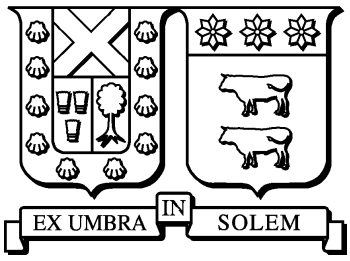
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INFORMÁTICA

SANTIAGO – CHILE



"EVALUACIÓN DE EFICIENCIA DE ALGORITMOS DE ESTIMACIÓN DE MODELOS DE TÓPICOS DINÁMICOS"

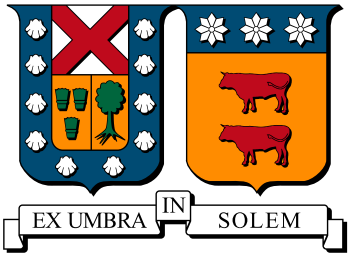
IGNACIO JAVIER ESPINOZA VILLARROEL

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: MARCELO MENDOZA

DICIEMBRE 2018

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO – CHILE



"EVALUACIÓN DE EFICIENCIA DE ALGORITMOS DE ESTIMACIÓN DE MODELOS DE TÓPICOS DINÁMICOS"

IGNACIO JAVIER ESPINOZA VILLARROEL

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: MARCELO MENDOZA

PROFESOR CORREFERENTE: VICTOR CODOCEDO

DICIEMBRE 2018

Agradecimientos

Agradezco totalmente a mi familia por el esfuerzo que entregaron para darme la posibilidad de estudiar en la universidad. Siempre han estado presentes en todo evento, preocupándose por mi tanto en las buenas como en las malas.

Quiero agradecer también los profesores del departamento con los que tuve la oportunidad de tener clases, por todo el conocimiento útil entregado. En especial quiero agradecer a Marcelo 'el profe' Mendoza por toda su disposición, tiempo y ayuda a lo largo de lo que fue este trabajo de título, en los proyectos pasados y la actual tesis de magíster. Es un siete en todo sentido y una gran gran persona.

Agradecer a todos quienes me acompañaron en mi estancia en la universidad, en especial las chiquillas Nicolás 'Nix' Aravena, Fernando 'Feño' Llorens, Victor 'Breco' Franchis, Nicolás 'Pollo' del Campo y Daniel Rivera. Por todos esos buenos ratos de ocio, diversión y estudios compartidos.

Finalmente quiero agradecer a la mejor persona del mundo, Fernanda Weiss, por todo su amor y cariño en este tiempo y por darme ánimos y fuerza para llegar a este punto.

Resumen

Hoy en día, hay un explosivo aumento de datos generados gracias al uso masivo de Internet, a variadas fuentes de información y a los millones de usuarios activos en redes sociales. Una parte importante de este contenido está en formato texto, el que describe un conjunto de temas o tópicos latentes que cambia en el tiempo, los que pueden ser utilizados para clasificar y/o confeccionar síntesis de información. El problema es que para un ser humano es imposible analizar todas estos datos sin utilizar herramientas automatizadas para esta tarea.

Algoritmos asociados al problema, como Dynamic Topic Models (DTM), analizan la evolución de los tópicos latentes en una colección de documentos a través del tiempo. Si bien DTM ha demostrado generar modelos de buena calidad demora en analizar grandes colecciones de datos, lo que lo inhabilita de trabajar en aplicaciones *online*, donde la latencia de respuesta debe ser baja. Por esto es menester crear algoritmos sofisticados que entreguen buenos resultados y en un menor tiempo.

En este trabajo se realiza un estudio y evaluación de tres algoritmos de modelado de tópicos dinámicos mediante un set de experimentos que miden la calidad de los modelos que generan y el tiempo asociado a este trabajo. Así, se pudo determinar bajo qué condiciones de operación los algoritmos presentan ventajas y desventajas frente al resto. A su vez, se busca analizar como escalan estos algoritmos en función de la cantidad de datos y el poder de cómputo asignado. Estos resultados sirven como guía para escoger un método a implementar en una aplicación de la vida real.

Abstract

Nowdays, there is an explosive increase of generated data thanks to the massive use of the Internet, various sources of information and the millions of active users on social networks. An important part of this content is in text format, which describes a set of themes or latent topics that change over time, which can be used to classify and/or make information synthesis. The problem is that for a human being it is impossible to analyze all this data without using automated tools for this task.

Algorithms associated with the problem, such as Dynamic Topic Models (DTM), analyze the evolution of latent topics in a collection of documents over time. Although DTM has demonstrated to generate models of good quality, it takes a long time analyzing big collections of data, what disables it of working in online applications, where the latency must be low. This is why it is necessary to create sophisticated algorithms that deliver good results and in a shorter time.

In this work, a study and evaluation of three dynamic topic modeling algorithms is performed through a set of experiments that measure the quality of the models they generate and the time associated with this work. Thus, it was determined under what operating conditions the algorithms have advantages and disadvantages compared to the rest. In turn, we seek to analyze how these algorithms scale based on the amount of data and the computing power assigned. These results serve as a guide to choose which method can be implemented in a real life application.

Índice de Contenidos

Índice de cuadros

Índice de figuras

Glosario

Tópico: Conjunto de términos relacionados capaces de describir el contenido de uno o más documentos.

Corpus: Colección de documentos de texto.

Stopword: Palabras que por si solas no tienen significado, como por ejemplo artículos, pronombres y preposiciones. Generalmente son eliminadas en el preprocesamiento de texto.

LDA: Latent Dirichlet Allocation

DTM: Dynamic Topic Model

CLDA: Clustered Latent Dirichlet Allocation

ToT: Topics Over Time

MPI: Message Passing Interface (Interfaz de Paso de mensajes).

Capítulo 1

Introducción

1.1. Definición del Problema

En los últimos años ha habido un aumento abismal de datos generados gracias al uso de Internet. Se estima que cada día son producidos 2.5 quintillones de bytes de datos[1] donde, por ejemplo, en cada minuto se producen 46.740 nuevos posts en Instagram, 15.220.700 de mensajes de texto son enviados y 456.000 tweets son generados por usuarios[2]. Esta cantidad de información puede resultar abrumadora para una persona común y corriente que desea buscar un sujeto en particular en su buscador preferido.

Una característica en particular que posee la información es que puede ser agrupada en diferentes temas o tópicos que describen a grandes rasgos el contenido expuesto en estas fuentes de datos. Al no ser toda esta información relevante para una búsqueda se necesita analizar el contenido de estos documentos, páginas webs, entre otros, para evaluar el valor que tiene para el usuario. Tareas como ésta necesitan de un set de herramientas que automaticen la búsqueda y descubrimiento en estas grandes bases de conocimiento. Además, el usuario espera que el tiempo de respuesta sea bajo y no le requiera destinar muchos recursos computacionales.

[1] https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/

[2] https://www.domo.com/learn/data-never-sleeps-5

Herramientas conocidas en este ámbito son los *Topic Models*, modelos probabilísticos que logran encontrar patrones de palabras en colecciones de documentos llamados tópicos. Uno de los primeros modelos es *Latent Dirichlet Allocation* (LDA) modelo generativo que permite encontrar tópicos subyacentes que puedan ser utilizados para explicar el conjunto de observaciones o corpus. El modelo asume que cada documento es producido por una distribución multinomial de documentos sobre un número definido de tópicos, cuyas palabras se atribuyen a la probabilidad de ocurrencia de la palabra en los tópicos que presenta ese documento. La distribución de documentos de un tópico es finalmente una representación de baja dimensionalidad, pudiendo ser utilizada para tareas de clasificación y búsqueda.

Al ser un modelo estático en el tiempo no captura la evolución que puedan tener los tópicos al ir cambiando el contenido de los documentos en el tiempo. Para solucionar este problema surgió un nuevo modelo generativo llamado *Dynamic Topic Models* (DTM), que extiende LDA tomando una distribución Gaussiana sobre los cortes temporales de un corpus, modelando cada tópico como una transición suave del mismo tópico pero del intervalo de tiempo inmediatamente anterior.

Si bien DTM ha demostrado generar buenos resultados de corpus separados por espacios de tiempo gracias a la evolución suavizada de los tópicos, no logra escalar para analizar grandes colecciones de datos. Esta falencia lo inhabilita para trabajar en aplicaciones que necesiten una respuesta con baja latencia y que no demanden una elevada cantidad de recursos para mantener los datos en memoria. Además, DTM no modela el nacimiento y desaparición de los tópicos, asumiendo que los tópicos extraídos son persistentes en el tiempo. Por ejemplo, en el escenario de clasificación de noticias divididas respecto al mes de publicación, un evento noticioso nuevo que no habla de los tópicos actuales será clasificado como uno de esos y las palabras nuevas en el corpus tendrán un menor peso debido a su baja ocurrencia en el conjunto de intervarlos de tiempo completo. En ese caso se esperaría tener un tópico que hable solamente de ese acontecimiento.

Otro problema que rodea a DTM es su difícil paralelización puesto que hay una fuerte dependencia entre los parámetros del modelo, parecidos a una Cadena de Markov. Por lo tanto, si se trabajara en un ambiente simplificado donde no hay una dependencia temporal habría

perdida de información entre un tiempo y otro, impactando en el resultado final del algoritmo.

Dada esta problemática, en este trabajo se propone hacer un estudio del estado del arte para ver qué variaciones de DTM han sido investigadas y cómo éstas buscan mejorar el desempeño en términos de tiempo de cómputo, sin comprometer la calidad de los modelos finales. Además, bajo la misma idea se buscará que algoritmos han logrado paralelizar el modelo original y qué impacto producen en como se trabajan los tópicos y su evolución en el tiempo. Esta investigación estará acompañada de una parte experimental para evaluar las soluciones encontradas.

1.2. Objetivos

Los objetivos generales y específicos se detallan a continuación:

1.2.1. Objetivos generales

El objetivo principal de este trabajo es el estudio y evaluación de eficiencia y eficacia de distintos algoritmos de estimación de modelos de tópicos dinámicos, o LDA dinámico, expuestos en la literatura.

1.2.2. Objetivos específicos

- Determinar bajo qué condiciones de operación los algoritmos presentan ventajas y desventajas sobre otros.
- Definir pruebas de escalabilidad que permitan analizar la eficiencia y eficacia de algoritmos de modelado de tópicos dinámicos.
- Estudiar cómo algoritmos de *clustering* pueden ser utilizados para modelar los tópicos de un conjunto de documentos.

1.3. Estructura del documento

Dicho esto, el presente trabajo de título está organizado de la siguiente manera.

El Capítulo 1 es la introducción al estudio a realizar en el documento, definiendo cuál es el problema y estableciendo los objetivos que guiarán el estudio del estado del arte y futura experimentación.

En el **Capítulo 2: Estado del Arte**, se revisará la literatura en base a los algoritmos de modelado de tópicos y qué avances se han hecho en este último tiempo. Además, se mostrará una herramienta de trabajo con texto y la comparación entre los modelos expuestos previamente.

Habiendo entendido el problema y sus actuales soluciones, en el **Capítulo 3: Experimentos** se analizarán los conjuntos de datos y algoritmos de modelado de tópicos a utilizar siguiendo el procedimiento experimental diseñado para responder a los objetivos expuestos en la introducción. También, serán detalladas las pruebas a efectuar, qué métricas medirán los resultados y la configuración de entorno necesaria para ejecutar los algoritmos.

Con este marco de trabajo, en el **Capítulo 4: Resultados**, se examinará, discutirá y analizará los resultados conseguidos sobre dos conjuntos de datos y en configuraciones de entorno diferentes, con el fin de entender los factores que influyen en el producto de los algoritmos.

Finalmente, en el **Capítulo 5 Conclusiones**, se expondrán las conclusiones del documento en base a los objetivos de la introducción, generales y específicos.

Los capítulos 6 y 7 son **Bibliografía**, referencias utilizada a lo largo de la investigación, y **Anexos**, donde se detallan los registros de cada experimento ejecutado.

Capítulo 2

Estado del Arte

En ese capítulo se presentará el estado del arte respecto a modelos de tópicos dinámicos, recientes estudios sobre el tema, además del contexto necesario para entender el trabajo a realizar en los siguientes capítulos.

2.1. Topic Models

Los algoritmos de modelado de tópicos, o *Topic Modeling* [1], son un conjunto de algoritmos probabilísticos que buscan descubrir y etiquetar grandes colecciones de documentos con información temática. Estos métodos probabilísticos analizan las palabras contenidas en los documentos para descubrir temas o *tópicos* latentes que se desarrollan a lo largo de ellos, cómo están conectados entre si y cómo evolucionan o cambian en el tiempo. Un *tópico* se define como una distribución de probabilidad sobre un vocabulario fijo de términos. Dentro de un documento se puede encontrar uno o más tópicos.

Los algoritmos no requieren que los documentos estén etiquetados previamente, pues los tópicos son extraídos al analizar los textos originales. Gracias a esto, es posible organizar y resumir archivos a gran escala que serían imposibles de realizar por un humano en un tiempo razonable.

2.1.1. Notación y terminología

Se definen los siguientes términos:

- Palabra *(word)*: unidad básica de datos, definida como un ítem de un vocabulario indexada por $\{1, \ldots, V\}$.
- Documento *(document)*: colección de N palabras denotada por $\mathbf{w}= (w_1, w_2, \ldots, w_N)$, donde w_n es la n-ésima palabra en la secuencia. Las palabras pueden aparecer cero o múltiples veces dentro de un documento.
- *corpus*: colección de M documentos denotado por $D = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M\}$.
- Vocabulario: Colección de todas las palabras sin repetir.

2.1.2. Primeros acercamientos al modelado de tópicos

Los primeros acercamientos por buscar técnicas para reducir el texto de un corpus a componentes significantes comenzaron con el método llamado *term-frecuency inverse-document-frecuency* (*tf - idf*). En este método la frecuencia de una palabra w en un documento d es comparada con la ocurrencia en todos los documentos de la colección D, como muestra la ecuación 2.1, estimando su importancia en el documento analizado. El resultado de esta operación proporciona información sobre cuán relevante es una palabra sin darle importancia a las *stopwords*[2].

$$tf - idf(t, d, D) = tf(t, d) \times idf(t, D) = \frac{f(t, d)}{max\{f(w, d) : w \in d\}} \times log\frac{|D|}{|\{d \in D : t \in d\}|} \quad (2.1)$$

A pesar de esto, *tf-idf* no reduce en gran parte la descripción del corpus, no entrega mayor información intra o inter-documentos y tampoco muestra una visión a gran escala de las tendencias [2]. Por ejemplo, un documento puede tener palabras como *avión* y *vuelo* con una alta frecuencia, pero no hay información sobre cuán común es esta combinación.

Latent Semantic Indexing (LSI) es un método de indexación automática que proyecta tanto documentos como términos a un espacio dimensional menor, intentando representar los conceptos semánticos en el documento. LSI utiliza *Singular Value Descompisition* (SVD) en la matriz de *tf-idf* para extraer las correlaciones clave entre términos. Este acercamiento puede lograr una compresión significativa en grandes corpus. Los autores expresan que además puede capturar nociones básicas de lingüística como sinónimos y polisemia [2] [3]. Para el ejemplo anteriormente mencionado, LSI detectará que *avión* y vuelo aparecen juntas con una frecuencia dada.

Probabilistic Latent Semantic Indexing (pLSI) [4], también conocido como *aspect model* extiende LSI en un contexto probabilístico. Se basa en los mismos principios y supuestos conceptuales que LSI, pero utiliza un proceso generativo probabilístico diferente para generar los términos en los documentos de un corpus. pLSI modela cada palabra de un documento como una muestra de un modelo de mezclas, donde las componentes son variables multinomiales aleatorias, las que pueden ser vistas como una representación de tópicos. De este modo cada palabra es generada por un único tópico, y diferentes palabras dentro de un documento pueden ser generadas por distintos tópicos. Cada documento es representado por una mezcla de tópicos.

Siendo este trabajo un gran avance en el modelado de texto, no provee un modelo probabilístico a nivel de corpus. El número de parámetros necesarios para modelar un documento crece linealmente con el tamaño del corpus, lo que genera *overfitting*.

2.2. Latent Dirichlet Allocation

El algoritmo más simple de *Topic modeling* es Latent Dirichlet Allocation (LDA) [2]. La idea básica es que los documentos son representados como mezclas aleatorias de tópicos latentes, donde cada tópico es caracterizado como una distribución de palabras de un vocabulario. Específicamente, se asume que existen K tópicos asociados a la colección, y que cada documento exhibe estos tópicos con diferentes proporciones. Esta suposición es natural puesto que documentos en un corpus tienden a ser heterogéneos, combinando un conjunto acotado

de ideas o temas principales compartidos por la colección.

LDA asume el siguiente proceso generativo para cada documento **d** en un corpus D:

1. Escoger N ~ Poisson(η).
2. Escoger $\theta_d \sim \text{Dir}(\alpha)$.
3. Para cada w_i con i en $i = 1 : N$:
   a) Escoger un tópico $z_n \sim \text{Multinomial}(\theta_d)$.
   b) Escoger una palabra w_n de $p(w_n \mid z_n, \beta)$, probabilidad multinomial condicionada al tópico z_n y a β.

