite leer y almacenar varios formatos de datos estándar y no estándar, generar varios tipos de redes clásicas y aleatorias, analizar estructuras, construir modelos, obtener medidas de centralidad y afinidad, entre muchas otras tareas.

3.2.2 Python-igraph

Adaptación a Python de una libreria popular para la manipulación de Grafos escrito en lenguaje R. Por la forma de su implementación, cuenta con una performance superior a NetworkX.

3.3 Pre-análisis y procesamiento de datos

3.3.1 Numpy, Pandas y Scipy

Numpy es una librería de Python que provee al lenguaje de mayor soporte para manipulación de vectores y matrices, entre otras cosas, como la confección de gráficos. Scipy es una librería de Python que expande a Numpy, proveyendo de más herramientas y algoritmos matemáticos. Pandas es una librería que extiende a Numpy simplificando la manipulación de algunas estructuras de datos complejas, sin perder eficiencia. Pandas hace *wrappers* que simplifican comportamientos y operaciones tabulares sobre matrices. Pandas funciona muy bien cuando nos toca trabajar con:

- Datos heterogéneos que pueden distribuirse de forma tabular.
- Series temporales
- Matrices

Se usó ampliamente para la extracción de datos sociales, debido a la posibilidad de escribir operaciones vectoriales y las ventajas en tiempo de ejecución y de desarrollo de prototipos. Numpy proveyó una buena cantidad de rutinas de cálculo matricial, mientras que scipy proveyó el uso de matrices dispersas o *sparse matrices*. Una *sparse matrix* es una matriz que posee una gran mayoría de elementos iguales a cero. Se almacena, no de forma entera (*dense matrix*) elemento por elemento, sino que se supone completamente vacía (con ceros) y se almacenan las posiciones de los elementos distintos a cero. Es así que si una matriz posee una gran cantidad de elementos con valor igual a cero, el uso de

sparse matrices trae un gran ahorro en memoria y disco en su manipulación con respecto a matrices densas.

3.3.2 Jupyter

Jupyter provee un entorno interactivo de desarrollo accesible desde un navegador web (notebooks), que permite combinar código Python, texto y visualizaciones. Se empleó para gran parte del ciclo de análisis y exploración de datos, y desarrollo de modelos; como así también en el procesamiento y visualización de resultados.

3.4 Aprendizaje Automático y PLN

3.4.1 Scikit-learn

Librería que proporciona un amplio conjunto de algoritmos de aprendizaje supervisado y no supervisado a través de una consistente interfaz en Python. Publicado bajo licencia BSD y distribuido en muchos sistemas Linux, favorece el uso comercial y educacional.

Esta librería se centra en el modelado de datos y no en la carga y manipulación de los mismos, para lo que utilizamos Pandas. Utilizamos scikit-learn para:

- Validación cruzada[24]
- Particionado de datos de entrenamiento y prueba
- Escalado y transformación de datos
- Entrenar Modelos de clasificación
- Ajuste de parámetros de modelos

3.4.2 NLTK y Spacy

NLTK (Natural Language Tool Kit)[36] es una librería de Python con algoritmos que facilitan tareas del procesamiento de lenguaje natural, fundamentalmente en el preproceso de los datos. La tokenización, limpieza de caracteres especiales son algunas de las funciones qué han sido de utilidad en este trabajo.

Spacy[37] es una libreria más completa en cuanto las tareas de PLN (Procesamiento del LenguajeNatural) que NLTK. Complementando los algoritmos que ofrece NLTK, suma la Lematización, y la posibilidad de diagramar pipelines de trabajo. Con un modelo preentrenado muy aceptable en casi todos los idiomas predominantes incluyendo entre ellos el Español, fue utilizado en nuestro caso para aplicar tokenización y remover texto no deseado como links, stopwords, etcétera y para experimentar con Lematización.

Chapter 4

Conjunto de datos

Como se dijo antes, este trabajo amplía y profundiza dos trabajos anteriores (Ver 2.6). Para permitir la comparabilidad con los resutados de trabajos anteriores, se mantuvo como red social predilecta a Twitter.

4.1 Usuarios y contenido

Siguiendo ambos trabajos, la selección de usuarios se hizo alrededor de un usuario central generador de un universo. Como se expone en ambos trabajos, el propósito era construir una red donde cada usuario tenga información lo suficientemente rica sobre sus seguidos. Para ello se siguió el procedimiento descripto a continuación, el cual consta de dos etapas.

En una primera etapa, comenzando con un usuario inicial u_0 y un conjunto $U_0 = \{u_0\}$, se realizan tres iteraciones al siguiente procedimiento: (1) Buscar todos los usuarios seguidos por usuarios en U_i; (2) De ese conjunto, mantener los usuarios que tengan más de 40 seguidores cómo así también más de 40 cuentas seguidas; (3) Incorporar el grupo resultante junto con sus relaciones al grafo extendido U_{i+1}. De este proceso obtenemos un universo $U := U_3$ con $2,926,181$ usuarios y $10,144,158$ relaciones.

En la segunda etapa, y para conseguir homogeneidad en información entre nuestros usuarios, se realizó el siguiente procedimiento:

- Se crea un subconjunto de usuarios semilla, S, determinado por aquellos nodos pertenecientes a U con valencia de salida igual a 50.

- Para cada uno de ellos se agregan a sus 50 usuarios más afines de entre sus seguidos. La afinidad entre dos usuarios se calcula como el *ratio* entre el número de usuarios seguidos por ambos y los seguidos por al menos uno de ellos.

- Se repite este último paso para cada nuevo usuario agregado, hasta que no hay nuevos usuarios por agregar.

Este procedimiento da como resultado un grafo G con $5,589$ nodos y $245,694$ aristas.

El conjunto de tweets, sin embargo, tuvo que ser completamente renovado, con respecto a los trabajos previos. Esto debido a que el trabajo aquí presentado requiere tener como dato de trabajo los momentos exactos en que un tweet fue publicado. Para los trabajos anteriores la temporalidad no fue considerada, por lo cual se decidió no persistir dicha información en medio alguno.

