Recomendación de Información basada en Análisis de Redes Sociales y Procesamiento de Lenguaje Natural [1]

Autor: Pablo Gabriel Celayes.
Director: Dr. Martín A. Domínguez.
Co-director: Ing. Rene Nederhand.
Facultad de Matemática, Astronomía, Física y Computación
Universidad Nacional de Córdoba

19 de julio de 2017

Resumen

El presente trabajo se origina en el estudio de técnicas de Análisis de Redes Sociales para mejorar la calidad de un recomendador de contenido para entornos corporativos.

Estudiamos el problema de recomendación de contenido basada en preferencias del entorno social de un usuario. Construimos un conjunto de datos de muestra extraído de la red social **Twitter** y a partir de estos datos entrenamos y evaluamos modelos de clasificación binaria SVM (*Support Vector Machines*) que predicen *retweets* de un usuario en base a los de su entorno. Se obtiene una calidad media de predicción $F1$ superior al 84 %, sin analizar el contenido de los *tweets*.

En los casos en que la predicción social pura no es tan buena, se estudian modelos aumentados con características extraídas del contenido, usando el modelo temático probabilístico LDA (*Latent Dirichlet Allocation*).

Palabras Clave: Análisis de Redes Sociales, Aprendizaje Automático, Procesamiento de Lenguaje Natural, Modelado Temático.

Clasificación (ACM CCS 2012):

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

II

Agradecimientos

A mi director, Martín Domínguez por su enorme predisposición y tantas horas acumuladas de mates, cenas y consejos académicos y no tanto.

A mi co-director Rene Nederhand por confiar en mí para comenzar el desarrollo de este trabajo dentro de su empresa.

A Laura, por el apoyo, la compañía, el amor y todo lo compartido durante el último año de este trabajo. Por creer en mí, en vos y en nosotros y animarnos a entender y transformar el mundo (en tu caso, el universo) desde la ciencia.

A mis viejos por darme la posibilidad de estudiar y desarrollarme como persona y profesional, y apoyarme y confiar hasta en mis planes más locamente optimistas. Por transmitirme humildad y cooperación por encima del éxito individual.

A mis hermanos el Mati y la Lore por tanta alegría y crecimiento compartidos.

A la abuela Aurora por el refugio, el oído, la comida y la contención.

Al puñado de amigos que siempre estuvieron cerca, por ayudarme en los momentos duros, por escucharme y distraerme mientras reacomodaba la vida y redibujaba mi camino.

Y a toda mi familia, abuelos, tíos y primos que me apoyan y acompañan.

A Laura Alonso i Alemany, Franco Luque y Pedro D'Argenio por completar el tribunal de este trabajo.

A la facultad, por dar la excelente capacitación profesional y la predisposición de todos, docentes, ayudantes, personal de la Biblioteca, Departamento de Alumnos, etc. También al Estado Nacional y todos los argentinos por seguir apostando a la educación universitaria pública y gratuita como motor del desarrollo económico, productivo y social del país.

A *la banda del tío*, por hacer de la facultad una experiencia de camaradería y amistad que continuaremos toda la vida.

Al GURI por la oportunidad de ser parte de la vida institucional y política de la facultad.

A la Olimpiada Matemática Argentina por haberme cambiado la vida para siempre al permitirme descubrir mi pasión por la resolución de problemas.

A todos mis compañeros de trabajos de desarrollo, de LavandaInk, Machinalis, Infoxel, BairesDev, OTA Expert, Cogfor, Booking.com y todos aquellos con los que compartimos experiencias freelance (vía Upwork o locales) en estos años.

A todas las empresas anteriores por darme la oportunidad de desarrollarme profesionalmente dentro de la industria, en momentos en los que creí que se me había pasado el tren.

A la comunidad de software libre en general no sólo por brindarme las herramientas que me permiten hacer mi trabajo (en particular **este** trabajo final), sino también por defender la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar y/o modificar el software.

A la comunidad de Python en particular, por su apertura y espíritu cooperativo e inclusivo, por los Pycamps, Scipy, PyData, EuroPython y demás eventos en los que he tenido la oportunidad de aprender y conocer muy buena gente.

A Open Data Córdoba por los momentos y desafíos compartidos y el trabajo conjunto en esta visión en común de la información como bien social y herramienta de transformación.

Índice general

Capítulo 1

Introducción

1.1. Origen y evolución de la idea central

Como ocurre con muchos trabajos de investigación, a lo largo del desarrollo de esta tesis se fueron produciendo cambios tanto en las condiciones externas como en nuestros intereses y entendimiento de la problemática tratada. Esto fue transformando la idea original en algo diferente, pero siempre relacionado a los intereses que motivaron el inicio de este trabajo.

Creemos que es valioso compartir esta experiencia del modo en que se desarrolló. No sólo para dar más claridad a nuestras ideas, sino porque estamos convencidos de que comunicar ciencia no debe limitarse a la presentación austera y "podada" de resultados finales, sino que debe extenderse a comunicar la experiencia completa y el camino recorrido para que otros puedan sacar el mayor beneficio posible de los aciertos y errores de nuestro trabajo.

Es por esto que dividiremos esta introducción en dos partes: la primera mostrando la idea original, y la segunda explicando cómo se terminó trabajando y cuáles fueron las razones detrás de los cambios de enfoque realizados.

1.1.1. Idea original

Dada la gran cantidad de información disponible en Internet y su velocidad de generación, se vuelve cada vez más difícil y tedioso encontrar contenido actualizado y de interés. Se genera entonces la necesidad de contar con aplicaciones que faciliten la clasificación de la información y el filtrado de artículos de valor informativo para cada usuario. La plataforma de recomendación de contenido **Cogfor** [16] (donde el autor se desempeñaba como programador/investigador al comienzo de este trabajo) se propone brindar una solución a este problema tanto para entornos corporativos como para uso personal. La motivación inicial de esta tesis se centraba en hacer un aporte a la inteligencia de filtrado de contenido de dicho proyecto.

La plataforma mencionada evalúa el contenido consumido por una organización diariamente, con el fin de clasificarlo y seleccionarlo para los integrantes de acuerdo a sus preferencias personales. Inicialmente, las recomendaciones generadas para cada usuario se basaban solamente en su uso de la plataforma, sin tener en cuenta al resto de los usuarios. La propuesta original de esta tesis era agregar una dimensión social al proceso, modelando no sólo las preferencias de cada usuario, sino también sus afinidades con otros usuarios y comunidades a las que pertenece. Esto nos daría la oportunidad de combinar dos áreas de investigación muy dinámicas y en boga, como son el Procesamiento de Lenguaje Natural (PLN) y el Análisis de Redes Sociales (SNA, por *Social Network Analysis*).

Un primer paso en la integración de estas dos áreas consistía en resolver el problema de *cold start*, que consiste en dar un modelo inicial para las preferencias de un usuario nuevo, sobre el que todavía no tenemos comportamiento registrado. Las técnicas de SNA nos permitirían descubrir usuarios existentes con gustos similares al nuevo, para así basar las sugerencias iniciales en lo ya conocido sobre dichos usuarios. A continuación, se investigarían métodos para mejora continua de recomendaciones basada en información sobre la afinidad y conectividad entre usuarios. Se estudiarían diversas maneras de extraer dicha información de fuentes externas (redes sociales como **Facebook**, **LinkedIn** o **Twitter**) o internas (e.g.: interacciones por mail en el caso corporativo) empleando técnicas de detección de comunidades para encontrar tanto temas de interés como usuarios afines en quienes basar nueva recomendaciones.

1.1.2. En qué devino...

Nuestro objetivo original de mejorar el recomendador de **Cogfor** dependía en gran medida de la adopción comercial a gran escala de la plataforma, lo que nos permitiría contar con una buena cantidad de usuarios sobre los que experimentar nuestras ideas. Si bien se generó mucho interés alrededor del proyecto y se lograron acuerdos con socios tecnológicos de relevancia en el negocio de las *intranets* corporativas en Holanda, al momento de concluir la etapa de recolección de datos de esta tesis no se contaba aún con una base de usuarios lo suficientemente grande para nuestros experimentos.

Es por esto que se decidió basar los análisis del presente trabajo en datos obtenidos de redes sociales de acceso público como **Facebook** o **Twitter**. Esto por un lado nos permitiría contar con un buen volumen de datos sobre el cuál entrenar y evaluar nuestros algoritmos, pero tenía la desventaja de ofrecernos menos información explícita sobre el tipo de contenido que un usuario encuentra interesante o no.

Los intentos de generar un buen *conjunto de datos* a partir de *Facebook* generaron muchas complicaciones técnicas que nos condujeron a basarnos en datos extraídos de *Twitter*, que ofrece una API mucho más abierta y que además como red social tiene una estructura mucho más simple y clara. Por otra parte, el uso de datos de Twitter introduce la desventaja de la limitada cantidad de contenido que se comparte en cada *tweet* (140 caracteres). Esto nos llevó a cambiar nuestro enfoque para dar más énfasis a modelos

basados en interacciones sociales que en análisis de contenido.

Resumiendo entonces, el núcleo de este trabajo consiste en:

- Generación de un conjunto de datos de usuarios, seguidores y tweets a partir de la red social *Twitter*.
- Estudio de modelos para aprender y predecir preferencias (retweets) sobre dicho conjunto de datos, a partir de información sobre el entorno social de cada usuario.
- Estudio de posibles mejoras a los modelos sociales de predicción, introduciendo técnicas de procesamiento de lenguaje natural

El trabajo constituye no sólo un interesante análisis de distintos algoritmos y técnicas, sino que también es una forma de entender qué tan influenciables son los usuarios por su entorno social.

1.2. Estructura

El resto de este trabajo se estructura como sigue: En el capítulo 2 se detallan los fundamentos teóricos de clasificación supervisada y modelado temático que se emplearon para la implementación de nuestros modelos predictivos. A continuación se da una breve descripción de las herramientas y tecnologías utilizadas. Por su parte, el capítulo 4 detalla el proceso de generación de datos de muestra, tanto de usuarios y conexiones como de contenido. El capítulo 5 constituye el núcleo del trabajo, donde se describen nuestros experimentos de predicción basada en entorno social y los intentos de mejorar sus resultados a través de análisis de contenido con técnicas de modelado temático. Luego de esto se hace una breve comparación con trabajos relacionados previos. Para finalizar, el capítulo 7 contiene nuestras conclusiones principales y varias líneas de trabajo futuro posibles. El capítulo 8 contiene recursos bibliográficos y enlaces a información sobre proyectos y tecnologías relevantes.

Capítulo 2

Fundamentos Teóricos

En este capítulo introducimos los principales conceptos teóricos necesarios para entender los análisis y experimentos realizados en este trabajo.

La primera sección se ocupa del problema de *Aprendizaje Automático* supervisado de *clasificación binaria*, el algoritmo *Support Vector Machines* para resolverlo y las métricas que emplearemos para evaluar la calidad de los modelos generados.

La segunda parte profundiza sobre la técnica de modelado temático *Latent Dirichlet Allocation*, que emplearemos para descubrir temas sobre un *corpus*, que luego se emplearán para generar descripciones numéricas breves de los textos basadas en las probabilidades de cada uno de estos temas dentro de cada texto.

2.1. Clasificación

Dentro del Aprendizaje Automático Supervisado el problema de clasificación binaria consiste en, dado un conjunto de datos de entrenamiento partidos en dos clases, inferir reglas que permitan, dados nuevos datos no observados durante el entrenamiento, asignarles una de las dos clases.

En nuestro trabajo se entrenarán clasificadores binarios para distinguir contenido interesante y no interesante para un usuario dado, con el objetivo de proveerle a futuro recomendaciones personalizadas de contenido.

Comenzaremos describiendo el algoritmo de clasificación a emplearse (Support Vector Machines), para luego introducir las métricas con las que evaluaremos la calidad de los modelos. Por último, describimos el fenómeno de *sobreajuste* y las técnicas de particionado de los datos que nos ayudan a prevenirlo.

2.1.1. Support Vector Machines

Utilizaremos para la clasificación de preferencias el algoritmo SVM (*Support Vector Machines*) [4]. Dentro de la tarea de clasificación, las SVMs pertenecen a la categoría de los clasificadores lineales, puesto que inducen separadores lineales (también llamados hiperplanos). Esta separación puede producirse en el espacio original de los datos de entrada, si éstos son linealmente separables o cuasi-separables (ruido), o en un espacio transformado, si los ejemplos no son linealmente separables en el espacio original. La búsqueda del hiperplano de separación en estos espacios transformados, normalmente de muy alta dimensión, se hace de forma implícita utilizando las denominadas funciones *kernel*.

En esta sección se describe brevemente la derivación matemática de las SVMs como técnica de clasificación binaria. Se comenzará con la descripción del enfoque para funciones de discriminación lineales, para luego extender el método a funciones no lineales.

Clasificación Lineal

Sean nuestros datos de entrenamiento $\{\mathbf{x}_i, y_i\}$ $i = 1, \ldots, l$, donde los $\mathbf{x}_i \in \mathbb{R}^n$ son vectores de características (*features*) de nuestros datos de entrada y los $y_i \in \{-1, 1\}$ son etiquetas que indican a qué clase pertenece cada ejemplo.

Diremos que los datos son *linealmente separables* si existe un hiperplano separador $\mathbf{x}\mathbf{w} + b = 0$ (con $\mathbf{x} \in \mathbb{R}^n$, $b \in \mathbb{R}$), donde

- $\mathbf{w}$ es perpendicular al hiperplano
- $\dfrac{|b|}{\|\mathbf{w}\|}$ es la distancia al origen
- $\|\mathbf{w}\|$ es la norma euclídea $\mathbf{w}$

En este caso, la SVM encuentra el hiperplano separador de mayor margen entre los datos, donde el margen se define como $m := d_+ + d_-$, siendo d_+, d_- las distancias del hiperplano a los puntos más cercanos dentro de cada clase.