La figura 2.1 muestra el proceso generativo de LDA, usando una cantidad conocida de K temas. Estos temas están compuesto por una distribución de palabras $\beta_{1:k}$, donde cada β_k es una distribución sobre un vocabulario. La proporción de tópicos en un documento d corresponde a θ_d. Los temas asignados al documentos d es Z_d, siendo $Z_{d,n}$ el tema asignado a la palabra n en el documento d. Las palabras observadas en el documento d corresponden a w_d, donde $w_{d,n}$ es la n-ésima palabra del documento. α y β son hiperparámetros del modelo. El primero regula la distribución de tópicos para los documentos. Para valores altos de α (mayores a uno) los documentos tendrán una mayor mezcla de tópicos. Por otro lado, si toma valores pequeños (menores a uno), es más probable que los documentos sean generados por pocos tópicos. El segundo hiperparámetro regula la distribución de palabras por tópico. Al igual que con α, para valores altos los tópicos tendrán una mayor mezcla de término, y cuando sea bajo, habrá una menor mezcla de términos.

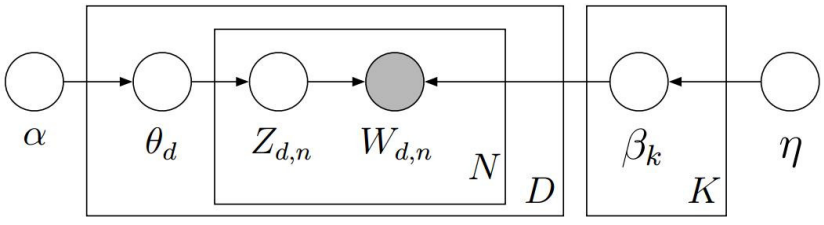


Figura 2.1: Representación gráfica del modelo LDA.

En modelado de tópicos probabilísticos se asume que los datos (en este contexto documentos) surgen de un proceso generativo que incluye variables latentes. Este proceso define una distribución de probabilidad conjunta sobre las variables observadas y las no observadas. Así, se computa la distribución condicional de las variables latentes dadas las variables observadas, también llamada distribución posterior.

$$p(\theta, z, d \mid \alpha, \beta) = \prod_{i=1}^{N} p(w_i \mid z_i, \beta) \cdot p(z_i \mid \theta_d) \cdot p(\theta_d \mid \alpha) \tag{2.2}$$

$$p(d \mid \alpha, \beta) = \int p(\theta_i \mid \alpha) \cdot \left[\prod_{i=1}^{N} \sum_{z_i \in Z} p(w_i \mid z_i, \beta) \cdot p(z_i \mid \theta_i) \right] d\theta_i \tag{2.3}$$

Para resolver la inferencia se debe calcular la distribución posterior entre las variables latentes (relacionadas con los tópicos) dadas las observaciones (documentos). El problema de este calculo es que las variables θ y β están acoplados al momento de sumar los tópicos latentes, por lo que hace que la integral no pueda ser calculada.

A pesar de que la posterior no es calculable para hacer una inferencia exacta, hay una variedad de algoritmos de inferencia aproximada que pueden ser utilizados para alcanzar una distribución similar. Uno de los algoritmos comúnmente usados debido a su simplicidad es *Gibbs Sampling*[5]. Gibbs Sampling es un método de Cadena de Markov Monte Carlo (MCMC) donde se construye una secuencia de variables aleatorias, cada una dependiente de las variables previas. Para poder hacer el cálculo aproximado se asume una distribución *a priori* Dirichlet β para el set de tópicos Φ, lo que puede ser integrado con una función multinomial. Por lo tanto, en cada paso el algoritmo actualizará cada tópico $z_{d,i} \in \mathbf{Z}$, realizando el siguiente muestreo [6]:

$$p\left(z_{d,i} = k \mid z^{\neg ij}, x^{\neg ij}, x_{ij} = w, \alpha, \beta\right) \propto \frac{N_{wk}^{ij} + \beta}{\sum_{\neg ij} N_k^{\neg ij} + W\beta} \left(N_{kd}^{ij} + \alpha\right) \tag{2.4}$$

donde $w \in W$ corresponde a una palabra en el vocabulario, $k \in [1, K]$ es uno de los tópicos, N_{wk}^{ij} es un conteo de veces que una palabra w es asignada al tópico k sin considerar $w_{d,i}$ y $z_{d,i}$, N_{kj}^{ij} es el número de veces que el tópico k ocurre en un documento d sin incluir w_{di}.

Cada vez que se asigna un tópico $z_{d,i}$ en el proceso de muestreo se actualizan tanto N_{wk}^{ij} como N_{kj}^{ij}. Transcurrida una serie de iteraciones del algoritmo, los parámetros Θ y Φ pueden aproximarse de la siguiente forma:

$$\theta_{d,k} = \frac{N_{\neg d,k}^{d} + \alpha}{N_{-d,k}^{(doc_i)} + K\alpha} \tag{2.5}$$

$$\phi_{w,k} = \frac{N_{\neg w,k}^{w} + \beta}{N_{w,k}^{(\cdot)} + W\beta} \tag{2.6}$$

donde $N_{-d,k}^{(\cdot)}$ es el número total de palabras asignadas al tópico j (no incluyendo al actual), $N_{\neg w,k}^{w}$ es el número de palabras del documento d_i asignado al tópico j (no incluyendo al actual), $N_{w,k}^{(\cdot)}$ y es el número total de palabras en el documento d_i. Esta mejora en el proceso de inferencia permite a LDA converger en una menor cantidad de iteraciones como se muestra en [5].

2.3. Parallel LDA

Siendo LDA una herramienta poderosa en el área de minería de texto, su alcance y capacidad para analizar corpus de gran tamaño es limitada. La dificultad de distribuir y paralelizar las actualizaciones de Gibbs sampling recae en el hecho de que es un proceso estrictamente secuencial.

A modo de solución al problema surgió PLDA, proveniente de trabajos entre investigadores de Google Beijing Research y la Universidad de Carnegie Mellon[7]. PLDA es una implementación, basada en MPI, del algoritmo Approximate Distributed LDA (AD-LDA)[8]. En este algoritmo los tópicos no son trabajados con una función de probabilidad sino como el conteo de cada palabra asignada a estos tópicos. Por ejemplo, si la palabra 'pizza' es generada por un tópico 4 veces y existen 40 palabras en total generadas por ese tópico, entonces AD-LDA registrará un valor de 4 mientras que LDA asignaría un valor de 0.1. Tal como dice

Algorithm 1 AD-LDA

repeat
  for each processor p in parallel **do**
    Copy global counts: $N_{wkp} \leftarrow N_{wk}$
    Sample z_p locally: LDA-Gibbs-Iteration$(x_p, z_p, N_{kjp}, N_{wkp}, \alpha, \beta)$
  end for
  Synchronize
  Update global counts: $N_{wk} \leftarrow N_{wk} + \sum_p (N_{wkp} - N_{wk})$
until termination criterion satisfied

su nombre, no se trabaja directamente con la posterior predictiva sino que con una aproximación a esta.

En AD-LDA, se implementa LDA en cada procesador (o nodo procesador), realizando simultáneamente Gibbs sampling de forma independiente en cada uno de los P procesadores. Con esta arquitectura cada procesador poseerá una versión local del modelo y existirá un modelo global que será actualizado con la información de todos los nodos en cada iteración.

El algoritmo distribuye D documentos de entrenamiento en P procesadores, con $Dp = \frac{D}{P}$ documentos en cada procesador. Además, se particionan las palabras de los D documentos tal que el procesador p solo almacene las palabras de los documentos que contiene su partición ($x = \{x_d\}_{d=1}^{D}$ se devide en $\{x_{|1}, \ldots, x_{|P}\}$) y la asignación correspondiente de topicos $z = \{z_d\}_{d=1}$ en $\{z_{|1}, \ldots, z_{|P}\}$, donde $x_{|p}$ y $z_{|p}$ existen solo en el procesador p. La matriz de conteo de documentos por tópicos N^{doc} es distribuida de igual forma, representando cada matriz en su procesador correspondiente como $N^{doc}_{|p}$. Cada procesador mantiene una copia de la matriz de conteo de palabras por tópico N^{word}, usando $N^{word}_{|p}$ para almacenar temporalmente los cambios acumulados de los documentos locales asignados a cada procesador.

En cada iteración de Gibbs sampling cada procesador p actualiza $z_{|p}$ haciendo un muestreo para cada $z_{ij|p}$ de la aproximación de la distribución a posteriori

$$p\left(z_{ij|p} = k \mid z^{\neg ij}, x^{\neg ij}, x_{ij|p} = w, \alpha, \beta\right) \propto \frac{N^{\neg ij}_{wk} + \beta}{N^{\neg ij}_{k} + W\beta}\left(N^{\neg ij}_{kj|p} + \alpha\right) \tag{2.7}$$

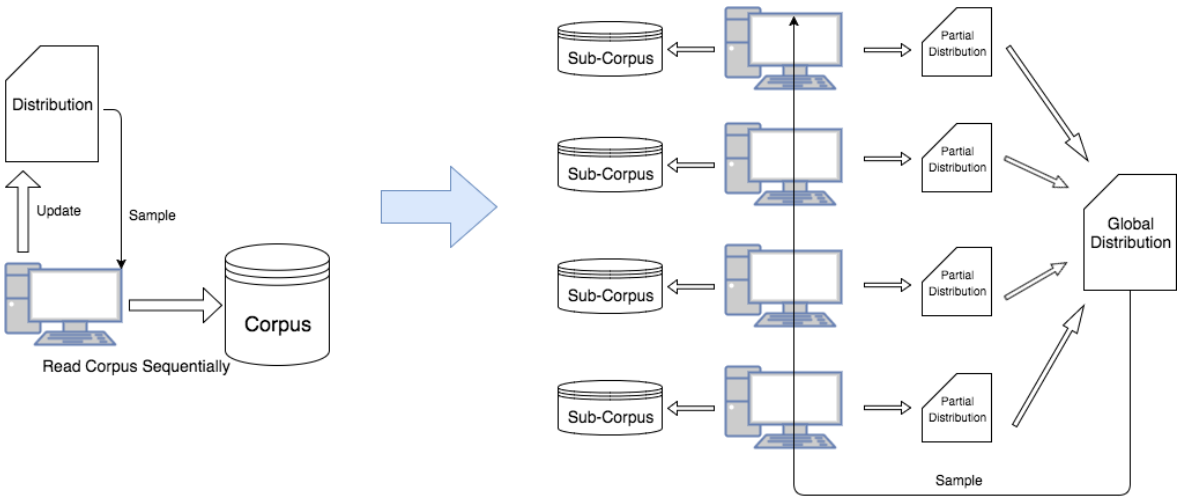


Figura 2.2: Diagrama de proceso de actualización de modelos. A la izquierda se encuentra LDA tradicional y a la derecha PLDA

lo que actualiza las matrices $N_{|p}^{doc}$ y $N_{|p}^{word}$. Luego de completar cada iteración, es decir un barrido completo por los datos, los procesadores vuelven a computar la matriz $C_{|p}^{word}$ que cada uno posee y comunican sus resultados por medio de la operación *AllReduce* [1] de MPI, que permite aplicar la operación *reduce* a todos las matrices para transmitir la nueva matriz C^{word} a todos los procesadores, tal como se muestra en la figura 2.2. Este proceso puede terminar luego de un número fijo de iteraciones o algún criterio de convergencia de MCMC.

El problema de este método es que en cada iteración los procesos no se comunican, teniendo que realizar operaciones de lectura y escritura en el disco para buscar y actualizar su información con el resto de los procesadores. Esto induce a una reducción de precisión global al tener datos obsoletos. Cada copia de los datos tiene que ser almacenada en memoria lo cual puede no ser conveniente cuando se trabaja con corpus de gran tamaño y computadores con pocos recursos. Además, como el proceso de actualización global debe ser sincronizado se debe esperar al procesador más lento para generar la carga de información. A pesar de esto, PLDA converge en la práctica y obtiene un gran ahorro de tiempo de proceso.

PLDA+ extiende el algoritmo anterior utilizando cuatro nuevas estrategias: colocación de datos, procesamiento por *pipelines*, agrupamiento de palabras y programación basada en

[1] https://www.mpich.org/static/docs/v3.1/www3/MPI_Allreduce.html

prioridades [9]. La colocación de datos permite a los *pipelines* utilizar los retrasos de tiempo generados por la comunicación con otros cálculos computacionales, trabajando con un set de palabras mientras se comunican los resultados de otro. La selección de estos set de palabras se realiza de forma que el tiempo de cálculo sea lo suficientemente largo como para realizar la comunicación, y se organizan en un cola circular en lugar de asignarse estáticamente a los procesos. Tanto los conjuntos de palabras existentes en la cola y el resto de ellos son manejados por uno de los procesadores mientras otro set computa Gibbs sampling, lo que aprovecha el paralelismo del modelo. Con esto logra un rendimiento y escalabilidad superior a PLDA.

2.4. Dynamic Topic Models

En el modelo LDA, las palabras de cada documento se asumen independientes y son generadas por una mezcla de multinomiales. Tratar cada término bajo el modelo de *bag-of-words* simplifica la identificación semántica de los temas subyacentes[2]. Este mismo fenómeno se considera para el orden de los documentos. Sin embargo, para muchas colecciones de interés asumir implícitamente la intercambiabilidad de los documentos no es apropiado. Algunos ejemplos, como artículos de noticias, papers de investigación y consultas en un buscador, reflejan una evolución en su contenido en el tiempo. Artículos de neurociencia de comienzos del siglo XX lucirán bastante diferentes a los del comienzo del siglo XXI. Los temas en el conjunto de documentos evoluciona en el tiempo y sería correcto poder modelar esta dinámica.

Dynamic Topic Models (DTM)[10] fue el primer acercamiento a resolver el problema de la dependencia temporal, capturando la evolución de los tópicos en un corpus organizado secuencialmente.

Considerar $\beta_{1:K}$ los K tópicos, cada uno con una distribución sobre un vocabulario. DTM supone que los datos están divididos en cortes temporales (*time slices*), por ejemplo en años o meses. Se modelan los documentos de cada corte con K-tópicos, donde los temas asociados al corte t evolucionaron del tiempo anterior $t-1$, según un proceso gaussiano [10]. Por lo

tanto, $\beta_{t,k}$ es el vector de distribución de palabras para el tópico k en el tiempo t.

$$\beta_{t,k} \mid \beta_{t-1,k} \sim \mathcal{N}(\beta_{t-1,k}, \sigma^2 I) \tag{2.8}$$

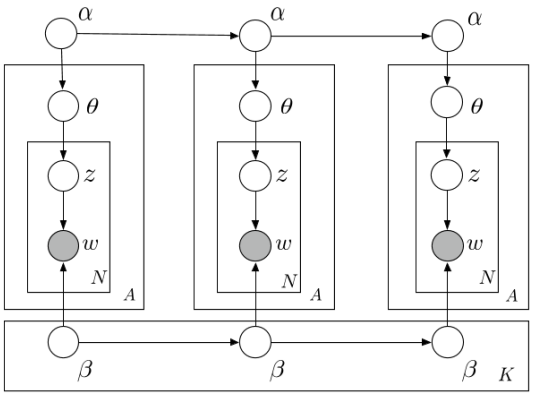
En LDA, la proporción de tópicos θ se genera de una Dirichlet con parámetro α. Para adaptarla al nuevo modelo, se utiliza una distribución log-normal con media α. Así, se define el siguiente modelo dinámico para capturar las estructuras entre modelos, donde α_t es la distribución de tópicos por documento en el tiempo t.

$$\alpha_t \mid \alpha_{t-1} \sim \mathcal{N}(\alpha_{t-1}, \delta^2 I) \tag{2.9}$$

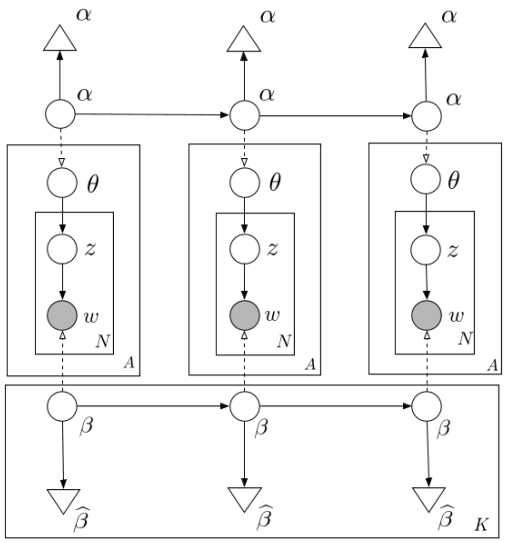
Notar que α_t y $\beta_{t,k}$ son generadas respecto a su distribución en un tiempo $t-1$, (α_{t-1} y $\beta_{t-1,k}$). Si η_t, d la distribución de tópicos para un documento d en t, $z_{t,d,n}$ el tópico para la palabra n en en d, el proceso generativo para un corte temporal t en un corpus secuencial es el siguiente:

1. $\beta_{t,k} \mid \beta_{t-1,k} \sim \mathcal{N}(\beta_{t-1}, \sigma^2 I)$
2. $\alpha_t \mid \alpha_{t-1} \sim \mathcal{N}(\alpha_{t-1}, \delta^2 I)$
3. Para cada documento:
    a) Escoger $\eta \sim \mathcal{N}_{t,d}(\alpha_t, a^2 I)$
    b) Por cada palabra w_n, con $n : 1 \ldots N$:
        1) $Z_{t,d,n} \sim Mult(\pi(\eta_{t,d}))$
        2) $W_{t,d,n} \sim Mult(\pi(\beta_{t,Z_{t,d,n}}))$

donde $\pi(x_i) = \frac{exp(x_i)}{\sum_i exp(x_i)}$. Al igual que LDA, las palabras son el único elemento observable. El modelo gráfico para este proceso generativo se puede observar en la figura 2.3a. Al eliminar las dependencias temporales se reduce a un modelo independiente por cada corte. Con la dinámica temporal, el k-ésimo tópico en el corte t ha evolucionado suavemente del k-ésimo tópico en el tiempo $t-1$.