Se llevó a cabo entonces una recolección de datos nueva, esta vez guardando todos los datos aportados por la API de Twitter. Se trató de mantener las mismas condiciones de recolección en dos aspectos principales: usuarios involucrados y tiempo de recolección de tweets.

Para cada uno de los usuarios de nuestro universo se recolectaron: todos los tweets disponibles en la API de Twitter (200 tweets por usuario hacia el pasado a partir del 30/9/2018), y una recolección diaria durante el período de un mes (del 30/9/2018 al 30/10/2018). Se obtuvo así un contenido resultante de $15,633,833$ tweets, de los cuales seleccionamos los escritos en Español, contando así con $9,441,951$ tweets.

La recolección en bruto de los tweets fue almacenada en formato JSON en una base de datos MongoDB. La recolección se hizo con una tarea programada que, a traves de 9 cuentas rotativas, todos los días guardaron los tweets hechos el día anterior por todos los usuarios de nuestro universo. De esta forma se supera la limitación de la API de Twitter que permite recolectar una cierta cantidad fija de tweets por usuario, ignorando limitaciones de fechas. Para mejor manejo del conjunto de datos se exportaron los mismos desde MongoDB a *Comma Separated Values* (CSV), y JSON (JavaScript Object Notation): JSON para los textos de los tweets ya que un tweet puede contener caracteres arbitrarios que deben ser parseados con cuidado, y CSV para el resto de los datos, que en nuestro trabajo son: idioma, identificador de tweet, momento exacto de publicación del tweet (timestamp), identificador del tweet original (en caso que sea un retweet), y momento de creación de tweet original (en caso que sea un retweet).

4.2 Ventana de tiempo de retweeteo (Retweet-Time-window)

Como se mencionó antes, la idea principal de este trabajo es reproducir el trabajo previo agregando la idea de ventanas de tiempo. Para ello, para cada tweet "original" (es decir, un tweet que no es un retweet de ningún tweet previo) tomamos ventanas de tiempo de diferentes longitudes, siempre tomando como punto de inicio de la ventana al momento de publicación de cada tweet original. Sea θ_0 el momento exacto de publicación de un tweet original t por su autor, la usuaria u, decimos que θ_0 es el timestamp de inicio de la ventana de tiempo. Entonces, para cada segmento de tiempo ω, definimos la *retweet-time-window* (o *ventana de tiempo de retweeteo*), para el tweet t y longitud ω como el

conunto de todos los retweets te t hechos por otrx usuarix, que tomó lugar antes del momento $\theta_0 + \omega$. Definiremos ademas $\mathbf{RTW}(\omega)$ que es la unión de todos los retweet-time-windows de longitud ω. Formalmente, $\mathbf{rtw}(t, \omega) := \texttt{retweets}(t) \cap \mathbf{T}^{\texttt{timestamp}(t)+\omega}$ and $\mathbf{RTW}(\omega) := \bigcup_{t \in \mathbf{T}_o} \mathbf{rtw}(t, \omega)$, donde $\texttt{timestamp}(t)$ es el momento de publicación de t, $\texttt{retweets}(t)$ son todos los retweets que t tuvo entre las usuarias de nuestro grafo $\mathcal{G}$ y $\mathbf{T}_o$ es el conjunto de todos los tweets originales en $\mathbf{T}$.

Selección de usuarios

Los usuarios inactivos van a ser omitidas en estos experimentos porque son impredecibles por naturaleza. Consideramos que un usuario en nuestro dataset es pasivo si tiene menos de 10 retweets en su timeline. Dejando de lado a esos usuarios, nos deja con un conjunto de $3,911$ usuarios activos en $\mathcal{G}$. Restringimos el análisis a tales usuarixs, removiendo además el contenido compartido por tales usuarios inactivos.

Chapter 5

Modelos predictivos

En esta sección describiremos el desarrollo y los resultados de los experimentos llevados a cabo. Dado que se trata de la unión de dos trabajos previos (Ver sección 2.6), separaremos la exposición en dos secciones.

5.1 Modelo predictivo de preferencias de retweeteo

En esta sección extenderemos los experimentos del trabajo previo, [25], para diferentes *ventanas de retweeteo* (o retweet-time-windows), usando el nuevo conjunto de datos descripto anteriormente. Explicaremos primero cómo construir el modelo predictivo. Luego describiremos cómo elegir los usuarios para la predicción y cómo construir todos los datasets necesarios. Terminamos la sección presentando los resultados, describiendo además la performance y su variación predictiva, dependiendo del tamaño de la ventana de tiempo tomada.

5.1.1 Configuración del experimento

En este modelo deseamos predecir, para un usuario dado u y un tweet dado t, si u va a compartir t basándonos en información sobre qué usuarios en el entorno de u retweetearon t *hasta cierto momento*. Dado que el proceso de extracción de features, modelado y selección de parámetros es computacionalmente caro, estos experimentos son llevados a cabo sobre un subconjunto selecto de usuarios. Comenzamos describiendo el criterio sobre el cual los usuarios fueron seleccionados. Luego describiremos cómo generamos, para cada usuario u el vecindario de usuarios E_u y un conjunto T_u de tweets potencialmente interesantes. Luego, describimos el proceso de extracción de features basado en T_u y el particionado en conjuntos de entrenamiento, tunning y prueba. Finalmente, explicamos el proceso de entrenar clasificadores y ajuste de parámetros.

5.1.2 Elección de usuarios de prueba

El proceso de extracción de características y ajuste de parámetros es computacionalmente costoso. A su vez, es deseable centrar los análisis en usuarios donde haya una buena cantidad de ejemplos positivos de los que aprender y a su vez un entorno E_u suficientemente grande para proveer buena información sobre los mismos.