Las condiciones de hiperplano separador se formalizan así:

$$\begin{aligned} \mathbf{x}_i\mathbf{w} + b &\geq +1, \quad y_i = +1 \\ \mathbf{x}_i\mathbf{w} + b &\leq -1, \quad y_i = -1 \\ &\equiv \\ y_i(\mathbf{x}_i\mathbf{w} + b) - 1 &\geq 0, \qquad \forall i \end{aligned} \tag{2.1}$$

Para los puntos más cercanos en cada clase, se cumplen las igualdades, y por lo tanto resulta:

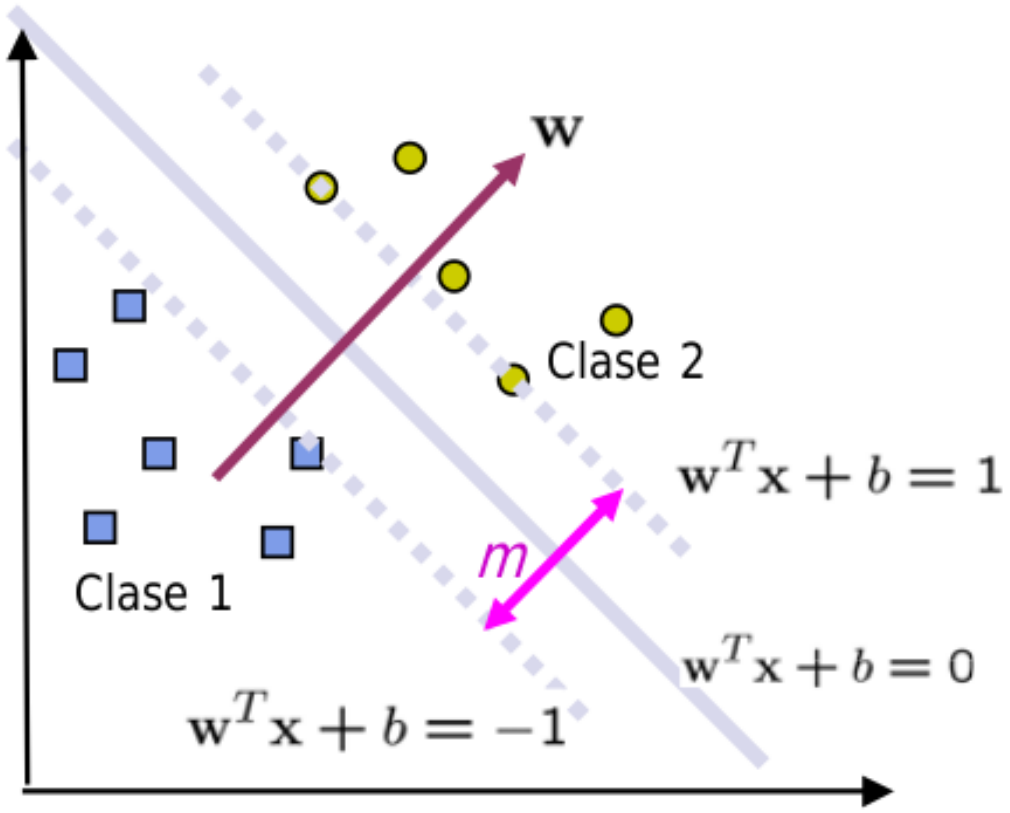

Figura 2.1: SVM busca una frontera de decisión tan lejos de los datos como sea posible

$$m = d_+ + d_- = \frac{|(-b+1)-(-b)|}{\|\mathbf{w}\|} + \frac{|-b-(-b-1)|}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|} \tag{2.2}$$

Maximizar el margen equivale entonces a minimizar $\|\mathbf{w}\|^2$ sujeto a las condiciones 2.1.

A partir de esta formulación se construye el dual mediante la técnica de los multiplicadores de Lagrange [5]. La formulación dual permitirá construir funciones de clasificación no lineales, lo que usualmente lleva a un mayor poder predictivo, como veremos en un momento. La formulación dual resulta ser:

$$\max_{\alpha} \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \mathbf{x}_j) \tag{2.3}$$

sujeto a

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

$$\alpha_i \geq 0, i = 1 \ldots l$$

Este es un problema de *optimización cuadrática* para el cuál existen varias técnicas de resolución conocidas.

Los puntos $\mathbf{x}_i$ con $\alpha_i > 0$ se denominan vectores soporte y pertenecen a los hiperplanos paralelos al separador que determinan el máximo margen. Son los únicos que participan en la construcción del hiperplano de clasificación, los demás puntos pueden removerse sin alterar la solución del problema.

Una vez encontrado el mejor hiperplano $\mathbf{x}\mathbf{w}^* + b^* = 0$, cualquier nuevo dato x se clasifica mediante la función de clasificación $\hat{y}(x) := \text{sgn}(\mathbf{w}^*\mathbf{x} + b^*)$, que simplemente indica de qué lado del hiperplano se encuentra x.

Se puede demostrar que $\mathbf{w}^* = \sum_{i=1}^{l} \alpha_i^* y_i \mathbf{x}_i$, con lo que la función de decisión puede reescribirse como:

$$\hat{y}(x) = \text{sgn}(\sum_{i=1}^{l} \alpha_i^* y_i (\mathbf{x}_i \mathbf{x}) + b^*)$$

Datos no Separables

Consideremos ahora el caso en que no existe un hiperplano separador, es decir, no es posible satisfacer todas las restricciones del problema 2.1. Con el fin de considerar un costo por cada observación mal clasificada, se introduce un conjunto adicional de variables $\xi_i, i = 1 \ldots l$.

En este caso, la SVM resuelve el siguiente problema de optimización:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i \tag{2.4}$$

sujeto a

$$\begin{aligned} y_i(\mathbf{x}_i\mathbf{w} + b) &\geq 1 - \xi_i \\ \xi_i &\geq 0, \quad i = 1 \dots l \end{aligned}$$

, donde C es una constante, denominada parámetro de *regularización*. El ajuste de éste parámetro permite hacer un balance entre la maximización del margen y la minimización de clasificaciones erróneas.

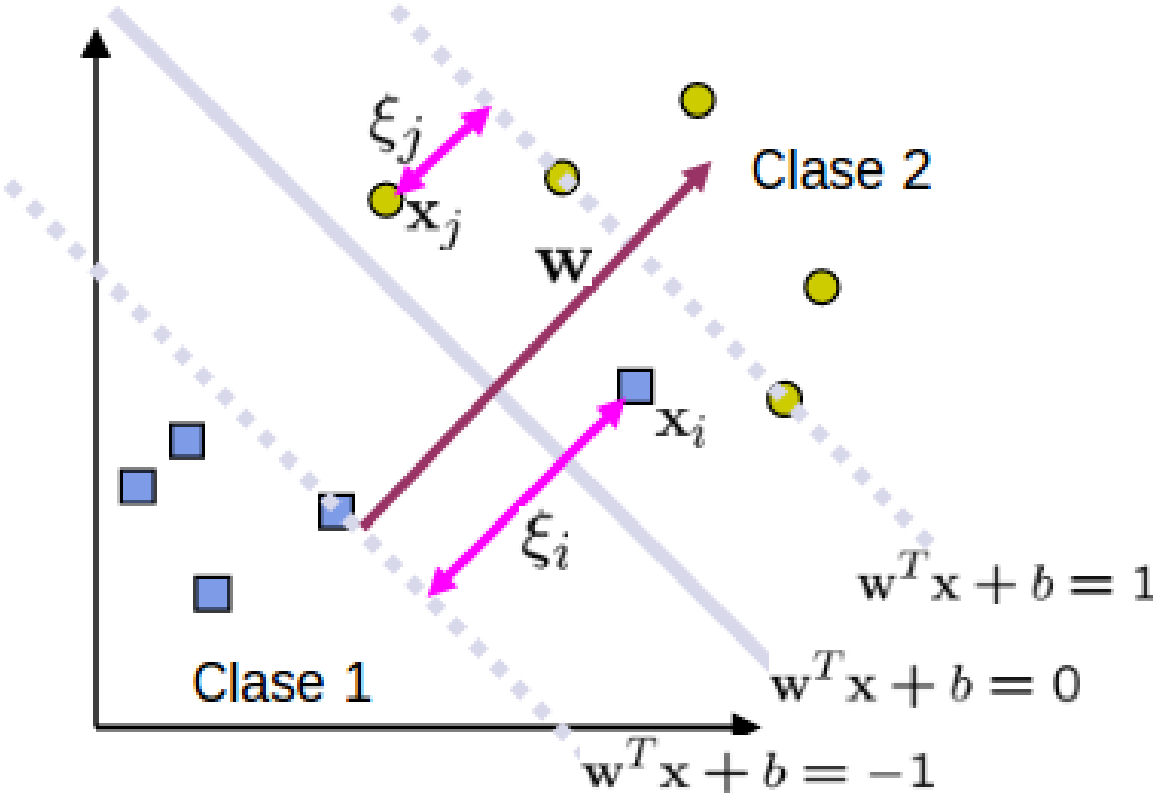

Figura 2.2: introducción de variables de error ξ_i para el caso no separable.

SVMs no Lineales

Para el caso no lineal, la SVM proyecta el conjunto de datos a un espacio de mayor dimensión $\mathcal{H}$ utilizando una función $\Phi : \mathbb{R}^d \to \mathcal{H}$, donde se construye un hiperplano separador de máximo margen. El problema de optimización cuadrática a resolver resulta:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i \tag{2.5}$$

sujeto a

$$\begin{aligned} y_i(\Phi(\mathbf{x}_i)\mathbf{w} + b) &\geq 1 - \xi_i \\ \xi_i &\geq 0, \quad i = 1 \dots l \end{aligned}$$

El Truco del Kernel

La formulación lagrangiana del problema anterior es la siguiente:

$$\max_{\alpha} \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j(\Phi(\mathbf{x}_i)\Phi(\mathbf{x}_j)) \tag{2.6}$$

sujeto a

$$\begin{aligned} \sum_{i=1}^{l}\alpha_i y_i &= 0 \\ \alpha_i \geq C, i &= 1 \dots l \end{aligned}$$

, donde $\mathbf{w} = \sum_{i=1}^{l}\alpha_i y_i \Phi(\mathbf{x}_i)$.

La función de clasificación toma la siguiente forma:

$$\begin{aligned} \hat{y}(x) &:= \operatorname{sgn}(\mathbf{w}^*\Phi(\mathbf{x}) + b^*) \\ &= \operatorname{sgn}((\sum_{i=1}^{l}\alpha_i^* y_i \Phi(\mathbf{x}_i)\Phi(\mathbf{x})) + b^*) \end{aligned} \tag{2.7}$$

Notemos que tanto en el problema de optimización como en la función de clasificación obtenida, los $\Phi(\mathbf{x}_i)$ aparecen siempre dentro de un producto punto.

Gracias a esto, el algoritmo SVM sólo necesita conocer la función $K(x_i, x_j) = \Phi(\mathbf{x}_i)\Phi(\mathbf{x}_j)$ sin necesidad de conocer la propia Φ. Esto permite proyectar sobre espacios $\mathcal{H}$ de dimensión muy alta (incluso infinita), donde los datos puedan separarse, pero manteniendo la complejidad del problema de optimización al mismo nivel que en el caso lineal.

El problema reformulado con el kernel K es el siguiente:

$$\max_{\alpha} \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{2.8}$$

sujeto a

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$
$$\alpha_i \geq C, i = 1 \dots l$$

y la función de clasificación que genera resulta:

$$\hat{y}(x) := \text{sgn}(\sum_{i=1}^{l} \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}) + b^*)$$

En este trabajo experimentaremos con los 2 kernels más usuales:

- *Radial basis function* (RBF) o Kernel Gaussiano: $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$.
- Kernel polinómico: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i \mathbf{x}_j + 1)^d$, donde $d \in \mathbb{N}$ es el grado del polinomio.

En ambos casos, γ es un parámetro que altera la forma de las fronteras de decisión. Por defecto suele usarse $\gamma = 1$.

2.1.2. Sobreajuste y Particionado

El sobreajuste (más conocido como *overfitting*) es la tendencia que tienen la mayoría de los algoritmos de Aprendizaje Automático a ajustarse a características demasiado específicas de los datos de entrenamiento que no tienen relación causal con la función objetivo que estamos buscando para generalizar. El ejemplo más extremo de un modelo sobreajustado es un modelo que sólo memoriza las respuestas correctas; este modelo al ser utilizado con datos que nunca antes ha visto va a tener un rendimiento azaroso, ya que nunca logró generalizar un patrón para predecir.

Todos los modelos de Aprendizaje Automático tienen tendencia al sobreajuste; es por esto que debemos aprender a convivir con el mismo y tratar de tomar medidas preventivas para reducirlo lo más posible. Las dos principales estrategias para lidiar con el sobreajuste son: la retención de datos y la validación cruzada.

En el primer caso, la idea es dividir nuestro conjunto de datos, en uno o varios conjuntos de entrenamiento y evaluación. Es decir, que no le vamos a pasar todos nuestros datos al algoritmo durante el entrenamiento, sino que vamos a retener una parte de los datos de entrenamiento para realizar una evaluación de la efectividad del modelo. Con esto lo que buscamos es evitar que los mismos datos que usamos para entrenar sean los mismos que

utilizamos para evaluar. De esta forma vamos a poder analizar con más precisión cómo el modelo se va comportando a medida que más lo vamos entrenando y poder detectar el punto crítico en el que el modelo deja de generalizar y comienza a sobre-ajustarse a los datos de entrenamiento.

2.1.3. Evaluación

Definimos a continuación las métricas que utilizaremos para medir la calidad de un modelo que genera una función binaria de clasificación $f : \mathbb{R}^d \to \{-1, 1\}$.

Todas estas medidas se calculan respecto de un cierto conjunto de datos etiquetados $\{\mathbf{x}_i, y_i\}$ $i = 1, \ldots, l$ y de una de las clases que consideraremos como *positiva*.

En nuestro caso particular, la clase positiva ($y = 1$) serán los tweets interesantes para un usuario dado.