(a) Representación gráfica de DTM, para tres cortes temporales. Cada hiperparámetro, $\beta_{t,k}$ y α_t, evoluciona en el tiempo.



(b) Representación gráfica de DTM con aproximación variacional, con $\hat{\beta}$ y $\hat{\alpha}$ parámetros calculados por Kalman filter.

Figura 2.3: Representación gráfica de DTM y DTM variacional.

Proceso de inferencia

Para el modelado de los parámetros se utiliza un proceso gaussiano para capturar la dinámica del tiempo. Como esta distribución no es conjugada con el modelo multinomial la inferencia a posterior no es posible de calcular[10]. Gibbs sampling ha funcionado para modelos estáticos pero la falta de una forma cerrada hace que sea más complicado utilizarla para modelos dinámicos. Por lo tanto, una estrategia es trabajar con una aproximación de la posterior usando métodos variacionales.

Optimizando los parámetros libres de la distribución sobre las variables latentes se obtiene un resultado cercano a la posterior verdadera en términos de divergencia Kullback-Leiber (D_{KL}). La divergencia KL es una medida no simétrica entre dos distribuciones P y Q que mide el grado de cercanía o similitud entre éstas. P es la distribución de las observación mientras que Q es el modelo que se buscar aproximar a P. A menor valor de divergencia KL mayor será la similitud o cercanía entre Q (aproximación) y P (original).

$$D_{KL}(P \parallel Q) = -\sum_{i} P(i) \cdot log\left(\frac{Q(i)}{P(i)}\right) \tag{2.10}$$

Se retiene la estructura secuencial del tópico proponiendo un modelo dinámico con observaciones variacionales gaussianas $\{\hat{\beta}_{k,1}, \ldots, \hat{\beta}_{k,T}\}$. Estos parámetros son adecuados para minimizar la divergencia KL entre el posterior resultante, que es gaussiana, y la verdadera posterior, que no es gaussiana [10]. En la figura 2.3b los parámetros relacionales se presentan como triángulos; ellos pueden ser pensados como un resultado hipotético del algoritmo Kalman filter. El modelo sufre el siguiente cambio

$$\hat{\beta}_t \mid \beta_t \sim \mathcal{N}(\beta_t, \hat{v}_t^2 I) \tag{2.11}$$

DTM es efectivo capturando la transformación de los tópicos dinámicos en el tiempo. Sin embargo, no posee forma alguna de identificar cuando los tópicos aparecen y desaparecen. Además, aunque los tópicos pueden evolucionar hasta cambiar completamente lo debe hacer de forma gradual, habiendo una pequeña distante entre un corte temporal y el siguiente. Finalmente, necesita saber previamente la cantidad de tópicos.

2.4.1. Topics Over Time

De forma paralela a DTM fue desarrollada otra solución llamada *Topics over Time* (TOT), método semejante a LDA que se enfoca en modelar la co-ocurrencia de las palabras conjuntas en el tiempo[11]. Logra capturar la estructura de datos de baja dimensionalidad y cómo estas estructuras cambian en el tiempo. El contenido del corpus es dinámico, y a diferencia de DTM, existe la posibilidad de que los tópicos aparezcan y desaparezcan, se dividan o mezclen, cambiando la correlación a lo largo del tiempo. En TOT, cada documento es generado como una mezcla de tópicos, influenciada por la co-ocurrencia de las palabras y las marcas temporales de los documentos.

Una de sus principales características es que TOT no supone un comportamiento Markoviano sobre el tiempo, sino que trabaja el tiempo como una variable continua observada. Otros

modelos toman el supuesto de que la distribución de palabras en los tópicos cambia en el tiempo. En este caso, TOT considera una distribución constante en cada tópico y es la co-ocurrencia de los tópicos lo que cambia en el tiempo. Gracias a esto se puede modelar el nacimiento y muerte de tópicos, como la separación y mezcla de éstos. En este sentido el vocabulario es estático.

Sea T el número de tópicos, ϕ_z la distribución multinomial de palabras para un tópico z, $\psi_{z_{di}}$ la distribución beta de un tiempo específico para un tópico z y t_{di} la marca temporal asociada con el token i en el documento d. El primer proceso generativo presentado utiliza Gibbs sampling para la estimación de los parámetros, como se muestra a continuación:

1. Escoger T multinomiales ϕ_z de una Dirichlet β, una para cada tópico z.
2. Por cada documento d, seleccionar una multinomial θ_d de una Dirichlet con parámetro α. Luego por cada palabra w_{di} en un documento d:
   a) Escoger un tópico z_{di} de una multinomial θ_d.
   b) Escoger una palabra w_{di} de una multinomial $\phi_{z_{di}}$.
   c) Escoger una marca de tiempo t_{di} de una Beta $\psi_{z_{di}}$.

El modelo gráfico es mostrado en la imagen 2.4. En este caso, todas las marcas temporales de las palabras en un documento son observadas igual que las marcas temporales del documento. La distribución a posteriori de los tópicos depende de la información tanto del texto como del tiempo.

En este modelo se utiliza Gibbs sampling para realizar la inferencia. Debido a que los cortes temporales son generados de una distribución Beta en vez de discretizar el tiempo, la dispersión de los datos no es un gran problema al ajustar la dimensión temporal al modelo. Por lo tanto, por simplicidad y velocidad de cómputo, se calcula la distribución beta ψ_z en cada iteración de Gibbs sampling.

El proceso generativo asocia una marca temporal a cada palabra, lo que se adecua perfectamente cuando diferentes partes de un documento hablan sobre diferentes períodos de tiempo. Sin embargo, es observado que los documentos suelen presentar solo una marca temporal.

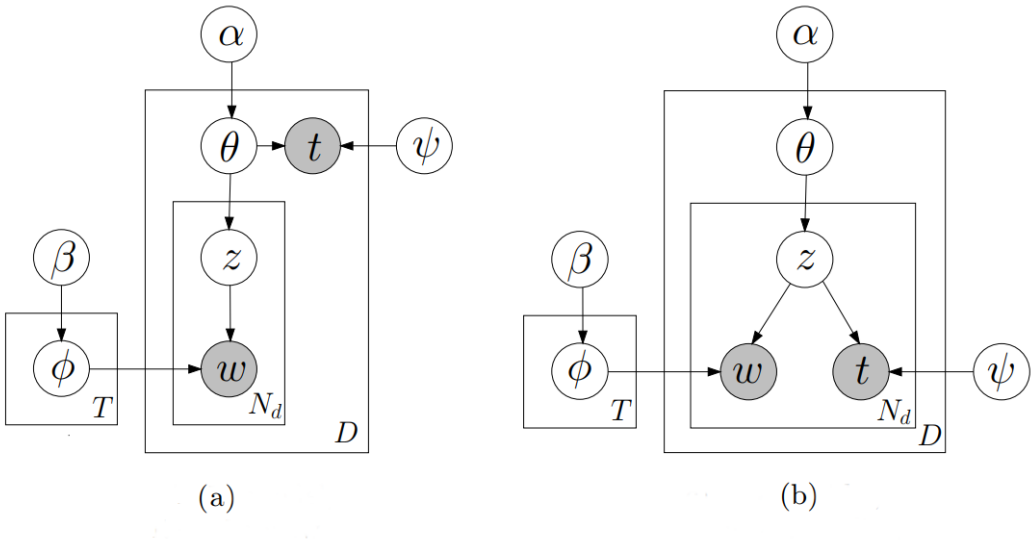


Figura 2.4: Representación gráfica del modelo Topics over Time (a) usando Gibbs sampling y (b) en su vista alternativa.

Esta es una deficiencia del modelo, pero aun así presenta buenos resultados en el modelado de grandes colecciones de texto.

Una solución alternativa cercana a lo que ocurre en la realidad (y que se acomoda mejor a la generación de nuevos documentos no pertenecientes al conjunto de entrenamiento), es la asociación de una única marca temporal a cada documento, generada por una mezcla de distribuciones Beta por tópico sobre el tiempo. Este modelo alternativo puede ser visto en la figura2.4(b). Este cambio puede generar cuál es la marca temporal de un documento dada sus palabras. Para facilitar esta predicción (y su comparación con LDA), las marcas temporales son discretizadas. Por lo tanto, dado un documento, se puede predecir su marca temporal escogiendo la marca temporal discretizada que maximice la probabilidad posterior la que es calculada por la multiplicación de las probabilidades de las marcas temporales de cada token de sus correspondientes distribuciones Beta a través del tiempo. Esto se resume por $argmax_t \prod_{i=1}^{N_d} p(t \mid \psi_{z_i})$. De la misma forma, se puede obtener la distribución sobre los tópicos, condicionada a las marcas temporales, obteniendo los patrones de ocurrencia de los tópicos en el tiempo.

2.4.2. Parallel DTM

Diferentes modelos de DTM han sido efectivos en la tarea de descubrir y capturar la evolución de datos en el tiempo. Para realizar la inferencia a posterior en DTM, los métodos existentes funcionan en base a algoritmos secuenciales (o *batch*) que hacen una revisión completa de los datos antes de cada actualización del modelo.

Como se mencionaba anteriormente, el modelo de DTM trabaja bajo procesos Markovianos encadenando las distribuciones de términos por tópicos a través del tiempo bajo parametrización Logística-Normal al igual que en el modelo de *Correlated Topic Model*[12]. Su principal debilidad es la no conjugación del modelo haciendo difícil la inferencia a gran escala.

Parallel DTM presenta una solución escalable para hacer la inferencia usando, dentro de Gibbs sampling, avances en *Stochastic MCMC* y *Stochastic Gradient Langevin*[13] para muestrear los parámetros logísticos normales por medio de la observación de *mini-batch* del corpus, procedimientos que convergen más rápido que los modelos secuenciales[14].

Dado que no es posible hacer la integración de los parámetros por la no-conjugación de las distribuciones gaussiana y multinomial, los autores deciden muestrear cada parámetro por separado. Así, se puede dividir el cálculo de cada uno de estos (α_t, $\eta_{d,t}$, $\phi_{k,t}$, y $Z_{d,n,t}$) en un *thread* diferente para su paralelización.

- $\alpha_\mathbf{t}$: alpha tiene una estructura Markoviana y está acondicionado a su *Markov Blanket* (MB). Usando la técnica de completar cuadrados, se llega a la siguiente forma para alpha:

$$\alpha_t \sim \mathcal{N}\left(\alpha_t \mid \hat{\mu}, \hat{\Lambda}^{-1}\right) \tag{2.12}$$

  donde la media y matriz de covarianza son evaluadas como:

$$\hat{\mu} = \overline{\alpha}_t + \overline{\eta}_{d,t} - \Lambda^{-1}\left(\frac{2}{\sigma^2}\overline{\eta}_{d,t} + \frac{D_t}{\psi^2}\overline{\alpha}_t\right) \tag{2.13}$$

$$\overline{\eta}_{d,t} = \frac{1}{D}\sum_{d=1}^{D_t}\eta_{d,t}, \quad \overline{\alpha}_t = \frac{\alpha_{t+1} + \alpha_{t-1}}{2}, \quad \overline{\Lambda} = \left(\frac{2}{\sigma^2} + \frac{D_t}{\psi^2}\right)I \tag{2.14}$$

  La evaluación de $\hat{\mu}$ es $O(K)$ siempre que se mantenga un registro de $\overline{\eta}_{d,t}$, dado que $\hat{\lambda}$ es matriz diagonal y que ésta es constante para todos los documentos del tiempo t.

- $\eta_{\mathbf{d,t}}$: la posterior condicionada a α_t y $Z_{d,t}$ es:

$$p(\eta_{d,t} \mid \alpha_t, Z_{d,t}) \propto \mathcal{N}(\eta_{d,t} \mid \alpha_t, \psi^2 I) \times \prod_{n=1}^{N_{d,t}} Mult(Z_{d,n,t} \mid \pi(\eta_{d,t})) \tag{2.15}$$

Este modelo es una regresión logística bayesiana con prior gaussiano y observaciones multinomiales $Z_{d,n,t}$. Para hacer inferencia sobre la posterior se utiliza *Stochastic Gradient Langevin Dynamics* (SGLD)[13]. SGLD es un algoritmo iterativo de aprendizaje que usa mini batches para realizar las actualizaciones. Está construido en base a *Gradient Descent* agregando ruido gaussiano en cada actualización, por lo tanto puede generar muestras de la distribución a posterior. Tomando el gradiente del logaritmo natural de la posterior condicional de $\eta_{d,t}$ se obtiene:

$$\nabla_{\eta^k_{d,t}} \log p\left(\eta_{d,t} \mid \alpha_t, Z_{d,t}\right)) = -\frac{1}{\psi^2}(\eta^k_{d,t} - \alpha^k_t) + C^k_{d,t} - (N_{d,t} \times \pi(\eta_{d,t})_k) \tag{2.16}$$

donde $N_{d,t}$ es el número de palabras en d para el tiempo t y $C^k_{d,t}$ es el número de veces que el tópico k ha sido observado en d en el tiempo t. Esta ecuación se analiza directamente con SGLD. Con este nuevo proceso, actualizar η toma tiempo $O(K)$.

- $\phi_{\mathbf{k,t}}$: se modela como una cadena de Markov similar a α_t y la posterior se condiciona a su MB. Se repite el arreglo realizado para α, obteniendo:

$$p(\phi_{k,t} \mid MB(\phi_{k,t})) = \mathcal{N}(\phi_{k,t} \mid \overline{\phi_{k,t}}, \frac{\beta^2}{2} I) \times \prod_{d=1}^{D_t} \prod_{n=1}^{N_{d,t}} Mult(W_{d,n,t} \mid \pi(\phi_{k,t})) \tag{2.17}$$

donde

$$\overline{\phi}_{k,t} = \frac{\phi_{k,t+1} + \phi k, t-1}{2} \tag{2.18}$$

El resultado es un modelo de regresión logística bayesiana con prior gausiano y observaciones multinomiales$W_{d,n,t}$. Se toma el gradiente del logaritmo natural y se consigue el operador para aplicar en SGLD. Esto reduce el tiempo de muestreo de ϕ_t a O(VK).

- $\mathbf{Z_d, n, t}$: como fue mencionado anteriormente, Z depende condicionalmente del resto de los parámetros del modelo. Es así como:

$$p(Z_{d,n,t} = k \mid \eta^k_{d,t}, \phi^w_{k,t} \propto \exp\left(\eta^k_{d,t}\right) \exp\left(\phi^w_{k,t}\right) \tag{2.19}$$

Mediante este cambio del *framework* de trabajo se logra reducir la complejidad computacional de cada parámetro de DTM a $O(K)$ para α, $O(D_m K)$ para η, $O(VK)$ para ϕ y $O(D_m N_d)$ para Z, con D_m el tamaño del mini-batch utilizado, N_d número de palabras en el documento d, K número de tópicos y V el tamaño del tópico asociado.

La implementación tiene dos niveles: multi-thread y distribuido. Dada la naturaleza de DTM se podría trabajar cada corte temporal en un nodo trabajador diferente, separando los datos correspondientes a cada tiempo. Se trabaja con MPI para comunicar α_t y ϕ_t al comienzo de cada iteración a los nodos vecinos. Después de eso no es necesaria comunicación entre núcleos o nodos de cómputo. Adicionalmente, en cada nodo hay un nivel de paralelismo multi-thread usando la librería de C++11 std::thread. Se muestrea η, Z y ϕ por separado, acondicionando η dado Z y ϕ de la iteración pasada y de la misma forma para Z y ϕ. Así, se pueden muestrear independientemente cada variable.

Trabajando de forma distribuida, se conserva el mismo principio de cálculo independiente de cada variable, pero se debe hacer un balanceo de carga entre los *threads* porque la cantidad de documentos se vuelve mucho mayor al tamaño del vocabulario.

2.4.3. Clustered LDA

DTM es efectivo capturando la transformación que sufren los tópicos a un nivel global a través del tiempo. Sin embargo, no hay un mecanismo que permita hacer un seguimiento de la aparición y desaparición de los tópicos. Además, la evolución del modelo asume que son reconocibles de un corte temporal al siguiente. Mientras un tópico puede evolucionar de forma gradual hasta ser muy diferente de su forma original, cada cambio o salto debe ser pequeño en relación a los cortes más cercanos[10]. La dependencia existente entre los segmentos de tiempo hace muy difícil separar el proceso de inferencia para poder distribuir el cálculo entre diferentes procesos o máquinas.

Otra debilidad que presenta DTM, y que se mantiene de LDA, es que se requiere conocer el número de tópicos existentes, cuando la cantidad óptima puede ir cambiando en el tiempo. Esta característica no es soportada por el modelo.