Siguiendo a [25], pero con nuevos datos, tomamos 1000 usuarios con el mayor coeficiente de centralidad de Katz [6] en el grafo G y, por otro lado, los 1000 usuarios con mayor actividad de retweeteo. Restringimos nuestro análisis a usuarios que pertenezcan a ambos conjuntos, lo cual deja un conjunto de 211 usuarios. Comparando con el trabajo predecesor a este, encontramos resultados similares: un conjunto de usuarios de prueba de 194.

Es importante aclarar que en los experimentos, nuestro universo de usuarios sigue siendo G, con todos sus usuarios y conexiones. G es usado para generar el entorno de cada usuario en U, cuya preferencia de retweeteo estamos tratando de predecir.

5.1.3 Tweets visibles

La API de Twitter no da información explícita sobre si un usuario vió un tweet dado, pero podemos al menos tomar un universo de tweets *potencialmente vistos*. Esto es simplemente el conjunto de todos los tweets escritos o compartidos por los usuarios seguidos por u.

Excluimos de este conjunto aquellos tweets escritos por u, dado que nuestro objetivos es reconocer contenido interesante ajeno a u, y no su producción de contenido. Formalmente este conjunto se define como:

$$T_u := \left(\bigcup_{x \in \{u\} \cup followed(u)} timeline(x) \right) - \{t \in T | author(t) = u\} \tag{5.1}$$

, donde $followed(u) := \{x \in G | (u, x) \in follow\}$ y $timeline(x)$ es el conjunto de todos los tweets escritos o compartidos por x, dentro de nuestro conjunto de tweets recolectados.

Adicionalmente, necesitamos definir los subconjuntos de tweets que son visibles bajo una ventana de tiempo de longitud ω. Formalmente: $\mathbf{RTW}_u(\omega) := \{t \in T_u | t \in \texttt{timeline}(x, \texttt{timestamp}(t)+\omega)$ para algún $x \in \{u\} \cup \texttt{followed}(u)\}$, donde $\texttt{timeline}(x, \theta)$ es el conjunto de todos los tweets en $\texttt{timeline}(x)$ que fueron escritos o compartidos por x hasta el momento θ.

Para algunos usuarios el conjunto T_u resultó ser demasiado grande, haciendo al experimento muy costoso computacionalmente. Decidimos entonces recortar cada T_u a un máximo de $10,000$ tweets. Dado que los tweets retweeteados son la clase mayormente minoritaria, decidimos mantener todos los ejemplos positivos y hacer el recorte, con selección aleatoria, sobre la clase negativa.

5.1.4 Entornos de usuario

Dado que un usuario u puede ver solamente los tweets compartidos por aquellos usuarios que sigue, la información sobre su red extendida puede proveer más información sobre el grado de interés en un tweet t. Es por ello que decidimos tomar de un entorno de usuarios, no solo los usuarios que sigue, sino extender el mismo un paso más en la relación $follow$ incluyendo los usuarios seguidos por los usuarios seguidos por u. Esto es, tomamos todos los usuarios (que no sean u) en 1 o 2 pasos desde u en el grafo dirigido G. Formalemente:

$$E_u = \left(\bigcup_{x \in \{u\} \cup followed(u)} followed(x) \right) - \{u\} \tag{5.2}$$

Información social de usuarios

Con estas definiciones podemos construir ahora el conjunto de features y los vectores objetivo necesarios para el modelo predictivo centrado en un usuario u con una ventana de tiempo ω. Dado $E_u = \{u_1, u_2, \ldots, u_n\}$, definimos para cada tweet $t \in \mathbf{RTW}_u(\omega)$ el siguiente vector booleano de features:

$v_u^\omega(t) := [\texttt{tw\_tl}(t, u_i, \omega)]_{i=1,\ldots,n}$,

donde $\texttt{tw\_tl}(t, u, \omega) := \begin{cases} 1 & t \in \texttt{timeline}(u, \texttt{timestamp} + \omega(t)) \\ 0 & \text{otherwise} \end{cases}$

Notar que el contenido del tweet t no es considerado, sino que solo incluimos información de quién retweeteó t dentro de la ventana de tiempo de longitud ω que comienza en la creación del tweet original.

Finalemente el vector objetivo $y_u(t)$ es un vector booleano que indica, para cada tweet $t \in \mathbf{RTW}_u(\omega)$, siempre que el usuario u ha retweeteado t.

Separación conjuntos de entrenamiento, evaluación y ajuste

Como es habitual para problemas de aprendizaje automático, conjuntos de entrenamiento y evaluación deben estar disponibles. Construimos además un conjunto de ajuste, para validar los modelos usando datos no vistos, pero que sean al mismo tiempo diferentes del conjunto de entrenamiento.

Para ello, decidimos separar de forma aleatoria T_u en 3 datasets diferentes para cada usuario u: entrenamiento (T_u^{tr}), ajuste (T_u^{tu}), y evaluación (T_u^{ev}). Los subconjuntos resultantes tienen 70%, 10% y 20% respectivamente, sobre el dataset T_u. Sean y_u^{tr}, y_u^{tu} y y_u^{ev} los vectores objetivos correspondientes para cada dataset. Sean ahora $T_u^{tr}(\omega)$, $T_u^{tu}(\omega)$, $T_u^{ev}(\omega)$ los conjuntos previamente definidos de entrenamiento, ajuste y evaluación con la información de la retweet-time-window de tamaño ω, como la definimos en la sección previa (ver . A continuación mostramos la información de retweeteo disponibles en $T_u^{tr}(\omega)$ para diferentes valores de ω. En la lista que mostramos a continuación, se pueden ver los

porcentajes de retweets disponibles contra el total de retweeteos para el dataset completo T_u^{tr} sin ventanas de tiempo. Estos porcentajes son promedios entre nuestros usuarios de prueba, que tienen una actividad de retweeteo total de $4,468.38$ en T_u^{tr}.