Precisión

La `precision` mide cuántos de los datos clasificados como positivos realmente lo son:

$$\texttt{precision} := \frac{|\{x_i | f(x_i) = 1 \text{ y } y_i = 1\}|}{|\{x_i | f(x_i) = 1\}|}$$

Exhaustividad (*Recall*)

La exhaustividad (o `recall`) mide cuántos de los datos de la clase positiva son identificados por el clasificador:

$$\texttt{recall} := \frac{|\{x_i | f(x_i) = 1 \text{ y } y_i = 1\}|}{|\{x_i | y_i = 1\}|}$$

Puntaje `F1`

Esta es la métrica final que utilizaremos en este trabajo, ya que provee un buen balance entre las dos anteriores. Se calcula como la media armónica entre ellas:

$$\texttt{F1} := \frac{2}{\frac{1}{\texttt{precision}} + \frac{1}{\texttt{recall}}} = \frac{2 * \texttt{precision} * \texttt{recall}}{\texttt{precision} + \texttt{recall}}$$

2.2. Modelos Temáticos Probabilísticos

Los algoritmos de modelado temático son métodos estadísticos que analizan las ocurrencias de palabras en un conjunto de textos pre-procesados para descubrir los temas

de los que se habla en ellos, cómo se conectan entre sí estos temas, y cómo cambian a lo largo del tiempo.

Estos algoritmos no requieren ningún tipo de anotaciones o etiquetado previo de los documentos: los temas emergen del análisis crudo de los textos y las ocurrencias de palabras y términos en los mismos.

Su aplicación va más allá de la exploración y el descubrimiento de temas dentro de los textos: también son muy útiles como herramientas de reducción de dimensionalidad, ya que permiten representar un texto por los temas que ocurren en este en lugar de las palabras que lo componen.

2.2.1. Latent Dirichlet Allocation

LDA propone un modelo generador [7] que modela cada documento de una colección dada (de aquí en adelante *corpus*) como una mezcla temas (distribución multinomial de probabilidad sobre temas), donde los temas a su vez están formados por una mezcla de términos del vocabulario del corpus (distribución multinomial de probabilidad sobre el vocabulario).

Formalmente, un *tema* será una distribución sobre un vocabulario fijo. Por ejemplo, el tema "Genética" tendrá probabilidad alta de palabras sobre genética (gen, mutación, ADN, etc.).

Se asume que estos temas existen antes de la generación de los datos observados. Ahora, para cada documento en la colección, se generan palabras según el siguiente proceso de 2 etapas:

1. Elegir al azar una distribución de temas.
2. Para cada palabra en el documento:
   (a) Elegir al azar un tema de la distribución del paso 1.
   (b) Elegir al azar una palabra de la distribución del tema elegido en a.

Este modelo estadístico refleja la intuición de que un documento puede tocar varios temas y que cada documento muestra distintas proporciones de cada tema (paso 1). A su vez, cada palabra en un documento se toma de uno de sus temas (paso 2.b), donde el tema elegido se toma de la distribución de temas por documento (paso 2.a).

El *modelo gráfico* de LDA, con las distintas distribuciones de probabilidad involucradas, puede observarse en 2.3: Cada nodo es una variable aleatoria etiquetada según su rol en el proceso generativo. Los nodos ocultos (proporciones de temas por documento θ_d, asignación de palabras a tema $Z_{d,n}$ y temas ϕ_k se muestran en blanco). Los nodos observados (las palabras en los documentos) se muestran sombreadas. Los rectángulos son notación de *plato*, que indica multiplicidad de variables del mismo tipo. El plato N

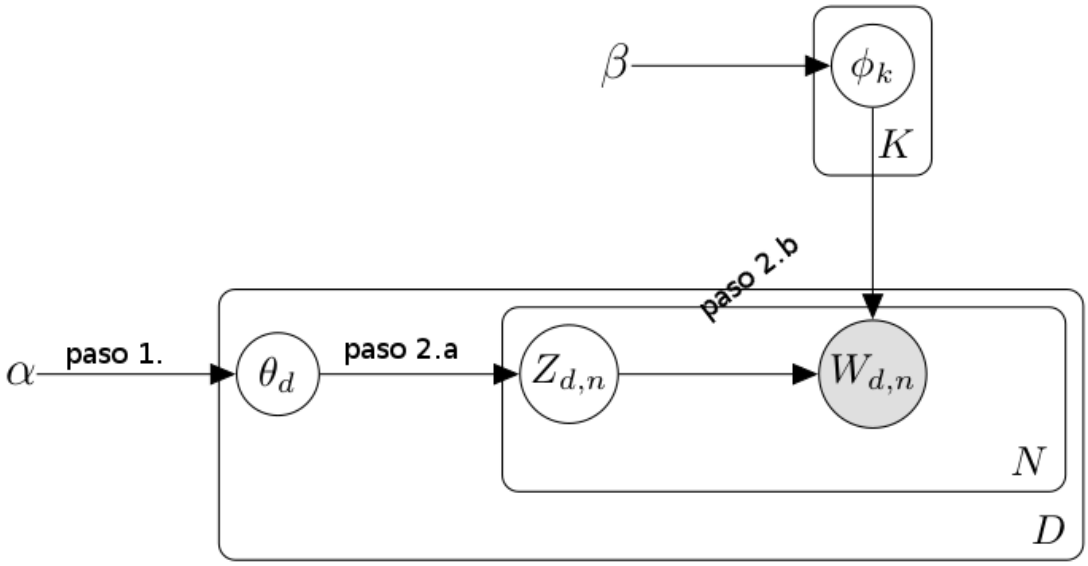

Figura 2.3: El modelo Latent Dirichlet Allocation en notación de plato.

denota la colección de palabras dentro de un documento, el plato D denota la colección de documentos en nuestro corpus, el plato K indica el conjunto de temas subyacentes.

Tanto θ_d como ϕ_k se toman de distribuciones de Dirichlet (α y β respectivamente). Una *distribución de Dirichlet* [8] es básicamente una distribución de distribuciones: Le asigna una probabilidad a cada posible distribución multinomial sobre un conjunto fijo de n eventos.

Estimación de parámetros

Los parámetros del modelo se ajustan usando la técnica de *Estimación de Máxima Verosimilitud* (EMV)[9]. Esto consiste en buscar las distribuciones α, β que maximizan la probabilidad de los documentos observados fijadas α y β .

El cálculo exacto de estas distribuciones es un problema intratable, pero existen varias técnicas que permiten aproximarlas. En este trabajo usamos el paquete `gensim` que implementa el algoritmo *online variational Bayes* descripto en [10].

Las principales ventajas de este algoritmo son:

1. Es continuo (*streamed*): los documentos de entrenamiento pueden consumirse secuencialmente, no se requiere acceso aleatorio al corpus completo. Gracias a esto, el tamaño del corpus de entrenamiento no afecta el uso de memoria y puede incluso procesar corpus más grandes que la RAM del sistema.

2. Es distribuido, lo que permite usar un cluster de máquinas y/o múltiples núcleos para acelerar la estimación del modelo.

Características LDA

LDA provee una distribución conjunta sobre las variables observadas y las ocultas. Esta distribución puede emplearse para obtener distribuciones *a posteriori* de las variables ocultas dados los D documentos observados. En particular, una vez estimados los parámetros del modelo, podemos estimar las distribuciones de temas por documento como:

$$\hat{\theta}_{d,k} = \mathbf{E}[\theta_{d,k}|w_{1:D,1:N}]$$

donde $w_{d,n}$ es la palabra n-ésima del documento d.

Esto permite generar para cada documento d un vector de características LDA $[\hat{\theta}_{d,k}]_{k=1}^{K}$ que intuitivamente puede interpretarse como *en qué medida habla de cada tema el documento*.

Capítulo 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.

Para más detalles sobre cada herramienta, pueden consultarse los enlaces de documentación incluidos en la bibliografía.

3.1. Datos

3.1.1. Tweepy

Esta librería permite comunicarse desde Python con Twitter y usar su API. Provee acceso autenticado a la API 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. Scrapy

Scrapy es un *framework* para extracción de datos web desarrollado en Python.

Es rápido (descarga los datos asíncronamente a través de *Twisted*), versátil y muy extensible. Provee soporte para extraer y seleccionar datos de fuentes HTML/XML usando selectores CSS y expresiones XPath, y para exportar los datos en formatos múltiples como JSON, CSV y XML.

Lo empleamos en combinación con el simulador de navegador Splinter para intentar generar un conjunto de datos de Facebook.

3.1.3. SQLAlchemy

SQLAlchemy es un ORM (*Object-relational mapper*) para Python, que permite convertir entre los tipos de datos usados en los lenguajes de programación orientados a objetos y los tipos de datos de algún sistema de base de datos relacional. Incluye soporte para SQLite, MySQL, PostgreSQL, Oracle, MS SQL, entre otros. Se utilizó para almacenar los datos recolectados en una base de datos SQLite y luego para acceder estos datos desde Python sin escribir consultas SQL.

3.2. Grafos

3.2.1. NetworkX

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

Se utilizó para almacenar y manipular redes de usuarios y hacer análisis de centralidad y afinidad entre los mismos (que finalmente no fueron utilizados en esta versión final del trabajo).

3.2.2. graphtool

graphtool provee funcionalidades similares a las de NetworkX pero implementadas de forma mucho más eficiente, ya que sus algoritmos y estructuras de datos principales están desarrollados en C++ con un fuerte énfasis en la eficiencia de ejecución. Provee una interfaz para acceder desde Python. Se empleó en algunos experimentos iniciales con conjuntos masivos de varios millones de usuarios, que luego fueron dejados fuera de este trabajo por su excesiva complejidad.

3.3. Análisis

3.3.1. pandas

Pandas es una librería que proporciona estructuras de datos flexibles y permite trabajar con la información de forma eficiente (gran parte de Pandas está implementado usando C/Cython para obtener un buen rendimiento).

Funciona muy bien cuando nos toca trabajar con:

- Datos heterogéneos que pueden distribuirse de forma tabular.

- Series temporales
- Matrices

Se empleó para tareas de análisis y limpieza de datos.

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.

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.
- Particionado de datos de entrenamiento y prueba
- Escalado y transformación de datos
- Entrenar Modelos de clasificación
- Ajuste de parámetros

3.4.2. NLTK

Natural Language Toolkit (NLTK) es un conjunto de bibliotecas y programas para el procesamiento del lenguaje natural (PLN) simbólico y estadístico en Python. Incluye demostraciones gráficas y datos de muestra y se acompaña de un libro que explica los conceptos subyacentes a las tareas de procesamiento del lenguaje implementadas en NLTK.

NLTK está destinado a apoyar la investigación y la enseñanza en PLN o áreas muy relacionadas, que incluyen la lingüística empírica, las ciencias cognitivas, la inteligencia artificial, la recuperación de información, y el aprendizaje automático. NLTK se ha utilizado con éxito como herramienta de enseñanza, como una herramienta de estudio individual,

y como plataforma para los sistemas de investigación de prototipos y construcción de modelos.

Lo empleamos en varias etapas del pre-procesado, limpieza y trasformación de textos.

3.4.3. gensim

Gensim es una plataforma de código abierto en Python para modelado vectorial de textos y modelado temático. Está diseñada específicamente para manejar grandes colecciones de textos, usando *streaming* de datos y algoritmos incrementales eficientes.

Utilizamos gensim tanto para extracción de frases relevantes como para modelado temático, usando su versión paralela distribuida de Latent Dirichlet Allocation (LDA).

3.5. Visualización

3.5.1. Gephi

Gephi es un software open-source de análisis de redes y visualización escrito en Java en la plataforma NetBeans.

Gephi ha sido utilizado en proyectos académicos de investigación, periodismo y otras áreas, por ejemplo para visualizar la conectividad global del contenido del New York Times y para examinar el tráfico de red de Twitter durante tiempos de malestar social, junto con temas de análisis de redes más tradicionales. Gephi es ampliamente utilizado dentro del campo de las humanidades digitales, una comunidad donde muchos de sus desarrolladores están involucrados.

Lo empleamos para visualizar la red de usuarios de Twitter estudiada en este trabajo.

3.5.2. pyLDAvis

Librería de Python para generar visualizaciones interactivas de modelos temáticos. Es una adaptación del paquete de *R* `LDAvis`, creado por Carson Sievert y Kenny Shirley.

pyLDAvis está diseñado para facilitar la tarea de interpretar los temas extraídos de un corpus de texto con el modelo LDA. El paquete utiliza información generada por un modelo LDA pre-entrenado para generar una visualización web interactiva.

En principio las visualizaciones están pensadas para ser usadas dentro de un *notebook* de IPython (Jupyter), pero también pueden guardarse en HTML para compartirlas más fácilmente.

3.5.3. Bokeh

Bokeh es una librería de Python para visualizaciones interactivas diseñada para funcionar en los navegadores web modernos. Su objetivo es proporcionar una construcción

elegante y concisa de gráficos modernos al estilo de D3.js, y ampliar esta capacidad con la interactividad y buen rendimiento sobre grandes volúmenes de datos. Bokeh permite crear en forma rápida y sencilla gráficos interactivos, dashboards y aplicaciones de datos. Puede crear tanto gráficos estáticos como gráficos interactivos en el servidor de Bokeh.

La utilizamos para la visualización de métricas de los modelos predictivos que desarrollamos.

Capítulo 4

Datos de muestra

El objetivo central de este trabajo es generar modelos capaces de predecir las preferencias futuras de un usuario dado en base a preferencias conocidas (tanto del usuario como de su entorno social).

Para ello necesitamos los siguientes ingredientes:

- Usuarios
- Conexiones entre usuarios (amistad, seguidores, etc.)
- Contenido *interesante* para cada usuario (artículos leídos, compartidos, favoritos, retweets, etc.)
- Contenido no interesante

Como además pretendemos evaluar la combinación de análisis de entorno social con técnicas de Procesamiento de Lenguaje Natural, es importante que el contenido analizado sea texto, dejando de lado contenido multimedia como imágenes, audios o videos.

En las próximas secciones describimos las distintas alternativas que fueron exploradas a la hora de generar nuestro conjunto de datos.

4.1. Plan A: usuarios propios

Originalmente la idea era obtener todos los datos a partir de los usuarios de la plataforma *Cogfor*, que consiste en un *feed* de noticias personalizado para miembros de una organización.