Buscando resolver el mismo problema de cluster de documentos, se propuso un enfoque diferente el cual no aplica directamente DTM. Clustered LDA (CLDA)[15] utiliza computación y descomposición paralela de datos, aplicando LDA a los segmentos de datos (en este caso independientes entre si) y luego combina los resultados usando algoritmos de clustering, que en este caso es k-means. CLDA aprovecha las implementaciones paralelas, tanto de LDA[9] como de k-means[16], para tener una mejora en desempeño de hasta dos ordenes de magnitud en tiempo en comparación a [10].

La información es dividida en múltiples segmentos temporales y en cada uno de estos se aplica LDA para estimar los *tópicos locales*. Una vez obtenidos los tópicos son fusionados y se realiza clustering para determinar los *tópicos globales*. La figura 2.5 [15] muestra el flujo de ejecución de CLDA usando notación de platos para representar LDA. A diferencia del modelo de DTM no existe una dependencia directa entre los cortes temporales consecutivos.

Algorithm 2 CLDA

1: **procedure** CLDA
2: SPLIT text corpus into S segments
3: $L \leftarrow$ number of local topics ▷ Lines 4-6 run independently in parallel
4: **for all** segments $s \in \{1, \ldots, S\}$ **do**
5: APPLY LDA to estimate local topics $\{t_n^i\}_{i=1}^{L}$
6: **end for** ▷ Line 7 run in parallel
7: $U \leftarrow$MERGE($\{t_n^i\}_{i=1}^{L}$ for $s \in \{1, \ldots, S\}$)
8: $K \leftarrow$number of global topics
9: CLUSTER U into K global topics
10: **end procedure**

Los pasos de CLDA, visibles en el algoritmo 2, son los siguientes:

1. *Segmentación de datos (SPLIT)*: el corpus se dividido en S conjuntos disjuntos. Estas divisiones normalmente se realizan en base a marcas de tiempo, aunque se pueden aplicar divisiones por ubicaciones geográficas o por la procedencia (fuente) de los datos.

   LDA requiere que el número de tópicos locales L sea seleccionado *a priori*, y puede ser

menor o mayor que el número de tópicos globales K. En general se obtienen mejores resultados cuando L es más grande que la cantidad esperada de tópicos K. Si muchos tópicos representan el mismo sujeto en cualquiera de los segmentos, son *clusterizados* juntos.

2. *Estimación tópicos locales:* cada segmento es analizado con LDA para obtener los tópicos locales. Esta operación puede ser altamente paralelizada si se utilizan implementaciones de LDA como PLDA+[9]. El resultado de esta ejecución es una colección de tópicos $\{t_s^i\}_{i=1}^{L}$ en cada segmento $s \in \{1, \ldots, S\}$, teniendo así un total de $S \cdot L$ tópicos locales. La unión de todos los tópicos es denotada por U.

3. *Combinación de tópicos locales (MERGE):* Se combinan los tópicos anteriormente obtenidos en U para obtener los tópicos globales del cluster. Dado que no todas las palabras aparecerán en todos los segmentos, se debe agregar las entradas faltantes a los tópicos para así poder tener resultados directamente comparables. Estas palabras son añadidas con una contribución igual a cero a los respectivos tópicos.

   Se asume que todos los tópicos locales poseen la misma escala e igual peso. Para asegurarse de lo último se realiza un proceso de normalización de los tópicos previamente a ser clusterizados.

4. *CLUSTER tópicos locales:* los tópicos locales son combinados para determinar los tópicos globales. El algoritmo k-means requiere conocer *a priori* el valor de los K tópicos globales. Si $K = 1$ se define un solo cluster que contenga todos los locales, y si $K = S \cdot L$ se define un cluster para cada tópico. Si $K < L$ no todos los tópicos globales tendrán una representación en cada segmento, por lo que podrán desaparecer y/o reaparecer. Si $K > L$ podrían desaparecer y reaparecer en diferentes segmentos, dependiendo de los resultados de k-means en el siguiente corte temporal. El input para k-means es la matriz tópicos $\times$ palabras.

   Dada la implementación de k-means, se puede utilizar como semilla inicial un conjunto de tópicos seleccionados.

Terminado el proceso de clustering se obtienen dos resultados del proceso. Los primeros son los centroides mismos, usables como tópicos, y por otro lado la asignación de los tópicos

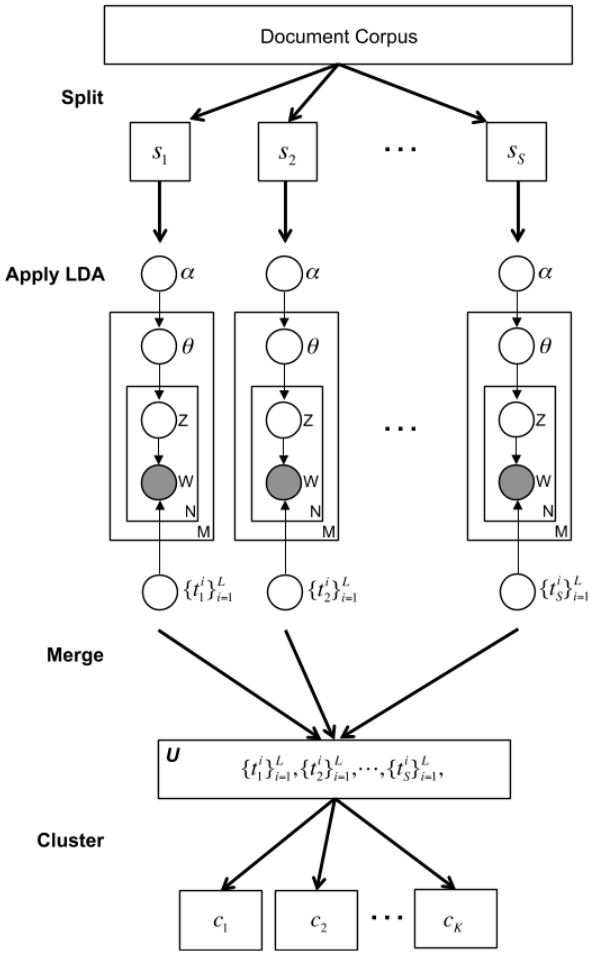


Figura 2.5: Diagrama de flujo de Clustered LDA.

originales a sus correspondientes clusters globales (membresía de cada tópico a un cluster). A diferencia de CLDA, DTM no provee como resultado una visión general de los tópicos a través del tiempo, sólo los tópicos locales en cada corte temporal.

Posteriormente, para los tópicos globales se calcula la proporción de palabras por tópico y las Top X palabras. Para los tópicos locales se determine la proporción de tópicos para cada documento y las Top X parabras para cada uno de estos.

En general los resultados realizados en [15] arrojan que CLDA es dos órdenes de magnitud más rápido que las existentes técnicas de modelado de tópicos dinámicos, con resultados de *perplexity* similares a los obtenidos por Blei[10].

Las debilidades del método recaen principalmente en que se asume que los clusters deben ser de tamaños similares y los datos deben estar separados considerablemente. Además, la selección de los K tópicos en K-means puede generar drásticos cambios en los resultados, y el algoritmo es conocido por ser muy sensible a las semillas iniciales.

2.5. Implementaciones

2.5.1. Gensim

Gensim [17] es una librería Open source escrita en Python para *Topic Modeling*, indexación de documentos y tareas de recuperación de información en grandes corpus. Está diseñada para manejar grandes colecciones de texto, usando streaming de datos y algoritmos incrementales eficientes.

Sus principales características son:

- **Independencia de memoria:** no es necesario tener un set de entrenamiento completo en memoria RAM para generar los modelos.
- **Implementaciones eficientes:** Tf-Idf, LSA distribuido e incremental, LDA distribuido, *Random projection*, entre otros.
- *Wrappers* e implementaciones eficientes para transformar diferentes tipos de datos con diversos formatos.
- *Queries* de similaridad para documentos para representaciones semánticas.

Actualmente posee una versión de LDA y DTM, pero esta última no ha sido optimizada lo que produce que el entrenamiento de los modelos sea muy costoso en tiempo.

2.6. Cuadros comparativos

Se realiza una comparación entre los distintos tipos de algoritmos de *Topic models* descritas anteriormente. En la tabla 2.1 se observan las diferencias y similitudes en alcance de las diferentes técnicas. Esta información será útil para determinar las técnicas a comparar en las siguientes secciones. Las características corresponden a:

- **Paralelización**: Posibilidad del algoritmo para ejecutarse en ambiente paralelo.
- **Componente temporal**: Capacidad de modelar diferentes intervalos de tiempo.
- **Evolución de tópicos**: Capacidad de modelar la evolución de tópicos a través del tiempo.
- **Vocabulario dinámico**: Usa un vocabulario dinámico que puede cambiar en el tiempo.
- **Número ilimitado de segmentos**: Cantidad de cortes temporales que se pueden modelar.
- **Nacimiento/muerte de tópicos**: Capacidad de modelar nuevos tópico y tópicos que no se reflejan en los documentos.

Cuadro 2.1: Cuadro comparativo de los enfoques existentes de modelado de tópicos.

	LDA[2]	PLDA+[9]	TOT[11]	DTM[10]	parallel DTM[14]	CLDA[15]
Paralelización	-	✓	-	-	✓	✓
Componente temporal	-	-	✓	✓	✓	✓
Evolución de tópicos	-	-	✓	✓	✓	✓
Vocabulario Dinámico	-	-	-	✓	✓	✓
Número ilimitado de segmentos	-	-	✓	✓	✓	✓
Nacimiento/muerte de tópicos	-	-	✓	-	-	✓

Capítulo 3

Experimentos

En este capítulo se detallará el procedimiento experimental para desarrollar las pruebas correspondientes a los objetivos del trabajo. Se realizará una descripción de los dataset a utilizar para correr los experimentos, destacando el tamaño en cantidad de documentos, su contenido y distribución de palabras. Además, se especificará cuáles fueron las pruebas seleccionadas para comparar los diferentes algoritmos de topic modeling y las métricas que medirán los resultados obtenidos. Finalmente, se explicará la configuración de entorno para correr los algoritmos de forma de que las pruebas puedan ser replicadas.

3.1. *Datasets*

Para poder evaluar los resultados y soluciones entregadas por los diferentes algoritmos de tópicos dinámicos, es necesario disponer de uno o más conjuntos de datos (o *datasets*), los que poseerán diferentes características permitiendo observar los distintos aspectos que tiene cada algoritmo al momento de ejecutarse sobre dichos datos. Estas características nos dirán, por ejemplo, qué tal se comporta un algoritmo cuando la cantidad de datos sea grande o pequeña, cómo responde a diferentes tipos de datos (numéricos, texto, atributos categóricos), o en qué afecta tener información repetida, entre otros.

En este trabajo, se ocuparán datasets de texto, también llamados *corpus*. Éstos corresponden

a conjuntos de documentos recolectados en el tiempo de sus respectivas fuentes (papers de conferencia y post en un foro de opinión). Dentro de sus características se encuentran el idioma de los documentos (pueden ser de uno o más idiomas), tamaño del corpus (cantidad de documentos), tamaño de los documentos, información temporal (los datos pueden estar divididos por años, meses, días, etc.), cantidad de términos y *tokens*, frecuencia de las palabras, entre otros. Cada una de estas dimensiones permiten al experimentador analizar diferentes aristas de un mismo problema.

A continuación se presentan los corpus elegidos para las tareas de descubrimiento de tópicos.

3.1.1. NIPS Proceedings

La conferencia *Neural Information Processing Systems* (NIPS) se caracteriza por contribuciones de numerosas comunidades de investigación en el área de algoritmos de aprendizaje. En la amplia gama de temas que cubre la conferencia se pueden encontrar con trabajos que van desde *Deep Learning* y *Computer Vision* hasta *Cognitive Science* y *Reinforcement Learning*.

El dataset contiene el texto completo de un conjunto de papers aceptados en la conferencia, y es frecuentemente usada como un punto de referencia para una variedad de técnicas y análisis en Machine Learning.

Se utilizará la versión disponible en [14], que comprende artículos del año 1990 hasta el 2002[1]. Para estos datos los autores realizaron un preprocesamiento de los datos eliminando stopwords y seleccionando las 8000 palabras más frecuentes del vocabulario.

En promedio los documentos tienen un total de 1188 palabras, con un mínimo de 296 y un máximo de 3745.

[1] http://www.cs.nyu.edu/ roweis/data. html

3.1.2. Reclamos.cl

Definido por sus creadores, Reclamos.cl[2] es el foro chileno de encuentro transparente entre personas, empresas y organizaciones. En esta plataforma las personas puede realizar reclamos respecto a empresas y servicios, tanto públicos como privados. El objetivo es poder dar al usuario un espacio en que pueda compartir sus molestias acerca de una entidad en específico, y que ésta tenga la oportunidad de comunicarse con su cliente para resolver los problemas que presenta.

La página comenzó su funcionamiento el año 2006, logrando una cantidad aproximada de más de 200.000 reclamos hasta la fecha actual. Los principales temas tocados en los reclamos son: telecomunicaciones, retail, bancos, salud, educación, gobierno, servicios, automotriz, construcción y transportes

El dataset obtenido consta de 201.853 documentos, en español, distribuidos a lo largo de 12 cortes temporales, con un total de 159.805 términos y 14.663.801 tokens. Esta gran cantidad de términos se explica por la forma de escribir de los usuarios pues una gran cantidad de palabras están mal escritas, y como los verbos en español tienen diferentes conjugaciones aumentan a su vez la cantidad de palabras con errores.

Como este dataset contiene mucho ruido (principalmente por palabras mal escritas) se realizó un preprocesamiento de los datos. Los pasos fueron los siguientes:

- Eliminar stopwords, definidas en la librería NLTK más un diccionario generado a mano por exploración de los documentos, palabras de largo 3 o menor, y que aparezcan en menos de 5 documentos.
- Stemming para acoplar diferentes conjugaciones o formas de palabras y verbos.
- Eliminar palabras con frecuencia menor a 10. Dada la gran cantidad de documentos existían muchas palabras con baja frecuencia de aparición en relación a la cantidad de tokens y documentos.
- Filtrar documentos de largo menor a 5.

[2]www.reclamos.cl

Finalmente, el resultado es un dataset con 123.197 documentos, 18.394 términos y 13.396.325 tokens. En promedio los documentos tiene un total de 90 palabras, con un mínimo de 6 y un máximo de 3254.

En la tabla 3.1 se observa en detalle la composición de los corpus. Cabe destacar que la principal diferencia que existe entre los dos datasets es la diferencia de idiomas de los documentos, siendo NIPS una colección de textos en inglés y Reclamos.cl documentos en español. Otra diferencia importante es la cantidad de documentos separándose por dos órdenes de magnitud. En relación al tamaño, este es el que ocupan los archivos posterior al preprocesamiento.

Cuadro 3.1: Información de los dataset a utilizar en la experimentación.

Dataset	**Time slices**	**Documentos**	**# Vocabulario**	**# Tokens**	**Tamaño**
NIPS	13	1.383	7.958	1.641.386	8.0 MB
Reclamos.cl	12	123.197	18.394	13.396.325	68.3 MB

3.2. Visualizaciones de los Datasets

En el gráfico 3.1 y 3.2 se visualiza la distribución de frecuencias de palabras en el corpus. Estos gráficos fueron obtenidos recolectando las palabras contenidas en todos los documentos, contando su frecuencia y ordenándolas de forma descendente. En ambos corpus existen pocas palabras con una gran frecuencia y una cola pesada de palabras con pocas repeticiones. Este fenómeno se condice con la Ley de Zipf [18], ley empírica que determina la frecuencia de aparición de un conjunto de palabras siguiendo una distribución aproximada por:

$$P_n \sim \frac{1}{n^a} \tag{3.1}$$

donde P_n es la frecuencia de la n-ésima palabra y a es un exponente que generalmente es levemente mayor a 1. Así, la segunda palabra se repetirá aproximada $1/2$ del primero, el tercero $1/3$ y así sucesivamente. Como en el corpus de Reclamos.cl aumenta tanto la cantidad

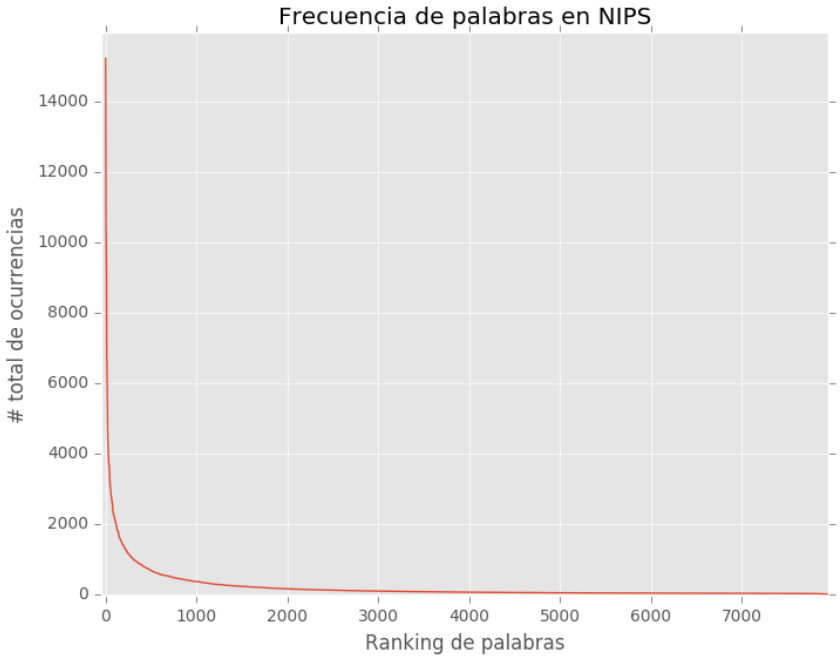


Figura 3.1: Gráfico de Ley de Zipf para el dataset NIPS luego del preprocesamiento del corpus.

de vocabulario como la cantidad de tokens, dada la Ley de Zipf, los primeros términos tienen una mayor repetición que el caso de NIPS.