1s	2s	10s	30s	2m	5m	15m	30m	60m	90m	120m	240m	540m
0%	0%	0.1%	2.18%	8.72%	15.93%	27%	35.47%	44.89%	51%	55.66%	66.72%	78.28%

Información de análisis de contenido

Para analizar el contenido de cada tweet, decidimos usar la implementación fastText [19] de *word embeddings*. Una de las características más interesantes para nuestro trabajo es la posibilidad de asignar vectores a palabras no vistas en la vectorización del conjunto de entrenamiento, buscando coincidencias a nivel de n-gramas de caracteres para vectorizar palabras fuera del vocabulario. Esto hace a nuestra vectorización más robusta para manejar palabras con errores de tipeo que son comunes en redes sociales. Usamos un modelo de 300 dimensiones pre-entrenado, incluido en la librería fastText [1]. A pesar de que fastText solo provee representación de vectores de palabras, un documento (conjunto de palabras) puede ser fácilmente obtenido tomando promedios de los vectores de las palabras que lo componen. En el trabajo previo de predicción de los retweets de un solo usuario [25], no se empleó fastText, pero en la sección 5.1.5, reportamos tales experimentos.

5.1.5 Resultados

Analizaremos ahora los resultados obtenidos de entrenar y evaluar los modelos centrados en un usuario usando los vectores de features descriptos en las secciones precedentes. Comenzamos con el modelo social puro (solo información social) y concluiremos la sección con los modelos que además consideran el contenido de los tweets.

Predicción Social

Para el modelo social centrado en un usuario usamos SVC [2], con `GridSearchCV` para la optimización de hiperparámetros.

Luego, para cada usuario u en nuestro conjunto U de usuarios de prueba, evaluamos el clasificado obtenido en el conjunto de prueba, obteniendo los resultados que se pueden ver en la Tabla 5.1. Luego, consideramos diferentes ventanas de retweet-time-window, ($\omega \in \{2, 5, 15, 30, 60, 120, 240, 540\}$ minutos).

Este análisis busca responder la pregunta de cuánto tiempo es necesario para predecir

[1] `https://fasttext.cc/docs/en/crawl-vectors.html`

[2] `Support Vector Classifier`, que es el nombre dado en la librería de Python `scikit-learn` a las `Support Vector Machines` (SVM)

Modelo Social				Social + fastText			
time	**Av. F_1**	**Av. Prec.**	**Av. Rec.**	**time**	**Av. F_1**	**AV. Prec.**	**Av. Rec.**
2m	53.76	43.18	80.09	2 m	55.38	47.95	78.65
5m	60.45	45.01	93.42	5 m	62.05	48.82	92.17
15m	63.76	49.32	91.45	15 m	64.47	51.43	90.36
30m	66.29	52.93	89.83	30 m	66.89	53.61	89.94
60 m	69.08	57.19	88.35	60 m	70.05	58.04	88.60
90 m	70.64	59.96	87.27	90 m.	71.36	60.57	87.82
120 m	72.07	62.43	86.45	120 m	72.61	62.83	86.91
240 m	75.2	68.24	84.98	240 m	75.52	69.02	84.98
540 m	78.65	75.96	82.61	540 m	78.92	76.28	82.61
n. w.	86.08	94.56	80.17	n. w.	86.24	93.78	80.93

Table 5.1: Performance del modelo social sobre el conjunto U de usuarios seleccionados.

con exactitud el comportamiento de retweeteo del usuario u. Este conjunto de experimentos muestra que, por un lado, cuando no se usa ventana de tiempo, el score F_1 obtenido es 86.08; este resultado es comparable a los resultados arrojados en trabajos previos [25] de 87.68. Por otro lado, al analizar los resultados obtenidos para los diferentes valores de ω, vemos que con tan solo 60 minutos de información, comenzamos con una performance alta de 69.08. De estos resultados podemos decir que 60 minutos son suficientes para asegurar una predicción aceptable.

Agregando información de contenido

Como vimos en los resultados de la Tabla 5.1, agregar features de contenido incrementa apenas la performance entre 0.15 y 1.62% para tamaños grandes de ventana. Sin embargo, el contenido juega un papel más significativo en la performance de las predicciones para ventanas más pequeñas. Es este un resultado interesante, dado que en la presencia de poca información social, por ejemplo 2 minutos de información, las features de contenido mejoran el score F_1 en 1.5. Es importante notar que el los trabajos previos [25] fastText no fue usado.

Nos interesa resaltar que en el presente trabajo, cuando se combinan los modelos sociales con fastText, apenas se obtiene una mejora de 0.15% en el score F_1, relativo al modelo puramente social. En el trabajo previo, la mejora conseguida con el modelo Twitter LDA, combinado con el modelo social, no alcanzó al 0.3%.

5.2 Modelo predictivo de tendencias

En esta sección, explicaremos cómo extender los experimentos del trabajo previo en [26], para diferentes retweet-time-windows usando el nuevo dataset descripto en la sección 4.

5.2.1 Configuración del experimento

Esta sección descriremos cómo construir modelos capaces de predecir con precisión la aceptación que un tweet t puede tener sobre una audiencia general de usuarios ($U_G \subset \mathcal{G}$), basándonos solamente en la reacción de un conjunto de influencers ($U_I \subset \mathcal{G}$) a la publicación. Además, mostraremos cómo configurar los modelos para este propósito sobre una selección de usuarios y tweets del dataset ($\mathcal{G}, \mathbf{T}$) definido antes.

De forma similar a la sección anterior, comenzaremos con modelos predictivos basados solo en features sociales, para luego extenderlos usando features de word embeddings.

Predicción social

El foco de atención de el presente trabajo es predecir si un tweet t va a tener la suficiente cantidad de retweets de un conjunto general de usuarios, para ser considerado como tendencia (*trending*), basandonos la información de cuáles de los influencers de U_I ha compartido la publicación hasta determinado momento.