La ventaja obvia de esto era poder contar con información explícita sobre contenido visto, compartido y preferido por cada usuario. Esto hubiera permitido definir con claridad la noción de contenido *interesante* y *no interesante*, dado que la plataforma nos provee

información sobre qué contenido fue visto por cada usuario, y dentro de lo que vieron, qué compartieron o marcaron como preferido.

Otra ventaja hubiera sido la facilidad de realizar *A/B testing* sobre nuestros usuarios, pudiendo comparar distintos modelos de recomendación entre grupos de usuarios.

Por otro lado, este enfoque planteaba dos dificultades principales:

- **Falta de información social explícita:** La plataforma no proveía una funcionalidad de conectar con otros usuarios o seguirlos. A esto planeábamos subsanarlo definiendo nuestra propia noción de conexión entre usuarios, basada en un análisis del contenido previamente consumido (por ejemplo: conectar dos usuarios si la cantidad de artículos leídos en común es mayor a cierto porcentaje de la cantidad total de artículos leídos por alguno de ellos).

- **Cantidad de usuarios dependiente del éxito comercial del proyecto:** Este punto fue el más difícil de superar, ya que dependía de la evolución comercial del producto y su adopción por parte de grandes clientes. Al momento de concluir la fase de definición del conjunto de datos para este trabajo (enero de 2015), existían acuerdos bastante prometedores con uno de los mayores proveedores de servicios de intranet de Holanda, pero aún no había disponibilidad suficiente de datos de preferencias de usuarios.

  Llegados a este punto decidimos generar nuestros propios datos extrayéndolos de redes sociales de uso público.

4.2. Plan B: Facebook

4.2.1. API

Al momento de comenzar a intentar obtener nuestros datos usando la *Graph API* de Facebook (enero de 2015), las condiciones de acceso a la información de ésta resultaban demasiado restrictivas para nuestras necesidades. En particular, no se nos permitía leer los muros de los contactos del usuario logueado, sin pedir además un permiso adicional a cada usuario.

Se evaluó la posibilidad de implementar una *app* a distribuir entre nuestros contactos, para que voluntariamente accedieran a aportar información de sus muros para el desarrollo de este trabajo, pero se descartó esta opción por considerarla engorrosa de implementar y de alcance incierto (dependiente de la respuesta del público).

4.2.2. *Scraping*

Para intentar obtener los datos que no nos daba la API, decidimos implementar un *scraper* que se autenticara con un usuario dado y navegara los muros de sus contactos

extrayendo información sobre artículos compartidos en un cierto periodo de tiempo.

Dada la complejidad del sitio de Facebook y el nivel intensivo de interactividad en el navegador (por ejemplo: necesidad de hacer *scrolling* para cargar historial dinámicamente), se decidió implementar esto usando la herramienta de *browsing* automático **Splinter**. [19]

Luego de mucho trabajo en este enfoque, se lo descartó por varias desventajas:

- Lentitud
- Difícil de hacer masivo
- Dificultad para distinguir contenido con artículos
- Dificultad para identificar artículos con el mismo contenido

4.3. Plan C: Twitter API

A diferencia de Facebook, la API de Twitter permite un acceso mucho más abierto a la información, tanto de contenido compartido como de conexiones entre usuarios.

Esto se ve favorecido por el propio modelo de uso de Twitter, en el que generalmente los usuarios comparten su contenido de forma pública y luego cualquier otro usuario puede decidir seguirlos. En Facebook, en cambio, debe aprobarse una solicitud de amistad para poder acceder al contenido compartido por un usuario.

Twitter provee una opción de privacidad adicional, donde sólo seguidores explícitamente autorizados pueden ver nuestro contenido, pero por lo general no es una opción muy usada.

Por otra parte, los datos de la API indican claramente cuando distintos usuarios han re-compartido un tweet original (*retweets*), punto fundamental para la construcción de nuestros datos, y que en el caso de Facebook resultaba dificultoso de determinar.

La principal desventaja de usar Twitter como fuente de datos es la longitud limitada de los textos compartidos (140 caracteres), que puede degradar la calidad de los algoritmos de PLN.

Otra desventaja, compartida con el caso de Facebook, es que no tenemos datos sobre qué vio cada usuario, solamente sabemos qué es lo que escribió, compartió o marcó como favorito. Esto complica un poco la separación entre contenido interesante y no interesante para un usuario. Veremos más adelante cómo se decidió resolver esto, al menos parcialmente.

4.3.1. Usuarios y contenido

Tomamos como grafo dirigido de muestra G al conjunto de usuarios seguidos por mi cuenta de Twitter (`@PCelayes`), junto con la relación `seguir` entre ellos. Se obtuvo así

un grafo de 1213 nodos y 40489 aristas (ver 4.1). Para cada usuario en G se extrajo un mes completo de tweets compartidos (del 18/3/2017 al 17/4/2017, ambos inclusive). Se obtuvo así un conjunto T de 163173 tweets compartidos. En el desarrollo de este trabajo nos centraremos en contenido en español, que resulta ser un conjunto de 109040 tweets.

G y T se almacenaron en una base de datos SQLite a través de modelos ORM implementados con SQLAlchemy, para facilitar la posterior consulta y manipulación de estos datos. Para cada tweet se almacenó su contenido, la fecha de creación, conteo de retweets y favoritos (global, extraído de la API). Se guardaron además las relaciones de favorito y retweet entre los usuarios G y los tweets en T y se guardó registro de la relación seguir entre usuarios.

La API de Twitter limita la cantidad de datos que pueden descargarse en un tiempo dado, por lo que se agilizó el proceso usando un conjunto rotativo de 9 cuentas de Twitter. Existen en Twitter usuarios con cuentas *protegidas* o privadas, las cuales sólo pueden seguirse pidiéndoles autorización. Mi usuario sigue algunas de éstas, pero las demás cuentas usadas para autenticar la API no lo hacen, por lo que la gran mayoría de cuentas en G son públicas. Esto es una limitación mínima, ya que la práctica común en Twitter es dejar las cuentas en modo público.

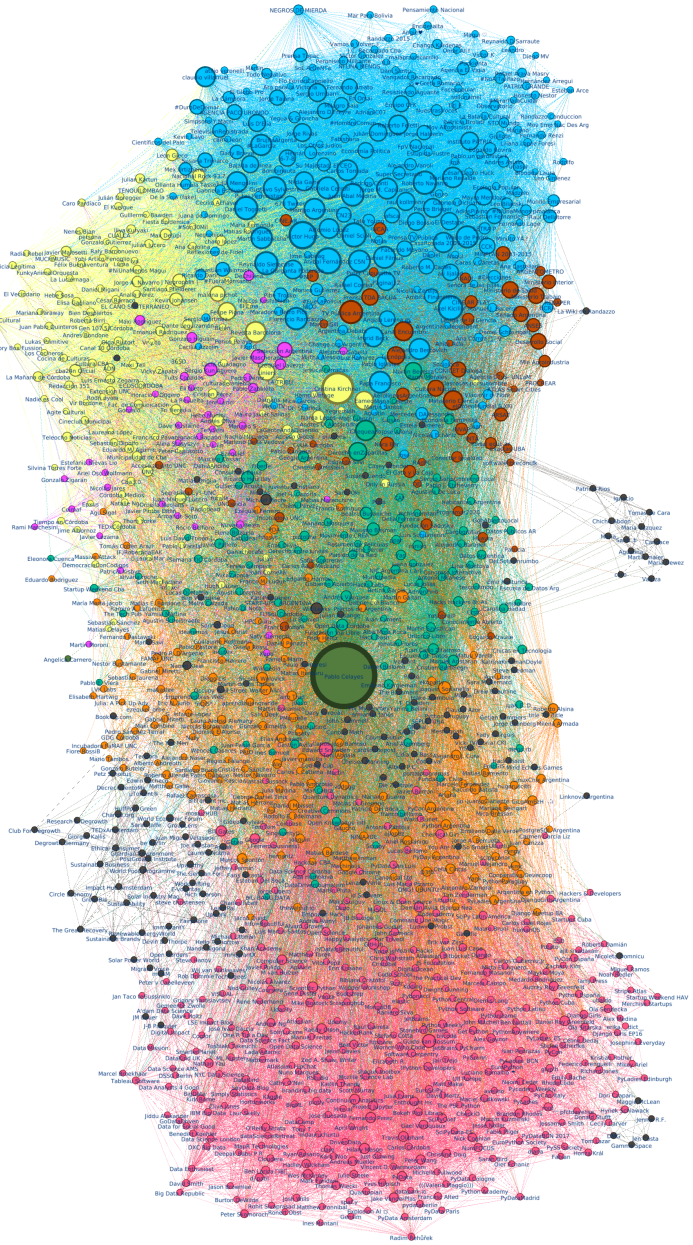

Figura 4.1: Mi red de usuarios seguidos en Twitter

Capítulo 5

Predicción de Preferencias

Nos interesa desarrollar modelos capaces de predecir las preferencias de un usuario, basándonos tanto en lo que conocemos de su entorno social como del contenido compartido previamente. En esta sección describiremos el desarrollo y los resultados de distintos experimentos que intentan lograr esto sobre nuestro conjunto de datos de Twitter.

Estudiaremos inicialmente modelos predictivos basados sólo en información del entorno social, para luego analizar la posibilidad de mejorar estas predicciones mediante técnicas de PLN que tengan además en cuenta el texto del contenido compartido.

5.1. Predicción Social Pura

Comenzamos estudiando el problema de predecir preferencias de un usuario dado u en base a su entorno social. El objetivo central será, dados un *tweet* e información sobre quiénes lo han compartido, predecir si u lo compartirá o no.

Dado que el proceso de extracción de características, entrenamiento de modelos y ajuste de parámetros es costoso computacionalmente, se realizarán estos experimentos sobre un conjunto limitado de usuarios. Comenzaremos describiendo el criterio con que se seleccionó este conjunto.

Describiremos luego cómo, para cada uno de ellos, se genera un entorno de usuarios E_u y un conjunto de *tweets* T_u potencialmente interesantes al usuario.

Continuamos explicando el proceso de generación de vectores de características en base a T_u y la partición de los mismos en conjuntos de *entrenamiento*, *validación* y *evaluación*.

Por último, describiremos el proceso de entrenamiento y ajuste de parámetros de los clasificadores y analizaremos la calidad de los modelos obtenidos.

5.1.1. 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.

Es por esto que decidimos restringir nuestros análisis a un subconjunto de usuarios activos y con entorno grande. Decidimos tomar aquellos con al menos 100 retweets en español y 100 usuarios en su entorno de segundo grado. Esto dio lugar a un subconjunto G_m de 98 usuarios. De aquí en adelante los modelos analizados estarán centrados en cada uno de estos usuarios.

Es importante destacar que el universo de usuarios de nuestros experimentos sigue siendo G, con todos sus usuarios y conexiones, que se usarán para generar los entornos de cada usuario en G_m cuyas preferencias tratemos de predecir.

5.1.2. Universo de tweets visibles

El objetivo inicial de este trabajo era desarrollar clasificadores que pudieran entrenarse sobre todo el contenido *visto* por un usuario, aprendiendo a separar lo que le interesó de lo que no.

En el caso de nuestro conjunto de datos de Twitter, no contamos con información explícita sobre qué vio o no realmente cada usuario, pero podemos al menos tomar un universo de tweets *potencialmente vistos*. Esto es sencillamente el conjunto de todos los tweets escritos o compartidos por los usuarios a los que u sigue, o por el propio u. Excluímos de este conjunto aquellos tweets *escritos* por el propio u, ya que el foco de nuestro problema es encontrar contenido externo interesante, y no estudiar la generación de contenido de un usuario particular. Formalmente, este conjunto se define como:

$$T_u := \left(\bigcup_{x \in \{u\} \cup \texttt{seguidos}(u)} \texttt{timeline}(x) \right) - \{t \in T | \texttt{autor}(t) = u\},$$

donde $\texttt{seguidos}(u) := \{x \in G | (u, x) \in \texttt{seguir}\}$ y $\texttt{timeline}(x)$ es el conjunto de todos los tweets escritos o compartidos por x dentro de los tweets recolectados en T.

Hay varias razones por las que un tweet *visible* podría no haber sido *visto* por u: por ejemplo, podría no haber chequeado su cuenta de Twitter por varios días, o podría estar siguiendo muchas cuentas, en cuyo caso Twitter le muestra sólo una selección de todo el contenido generado por sus seguidos. O en el caso de usuarios que u ha empezado a seguir recientemente, no habría estado expuesto al contenido compartido por estos usuarios durante el periodo completo de recolección de tweets.

Más allá de estas salvedades, sigue siendo una buena primera aproximación (y un problema interesante en sí mismo), desarrollar modelos que distingan tweets *interesantes*

(futuros retweets) dentro del universo de los tweets *visibles*.

Nos encontramos con algunos usuarios para los cuales el conjunto T_u resultaba muy grande, lo cual enlentecía el proceso de experimentación y entrenamiento de modelos. Se decidió en estos casos limitar T_u a un máximo de 10000 tweets. Como la clase de tweets interesantes al usuario es en general minoritaria, se redujo el tamaño del conjunto tomando una submuestra apropiada de los tweets no-interesantes.

5.1.3. Entorno de usuario

Para cada usuario u, usaremos el comportamiento de los usuarios en un entorno cercano para predecir el comportamiento del usuario elegido. Más precisamente, para cada tweet t nos interesa predecir si u compartió t en base a lo que conocemos sobre quiénes lo compartieron dentro de su entorno.

Más allá de que en principio un usuario sólo puede ver tweets compartidos por aquellos a quienes sigue, la información sobre su red ampliada puede proveer más indicadores del grado de interés de un tweet.

Es por ello que decidimos tomar como entorno de un usuario no sólo a los usuarios que éste sigue, sino además continuar un paso más en la relación `seguir` e incluir a los usuarios seguidos por éstos. Es decir, tomamos a todos los usuarios (distintos del propio u) a 1 o 2 pasos de distancia de u en G:

$$E_u = \left(\bigcup_{x \in \{u\} \cup \texttt{seguidos}(u)} \texttt{seguidos}(x) \right) - \{u\}$$

5.1.4. Creación de *características* de entorno

Teniendo ya definidos para cada usuario u un universo de tweets visibles T_u y un entorno de usuarios cercanos E_u, pasamos a construir los conjuntos de datos vectorizados con los que se entrenarán y evaluarán nuestros modelos predictivos.