La figura 3.3 de NIPS se muestran el Top50 de palabras con mayor frecuencia. Dentro de las primeras palabras con mayor frecuencia están: **network, learning, model, neural, input, data, fuction, networks, figure** y **time**. Estas palabras son regularmente utilizadas en los trabajos de esta conferencia, donde los temas *Machine learning*, *Deep learning*, *Cognitive Science* y *Algorithms* usan estas palabras.

Para Relamos.cl, como se aplicó stemming, hay palabras que aparecerán más veces si es que tiene más flexiones verbales o nominales de esta. Por ejemplo, palabras como *llama*, *llamada*, *llamando* y *llamado*, y otras conjugaciones del verbo *llamar*, serán reducidas a la raiz *llam*, y su frecuencia será la suma de las frecuencias de las 4 palabras. Así, algunas de las palabras con mayor frecuencia en el corpus, mostradas en la figura 3.4, son: **tod, llam, pag, hac, pas, compr, deb, sol, lleg, esper, empres** y **servic**. En el dominio de los reclamos es común ver palabras relacionadas con llamadas, página o pago, comprar, empresa

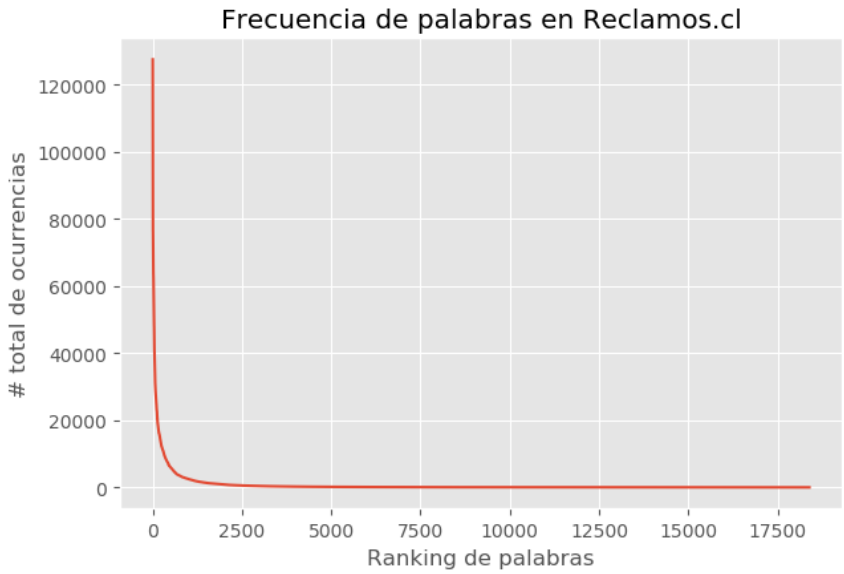


Figura 3.2: Gráfico de Ley de Zipf para el dataset Reclamos.cl luego del preprocesamiento del corpus.

y servicio, términos que se utilizan cuando personas tienen una queja respecto a una empresa que entrega servicios o venden productos, entre otros casos.

3.3. Procedimiento Experimental

El orden para proceder con la experimentación será, primero seleccionar el conjunto de algoritmos competentes a la problemática según sus características descritas en el estado del arte. Luego serán seleccionados datasets sobre los cuales serán ejecutados los algoritmos en cada experimento. Finalmente, se definirán las métricas que se utilizarán para evaluar los resultados obtenidos.

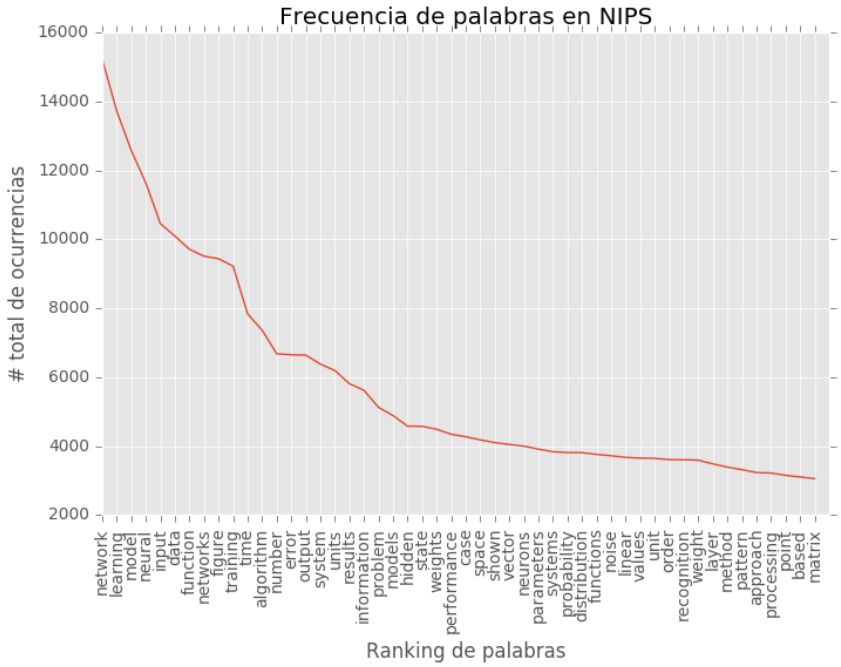


Figura 3.3: Gráfico de Ley de Zipf para las Top 50 palabras para dataset NIPS luego del preprocesamiento del corpus.

3.3.1. Algoritmos seleccionados

Los algoritmos de modelado de tópicos dinámicos seleccionados para llevar a cabo las pruebas fueron seleccionados de acuerdo a sus características respecto a escalabilidad, considerando la posibilidad de ejecutar pruebas paralelas/distribuidas, y la capacidad de modelar el comportamiento de los datos en el tiempo. Dentro de los algoritmos que cumplían con estos requisitos fueron seleccionados Parallel DTM y Clustered LDA por cumplir ambas características. Adicional a los dos algoritmos se escogió trabajar con DTM pues es el algoritmo base para comparar resultados, y el cual debería tener los mejores resultados en términos de eficacia. Topics Over Time, a pesar de cumplir con el hecho de modelar el comportamiento temporal, no fue seleccionado pues el mecanismo con que trabaja el vocabulario no es dinámico, no está diseñado para correr de forma paralela y tampoco modela la dependencia de los datos en el tiempo.

Para la experimentación se utilizará las implementaciones disponibles en Github por cada

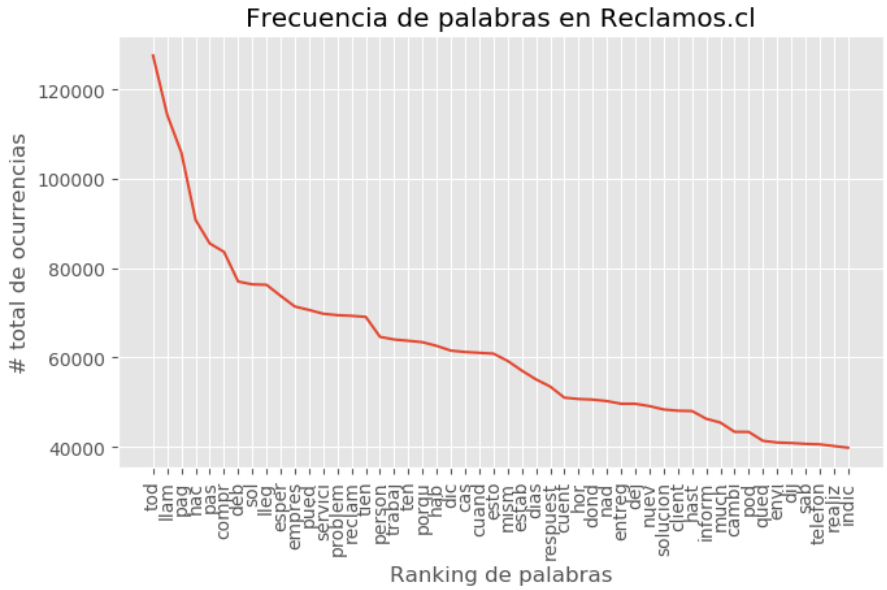


Figura 3.4: Gráfico de Ley de Zipf para las Top 50 palabras para dataset Reclamos.cl luego del preprocesamiento del corpus.

uno de los autores. La implementación de DTM [3] está programada en C y pDTM [4] en C++. Por otro lado, CLDA [5] tiene sus principales componentes, PLDA y K-means, escritos en C y un conjunto de scripts en Python que manejan el uso de datos para analizar los inputs y outputs de cada algoritmo.

3.3.2. Pruebas

Se han confeccionado tres pruebas diferentes para los algoritmos: eficiencia, eficacia y rendimiento en entorno paralelo.

En las pruebas de eficiencia se comparará el tiempo de ejecución de cada algoritmo en procesar los dataset presentados anteriormente. Para obtener diferentes datasets se recurrirá a la técnica de *oversampling* de manera de obtener un conjunto de datasets que requieran distintos niveles de poder de procesamiento. En este proceso se añade redundancia a los datos, por

[3] https://github.com/blei-lab/dtm
[4] https://github.com/Arnie0426/FastDTM
[5] https://github.com/groppcw/CLDA

lo tanto es crucial hacer la distinción

Además, se utilizará la técnica de *undersampling* para obtener un conjunto de datasets de menor tamaño. Así se analizarán los resultados de los algoritmos al aumentar la cantidad de información en cada dataset, sin que estos datos generen ruido ni redundancia con sucede en *oversampling*. Para ambas técnicas se ocuparán diez factores y se registrará el tiempo de ejecución guardando los resultados para medir su calidad con las métricas expuestas a continuación.

Las pruebas de rendimiento en un entorno paralelo tienen el fin de observar el desempeño de los algoritmos al tener disponibles más núcleos de trabajo. Se trabajará con datasets con *oversampling* y *undersampling* y se estudiará como cambian los resultados y el tiempo de procesamiento al aumentar la cantidad de *threads* asociados a cada proceso.

3.3.3. Medida de evaluación

El principal objetivo de los algoritmos de modelado de tópicos dinámicos es poder analizar el comportamiento de la información a través del tiempo, proyectando cómo evoluciona un corpus y cuáles tópicos y palabras son más relevantes para poder describir el conjunto de datos. Dependiendo del análisis que se quiera llevar a cabo tomaremos dos ejes principales para el proceso de experimentación: la eficacia de los algoritmos a ejecutar, y la calidad de las soluciones que éstos produzcan.

Por un lado, para evaluar la eficiencia de los algoritmos se utilizará el tiempo total de ejecución de cada algoritmo como métrica, considerando el tiempo de lectura, tiempo de procesamiento de los datos y tiempo de entrega de resultados.

Por otro lado, para evaluar la calidad de los modelos generados por cada algoritmo se utilizará la medida de *perplexity* (perplejidad).

Perplexity

En orden de tener buenos resultados, tanto para pDTM como CLDA, los tópicos generados por estos algoritmos deben ser mejores que los producidos por DTM, o incluso superiores a él. Actualmente, como medir la calidad de un modelo de tópicos es una pregunta abierta[19]. Además, las métricas existentes no siempre se correlacionan con el juicio humano de la calidad de los tópicos [20] ni capturan conceptos que las personas consideran relevantes e interpretables[21].

Una medida bastante usada es *perplexity*, que captura qué tan bien el modelo generado coincide con los datos proporcionados. Un valor bajo de perplexity indica que el modelo se ajusta mejor a los documentos de entrenamiento.

Para determinar la *perplexity* se debe calcular la probabilidad de que el corpus de entrada pueda ser generado por el modelo, equivalente al producto de la probabilidad de generar cada documento en el set de datos. En este caso, la probabilidad de generar cada documento es el resultado del producto de probabilidades de generar cada palabra contenida en él. Estos productos al ser tan pequeños se expresan como sumas de logaritmos, evitando así problemas con aritméticas de punto flotante. Por lo tanto, el valor de *perplexity* viene dado por la siguiente ecuación[22]:

$$perplexity = exp\left(-\frac{\sum_{d\in D}\sum_{w\in d} logP_z(w)}{\sum_{d\in D} N_d}\right) \tag{3.2}$$

donde

$$P_z(w) = \sum_z P(w \mid z)P(z \mid d) \tag{3.3}$$

y es equivalente a

$$P_z(w) = \Theta_{d_i} \cdot \Phi^t_{.,w} \tag{3.4}$$

d corresponde a un documento en el corpus D, w es una palabra y N_d corresponde al número de tokens. Es igual a calcular la exponente de la entropía. Calcular $P(w)$ depende del modelo usado para inferir los tópicos, pero en general la probabilidad de que una palabra aparezca en un documento es basada en la mezcla de tópicos de ese documento. Cada token tiene una probabilidad de ser elegida de cualquiera de los tópicos que conforman el documento,

cada uno de los cuales tiene una probabilidad de generar una palabra en particular[15]. Así, la probabilidad de que un token aparezca en un documento viene dado por la ecuación 3.3, donde $P(w \mid z)$ es una entrada en la matriz de tópicos Θ y $P(t)$ una entrada de la matriz de mezcla de tópicos Φ. Este resultado es equivalente a la ecuación 3.4.

3.4. Configuración de entorno de trabajo

Para ejecutar las pruebas y experimentos se trabajó en un entorno sobre el sistema operativo Ubuntu 16.04, distribución del sistema operativo GNU/Linux, basada en Debian. Se seleccionó este entorno por su facilidad de trabajo e instalación de las diferentes dependencias necesarias para correr las implementaciones de los algoritmos. También, es más fácil la instalación y modificación de archivos de configuración del sistema, sumado a la compilación de los códigos fuente. El servidor posee un procesador Intel Core i7-6700 CPU, con 4 cores y 64GB de ram.

3.4.1. Instalación de Eigen y Cmake

Para poder ejecutar la implementación de pDTM se deben poseer, dentro del entorno de trabajo, dos principales componentes: Cmake y Eigen.

CMake[6] es un sistema extensible de código abierto que gestiona el proceso de compilación en un sistema operativo y de forma independiente del compilador. Está diseñado para ser utilizado en conjunto con el entorno de compilación nativo. Los archivos de configuración simples ubicados en cada directorio de origen (llamados archivos CMakeLists.txt) se utilizan para generar archivos de compilación estándar (por ejemplo, archivos make en Unix y espacios de trabajo en Windows MSVC) que se utilizan de la manera habitual. Puede generar un entorno de compilación nativo que compilará código fuente, creará bibliotecas, generará contenedores y compilará ejecutables.

[6] cmake.org

Para la instalación de CMake, primero se debe descargar la última versión[7]. Luego ir a la carpeta de descarga y descomprimir el archivo.

```
$ tar -xzf cmake-3.10.2.tar.gz
$ cd cmake-3.10.2
```

Una vez dentro de la carpeta, si es que no existe una instalación previa se deben ejecutar los siguientes comandos en una consola.

```
$ ./bootstrap
$ make
$ make install
```

En el caso de que haya una instalación previa la instalación puede ser utilizada para actualizar a una versión actualizada.

```
$ cmake .
$ make
$ make install
```

Finalmente, si la instalación ha tenido éxito, al comprobar su versión debería aparecer un mensaje similar al siguiente.

```
$ cmake --version
cmake version 3.9.3
CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

Eigen[8] es una librería de código abierto de *templates* para C++ con algoritmos de álgebra lineal, operaciones de matrices y vectores, transformaciones geométricas, entre otros. Sus principales características son proveer soporte a matrices de diferentes tamaños (pequeñas, grandes, *sparse*), permite evaluación perezosa (lazy evaluation) para mejorar los tiempo de ejecución y utiliza algoritmos altamente optimizados.

Para instalar Eigen, primero se deben descargar los archivos de su página (utilizaremos la

[7] https://cmake.org/download/
[8] http://eigen.tuxfamily.org

versión 3.2.8[9]. En el caso de no tener instalado CMake, las librerías pueden ser usadas directamente del directorio de Eigen, agregando el parámetro al momento de compilar el código fuente.

```
$ g++ -I /directorio/de/eigen/ ejemplo_programa -o ejemplo_programa
```

Usando CMake simplemente debemos acceder al directorio y ejecutar los siguientes comandos.

```
$ cd build_dir
$ cmake source_dir
$ sudo make install
```

3.4.2. Instalación de MPICH

MPI o *Message Passing Interface* es un estándar que define la sintaxis y semántica para el traspaso de mensajes, diseñada para ser usada en programas que requieran la existencia de múltiples procesadores, tanto de forma paralela como distribuida. Por si misma no es una librería, sino que son especificaciones de cómo una librería debería ser construida. Debido a esto, es que existe una amplia variedad de implementaciones y librerías basadas en MPI para los diferentes lenguajes de programación existentes. Una de las implementaciones más populares es MPICH2, una versión portable y para procesos de alto rendimiento, enfocada en proporcionar soporte de manera eficiente para diferentes plataformas de computación y comunicación, como *commodity clusters*, redes de alta velocidad y supercomputadores.

Para su instalación, primero se debe descargar la última versión de su portal[10]. Luego ir a la carpeta de descarga y descomprimir el archivo.