Comenzamos esta sección con una explicación de nuestro proceso de selección de usuarios y tweets relevantes para este trabajo. A continuación, detallaremos cómo proceder para obtener un conjunto de influencers U_I de $\mathcal{G}$ y qué algoritmos usamos para tal propósito. Finalmente, explicaremos el proceso de extracción de features y la división del conjunto de datos para entrenamiento y evaluación de los modelos, evitando la superposición de datos al realizar dichas tareas.

Tweets que son tendencia (trending)

Diremos que un tweet es tendencia (*trending*) si lo consideramos lo suficientemente popular como para convertirse en un Trending Topic. Este término está relacionado a la cantidad de retweets que consigue sobre un público general de usuarios U_G. Para conseguir nuestro valor de oro (*golden value*) de retweeteo, entendiéndolo como la cantidad de retweets que una publicación debe tener para ser considerado popular, construimos un histograma de cuántos retweets tuvo cada tweet en $\mathbf{T}$.

Inicialmente, deseabamos usar el valor en el percentil 90 como nuestro valor de oro, pero dado el hecho que la gran mayoría de los tweets son compartidos sólo por su autor, este valor resultó ser igual a 1. Decidimos entonces descartar todos los tweets con menos de 3 retweets, lo cuál hizo que dicho percentil crezca a 14, permitiéndonos implementar modelos más precisos.

Es así que consideramos que un tweet es popular si fue retweeteado al menos 14 veces en nuestro dataset[3]

Es importante remarcar que los experimentos llevados a cabo tienen sentido sólo en el contexto de usuarios U_G, teniendo en mente que el objetivo de deste trabajo es analizar la influencia de un conjunto de usuarios U_I sobre un conjunto general de usuarios. Es esta la razón por la cual nos interesan sólo los tweets de $\mathbf{T}$ que aparecen al menos una vez en el *timeline* de al menos un usuario en U_G, definiendo así:
$T' := \left(\bigcup_{x \in U_G} timeline(x)\right)$.

Detección de influencers

Siguiendo a los trabajos previos, decidimos usar las ideas incluidas en [43], donde se propone una combinación de tres tipos de features: centralidad en la red, nivel de actividad y características de perfil.

Dado que no poseemos información sobre perfiles en nuestro dataset, nos concentramos en la centralidad y la actividad. Esto tuvo la ventaja de hacer los resultados más generalizables a otras redes sociales sin depender de información específica que puede estar disponible sólo en Twitter, y para algunos usuarios.

Para medir la *centralidad* de un usuario, sacamos un promedio de métricas calculadas con los siguientes algoritmos, todos incluidos en el paquete de Python `igraph`[34]:

- PageRank
- Betweenness
- Closeness
- Eigenvector centrality
- Eccentricity

El nivel de *actividad* de un usuario es calculado simplemente como un promedio entre el número de tweets y el número de retweets publicados por un usuario.

Siguiendo la misma línea que el trabajo previo, para decidir la mejor forma de clasificar usuarios como influencers, comparamos diferentes combinaciones entre centralidad y actividad, $\alpha * Centrality + (1 - \alpha) * Activity$, donde α controla la importancia que se le da a la *centralidad*. Encontramos que un valor de $\alpha = 0.5$, es la mejor opción. Para comparar la performance de estas opciones un subconjunto aleatorio de 500 tweets de $\mathbf{T}$ fue separado, como un conjunto de ajuste. Este subconjunto llamado T_{SI} es removido de $\mathbf{T}$ para evitar considerarlo como parte del conjunto de evaluación, donde los modelos de predicción de tendencia serán evaluados más adelante.

[3]En el trabajo previo a este [26] dicho valor resultó ser 13.

Este análisis revela que un usuario muy central no tiene mucho uso para este estudio si tiene un bajo nivel de actividad y, similarmente, un usuario muy activo no tiene valor como influencer si no está lo suficientemente conectado a la red de usuarios. La comparación de estos resultados indica que la mejor opción para medir el nivel de influencia de los usuarios es un promedio entre centralidad y actividad.

Ahora que seleccionamos nuestra métrica, la aplicamos a $\mathcal{G}$ sin los 500 tweets de T_{SI}, para obtener un ranking de todos los usuarios según su nivel de influencia. Tomamos el 25% con mayor influencia como nuestro conjunto de *influencers* y lo llamamos U_I, llamando a su vez al conjunto restante de usuarios U_G, es decir, nuestra población general de usuarios.

El objetivo de los modelos sociales descritos antes es predecir el nivel de aceptación que un tweet tiene sobre una población general. U_G, basándonos en el conocimiento de la actividad de los influencers U_I. La idea de estos experimentos descriptos en las siguientes secciones es variar el número de influencers tomados de U_I para predecir la popularidad de los tweets.

Información social

Como se mencionó antes, necesitamos entrenar nuestro modelo de clasificación para hacer predicciones. Para lograr esto, es necesario definir vectores de features y el vector objetivo a predecir. Para el vector de features, en el modelo social, sólo consideramos la actividad de retweeteo de nuestro conjunto de influencerssobre los tweets de nuestro conjunto de entrenamiento.

Para cada tweet t, podemos definir un vector binario
$V_t^\omega := \begin{bmatrix} i_{t1} & i_{t2} & \dots & i_{tn} \end{bmatrix}$
donde n es el número de influencers, y cada i_{tj} es igual a 1 si el tweet t fue retweeteado por el influencer j antes de $\texttt{timestamp}(t) + \omega$, y 0 en caso contrario.