Si $E_u = \{u_1, u_2, \ldots, u_n\}$, podemos definir para cada tweet $t \in T_u$ el siguiente vector de *características booleanas*:

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

donde

$$\texttt{tweet\_en\_tl}(t, u) := \begin{cases} 1 & t \in \texttt{timeline}(u) \\ 0 & \text{caso contrario} \end{cases}$$

Notar que no se está teniendo cuenta el contenido de t, sólo indicando qué usuarios del entorno elegido lo han compartido y cuáles no.

Reuniendo los vectores de todos los tweets en $T_u = \{t_1, \ldots, t_m\}$ en una sola matriz queda construido nuestro conjunto de datos vectorizado:

$$M_u := [\texttt{tweet\_en\_tl}(t_i, u_j)]_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}}$$

En esta notación, la variable a predecir para cada tweet t es $\texttt{tweet\_en\_tl}(t_i, u)$. Queda definido entonces para cada $u \in G_m$, el siguiente vector objetivo:

$$y_u := [\texttt{tweet\_en\_tl}(t_i, u)]_{1 \leq i \leq m}$$

5.1.5. Particionado del conjunto de datos

Siempre es importante poder evaluar el desempeño de nuestros modelos en datos no vistos, lo que nos lleva a separar algunos datos para evaluación, que no sean vistos durante el entrenamiento.

Por otra parte, durante el proceso de selección de algoritmos y evaluación, necesitaremos tomar decisiones basadas en la calidad de los modelos que a su vez queremos que no incidan en la evaluación final. Esto nos lleva a crear una partición adicional de los datos de evaluación, tomando un subconjunto que llamaremos de *ajuste*.

Evaluaremos las decisiones intermedias en este conjunto de ajuste, a fines de poder finalmente evaluar en un conjunto de datos que no sólo no haya sido observado por el algoritmo final, si no que tampoco haya influido en la elección del mismo o sus parámetros.

Decidimos entonces, para cada u particionar aleatoriamente M_u en conjuntos de entrenamiento, ajuste y evaluación (M_u^{en}, M_u^{aj}, M_u^{ev} respectivamente), conteniendo aproximadamente el 70 %, 10 % y el 20 % de los datos de M_u.

A cada una de estas partes le corresponde una parte de y_u: y_u^{en}, y_u^{aj}, y_u^{ev}.

5.1.6. Entrenando clasificadores

A continuación analizamos los resultados de entrenar y evaluar modelos `SVC` [1] en `scikit-learn` sobre todos los usuarios de G_m. En cada caso se hizo una optimización de *hiperparámetros* [2] `GridSearchCV` sobre M_u^{en} con la siguiente grilla de parámetros:

[1] Nombre de las SVM clásicas en `scikit-learn`, por *Support Vector Classifier*. El modelo se llama así para distinguirlo de la variante de SVM aplicada al problema de regresión (`SVR`, por *Support Vector Regression*).

[2] Se denomina así a los parámetros de un modelo que se fijan previamente, para distinguirlos de aquellos que se ajustan a los datos de entrenamiento.

```
{
    "C": [ 0.01, 0.1, 1 ],
    "class_weight": [ "balanced", None ],
    "gamma": [ 0.1, 1, 10 ],
    "kernel": [ "rbf", "poly" ]
}
```

Se evaluó luego el modelo obtenido para cada u sobre M_u^{aj}. Se obtuvo un puntaje `F1` promedio de $0,84$, con la distribución de puntajes sobre los 98 usuarios de G_m que se observa en la figura 5.1.

La configuración óptima de hiperparámetros varía bastante de un usuario a otro, pero se observa mayor preponderancia de ciertos valores. Los conteos de ocurrencias de los distintos valores de cada hiperparámetro fueron los siguientes:

```
{
    "C": {0.01: 73, 0.1: 7, 1: 18},
    "class_weight": {"balanced": 25, None: 73},
    "gamma": {0.1: 6, 1: 24, 10: 68},
    "kernel": {"poly": 63, "rbf": 35}
}
```

La `precision` promedio es $0,93$ y la `recall` promedio $0,78$, con las distribuciones que se observan en las figuras 5.2 y 5.3, respectivamente.

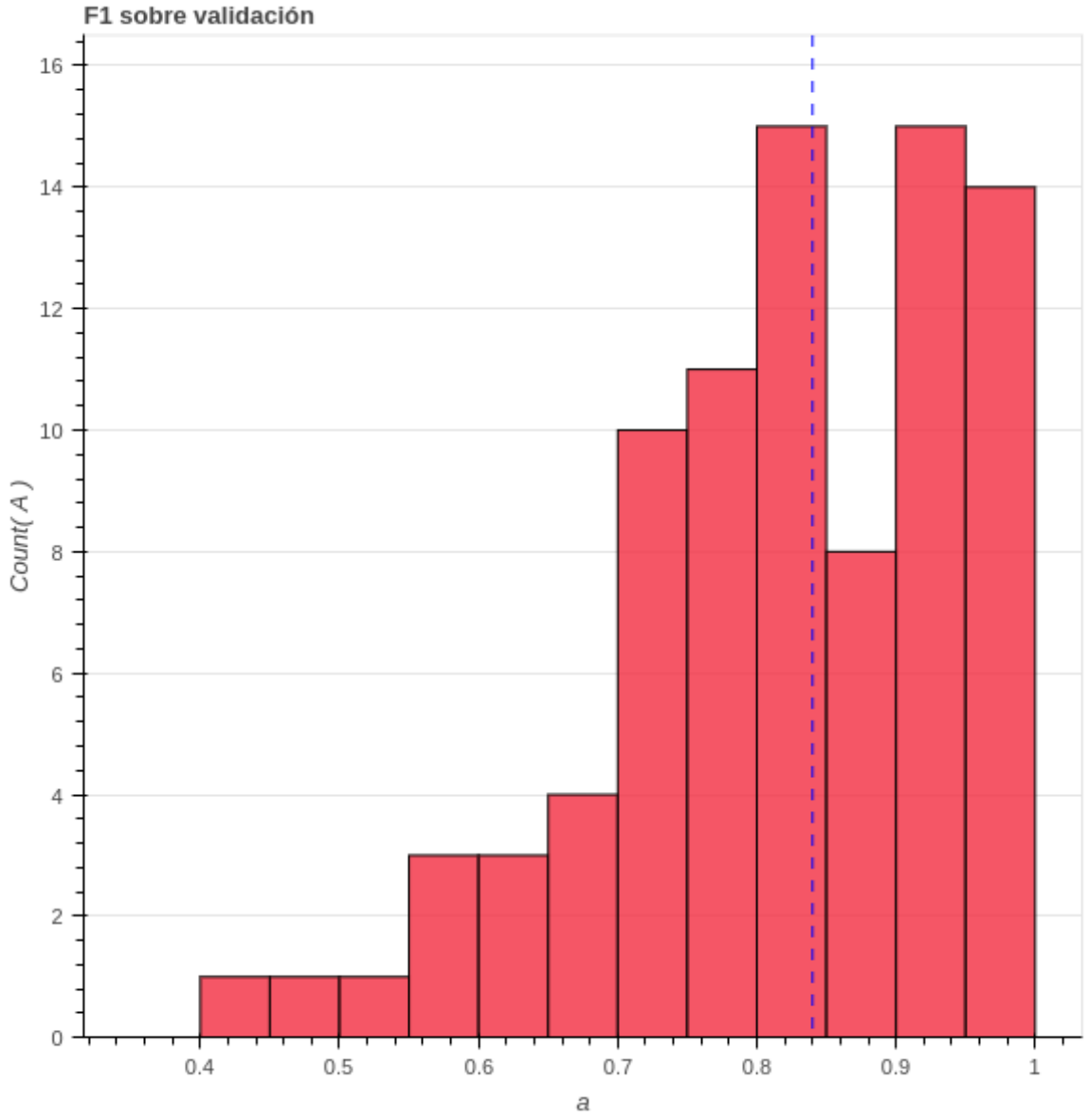

Figura 5.1: histograma de puntajes `F1` sobre M_u^{aj} y valor promedio.

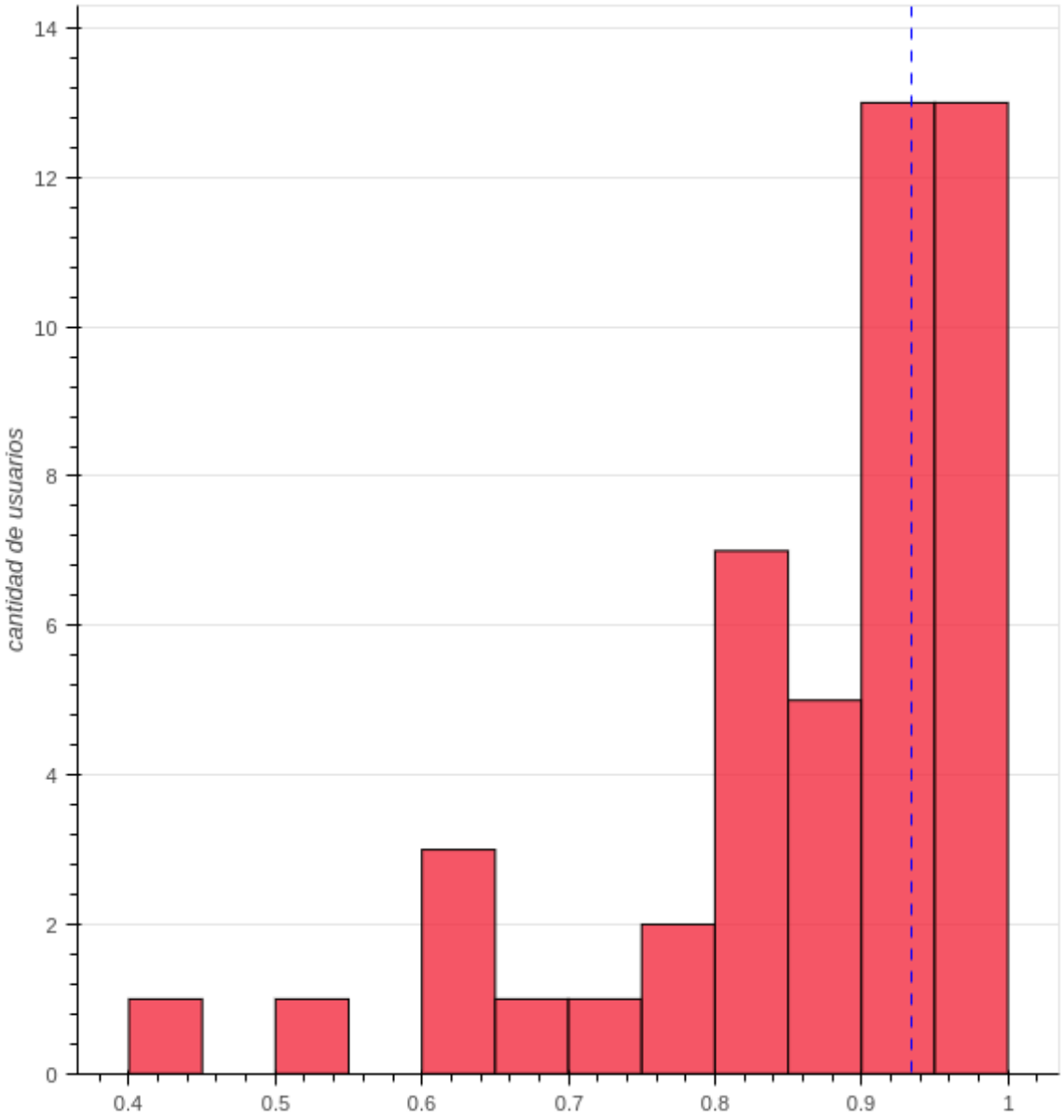

Figura 5.2: histograma de puntajes `precision` sobre M_u^{aj} y valor promedio.