```
$ tar -xzf mpich-3.2.tar.gz
$ cd mpich-3.2
```

Habiendo descomprimido los archivos se puede configurar la instalación para evitar algunos paquetes. En este caso, evitaremos la instalación de los componentes para Fortran.

[9] http://bitbucket.org/eigen/eigen/get/3.2.8.tar.bz2
[10] www.mpich.org

```
$ ./configure --disable-fortran
```

Cuando la configuración se haya completado, es tiempo de compilar e instalar MPICH2.

```
$ make; sudo make install
```

Si la instalación fue exitosa, al comprobar la versión debería verse información similar a esta.

```
$ mpiexec --version
HYDRA build details:
    Version:                                3.2
    Release Date:                           Wed Nov 11 22:06:48 CST 2015
    CC:                                     gcc
    CXX:                                    g++
    F77:
    F90:
```

Con eso se completa la instalación de MPICH.

3.4.3. Instalación de Python

Para trabajar con Python[11] se debe compilar e instalar los archivos fuente de la página oficial de Python. En el caso de la mayoría de las distribuciones de Linux vienen con diversas versiones preinstaladas, teniendo que añadir solo las librerías adicionales para el trabajo que se quiera desarrollar. Además, existen distribuciones de Python que facilitan el proceso de instalación, incluyendo un conjunto de librerías útiles comúnmente utilizadas. Dentro de las distribuciones populares y de código abierto está Anaconda[12], la cual incluye librerías como NumPy, SciPy, Pandas, Matplotlib y Scikit-Learn. Su instalación es bastante sencilla, solo se debe descargar el script instalador y ejecutarlo en una consola con el siguiente comando (para la versión 3.5):

```
$ bash Anaconda3-5.0.1-Linux-x86_64.sh
```

[11] https://www.python.org/
[12] https://anaconda.org/

O para la versión 2.7:

```
$ bash Anaconda2-5.0.1-Linux-x86_64.sh
```

Con la instalación básica es suficiente, puesto que no es necesaria ninguna librería adicional para ejecutar los algoritmos.

Capítulo 4

Resultados

En este capítulo se presentarán los resultados entregados producto de aplicar el procedimiento experimental detallado el en capítulo anterior. Los resultados serán de dos tipos: numérico y temporal. Se evaluará el tiempo de ejecución de los algoritmos sobre diferentes instancias de un corpus y se medirá la calidad de las soluciones generadas en base a la medida *perplexity*. Para el algoritmo CLDA, dado que los resultados no son deterministas (varía el resultado cada vez que se ejecuta la implementación), se tomó un promedio entre el conjunto de repeticiones. Este promedio se calcula tanto para la perplexity como para el tiempo de ejecución, siendo este último el que tiene una menor variación entre repeticiones en comparación al primero.

Para el corpus NIPS existen 13 intervalos de tiempos y para Reclamos.cl existen 12 intervalos. Para ambos datasets se ejecutarán pruebas con 10 tópicos.

4.1. Pruebas de Eficiencia

Para poder hacer una comparación entre los algoritmos, midiendo la eficiencia de cada uno, se utilizará la técnica de *oversampling* sobre el corpus NIPS. De esta forma, el corpus compuesto por 1.383 documentos verá aumentada la cantidad de registros proporcionalmente al factor de *oversampling* aplicado, respecto a la cantidad de datos inciales. Los factores de

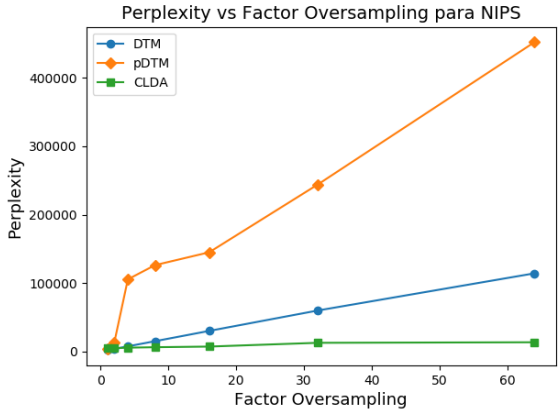
oversampling utilizados son 7 potencias de 2: 1, 2, 4, 8, 16, 32 y 64. Así, son generados 7 corpus con cantidades de documentos y tamaño en disco diferentes. Esta carga podrá variar los resultados de perplexity, como el tiempo de cómputo y los recursos necesarios para realizarlo. Así mismo, permitirá analizar qué ocurre cuando los datasets tienen datos repetidos.

Como el dataset de Reclamos.cl ya posee una gran cantidad de documentos (123.197), se escoge no hacer oversampling sobre él, sino que aplicar undersampling y analizar el comportamiento de los algoritmos frente a este escenario. Los factores a utilizar son: 10 %, 20 %, 30 %, 40 %, 50 %, 60 %, 70 %, 80 %, 90 % y 100 %. Serán generados 10 corpus cuya cantidad de documentos será la proporción del factor utilizado respecto al total de los datos del dataset, es decir, al crear el corpus con factor igual a 40 % se mantendrá un 40 % de los datos originales. Esta selección es aleatoria y se empleará sobre cada uno de los intervalo de tiempo. El fenómeno a estudiar es cuál es el desempeño al agregar nuevos documentos a un dataset. Esto difiere al estudio con oversampling debido principalmente a que los dataset no contendrán datos repetidos, sino que se presentará nueva información al procesar un porcentaje mayor de los datos. Este fenómeno también se estudiará con NIPS.

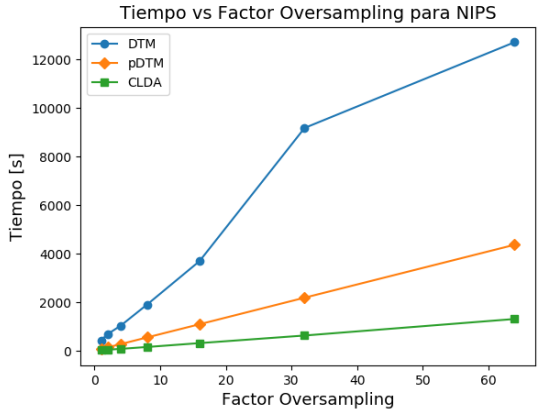
4.1.1. NIPS con diferentes factores de Oversampling

Luego de correr las pruebas, se obtuvo la perplexity y tiempo de cada instancia con los tres algoritmos. Estos datos fueron registrados en la tabla A.1 del apéndice. Una vez obtenidos estos datos, según factor de oversampling y algoritmo de modelado de tópicos, se generan los gráficos de la figura 4.1 para comparar descriptivamente cómo varía la perplexity, en la sub-figura 4.1a, y el tiempo de ejecución, en la sub-figura 4.1b. En ambos gráficos las líneas que unen los puntos no corresponden a una función de los datos sino que un artefacto para visualizar la continuidad entre los resultados de cada algoritmo.

En la figura 4.1a, es difícil diferenciar, para los primeros valores de *oversampling* (de 1 a 4), cuánto varían los algoritmos entre si. Con los datos de la tabla A.1 queda claro en un comienzo que DTM tiene un resultado menor al de pDTM y CLDA. A medida que crece el factor de *oversampling* DTM va aumentando casi linealmente (el doble) en función de los datos en el corpus. pDTM tiene un explosivo aumento al cuadruplicar el tamaño del corpus,



(a) Perplexity vs F. Oversampling NIPS



(b) Tiempo vs F. Oversampling NIPS

Figura 4.1: (a) Valores de perplexity en función del factor de oversampling para dataset NIPS utilizando los tres algoritmos en estudio. (b) Tiempo de ejecución versus factor de oversampling para las pruebas de eficiencia sobre NIPS.

siendo el que mayor perplexity arroja con el resto de los experimentos. CLDA por otro lado, parte con la perplexity más alta, pero a medida que van aumentando los datos la perplexity de los modelos generados son menores en comparación al resto de los algoritmos, incluso quedando bajo de DTM. Por la magnitud de los resultados, se puede sospechar que CLDA tiene menos problemas modelando corpus con documentos repetidos. Este es un efecto natural de los algoritmos de clustering, donde se favorecen de ambientes donde hay datos similares. Esta misma idea se lleva a la composición de tópicos que realiza CLDA usando k-means. Así, como se describió anteriormente en el estado del arte, se agrupan los tópicos por cercanía para extraer la evolución de los tópicos en el tiempo.

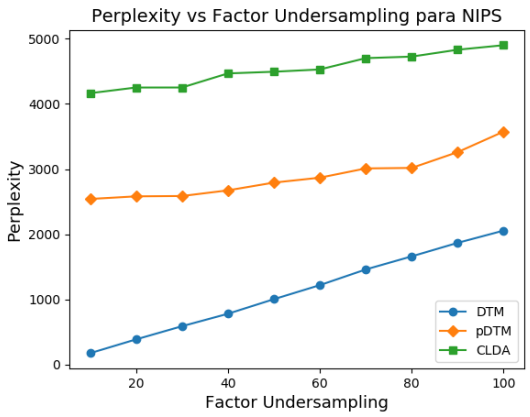
Sobre los tiempos de ejecución, CLDA queda por debajo de los dos algoritmos para todos los factor de *oversampling* demorándose 10 veces menos que DTM. El aumento del tiempo puede ser modelado como una función lineal donde de un valor de factor a otro aumenta aproximadamente al doble. El orden de mayor a menor tiempo de ejecución para las últimas pruebas es DTM (12,708.28 s), pDTM (4,366.09 s) y CLDA (1,310.63). Tanto pDTM como CLDA presentan un mejor desempeño en eficiencia con respecto a los tiempo obtenidos, para tamaños de dataset grandes. Sus variantes en términos de estimación de parámetros y construcción de tópicos tienen un impacto positivo cuando se necesita explorar una gran base

de datos, deficiencia que tiene la implementación de DTM pues almacena grandes matrices de tópicos y mezcla de tópicos en memoria.

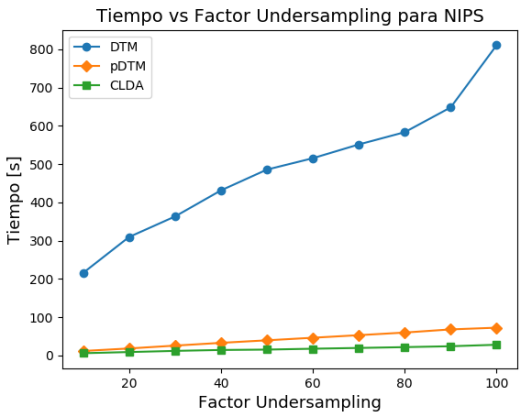
4.1.2. Datasets con diferentes factores de Undersampling

Repitiendo los experimentos, pero usando la técnica de *undersampling*, se obtuvo la perplexity de los modelos y los tiempos de ejecución para las pruebas analizando los dos corpus en estudio. Estas información fue registrada en las tablas A.2 (perplexity) y A.3 (tiempo ejecución) del apéndice. Cada una de las entrada del algoritmo CLDA corresponde a la media aritmética de los experimentos registrados en A.5 para NIPS y A.4 Reclamos.cl.

A diferencia de la sección anterior, al aumentar el factor de *undersampling* los corpus aumentarán en tamaño pero con nuevos documentos. Así, las curvas a estudiar en los gráficos podrían cambiar. Con los datos de A.2 y A.3 se generan las figuras 4.2a y 4.2b.



(a) Perplexity vs F. Undersampling



(b) Tiempo vs F. Undersampling

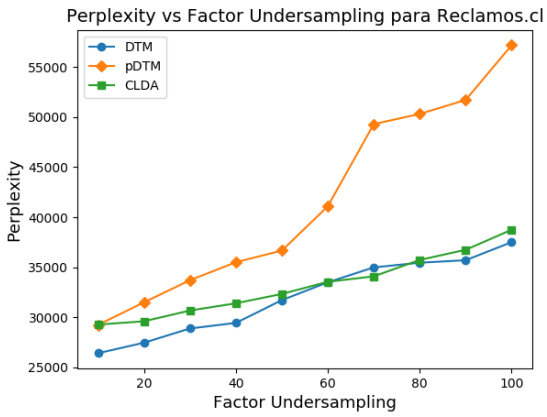
Figura 4.2: (a) Valores de perplexity en función del factor de undersampling para dataset NIPS utilizando los tres algoritmos en estudio. (b) Tiempo de ejecución versus factor de undersampling para las pruebas de eficiencia sobre NIPS.

Las curvas de perplexity cambian en este experimento, estando DTM como *baseline*, siguiéndole pDTM y finalmente CLDA. Las tres curvas tienen un comportamiento lineal creciente y pareciera haber una brecha entre cada algoritmo de aproximadamente 1,400 puntos

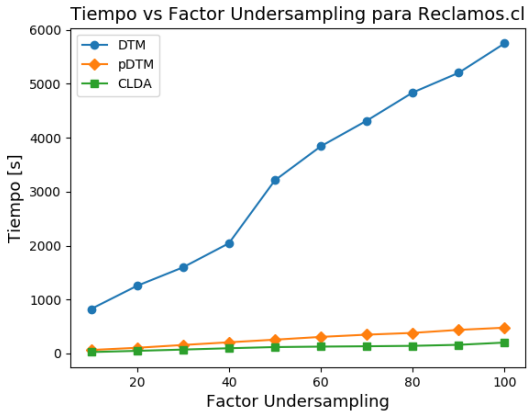
entre DTM y pDTM, y 1,500 entre pDTM y CLDA. Además, las pendientes de las curvas son mucho menor a las exhibidas en las pruebas de *oversampling*, siendo el caso de pDTM y más afectado al aumentar cien veces su perplexity con un factor de 64 respecto a la medida inicial con factor igual a 1. Por lo tanto, la calidad de los modelos entregados por los tres algoritmos tienen una reacción visible frente a datos repetidos, agregando mucho ruido al proceso de inferencia. En cambio, cuando se agregan nuevos datos a un corpus, las perplexities calculadas para DTM aumentan con una pendiente aproximada a 2, mientras que los otros dos algoritmos aumentan en un menor porcentaje.

Los tiempos de ejecución si mantienen el mismo orden de desempeño de los algoritmos, nuevamente con CLDA con menor tiempo en cada una de las pruebas realizadas. A pesar de que NIPS es un dataset con pocos documentos, CLDA demora cerca de 40 veces menos trabajando con el 10 % de los datos y casi 30 veces menos con el 100 % de los documentos.

Haciendo un cambio en el dataset a analizar, se recojen los registros de pruebas sobre Reclamos.cl y se grafica tanto la perplexty en 4.3a como el tiempo de ejecución en 4.3b, ambos en función del factor de *undersampling*.



(a) Perplexity en función de F. undersampling.



(b) Tiempo de ejecución en función de F. undersampling.

Figura 4.3: (a) Valores de perplexity en función del factor de undersampling y (b) Tiempo de ejecución en función del factor de undersampling para las pruebas de eficiencia sobre el dataset de NIPS.

Al ver los resultados hay un contraste entre como trabaja CLDA con poco y muchos datos. En el caso de Reclamos.cl DTM sigue obteniendo mejores resultados pero CLDA le sigue de cerca llegando a estar por debajo del algoritmo base al construir los modelos con el 60 % y 70 % de los datos. Esta mejora es sustancial de modo que hay un ahorro de un orden de magnitud en tiempo. Como CLDA además está diseñado para ser distribuido y paralelizado, los tiempo de cómputo podrían ser aun menores. Sin embargo, no sabemos aun cuál es el impacto en la calidad de los modelos al ejecutar una tarea en más de un core de procesamiento.

Hay dos puntos importantes de analizar en esta sección. DTM genera, en la mayoría de los casos, soluciones de mejor calidad para un dataset con cerca de 1.000 documentos y otro con 100.000. El problema asociado a esta mejor calidad es que el algoritmo se demora más (una orden de magnitud) que las propuestas que buscan mejorar esos tiempo pero a un coste de calidad de resultados. Este balance entre tiempo de cómputo y calidad de solución debe hacerse dependiendo del uso que uno le de a los resultados y la frecuencia de actualización con la llegada de nuevos datos. Para sistemas *offline* (que no necesiten una actualización permanente de los datos) es posible utilizar algoritmos que tomen más tiempo en el procesamiento de los datos pues se preferirá un mejor modelado a tener una constante actualización. Si se necesita un sistema web, por ejemplo, alimentado constantemente con datos y donde se quiera hacer análisis de estos sin tener que leer su contenido es posible utilizar los dos algoritmos propuestos, perdiendo calidad final pero ganando mucho tiempo de cómputo.

4.2. Pruebas paralelas

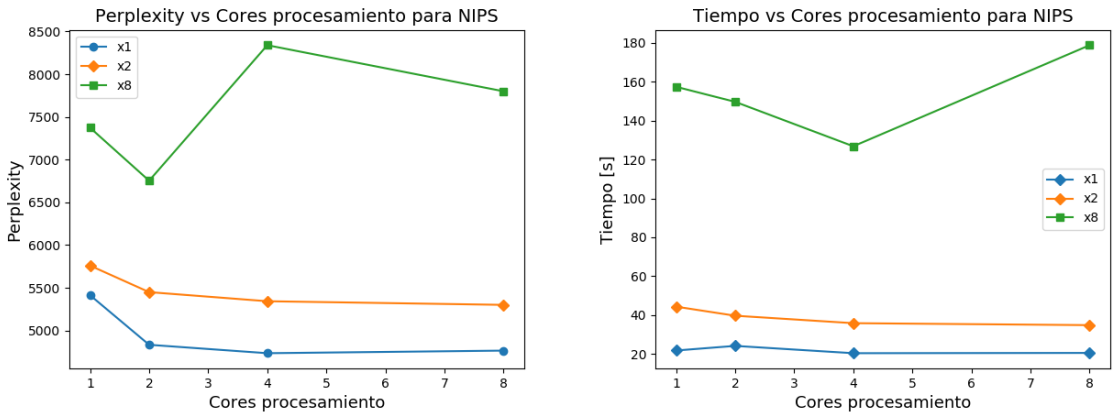
Debido a la disponibilidad de código, solo es posible probar el algoritmo CLDA para las pruebas de rendimiento en paralelo. A la fecha, el código liberado por los autores de pDTM no es la versión final con que presentaron el paper y, a causa de esto, no contiene el componente paralelo ni distribuido. Quedará como trabajo a futuro la implementación del algoritmo en su versión paralela, siguiendo las instrucciones tal como se indica en [14].

En esta sección se experimentará con el algoritmo CLDA, ejecutando nuevamente las pruebas sobre los corpus NIPS y Reclamos.cl, pero esta vez aumentando la cantidad de cores

asignados al procesamiento de datos. Para aprovechar los recursos del servidor, fueron ejecutadas pruebas paralelamente usando todos los núcleos disponibles (ej: para pruebas con $cores = 2$, se ejecutaron cuatro pruebas simultaneas). Se optó nuevamente con trabajar con potencias de 2 hasta el máximo disponible que es 8. Se esperaría que al paralelizar un proceso, este demore menos tiempo en completarse comprometiendo la calidad debido a la desconexión entre parámetros, cortes temporales, entre otros elementos del problema.

4.2.1. Oversampling

Se ejecuta CLDA con 4 configuraciones de cores de procesamiento sobre 3 versiones de NIPS: datos originales, con factor de *oversampling* x2 (doble de los datos) y x8 (ocho veces la cantidad original). Se mide la perplexity y tiempo de ejecución registrando los resultados en las tablas A.6 y A.7 del apéndice (como fueron realizadas primero las pruebas de *undersampling* los valores asociados al uso completo de los datasets se encuentran en esas tablas del apéndice). Con estos registros se graficó perplexity en función de los cores y el tiempo en función del mismo parámetro.



(a) Perplexity en función de Cores.

(b) Tiempo en segundos en función de cores

Figura 4.4: (a) Valores de perplexity y (b) tiempo en función de la cantidad de cores de procesamiento para NIPS usando técnica de oversampling.

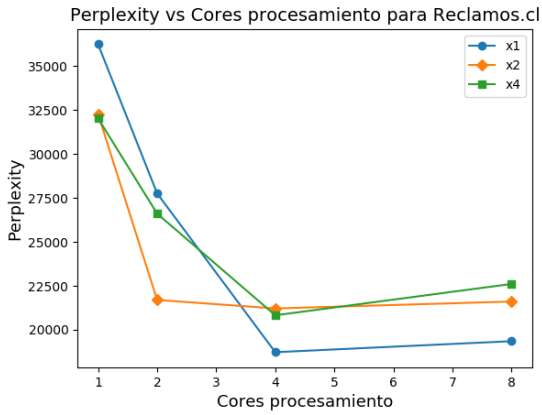
Visualizando los registros se observa una disminución tanto de perplexity como de tiempo al aumentar la cantidad de cores. No obstante, la disminución de tiempo de cómputo fue menor

a la esperada sospechando que existe, dentro de los pasos descritos en el estado del arte para CLDA, un 'cuello de botella' que repercute en evitar la disminución drástica del tiempo. Observando el código del algoritmo se ve claramente que hay dos secciones que se pueden paralelizar (LDA y K-means) pero el resto de los pasos son scripts en Python para el manejo de datos y comunicación entre LDA y K-means. Al no estar completamente paralelizado el algoritmo se producen pasos que toman más tiempo al aumentar la cantidad de datos. Es por esto que CLDA, como lo presentan los autores en su código fuente, es un algoritmo pseudo-paralelo. Esto podría mejorarse usando un framework para tareas distribuidas y paralelas como lo es Apache Spark.

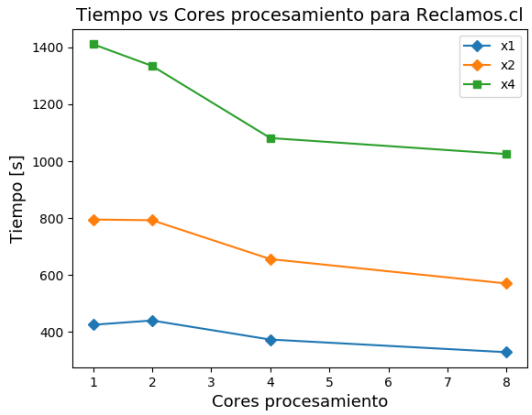
Otro elemento de sorpresa es que las curvas de perplexity disminuyen en el tiempo cuando la suposición inicial era que estas irían en aumento. Por lo tanto, el paralelizar el algoritmo no afecta a la comunicación de datos entre procesos. Un caso anómalo es el conjunto de datos con factor de *oversampling* igual a 8 cuya perplexity baja en los primeros dos pasos pero vuelve aumentar. Lo mismo ocurre con el tiempo de procesamiento. La conclusión más posible es un error experimental en la toma datos y/o configuración del algoritmo para esas pruebas.

Repitiendo los mismos pasos, se registran los resultados en las tablas A.8 y A.9 del apéndice. Nuevamente los datos para el corpus completo están en la sección de *undersampling*. Esta información es graficada en las siguientes figuras.

Los resultados para Reclamos.cl difieren de la experimentación con NIPS. Los puntos están más cercanos y al aumentar la cantidad de cores cambia el orden de cuál algoritmo es mejor, efecto no visto para las pruebas anteriores donde es constante el orden. Si se cumple finalmente que el dataset con datos originales tiene perplexity más baja que los dataset con datos repetidos, pero para beneficio del usuario, cuyos datasets pueden contener más de una instancia de los datos (ej: twitter y retweets), CLDA no sufre tanto en calidad como DTM y pDTM. El principal determinante de esto sería K-means en la fase de determinar los tópicos globales debido a que, al existir datos repetidos, estos serían combinaciones particularmente probables de valores, que deberían tener un mayor peso debido a eso. Finalmente, las observaciones con mismo valores no se transforman en datos redundantes.



(a) Perplexity en función de Cores



(b) Tiempo en segundos en función de cores

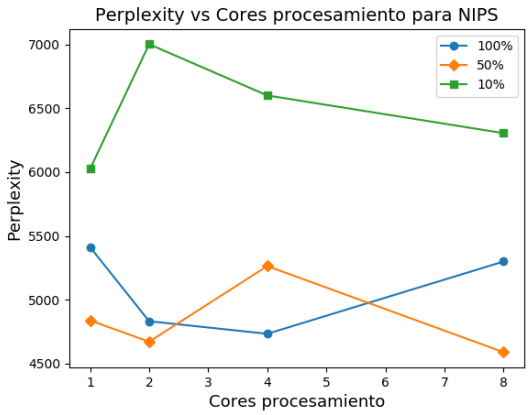
Figura 4.5: (a) Valores de perplexity y (b) tiempo de ejecución en función de la cantidad de cores de procesamiento para Reclamos.cl usando técnica de oversampling.

Los tiempos de ejecución son decrecientes consecuencia de aumentar el número de cores. Es esperable, al igual que los experimentos con NIPS, que a mayor volumen de datos el algoritmo demore más en procesarlos.

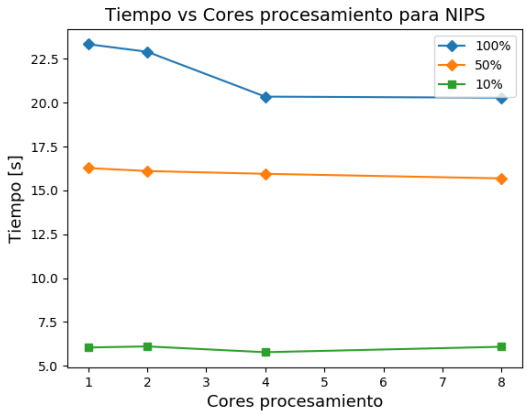
4.2.2. Undersampling

Para las pruebas de *undersampling* se trabajó con tres versiones de ambos corpus: con 100 %, 50 % y 10 % de los datos originales. Como se mencionó anteriormente, la reducción de datos se hizo de forma aleatoria y proporcional a los documentos en cada corte temporal. Los experimentos sobre NIPS fueron registrados en las tres tablas del anexo A.4. Cada una representa uno de los tres porcentajes de *undersampling*. Con los datos recuperados se construyeron los siguientes gráficos.

Los resultados, en relación a la cantidad de datos, es contraria a lo observado al aplicar *oversampling*, incluso con los experimentos de *undersampling* sin ejecutar CLDA en un entorno distribuido. CLDA no estaría funcionando bien en ambientes de baja cantidad de datos. Si el 10 % de NIPS son solamente 135 documentos, divididos entre 13 intervalos temporales,



(a) Perplexity en función de Cores



(b) Tiempo en función de Cores

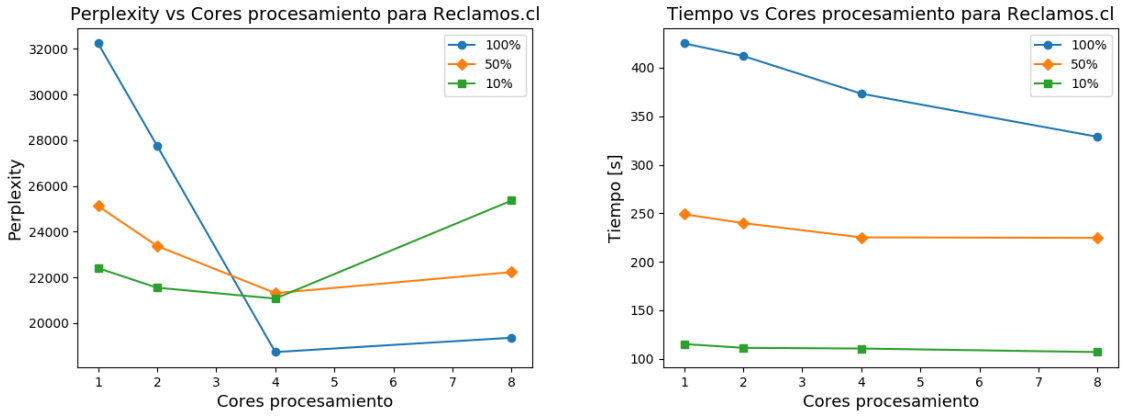
Figura 4.6: (a) Valores de perplexity y b tiempos de ejecución en segundos en función de la cantidad de cores de procesamiento para NIPS con undersampling.

cada uno de estos tendrá en promedio 10 entradas. Estas secciones son modeladas independientemente con LDA (específicamente pLDA), método sensible al número de documentos y el largo en cantidad de palabras de los ejemplos seleccionados en ese 10 %. Es por esto que la información utilizada para generar los modelos tiene impacto directo en el resultado, que utiliza k-means como herramienta para extraer los tópicos finales. Probablemente sería mejor bajar la cantidad de tópicos, o trabajar con un número dinámico de tópicos por tiempo, pero se perdería la cantidad de temas que inicialmente se se deseaba descubrir en el dataset (asumiendo un conocimiento previo de qué contenido se espera encontrar).

Los tiempos de ejecución de la figura 4.6b muestran que a mayor cantidad de datos en el corpus a procesar más se demora el algoritmo en ejecutarse. Esto debe pasar independiente si se está aplicando *oversampling* o *undersampling*. No obstante, al aumentar los cores de procesamiento el tiempo de procesamiento no disminuye considerablemente, demorándose unas décimas de segundo más para el dataset del 10 %. CLDA no estaría aprovechándose del entorno paralelo cuando trabajó con pocos datos. Su implementación que no es completamente paralela debe tener fases que toman más tiempo en completarse que la misma estimación de tópicos locales de LDA y la fase final de tópicos globales con K-means.

Se repitió el experimento para el dataset de Reclamos.cl registrando los resultados en el

anexo A.5. A diferencia de las pruebas paralelas con NIPS, los intervalos de tiempo ahora contendrán un mayor número de documentos. Este aumento debería cambiar los resultados pues, a diferencia de NIPS donde existe la posibilidad de que en un corte temporal quede una cantidad reducida de documentos para modelar sus tópicos, en cada sección habrían suficientes entradas como para que la fase de LDA en CLDA capture bien los temas (considerando 10 tópicos). Para corroborar este argumento, se graficaron los datos del anexo mencionado.