Formalmente, supongamos que queremos hacer predicciones con la información disponible hasta el momento ω después de que el tweet t es creado. Luego, si j es un influencer $\mathbf{RTW}_j(\omega)$ devuelve el conjunto de m tweets en el timeline de j hasta el momento $\texttt{timestamp}(t) + \omega$. Al agrupar en una matriz todos los vectores asociados con los m tweets, la entrada al modelo predictivo es, para cada fila t:

$$features_t := \begin{bmatrix} i_{t\,1} & \dots & i_{t\,j} & \dots & i_{t\,n} \end{bmatrix}_t \text{ donde } i_{tj} = \begin{cases} 1 & \text{si } t \in \mathbf{RTW}_j(\omega) \\ 0 & \text{caso contrario} \end{cases}$$

Notar que el contenido de cada tweet t no es considerado, sino que solo incluimos la información sobre qué usuario en U_I retweeteó t. Ahora, como parte del modelo supervisado, usamos el siguiente vector objetivo, calculado sobre el conjunto de entrenamiento de tweets. Sea $R^\omega(t)$ una función que devuelve el número de retweets que el tweet t tuvo en U_G hasta el momento $\texttt{timestamp}(t) + \omega$; definimos el vector objetivo de la siguiente manera:

$$classification = \begin{bmatrix} r_1 \\ \dots \\ r_m \end{bmatrix} \quad \text{donde} \quad r_t = \begin{cases} 1 & R^{\omega}(t)(s) >= \textit{valor de oro} \\ 0 & \text{caso contrario} \end{cases}$$

División del dataset

Para evaluar la performance de nuestros modelos, dividimos nuestro dataset de tweets en 2 partes, una para entrenamiento y la otra para evaluación. Como es usual, estos conjuntos de datos no se superponen. En otras palabras, el conjunto de evaluación no es visto por los algoritmos de entrenamiento.

Sin importar el número de influencers elegido para la predicción, queremos al conjunto de entrenamiento y de evaluación que permanezcan disjuntos. Como lo explicamos antes en esta sección particionamos el conjunto $\mathcal{G}$ en dos partes disjuntas, U_I (influencers) y U_G (usuarios generales).

Para los experimentos restantes en este trabajo, U_I se define como el 25% de usuarios mejor rankeados de $\mathcal{G}$, usando el promedio de centralidad y actividad para detectar los influencers. Notar que al dividir el conjunto de datos, para hacer el conjunto de evaluación fijo, al variar el número de influencers tomado para hacer predicciones.

Para determinar que los conjuntos de entrenamiento y evaluación están bien formados, dejamos de lado de $\mathbf{T}$ los tweets publicados por usuarios en U_I, llamados T_I. Además es necesario sacar de $\mathbf{T}$ el conjunto T_{SI} usado previamente en esta sección para detectar influencers. Los tweets restantes, i.e. $T_G = \mathbf{T}' - T_I - T_{SI}$ son particionados nuevamente. Para ello T_G es particionado de forma aleatoria en conjuntos de entrenamiento T_G^{train} (75%) y evaluación T_G^{test} (25%).

Sean ahora $T_G^{train}(\omega)$ y $T_G^{test}(\omega)$ los conjuntos definidos antes de entrenamiento y evaluación, con la información de la retweet-time-window ω. A continuación mostramos la información de retweeteo disponible en $T_G^{train}(\omega)$ para diferentes ventanas ω. En la siguiente lista, mostramos el porcentaje de retweets disponibles sobre el total de retweets para el conjunto T_G^{train} sin ventanas de tiempo:

1s	2s	10s	30s	2m	5m	15m	30m	60m	90m	120m	240m	540m
0%	0%	0.04%	0.7%	3%	7.7%	15.1%	21.1%	29.3%	35.7%	40.4%	53.6%	68.5%

Agregando información de contenido

Para conseguir un incremento en la calidad de predicción de tweets populares, aplicamos técnicas de NLP para extender el modelo social puro con features de contenido, en forma similar a la sección anterior.

5.2.2 Resultados

Describiremos ahora cómo construimos nuestros modelos predictivos y los resultados obtenidos con y sin análisis de contenido para diferentes retweet-time-windows. Vamos a comparar nuestros modelos contra un modelo de referencia construido de un modelo social puro donde los usuarios considerados influencers son elegidos aleatoriamente en vez de usar nuestro algoritmo de selección de influencers.

Con esto queremos mostrar la utilidad de usar el algoritmo de detección de influencers, y la información relevante que ellos proveen para aprender sobre el comportamiento de los usuarios generales. Los experimentos fueron ejecutados en dos servidores de 14 cores CPU E5-2680 v4 2.40GHz, con 128Gb of RAM.

Modelo de referencia (Baseline)

Como modelo de referencia usamos un modelo lo suficientemente demandante para ser comparado en nuestros propósitos. Decidimos usar el mismo tipo de features que en el modelo social puro, pero eligiendo de forma aleatoria un 25% de usuarios de $\mathcal{G}$ como nuestro conjunto de influencers U_I.

Para hacer una comparación justa entre nuestros modelos hacemos una nueva división de nuestro dataset $\mathbf{T}$ en T_I y T_G con el contenido de usuarios en la selección aleatoria de U_I y U_G respectivamente. Asimismo, se hace una división de $75\% - 25\%$ para entrenamiento-evaluación sobre T_G para los modelos de referencia, bajo las mismas condiciones en el modelo social puro. Mantenemos los datasets disjuntos y evaluamos sobre la población general de usuarios con el comportamiento de los influencers como entrada.

Siguiendo el mismo patrón que en los otros modelos sociales, procedemos a evaluar el modelo social de referencia sobre un número cada vez más grande de usuarios de U_I como fuente de features sociales. En este caso no disponemos de un ranking de usuarios para elegir, sino que hacemos elecciones al azar.

Para calcular la permormance del modelo de referencia, para cada cantidad diferente de influencers (llamemosle k un valor), elegimos k usuarios diferentes de U_I, y entrenamos y evaluamos un modelo usando la división de entrenamiento-evaluación de T_G.

Para evitar golpes de suerte y resultados potencialmente engañosos, repetimos este proceso cinco veces para cada valor de K, reportando el valor promedio del score $F1$.

Modelos combinados y sociales

Mostraremos ahora los resultados obtenidos de entrenar y evaluar los modelos de predicción de tendencia con las features descritas en la sección 5.2.1. Usamos modelos de Support Vector Machine para la clasificación, más precisamente el `SVC`.