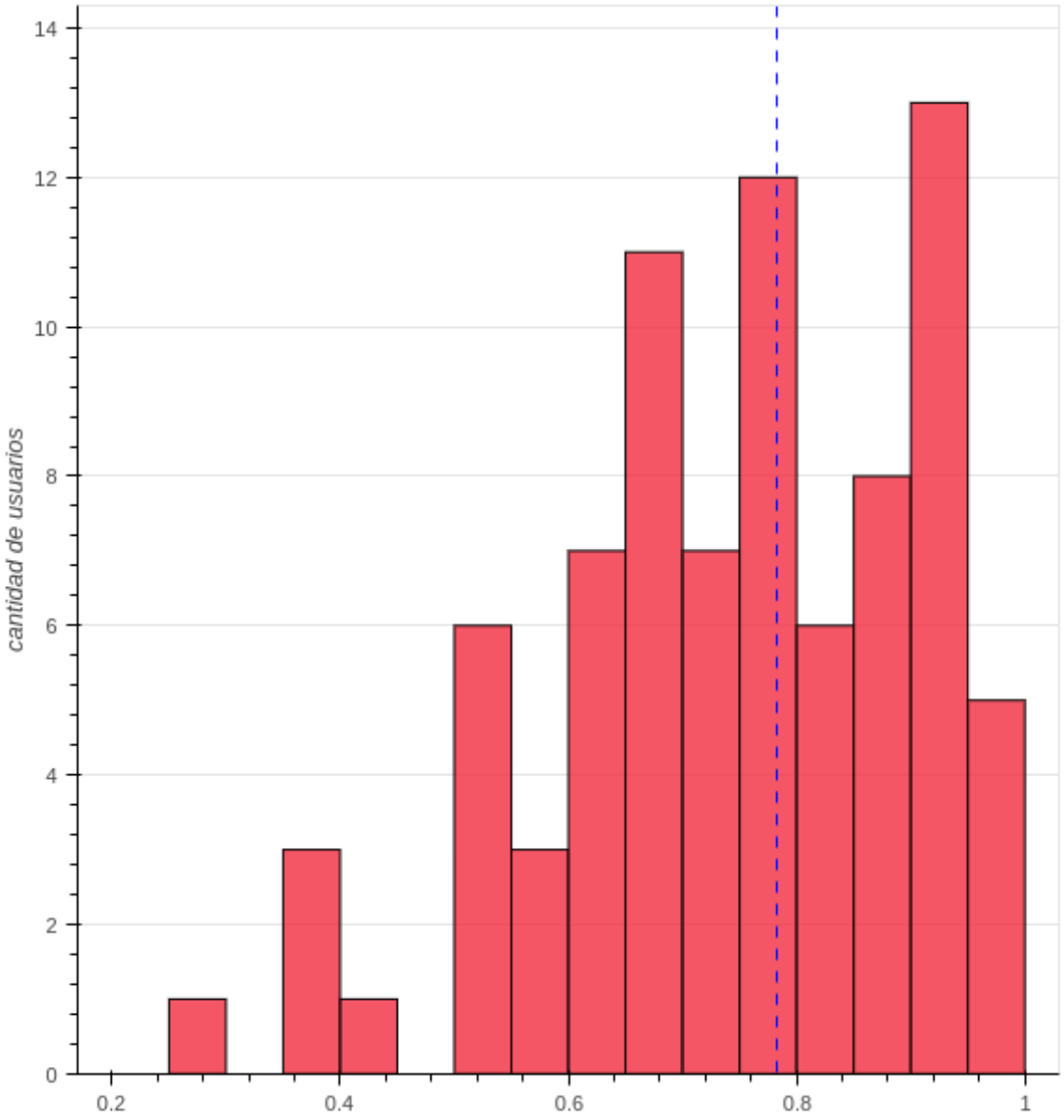

Figura 5.3: histograma de puntajes `recall` sobre M_u^{aj} y valor promedio.

5.2. Agregando análisis de contenido

Estudiamos a continuación la posibilidad de mejorar los clasificadores de la sección anterior con características extraídas del texto de los tweets, usando técnicas de Procesamiento de Lenguaje Natural.

Describiremos el proceso de transformación de texto en características numéricas en todas sus etapas, desde la normalización y limpieza del texto, pasando por su tokenizado y terminando por la reducción de dimensionalidad por medio de un modelado temático LDA.

Analizaremos el fenómeno de sobreajuste que se presenta, y propondremos algunas estrategias posibles para mitigarlo.

5.2.1. Selección de usuarios

En principio, queremos concentrar el análisis en aquellos usuarios para los cuales el modelo social tuvo un desempeño de menor calidad. Tomamos entonces todos los usuarios que tuvieron una medida `F1` menor a $0,75$ en M_u^{aj}. Nos interesa ver además si se produce alguna mejora en casos de mejor calidad, por lo que extendemos la muestra con una selección aleatoria de 10 usuarios más.

Esto da lugar a un conjunto G_{PLN} de 33 usuarios, en los que basaremos todos los análisis de esta sección.

5.2.2. Pre-procesamiento

A la hora de filtrar artículos de interés, nuestros algoritmos necesitan "entender" el contenido de los textos que se le presentan. En este contexto, "entender" significa ser capaz de identificar cuándo dos artículos comparten el mismo contenido y qué tan similares son entre ellos.

Claramente, comparar las representaciones binarias de dos textos no es suficiente, ya que la codificación de un texto no da demasiada información explícita sobre su contenido. Por otra parte, incluso si las palabras aparecen en diferente orden, el significado podría ser similar aunque la codificación fuera diferente. Comparar textos palabra por palabra tampoco ayuda, porque cada idioma permite a un autor expresar la misma idea de varias maneras diferentes.

Los textos además están plagados de *stopwords* como verbos auxiliares o artículos. Éstas ocurren en casi cualquier texto y no son tan significativas como por ejemplo los sustantivos. También se encuentran con frecuencia homónimos y sinónimos que dificultan a una computadora la tarea de entender dos textos por simple comparación de las palabras que ocurren.

En esta sección describiremos cómo nuestros algoritmos extraen características de los artículos más allá de la simple ocurrencia de palabras, para así poder lograr comparar

su contenido de forma efectiva. Usualmente, la computadora aprende un espacio vectorial dentro del cuál cada documento se transforma en un vector. A su vez, será posible convertir entre distintas representaciones mediante transformaciones lineales (matrices).

Describimos ahora las diferentes etapas de procesamiento del texto, que convierten a un tweet en un vector de características numéricas aptas para ser procesadas por algoritmos de clasificación.

Normalización

Como primer paso, a cada tweet se le aplican las siguientes transformaciones de normalización:

- remover URLs
- pasar a minúsculas
- quitar acentos
- remover caracteres inusuales
- contraer repeticiones de vocales (ej.: `goooooool` a `gol`)
- normalizar espacios en blanco

Tokenizado

A continuación procedemos a tokenizar el texto, a través de las siguientes etapas:

1. remover puntuación
2. partir en palabras
3. remover *stopwords*
4. reducir palabras a su raíz (stemming)
5. remover palabras de 1 sola letra

Para el paso 2 se usó el método `word_tokenize` de `NLTK`, de donde también se obtuvo la lista de *stopwords* en español y se utilizó el *stemmer* `SnowballStemmer` que soporta también este idioma.

Extracción de frases

Existen en todo corpus de texto expresiones de dos o más palabras (frases) que se usan frecuentemente y representan un concepto relevante. A fines de mejorar la representación de conceptos de nuestros modelos es importante detectar estas expresiones y tratarlas como un término único, más informativo que la simple ocurrencia de cada una de sus componentes.

Para este paso se utilizó la clase `Phraser` incluida en `gensim`, que provee un método para detectar frases basado en el conteo de co-ocurrencias. Los pares de palabras adyacentes que aparecen juntos con más frecuencia de la esperada se unen con el caracter `_` y son tratadas como un solo término.

Se corrió este algoritmo sobre el corpus completo T previamente tokenizado, ignorando todas las palabras y frases que ocurren menos de $min\_count = 5$ veces y tomando como frases relevantes aquellos bigramas ab tales que $\frac{(cnt(a,b)-min\_count)*N}{cnt(a)*cnt(b)} > 10$, donde cnt cuenta las ocurrencias de un token en el corpus y N es el tamaño del vocabulario (cantidad de términos diferentes).

A fines de incluir frases de 3 palabras, se hizo una segunda pasada del mismo algoritmo sobre el corpus retokenizado con frases de 2 palabras.

Puede verse en el siguiente bloque de código la definición de un iterador sobre el corpus tokenizado [3] y la aplicación del proceso de extracción de frases descripto:

```
class get_docs(object):
    def __init__(self, corpus):
        self.corpus = corpus

    def __iter__(self):
        for doc in self.corpus:
            tokens = tokenize(preprocess(doc), remove_stopwords=True)
            yield tokens
```

```
from gensim.models.phrases import Phraser, Phrases
```

```
phrases = Phrases(get_docs(corpus), min_count=5)
bigram = Phraser(phrases)
trigram = Phrases(bigram[get_docs(corpus)], min_count=5)
```

Figura 5.4: proceso de extracción de frases

[3] Es un patrón muy común en `gensim` recorrer documentos a través de iteradores. Esto le permite procesar un corpus grande de a un documento a la vez, sin cargar el corpus completo en memoria.

Diccionario de términos

Una vez extraídas las frases y retokenizado el corpus T uniendo las frases extraídas como términos simples, pasamos a generar un diccionario con el vocabulario relevante a nuestro corpus.

En este paso excluimos todo término demasiado infrecuente (que ocurra menos de 3 veces) o demasiado común (presente en más del 30 % de los tweets).

Como resultado obtuvimos un diccionario D de 26201 términos.

Bag of terms

Una vez creado nuestro diccionario de términos (tokens o frases), podemos representar cualquier texto como un multiconjunto (*bag*) de los términos que ocurren en él (donde no importa el orden, pero sí se cuenta la cantidad de repeticiones). En nuestro caso, al tratarse de textos cortos (≤ 140 caracteres), en la práctica estas representaciones siempre resultaran ser un conjunto simple de términos (con 0 o 1 ocurrencia de cada término).

Dado que el diccionario de términos relevantes $D = \{t_1, \ldots, t_{26201}\}$ está fijo, esta representación da lugar a un vector de características enteras (que en nuestro caso podemos pensar como booleanas):

$$v_{BOT}(tweet) := [count(t_i, tokens(tweet))]_{i=1}^{26201}$$

donde $count(t, tokens)$ cuenta las ocurrencias del término t en la lista $tokens$.

En la práctica, v_{BOT} se representa como un vector ralo (*sparse*) en el que sólo se indican los valores de sus posiciones no nulas, como puede verse en la figura 5.5.

```
ind = 90172
```

```
print all_tweets_text_es[ind]
```

```
#HaceInstantes Abrazo al Congreso organizado por gremios docentes después de la represión policial de ayer. http
s://t.co/ToyertpI70
```

```
print [dictionary[i] for (i,c) in bow[ind]]
```

```
[u'represion_policial', u'gremi_docent', u'ayer', u'organiz', u'abraz_congres', u'despu']
```

```
print bow[ind]
```

```
[(221, 1), (1176, 1), (3055, 1), (6257, 1), (14203, 1), (25141, 1)]
```

Figura 5.5: Ejemplo de un tweet t, sus términos y su representación *bag-of-terms* rala

LDA

Entrenar modelos sobre vectores de dimensionalidad muy alta acarrea problemas tanto de eficiencia como de sobreajuste. Es por esto que necesitamos reducir las representaciones

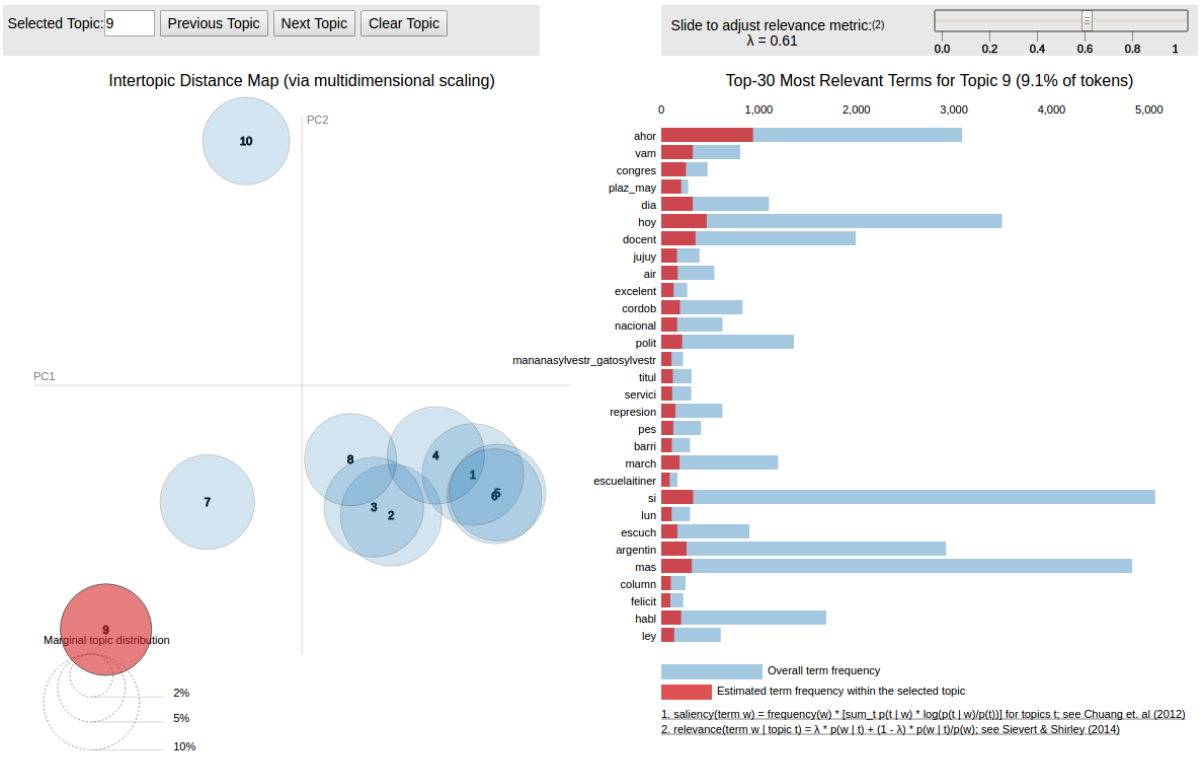

Figura 5.6: LDA de 10 temas y términos representativos de uno de ellos.

bag-of-terms a una representación que tenga menos dimensiones, pero que siga capturando información importante sobre el contenido de cada tweet.

Para tal fin, utilizamos el modelo LDA, que descubre temas subyacentes dentro de un corpus dado y los representa como distribuciones de probabilidades de ocurrencia de términos. Una vez generados estos temas, el modelo permite transformar cada vector bag-of-terms en un vector de probabilidades sobre cada tema.

Se realizaron pruebas con modelos de 10 y 20 temas, entrenados sobre el corpus completo de tweets en español usando el algoritmo `LdaMultiCore` de `gensim`, usando 10 pasadas del mismo en cada caso.

Podemos ver en la figura 5.6 una visualización `pyLDAvis` del modelo de 10 temas, junto con los términos más relevantes de uno de los temas extraídos.

5.2.3. Clasificación social + PLN

Presentamos ahora los resultados de las pruebas de clasificación al agregar las características PLN descritas anteriormente. Experimentamos con dos variantes de LDA con 10 y 20 temas, sobre cada uno de los 33 usuarios de G_{PLN}.

En todos los casos se entrenan los algoritmos sobre el mismo conjunto de entrenamiento M_u^{en} extendido con las características PLN correspondientes (`LDA10`, `LDA20`).

Para la comparación del desempeño de las distintas variantes se utilizará el conjunto de evaluación M_u^{ev}, que contiene datos que no han sido utilizados en el entrenamiento ni tampoco se han empleado para la selección de los usuarios G_{PLN} (elegidos en base a la calidad de los modelos sociales sobre el conjunto de ajuste M_u^{aj}).

Se utilizará al igual que antes una validación cruzada de 3 partes para elegir los mejores parámetros de un clasificador `SVC`.

Al agregar características lingüísticas, comenzamos a encontrarnos con varios casos de usuarios para los que la optimización de parámetros tardaba mucho o no terminaba debido a problemas de no convergencia. Es por esto que decidimos imponer un máximo de 5000 iteraciones al entrenamiento de clasificadores en esta sección.

Se agregó además un paso previo de escalado de columnas para mejorar la estabilidad numérica de los algoritmos y balancear la importancia de características sociales y lingüísticas.

Evaluación

Como puede observarse en la figura 5.7, los modelos extendidos con LDA tienden a producir mejoras respecto del modelo social sólo en casos de `F1` bajo, y van degradando su desempeño a medida que el `F1` del modelo social crece.

El modelo con $LDA10$ produjo mejoras sólo para 4 de los 33 usuarios en G_{PLN} y produjo sobre estos una mejora media de $2,8\,\%$. El modelo $LDA20$ mejoró el `F1` de 5 usuarios, con una mejora media de $3,8\,\%$.

Sobreajuste y características booleanas

Analizando el desempeño sobre los datos de entrenamiento (ver 5.8), vemos que en ambos casos se produce un salto abrupto en la medida `F1` sobre los datos de entrenamiento. Esto contrasta fuertemente con lo observado sobre los datos de evaluación: mínimas mejoras para algunos usuarios y pérdida de calidad en el resto. Todo esto indica una fuerte tendencia al sobreajuste cuando se agregan las características LDA.

Una causa frecuente de sobreajuste es tener características demasiado descriptivas sobre un conjunto de datos no tan grande. Esto aumenta la posibilidad de que un algoritmo de clasificación aprenda una descripción demasiado ajustada a los datos vistos que no generalice bien a otros datos.

Una forma sencilla de intentar reducir la descriptividad de características temáticas como las producidas por LDA es discretizarlas. Podemos por ejemplo generar características booleanas imponiendo un valor umbral a los probabilidades por tema. Es decir, en lugar de describir un texto con las proporciones de cada tema que lo componen, lo representamos simplemente por el conjunto de los temas más relevantes.

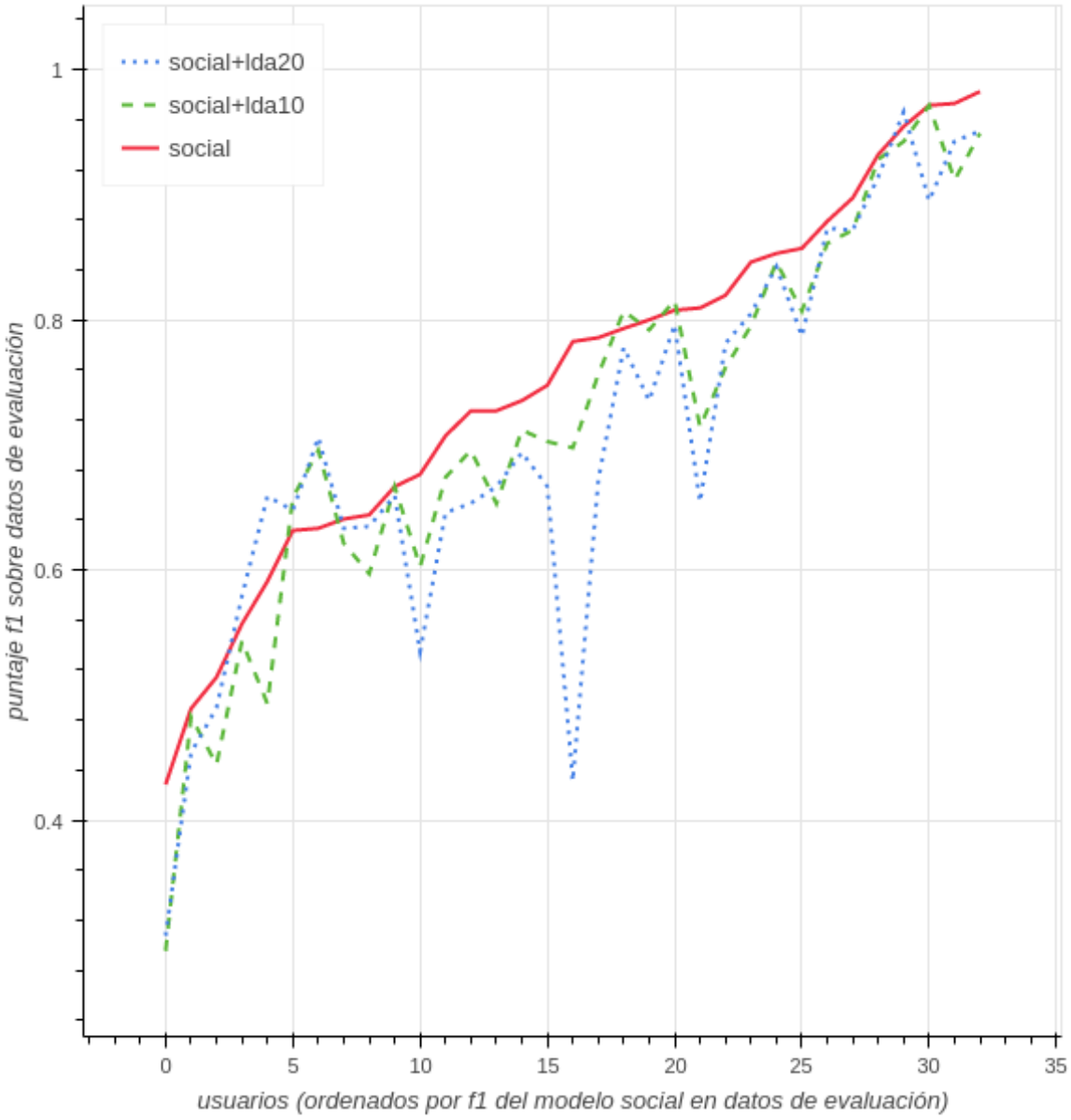

Figura 5.7: puntajes `F1` con características LDA sobre M_u^{ev}

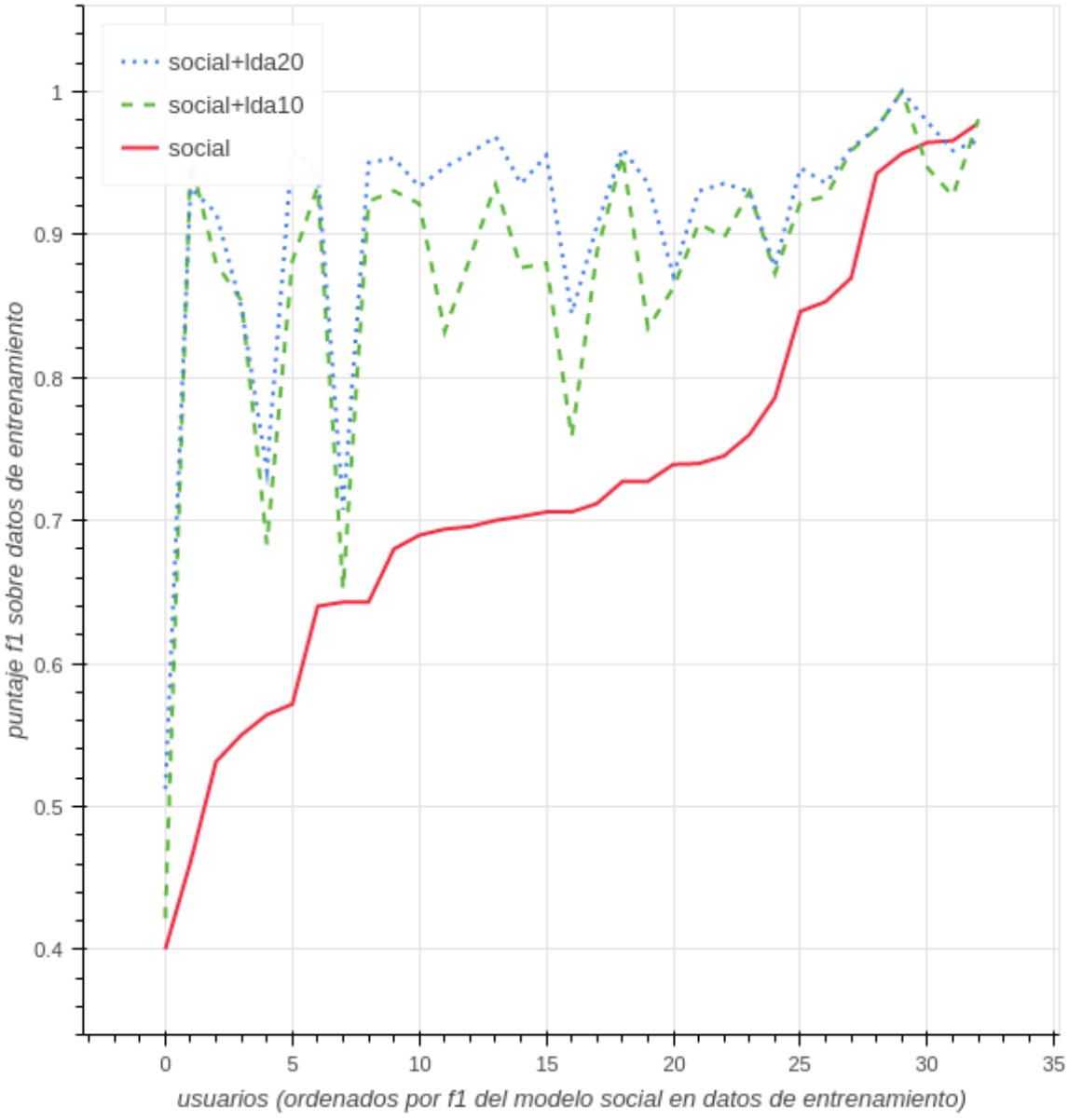

Figura 5.8: puntajes `F1` con características LDA sobre M_u^{en}

Discretizamos las características LDA asignando un 1 para cada tema con puntaje $\geq 0,25$ y 0 en caso contrario. Repetimos el proceso de selección por validación cruzada de modelos con estas nuevas características (siempre en combinación con las características sociales), pero no observamos una mejora significativa en los resultados (5.9): si bien los conjuntos de usuarios con mejor `F1` crecen un poco (8 usuarios en el caso de `LDAbool10`, 7 para `LDAbool20`), la mejora media es menor que antes de la discretización ($2,4\,\%$ y $2,5\,\%$ respectivamente).

Vemos además en (5.10) que no ha habido una reducción apreciable del sobreajuste.