Decidimos hacer foco en los experimentos considerando al 10%, 15% y 20% de $\mathcal{G}$ como influencers. Probamos los modelos para las time-retweet-window de tamaño $\omega \in \{15, 30, 60, 90, 120, 240\}$ en minutos, sumando además el estudio sin ventana, es decir con

time	Top-10% inf.				Top-15% inf.				Top-20% inf.			
	FT	**Base .**	**Soc.**	**S+FT.**	**FT**	**Base .**	**Soc.**	**S+FT.**	**FT**	**Base .**	**Soc.**	**S+FT.**
15m		25.62	46.72	53.06		39.12	52.83	57.32		42.83	58.44	63.95
30 m		32.22	61.54	64.61		39.23	62.91	66		42.15	64.54	67.93
60 m		23.38	64.61	68.98		32.06	65.67	69.47		48.66	67.87	71.21
90 m		32.38	65.17	69.78		32.51	68.68	70.32		47.01	70.77	73.12
120 m	50.71	35.72	69.78	73.18	50.71	39.62	71.66	74.73	50.71	50.08	73.66	75.53
240 m		39.2	73.7	75.99		45.21	74.4	76.32		53.96	76.43	78.45
540 m		43.46	75.25	80.66		51.38	77.51	82.27		56.23	79.93	84.07
n.w.		46.06	77.85	82.69		54.31	80.73	84.72		57.44	82.22	89.17

Figure 5.1: Performance en score F_1 para todos los modelos, evaluados sobre U_G, usando el top 10%, 15% y 20% de usuarios como influencers. Columnas: FT=fastText Model ; Base = Baseline; Soc = Modelo social; S+FT = modelo Social y fastText combinado.

todos los retweets recolectados en el dataset.

En la tabla 5.1, reportamos los resultados para todos los modelos evaluados usando el score $F1$. Los modelos reportados son fastText (**FT**), el modelo de referencia descripto en la sección anterior (**Base**), social **Soc.**), y finalmente el modelo que combina el modelo social con fastText (**S+FT**).

En la tabla se puede observar que los resultados obtenidos en el trabajo anterior [26], sobre otro conjunto de datos es comparable para el modelo social con 10% de influencers ($F_1 = 79.2$ en el trabajo previo contra 77.85 en este trabajo). En todos los casos, el modelo social se desempeña mejor que el baseline.

Los resultados muestran además que la menor ventana para obtener predicciones razonables con modelos puros sociales es alrededor de los 120 minutos, tomando 10% de los usuarios como influencers, y que este número puede ser reducido a 60 minutos con 20% de influencers.

Combinamos luego el modelo social con los vecores de 300 dimensiones del modelo fastText descripto en secciones previas. En la tabla 5.1, columna **S+FT.** , podemos ver que los mejores resultados son obtenidos combinando el 20% de los influencers para entrenar el modelo social con fastText. Esta combinación logra un score $F1$ de 89.17. Los resultados obtenidos del modelo combinado muestran que, un ω entre 30 y 60 minutos, logra un score $F1$ por sobre el 70%.

Chapter 6

Conclusiones y Trabajo Futuro

Como conclusión general, confirmamos que la información social de las conexiones entre usuarios de Twitter y las primeras horas de actividad sobre un tweet, pueden ser esenciales para determinar la preferencia del tweet sobre un usuario en particular o sobre si ese tweet se va a hacer popular entre un público general de usuarios. En ambos casos obtuvimos una buen performance sin análisis de contenido.

En el caso particular de predecir la preferencia de un usuario, con 60 minutos de información obtuvimos una performance de alrededor de 69.08%, y luego de agregar información sobre el contenido, la performance crece levemente a 70.05%. Observamos que considerando más tiempo del comportamiento, el modelo social obtiene mejores niveles de performance que son levemente incrementados al ser combinados con las características de contenido de fastText.

Para el caso de predicción de popularidad, considerando el top 20% de los usuarios más centrales y activos como influencers, necesitamos de 90 minutos de información sobre su comportamiento para obtener scores F_1 mayores a 70%. Si extendemos el modelo con análisis de contenido, estos niveles de performance se logran más temprano, usando entre 30 y 60 minutos de información.

En todos los casos, los resultados parecen sugerir que la fuente de dónde proviene la información tiene una mayor influencia que el contenido cuando se trata de esparcirlo através de la red. El score del modelo basado solamente en contenido estuvo muy por debajo del score del modelo basado solamente en información social, lo cual refuerza la idea de que aveces nuestra lista de contactos da más información sobre nosotros que nuestro timeline.

En cualquier caso, los modelos combinados de información social y análisis de contenido incrementan la performance significativamente, lo cual indica que el contenido todavía tiene un nivel de importancia cuando se lo considera dentro de un determinado contexto social, especialmente temprano en el tiempo de vida de los tweets, cuando la información social sobre los mismos es todavía limitada. fastText parece estar particularmente adecuado para lidiar con análisis de contenido de Twitter, en gran parte debido a su habilidad de

obtener representaciones para palabras no vistas antes o con errores de ortografía.

Esta investigación abre varias puertas para desarrollar aún más el modelo. Las más relevante para nosotros son descriptas a continuación.

Uno de los experimientos a seguir es agregar features que tengan en cuenta el tiempo transcurrido luego de la publicación del tweet original. Otra linea de investigación es tratar de cambiar los modelos de SVM con modelos de Deep Learning. Además tenemos la idea de usar redes neuronales Long Short-Term Memory (LSTM), para representar la actividad de los tweets como una serie temporal.

Para la detección de influencers, alternativas como [42] y [43] podrían aplicarse para mejorar la selección de usuarios relevantes.

Proponemos además la investigar la fórmula de agregación para embeddings de oraciones o documentos. Usamos un simple promedio de vectores de las palabras que los componen, sin embargo existen otras funciones más sofisticadas, como el promedio pesado por la Frecuencia Inversa de Documento (Inverse Document Frequency [44]).