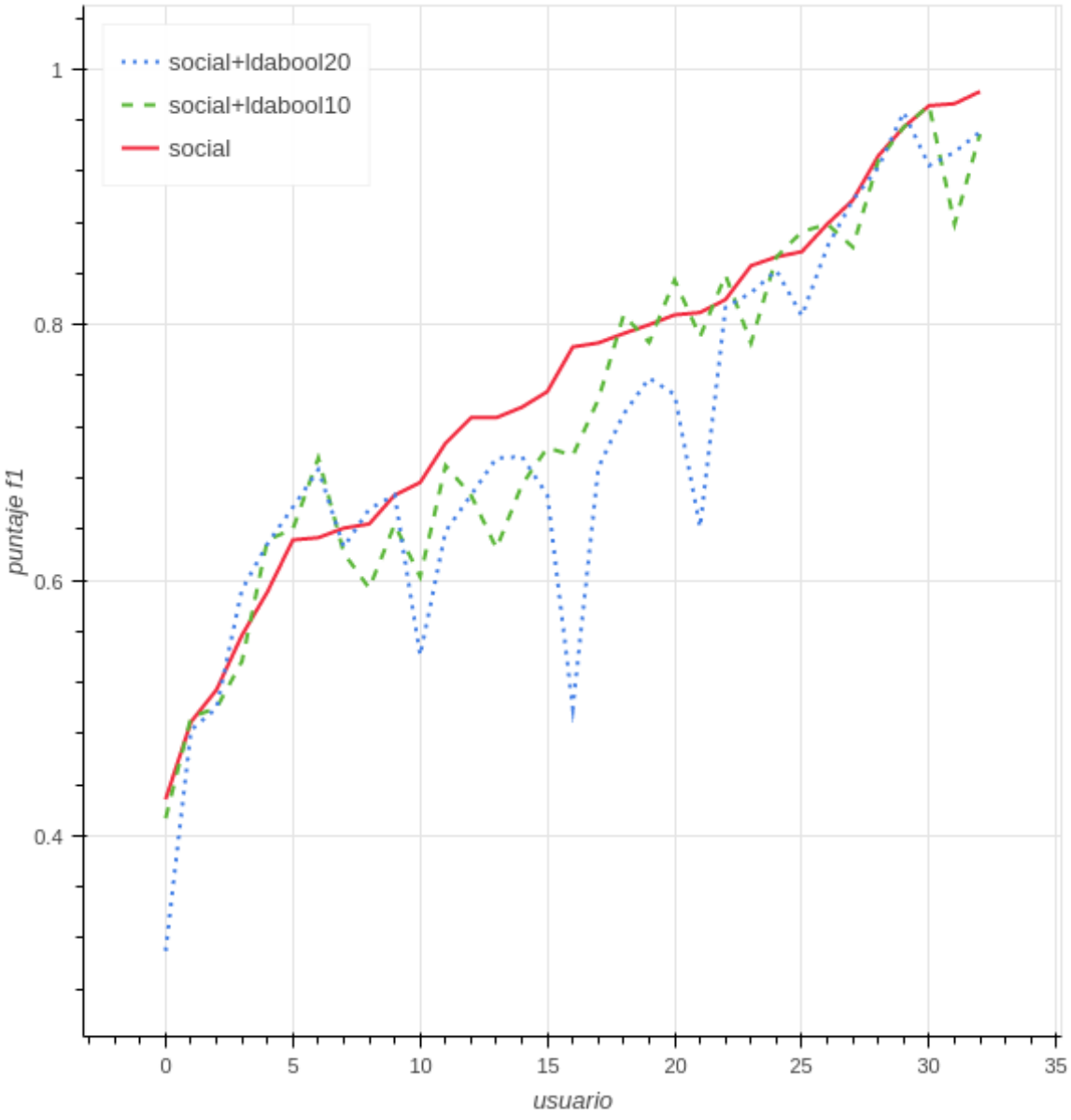

Figura 5.9: puntajes `F1` con características `LDAbool` sobre M_u^{ev}

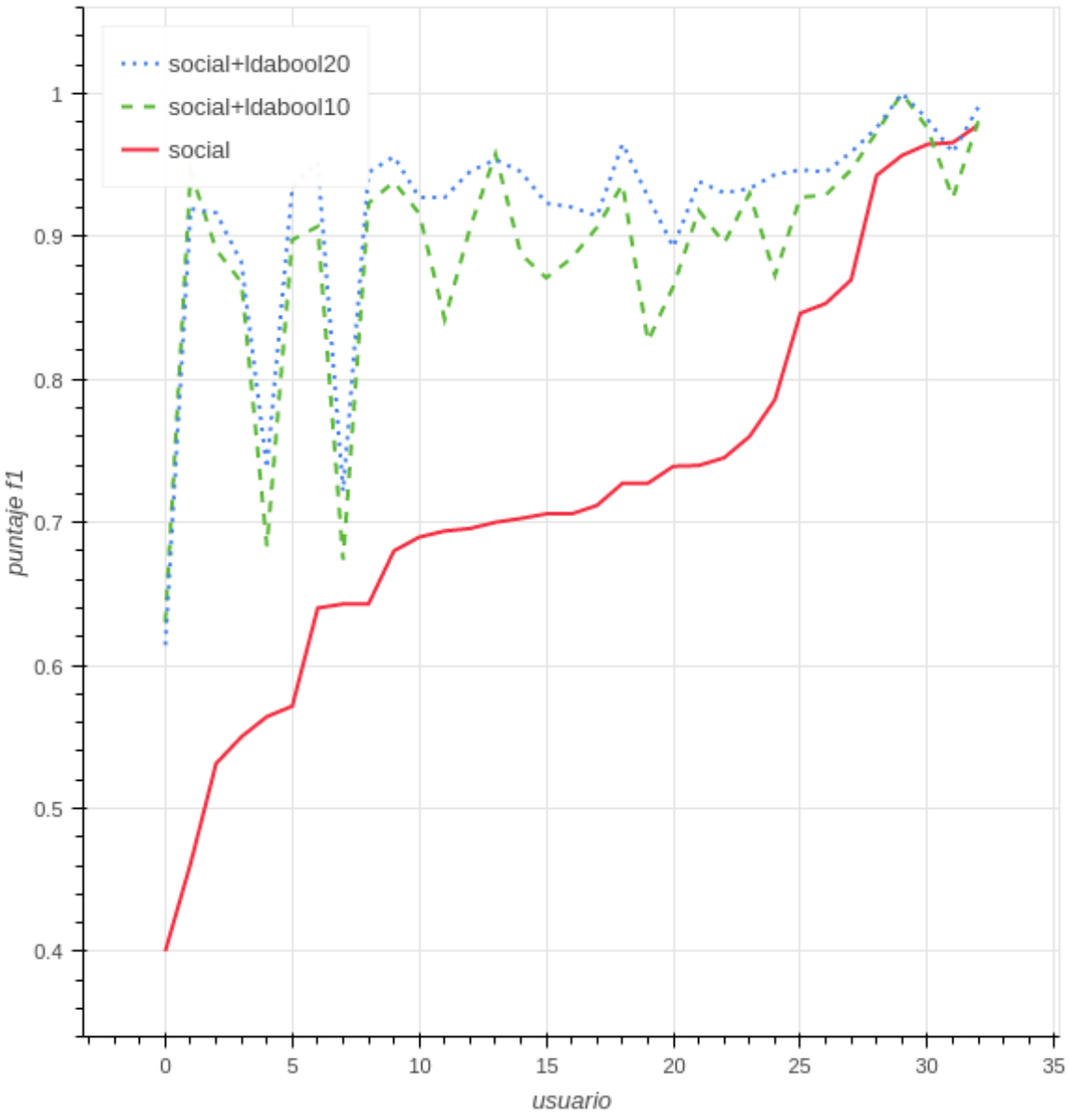

Figura 5.10: puntajes `F1` con características `LDAbool` sobre M_u^{en}

Capítulo 6

Trabajos relacionados

Exploramos en este capítulo algunos trabajos con temáticas afines al nuestro, describiendo cómo se relacionan y diferencian.

6.1. Re-tweeting from a Linguistic Perspective [11]

Al igual que este trabajo, hace análisis de *retweets*, pero como fenómeno global, no centrado en un usuario en particular como en nuestro caso. Estudian qué características del contenido de un tweet lo hacen más propenso a recibir muchos retweets, enfocándose en características lingüísticas como estilo de escritura del tweet, tiempos verbales usados, empleo de emoticones, hashtags, etc.

Se enfocan además en 3 tipos de tweets según la intención del emisor (*Opinión*, *Anuncio* e *Interacción*), y estudian el fenómeno de *retuiteo* en cada clase. Analizan también cómo las características lingüísticas pueden ayudar a identificar automáticamente la clase de un tweet, lo que ayuda a determinar mejor su potencial de retuiteo.

6.2. Recommendation as Classification: Using Social and Content-Based Information in Recommendation [12]

Analiza la combinación de información social y de contenido del ítem a recomendar (en este caso películas). Al igual que nosotros, plantea la recomendación como un problema de clasificación binaria entre interesante/no-interesante, pero se diferencia en que los datos de entrenamiento no están dados en este mismo formato, sino en forma de puntajes que los usuarios han dado a las películas, problema que resuelven poniendo umbrales a los puntajes.

La información social se codifica como características de conjunto en lugar de booleanas como en nuestro caso. La información de contenido no se da en forma de texto plano, sino que consiste en atributos de las películas a recomendar (director, actores, idioma, locación, etc.), en general de naturaleza categórica.

Una diferencia fundamental es que emplean un algoritmo de aprendizaje inductivo para la clasificación (*Ripper*) que intenta aprender reglas `IF-THEN` a partir de los datos de entrenamiento.

6.3. Feature Weighting in Content Based Recommendation System Using Social Network Analysis [13]

Estudian también la posibilidad de mejorar recomendaciones de contenido usando análisis de redes sociales. A diferencia de nuestro caso, la red se construye sobre los ítems a recomendar (no necesariamente textos) y no sobre los usuarios. Se usan los pesos de las aristas en esta red (cantidad de usuarios con interés en ambos ítems) para, a través de ciertas ecuaciones lineales, determinar pesos para las características de los ítems. Éstos luego se emplean para intentar definir una medida de similaridad de ítems que se correlacione bien con el nivel de acuerdo social sobre los mismos.

Capítulo 7

Conclusiones y Trabajo Futuro

Damos a continuación algunas conclusiones y reflexiones sobre el trabajo realizado, y delineamos las que a nuestro entender son algunas de las líneas de posible trabajo futuro más interesantes.

7.1. Conclusiones

Confirmamos durante este trabajo nuestra idea de que el análisis de redes sociales puede proveer herramientas muy útiles a la hora de implementar recomendadores de contenido y de entender mejor las conexiones entre los intereses de un usuario y su entorno social. Nos resultó sorprendente comprobar el nivel de calidad de las predicciones basadas en entorno social, sin siquiera tener en cuenta el contenido.

Queda también en evidencia que la extracción de temas con LDA, más allá de su utilidad en tareas de exploración y comprensión de corpus, tiene bastante potencial para describir contenido de texto en pocas dimensiones. De momento sólo hemos logrado mejoras en los casos de menor calidad del modelo social. Para hacer un mejor aprovechamiento de las características LDA en el caso general, restaría implementar técnicas que reduzcan el sobreajuste que actualmente se produce al agregarlas.

Desde el lado de la obtención de datos, encontramos que Twitter se presenta como una fuente muy rica, dinámica y accesible de datos de texto e interacciones sociales, lo cual abre posibilidades para muchos otros trabajos de análisis y modelado computacional de fenómenos sociales.

7.2. Trabajo Futuro

7.2.1. Reducir sobreajuste

Un primer paso para mejorar lo desarrollado es buscar técnicas para reducir el sobreajuste al agregarse características LDA. Un camino posible es intentar mejorar la calidad y generalidad de los temas extraídos usando adaptaciones de LDA específicamente pensadas para tratar con tweets. El modelo LDA está originalmente pensado para textos largos que posiblemente traten de varios temas. En el caso de Twitter, lo más común es que un tweet hable de un solo tema principal. Además, la corta longitud de los textos también reduce el nivel de co-ocurrencia de términos de un mismo tema, lo cual dificulta aprender buenos modelos. Existen diversas adaptaciones de LDA que proponen atacar estos problemas con estrategias de sumarización o modificaciones al modelo generativo subyacente. Hemos comenzado a implementar los enfoques propuestos en [14] y [15], pero no llegamos a incluirlos en este trabajo.

7.2.2. *Cold start*

En sistemas de recomendación basados en contenido como **Cogfor** se da el problema del *"cold start"*, que consiste en no saber qué recomendarle a un usuario nuevo, por no contar con interacciones previas con las que entrenar un recomendador.

Se puede explorar, integrando con redes sociales como Facebook o Twitter, la posibilidad de encarar este problema a través de la recomendación basada en entorno social aquí propuesta.

7.2.3. Características adicionales

Algunas características que planeábamos agregar a nuestros modelos y no tuvimos tiempo de considerar:

- número/porcentaje de retweets entre usuarios seguidos.
- número/porcentaje de retweets entre seguidos de seguidos.
- número/porcentaje de retweets entre *amistades* (usuarios que se siguen mutuamente con el usuario central)
- *afinidad* promedio con el usuario elegido entre los *retweeters* del tweet observado, para alguna medida de afinidad en grafos.

7.2.4. Temporalidad

El particionado de nuestros conjuntos de datos en conjuntos de *entrenamiento*, *validación* y *evaluación* se realizó al azar. Esto genera que en cada porción haya tweets de cualquier momento del período de tiempo del cual se tomó la muestra. También la forma en que construimos los vectores de características en principio está contaminada de información sobre el futuro: muchos de los retweets de vecinos usados para construir el vector pueden haberse realizado luego del retweet del usuario elegido.

Sería interesante rehacer los experimentos usando particiones que tengan en cuenta el orden temporal de los tweets, para tener una mejor idea de la capacidad de aprendizaje en un contexto *online*. A su vez, podríamos replantear la construcción de características para tener en cuenta sólo retweets anteriores al que se quiere predecir.

7.2.5. Modelos independientes del usuario

Los modelos que exhibimos en este trabajo dependen fuertemente del usuario central elegido y su entorno de usuarios seguidos (de hecho, cada característica está asociada a un usuario puntual de este entorno). Usando medidas de afinidad en grafos se pueden agrupar los usuarios de un entorno en una cantidad fija de grupos, y tomar como características las cantidades de retweets en cada uno de estos grupos. Modelar el entorno social de esta manera permitiría entrenar modelos que puedan ser aplicados a cualquier usuario en lugar de a un usuario central elegido.

Bibliografía

[1] Steven Bird, Ewan Klein, Edward Loper: "Natural Language Processing with Python". *O'Reilly Media, 2009*

[2] Matthew A. Russell: "Mining the Social Web". *O'Reilly Media, 2013*

[3] Tom M. Mitchell: "Aprendizaje Automático". *McGraw-Hill Science/Engineering/Math; (Marzo 1997)*

[4] S. Maldonado, R. Weber: "Modelos de Selección de Atributos para Support Vector Machines". *McGraw-Hill Science/Engineering/Math; (Marzo 1997)*

[5] `https://es.wikipedia.org/wiki/Multiplicadores_de_Lagrange`

[6] David Blei "Probabilistic Topic Models". *Communications of the Association for Computing Machinery (Abril 2012).*

[7] `https://en.wikipedia.org/wiki/Generative_model`

[8] `https://en.wikipedia.org/wiki/Dirichlet_distribution`

[9] `https://es.wikipedia.org/wiki/Máxima_verosimilitud`

[10] M Hoffman, FR Bach, DM Blei "Online learning for latent dirichlet allocation". *advances in neural information processing systems, 2010.*

[11] Aobo Wang, Tao Chen, Min-Yen Kan "Re-tweeting from a Linguistic Perspective". *Proceedings of the 2012 Workshop on Language in Social Media (LSM 2012).*

[12] Chumki Basu, Haym Hirsh, William Cohen "Recommendation as Classification: Using Social and Content-Based Information in Recommendation". *AAAI-98 Proceedings.*

[13] Souvik Debnath, Niloy Ganguly, Pabitra Mitra "Feature Weighting in Content Based Recommendation System Using Social Network Analysis". *WWW 2008 - Poster Paper - Abril 21-25, 2008 · Beijing, China.*

[14] Rishabh Mehrotra, Scott Sanner, Wray Buntine, Lexing Xie "Improving LDA Topic Models for Microblogs via Tweet Pooling and Automatic Labeling". *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval - Julio 2013.*

[15] Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan, Xiaoming Li "Comparing Twitter and Traditional Media using Topic Models". *Proceedings of the 33rd European Conference on Advances in Information Retrieval - 2011.*

[16] Cogfor `https://www.linkedin.com/company-beta/3202502/`.

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

[18] Scrapy https://docs.scrapy.org/en/latest/.

[19] Splinter https://splinter.readthedocs.org/en/latest/.

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

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

[22] graph-tool https://graph-tool.skewed.de/static/doc/index.html.

[23] Pandas http://pandas.pydata.org/pandas-docs/stable/.

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

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

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

[27] gensim https://radimrehurek.com/gensim/tutorial.html.

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

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

[30] Bokeh http://bokeh.pydata.org/en/latest/docs/user_guide.html.