Finalmente, una interesante línea de investigación es tratar de replicar los experimentos para otras redes sociales como Facebook e Instagram, y ver hasta qué punto nuestras conclusiones se aplican a aquellas. En particular, el modelo social puro puede extenderse a cualquier red de usuarios que compartan contenido, lo cual lo hace posible de evaluar incluso en redes basadas en imágenes como Instagram. Sin embargo estamos limitados por la disponibilidad de información para construir datasets.

Bibliography

[1] Celayes Pablo, Dominguez Martin Recomendación de Información basada en Análisis de Redes Sociales y Procesamiento de Lenguaje Natural

[2] Freeman, Linton A set of measures of centrality based on betweenness

[3] Brin, Sergey and Page, Lawrence The Anatomy of a Large-Scale Hypertextual Web Search Engine

[4] Bryan, K., Leise, T. The 25,000,000,000 eigenvector: the linear algebra behind google. SIAM Review 48, 569–581 (2006).

[5] Buckley, F., Harary, F Distance in graphs. Addison-Wesley (1990)

[6] Katz, L. (1953). A New Status Index Derived from Sociometric Analysis. Psychometrika, 39–43.

[7] Sabidussi, G The centrality index of a graph. Psychometrika 31(4) (1966)

[8] Rosvall, M. and Bergstrom, C. T. An information-theoretic framework for resolving community structure in complex networks. Proceedings of the National Academy of Sciences 104, 7327–7331 (2007).

[9] Peel, L. Estimating network parameters for selecting community detection algorithms. In 13th Conference on Information Fusion 1–8 (IEEE, 2010).

[10] Mukherjee, A., Choudhury, M., Peruani, F., Ganguly, N., Mitra, B. Dynamics On and Of Complex Networks, Volume 2: Applications to Time-Varying Dynamical Systems (Springer Science and Business Media, 2013)

[11] Blondel, V. D., Guillaume, J.-L., Lambiotte, R. and Lefebvre, E. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008, P10008 (2008).

[12] Xie, J., Szymanski, B. K. Community detection using a neighborhood strength driven label propagation algorithm. In Network Science Workshop 188–195 (IEEE, 2011).

[13] Pons, P., Latapy M.: Computing communities in large networks using random walks. In Computer and Information Sciences-ISCIS 2005, 284–293 (Springer, 2005).

[14] Xie, J., Szymanski, B. K.: Community detection using a neighborhood strength driven label propagation algorithm. In Network Science Workshop 188–195 (IEEE, 2011).

[15] Rousseeuw, P.J.; Kaufman, L. (1990). Finding Groups in Data: An Introduction to Clúster Analysis. Wiley.

[16] David M. Blei, Andrew Y. Ng, Michael I. Jordan Latent Dirichlet Allocation

[17] https://es.wikipedia.org/wiki/Distribuci%C3%B3n_de_Dirichlet.

[18] https://fasttext.cc/docs/en/pretrained-vectors.html.

[19] Grave, E., Mikolov, T., Joulin, A., Bojanowski, P. Bag of tricks for efficient text classification. En: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017n. pp. 427–431. Spain (2017), https://fasttext.cc/

[20] https://www.wikipedia.org/.

[21] Breiman, Leo (2001) Random Forests

[22] Cortes, Corinna, Vapnik, Vladimir N. (1995). Support-vector networks

[23] Shawe-Taylor, J., Cristianini, N. (2004). Kernel Methods for Pattern Analysis.

[24] https://en.wikipedia.org/wiki/Cross-validation_(statistics)

[25] Pablo Celayes, Martin Dominguez: Prediction of User Retweets Based on Social Neighborhood Information and Topic Modelling

[26] Matías Silva, Pablo Celayes, Martin Dominguez: Analyzing the Retweeting Behavior of Influencers to Predict Popular Tweets, with and without Considering their Content

[27] Flaviano Morone: Collective Influence Algorithm to find influencers via optimal percolation in massively large social media". *Nature, 2016*

[28] ZhaoYang: A Comparative Analysis of Community Detection Algorithms on Artificial Networks. Prediction of User Retweets Based on Social Neighborhood Information and Topic Modelling

[29] Wayne Xin Zhao, Jing Jiang et al. Comparing Twitter and traditional media using topic models.

[30] Tweepy http://tweepy.readthedocs.org/en/v3.2.0/.

[31] SQLAlchemy http://docs.sqlalchemy.org/en/latest/.

[32] Jupyter https://jupyter.readthedocs.io/en/latest/index.html.

[33] NetworkX http://networkx.readthedocs.io/en/stable/.

[34] Python-igraph http://igraph.org/python/

[35] Scikit-learn http://scikit-learn.org/stable/documentation.html.

[36] NLTK http://www.nltk.org/.

[37] Spacy https://spacy.io/

[38] Gephi https://gephi.org/users/.

[39] pyLDAvis http://pyldavis.readthedocs.io/en/latest/

[40] PlotLy https://plot.ly/python/user-guide/

[41] PySpark https://spark.apache.org/docs/latest/api/python/pyspark.html

[42] Morone, F. and Min, B. and Bo, L. and Mari, R. and Makse, H. A. Collective Influence Algorithm to find influencers via optimal percolation in massively large social media. En: Scientific Reports 6, p 30062 (2016)

[43] Azcorra, A. and Chiroque, L. F. and Cuevas, R. and Fernández Anta and Laniado, H. and Lillo, R. E. and Sguera, C. Unsupervised Scalable Statistical Method for Identifying Influential Users in Online Social Networks. En: Scientific Reports 8, p 6955 (2018)

[44] Sanjeev Arora, Yingyu Liang, Tengyu Ma A simple but tough-to-beat basekube fir sentence embeddings. En: Proceeding of International Conference on Learning Representations, ICLR 2017, April 24 - 26, Toulon, France, (2017)