(a) Perplexity en función de Cores

(b) Tiempo en función de Cores

Figura 4.7: (a) Valores de perplexity y (b) tiempos de ejecución en segundos en función de la cantidad de cores de procesamiento para Reclamos.cl con undersampling.

La figura 4.7a muestra los resultados de perplexity para tres versiones del corpus Reclamos.cl. Como se mencionó anteriormente este dataset tiene un número elevado de documentos (100 veces más que NIPS). En esta oportunidad, cuando la tarea es ejecutada sobre un solo núcleo observamos un fenómeno igual al que cuando se experimentó con *undersampling*: a mayor cantidad de datos es mayor la perplexity del modelo, es decir, es más difícil determinar una mejor representación en el tiempo para los datos. Sin embargo, al aumentar la cantidad de cores es el corpus de 100 % quien tiene menor perplexity, y no sus contrapartes con menos datos. Con 50 % y 10 % hay mejores resultados con baja cantidad de cores pero al aumentar empeoran en calidad, siendo el orden de mejor a mayor el siguiente (bajo 8 cores): 100 %, 50 % y 10 %. Así, se puede concluir que CLDA se desempeña mejor en ambientes con más datos, conclusión también sacada para las pruebas sobre NIPS.

Es importante la deducción anterior pues se buscan algoritmos con buen desempeño, que

sean eficientes y eficaces, acercándose a los resultados de DTM o incluso superándolos. La ventaja de CLDA sobre el resto de los métodos expuestos es su posibilidad de paralelizarse, y por lo visto, aumentando la cantidad de cores mejora la calidad del modelo. Quien sería el responsable de esto es el algoritmo pLDA+ empleado por CLDA. Este algoritmo divide los datos proporcionalmente a la cantidad de cores y genera modelos locales que capturan mejor el contenido de cada sub-corpus. Al hacer la mezcla de todas estas piezas se tiene un modelo con mayor información que uno que analiza por completo un corpus de documentos.

Siguiendo la misma idea incial del párrafo anterior, y analizando el gráfico de tiempos, hay una equivalencia entre el tamaño del corpus y cuánto demora en ejecutar el algoritmo. Esta es lineal con respecto al factor de *undersampling* donde el corpus con 50 % de los datos demora aproximadamente el doble que los tiempos del 10 % de los datos.

CLDA en estas 4 pruebas de paralelismo ha demostrado mejores tiempos de ejecución que pDTM y DTM. Comparando los resultados entre pruebas paralelas no hay una gran mejora entre ejecutar el algoritmo con más o menos cores. Hay una disminución entre las medias de duraciones pero era de esperar una disminución proporcional al aumento de fuentes de procesamiento, es decir, al ejecutar con 4 cores se esperaba que la disminución en tiempo hubiese sido más o menos de un cuarto del tiempo con 1 solo core. Como se mencionó anteriormente, la implementación del algoritmo no es por completo paralelizada teniendo cuellos de botella en las secciones que convierten la salida de un algoritmo (LDA) en la entrada del otro (K-means). Siendo estas principalmente matrices crecen con el tamaño del corpus y del vocabulario. Una solución para este problema es llevar la implementación a un framework que soporte paralelismo como lo es Spark.

Capítulo 5

Conclusiones

En este capítulo final se exponen las conclusiones obtenidas a partir de la realización de un proceso de investigación, cuyo foco es hacer una comparación entre métodos de modelado de tópicos dinámicos, utilizando el algoritmo clásico en la literatura y dos nuevas propuestas que usan enfoques diferentes para resolver el problema buscando reducir el tiempo de cómputo, especialmente con grandes bases de datos de texto.

El objetivo principal de este estudio fue la evaluación de distintos algoritmos de estimación de modelos de tópicos dinámicos expuestos en la literatura, en términos de eficiencia y eficacia. Dentro de esta tarea estaba entender cómo funcionaban los métodos y cuáles eran las principales diferencias entre ellos.

Los algoritmos de modelado de tópicos dinámicos permiten extraer los tópicos latentes, en un conjunto de documentos llamado corpus, a través de un proceso gaussiano. Además, captan la evolución de estos temas en el tiempo y cómo cambian en función del contenido (documentos y palabras) de cada intervalo. Dentro de estos algoritmos tenemos a Dynamic Topic Models como modelo base y otros más modernos como Parallel DTM y Clustered LDA, los que exploran otras aristas del problema para mejorar los tiempo de ejecución del algoritmo base, además de generar resultados de igual o mejor calidad.

Se diseñaron pruebas para estos tres algoritmos que pudiesen medir, de forma experimental, el rendimiento en tiempo y la calidad de modelos que podían generar dado un corpus,

usando como métricas el tiempo de ejecución y perplexity respectivamente. Para esto fueron utilizados dos datasets, NIPS y Reclamos.cl, cuyas composición es diferente en cantidad de documentos, palabras, lenguaje y dominio (uno de papers de conferencias y el segundo de reclamos sobre servicios o empresas). Se aplicaron las técnicas *Undersampling* y *Oversampling*, para que las pruebas pudiesen analizar que pasaba al añadir nuevos datos a un corpus, generar ruido al añadir datos repetidos, y en el mismo esfuerzo observar rendimientos al variar tamaños de datasets. Además, fueron confeccionadas pruebas que midieron el desempeño de los algoritmos en un entorno paralelo, tomando como entrada corpus con cantidad de documentos diferentes y variando los cores de procesamiento.

Para las pruebas de *oversampling* CLDA posee los mejores resultados de perplexity al aumentar la cantidad de datos y, además, mejora los tiempos de ejecución de pDTM y DTM. Por lo tanto, CLDA no es tan afectado por el aumento de datos repetidos como le sucede a DTM y especialmente a pDTM cuya curva sube aceleradamente. Los tiempos de ejecución cambian pues es DTM quien demora más en todo tiempo y CLDA con pDTM logran su objetivo de bajar los tiempos de ejecución. Por lo tanto, sus modelos aceleran la generación de tópicos.

En las pruebas de *undersampling* el escenario varía según el dataset. Para NIPS, DTM se queda con los mejores resultados, seguido de pDTM y CLDA. Dado el cambio de posición de CLDA se daría a entender que este algoritmo trabaja cuando hay una mayor cantidad de datos, cuando los otros dos algoritmos generan mejores modelos al considerar que los tópicos tienen una dependencia entre si, y no como CLDA que al procesar pocos datos por corte temporal no logra captar estas cercanías entre los tópicos de cada tiempo. El panorama cambia con Reclamos pues CLDA y DTM tiene resultados muy cercanos, siendo el primero mejor en dos instancias. Aunque se haya disminuido la cantidad de datos, hay suficientes como para que CLDA pueda funcionar con resultados de buena calidad. En ambos datasets, el tiempo de ejecución pDTM y CLDA se mantienen cerca entre si, pero mejoran con creces a DTM.

Por lo tanto, para las primeras pruebas CLDA es un algoritmo competente dependiendo de la cantidad de datos que haya en cada dataset, y sus tiempos de ejecución siempre son los menores. pDTM es un algoritmo que pierde calidad cuando hay muchos datos y en mayor

proporción cuando están repetidos. Sin embargo, sus tiempo son menores a los de DTM. DTM se mantiene con las mejores soluciones excepto cuando hay datos repetidos, además que es el más lento de los tres. Este algoritmo es bueno utilizar cuando el tiempo de cómputo no es un factor clave en la aplicación a realizar. También, hay un *trade-off* entre el tiempo y calidad de solución, tomando más tiempo las soluciones con mayor valor de perplexity. La elección de cuál algoritmo utilizar dependerá del contexto de los datos.

Para las pruebas paralelas se descubrió que la implementación de CLDA no es completamente paralelizada ganando muy poco en tiempo al aumentar los cores de procesamiento. Una observación importante es que al aumentar los cores de procesamiento si disminuye la perplexity de los modelos por lo que se sospecha que, pLDA+ encargado de calcular los tópicos en la primera parte del algoritmo, se beneficia de la partición de datos para generar modelos más informativos dado su esquema de trabajo visto en la figura 2.2.

Finalmente, se estudia como algoritmos de clustering como k-means pueden ser utilizados para trabajar con modelado de tópicos dinámicos, donde los centrodies encontrados indicarían cuáles son los tópicos globales que agrupan los tópicos locales de cada corte temporal. Al ser K-means uno de los algoritmos de agrupamiento más simples, este trabajo podría ser mejorado implementando otro algoritmo de clustering como DBSCAN, algoritmo no supervisado que trabaja en base a la densidad de los datos.

5.1. Trabajo a futuro

- **Estudio implementación de algoritmos en Apache Spark**: Apache Spark es un poderoso *framework* para el cómputo en paralelo y distribuido de datos. Su núcleo contiene una librería para Aprendizaje de máquinas llamada MLlib, conteniendo implementado un grupo de algoritmos clásicos para el análisis de datos, dentro de los cuales está LDA. Sería interesante explorar la posibilidad de implementar DTM pues no existe una a la fecha.
- **Implementación de CLDA en Apache Spark**: el principal problema que tiene CLDA

es el cuello de botella por su pseudo-paralelización. Esto puedo ser mejorado implementando el conjunto de pasos con Spark, considerando que MLlib contiene una implementación de LDA y de K-means. También, cómo cambian los resultados usando otro algoritmo de clustering.

- **Pruebas paralelas con pDTM**: lamentablemente, por derechos de autor, el código completo no está disponible. Sería interesante poder contar con esta implementación para compararla con CLDA y cuantificar qué algoritmo se beneficia del paralelismo y carga distribuida en diferentes nodos de procesamiento.

Apéndice A

Apéndice: Tablas de Experimentos

A.1. Resultados pruebas de eficiencia

Los siguientes resultados corresponden a las pruebas de eficiencia sobre el dataset NIPS usando 7 factores de Oversampling. Posteriormente, se presentan los resultados de las pruebas sobre NIPS y Reclamos.cl usando Undersampling.

Cuadro A.1: Tiempos de ejecución y valores de perplexity para las pruebas de eficiencia con diferentes factores de oversampling, sobre el dataset de NIPS.

Factor	pDTM		CLDA		DTM	
Oversampling	tiempo [s]	Perplexity	tiempo [s]	Perplexity	tiempo [s]	Perplexity
1	72.35	3,575.37	25.27	4,932.33	433.43	2055.84
2	140.66	13,233.07	44.3	5,759.50	687.30	4009.53
4	277.32	105,571.01	80.95	4,920.54	1037.81	7730.14
8	550.76	126,307.72	157.42	7,373.65	1902.18	15197.45
16	1093.96	145,087.34	316.96	6,417.03	3692.43	30247.68
32	2187.31	243,964.31	629.85	13,653.67	9176.54	60004.68
64	4366.09	452,031.52	1310.63	12,847.90	12708.28	114341.28

Cuadro A.2: Valores de *perplexity* con diferentes factores de undersampling para los algoritmos DTM, pDTM y CLDA sobre los dataset de NIPS y Reclamos.cl.

Factor	NIPS			Reclamos		
Undersampling	DTM	pDTM	CLDA	DTM	pDTM	CLDA
100 %	2055.84	3575.37	4901.29	37502.71	57201.41	38741.67
90 %	1868.06	3260.29	4831.89	35697.09	51723.95	36739.87
80 %	1660.56	3010.20	4726.22	35446.50	50326.99	35722.77
70 %	1459.71	2868.01	4702.80	34978.73	49305.74	34085.46
60 %	1218.83	2673.53	4529.07	33487.53	41093.40	33538.40
50 %	1003.22	2581.64	4251.79	31703.87	36657.67	32323.14
40 %	779.57	2541.26	4252.29	29427.52	35498.25	31388.86
30 %	588.35	2586.74	4164.90	28870.35	33693.44	30664.15
20 %	386.11	2792.62	4494.08	27446.50	31489.04	29260.52
10 %	174.38	3017.83	4468.29	26397.53	29226.23	29589.52

Cuadro A.3: Tiempo de ejecución de algoritmos (en segundos) en las pruebas de undersampling, para los datasets NIPS y Reclamos.cl.

Factor	NIPS			Reclamos.cl		
Undersampling	DTM	pDTM	CLDA	DTM	pDTM	CLDA
100 %	810.96	72.35	27.74	5749.14	478.02	202.39
90 %	648.29	67.87	23.87	5203.20	437.85	160.90
80 %	583.54	59.63	21.53	4837.87	381.63	141.84
70 %	551.44	52.79	19.40	4316.13	349.00	134.11
60 %	515.46	46.06	17.36	3841.01	307.36	127.69
50 %	485.66	39.34	15.11	3209.80	256.25	119.05
40 %	431.27	32.71	14.10	2043.20	207.55	97.23
30 %	363.06	25.45	11.69	1596.66	157.52	72.61
20 %	309.36	18.05	8.66	1254.07	105.92	48.98
10 %	215.42	11.44	5.65	825.08	64.33	26.55

Cuadro A.4: Resultados de *perplexity* en las pruebas de undersampling para el algoritmo CLDA, sobre el datasets de Reclamos.cl.

	Factor Undersampling									
Prueba	100 %	90 %	80 %	70 %	60 %	50 %	40 %	30 %	20 %	10 %
1	55069.34	24595.59	46367.26	25568.21	47654.02	52886.61	26134.45	31032.53	42046.62	39240.58
2	59661.39	61531.40	23497.56	25335.28	25370.09	23723.99	26081.98	22484.61	20329.28	18984.84
3	62916.77	25894.38	47417.33	25222.86	23482.29	24964.00	24240.01	35828.17	20301.16	18934.44
4	25379.49	28049.20	23497.56	60166.91	55752.50	54777.88	22809.45	48686.36	35286.90	32953.10
5	21870.50	26741.14	29512.09	43161.80	19974.05	53595.21	32959.92	21016.38	15088.19	26498.20
6	24698.09	60502.34	55066.37	25568.79	30241.09	16209.75	30866.38	32000.20	44400.35	36562.31
7	63221.90	18238.30	58386.12	21308.04	27225.45	24202.62	30171.18	35472.17	21579.65	25600.75
8	24353.44	60057.36	24742.40	33663.05	24799.04	24756.37	23870.66	40608.90	44582.07	35826.16
9	24942.52	36239.01	23739.39	57310.04	56253.29	23862.79	45568.61	24872.07	28897.43	23639.26
10	25303.29	25549.96	25001.63	23549.64	24632.21	24252.14	51185.96	14640.15	20093.52	37655.58
MAX	63221.90	61531.40	58386.12	60166.91	56253.29	54777.88	51185.96	48686.36	44582.07	39240.58
MIN	21870.50	18238.30	23497.56	21308.04	19974.05	16209.75	22809.45	14640.15	15088.19	18934.44
MEDIA	38741.67	36739.87	35722.77	34085.46	33538.40	32323.14	31388.86	30664.15	29260.52	29589.52

Cuadro A.5: Resultados de *perplexity* en las pruebas de undersampling para el algoritmo CLDA, sobre el datasets de NIPS.

	Factor Undersampling									
Prueba	100 %	90 %	80 %	70 %	60 %	50 %	40 %	30 %	20 %	10 %
1	5217.80	4591.77	4795.40	4176.89	4469.70	4358.00	4428.99	3918.69	4581.10	1642.69
2	4477.09	5541.22	4225.39	4512.45	3770.45	4734.09	3754.21	4284.10	4819.91	4137.89
3	5763.52	4516.20	5156.10	4732.26	5472.40	4550.00	4867.84	4721.19	4766.28	5336.32
4	5086.34	5044.39	5131.61	5111.07	4241.38	4322.10	4497.26	4141.74	4257.41	5336.32
5	4923.87	5009.71	4795.40	5301.01	4520.29	4235.94	3483.26	4018.76	4360.19	4195.19
6	4939.16	6062.61	3872.57	4774.11	4057.59	4603.13	3892.75	4218.23	4282.15	5406.76
7	4508.15	2296.80	5278.36	4428.86	4845.29	4894.70	4484.91	3918.47	4510.49	4288.60
8	4849.54	5148.11	5195.26	4765.91	4520.29	2508.19	4368.12	4143.11	4581.18	4708.10
9	4340.64	5188.35	4266.19	4774.11	4352.32	4531.83	4640.31	4118.57	4524.68	5232.75
10	4906.76	4919.72	4545.96	4451.36	5041.00	3779.96	4105.25	4166.14	4257.41	4398.27
MAX	5763.52	6062.61	5278.36	5301.01	5472.40	4894.70	4867.84	4721.19	4819.91	5406.76
MIN	4340.64	2296.80	3872.57	4176.89	3770.45	2508.19	3483.26	3918.47	4257.41	1642.69
MEDIA	4901.29	4831.89	4726.22	4702.80	4529.07	4251.79	4252.29	4164.90	4494.08	4468.29

A.2. Anexo: Resultados pruebas paralelas con NIPS usando Oversampling

Los siguientes resultados corresponden a diversas pruebas realizadas con el algoritmo CLDA sobre el corpus de NIPS. En cada oportunidad se corrieron 15 pruebas donde se varía el factor de oversampling del corpus y se ejecuta la implementación con asignándole 2, 4 u 8 cores de procesamiento. Se registra la *perplexity* y el tiempo de ejecución. Cada fila de las tablas corresponde a un experimento diferente.

Cuadro A.6: Valores de *perplexity* para las pruebas de oversampling con factor igual 2, con diferentes cantidades de cores asociados al proceso, sobre el dataset NIPS. Cada línea representa una prueba del algoritmo CLDA.

Factor: x2	Perplexity			Tiempo [s]		
N° cores	2	4	8	2	4	8
1	4981.03	5691.80	6516.92	39.85	35.00	34.74
2	6249.97	5880.96	5350.19	39.94	36.67	34.67
3	5152.43	4585.16	6095.70	39.46	35.80	34.75
4	5972.00	6266.69	4508.96	39.47	35.42	34.85
5	4889.17	4287.05	4027.15	39.46	36.12	34.95
MAX	6249.97	6266.69	6516.92	39.94	36.67	34.95
MIN	4889.17	4287.05	4027.15	39.46	35.00	34.67
MEDIA	5448.92	5342.33	5299.78	39.64	35.80	34.79

Cuadro A.7: Valores de *perplexity* para las pruebas de oversampling con factor igual a 8, con diferentes cantidades de cores asociados al proceso, sobre el dataset NIPS. Cada línea representa una prueba del algoritmo CLDA.

Factor: x8	Perplexity			Tiempo [s]		
N° de cores	2	4	8	2	4	8
1	5402.19	8133.87	7148.57	150.12	129.56	178.84
2	5940.04	8257.80	7269.07	149.77	126.37	178.73
3	7439.55	9877.63	7070.88	149.53	130.05	179.00
4	8711.87	6088.87	9537.25	149.73	120.51	178.62
5	6273.41	9340.93	7975.09	149.30	127.63	178.91
MAX	8711.87	9877.63	9537.25	150.12	130.05	179.00
MIN	5402.19	6088.87	7070.88	149.30	120.51	178.62
MEDIA	6753.41	8339.82	7800.17	149.69	126.82	178.82

A.3. Anexo: Resultados pruebas paralelas con Reclamos.cl usando Oversampling

Los siguientes resultados corresponden a diversas pruebas realizadas con el algoritmo CLDA sobre el corpus de Reclamos.cl. En cada oportunidad se corrieron 20 pruebas donde se varía el factor de oversampling del corpus (x2 y x4) y se ejecuta la implementación con asignándole 1, 2, 4 u 8 cores de procesamiento. Se registra la *perplexity* y el tiempo de ejecución. Cada fila de las tablas corresponde a un experimento diferente.

Cuadro A.8: Valores de *perplexity* para las pruebas de oversampling con factor igual a 2, con diferentes cantidades de cores asociados al proceso, sobre el dataset de Reclamos.cl. Cada línea representa una prueba del algoritmo CLDA.

Factor:x2	Perplexity				Tiempo [s]			
N° cores	1	2	4	8	1	2	4	8
1	18882.03	23372.44	24102.30	21432.50	797.16	790.83	649.12	571.60
2	45198.24	22346.86	14773.77	19150.52	790.59	799.56	662.98	572.10
3	49561.44	23544.16	22605.73	22883.75	801.19	784.83	658.91	568.53
4	23010.96	18682.17	20130.35	20898.78	793.60	798.28	657.12	568.37
5	24705.91	20559.23	24468.13	23679.21	792.41	788.95	651.86	570.61
MAX	49561.44	23544.16	24468.13	23679.21	801.19	799.56	662.98	572.10
MIN	18882.03	18682.17	14773.77	19150.52	790.59	784.83	649.12	568.37
MEDIA	32271.72	21700.97	21216.06	21608.95	794.99	792.49	656.00	570.24

Cuadro A.9: Valores de *perplexity* para las pruebas de oversampling con factor igual a 4, con diferentes cantidades de cores asociados al proceso, sobre el dataset de Reclamos.cl. Cada línea representa una prueba del algoritmo CLDA.

Factor:x4	Perplexity				Tiempo [s]			
N° cores	1	2	4	8	1	2	4	8
1	49497.56	35336.76	27271.53	15569.18	1416,45	1343.14	1064.88	1025.46
2	46213.77	26142.02	24723.73	26747.14	1417.23	1328.07	1089.67	1024.75
3	23891.49	23180.98	15716.34	20870.53	1416.32	1322.65	1076.60	1022.06
4	14460.52	21344.02	18302.85	25975.88	1407.47	1328.79	1089.92	1025.55
5	26068.10	27040.18	18116.86	23886.75	1403.36	1350.57	1085.87	1026.86
MAX	49497.56	35336.76	27271.53	26747.14	1417.23	1350.57	1089.92	1026.86
MIN	14460.52	21344.02	15716.34	15569.18	1403.36	1322.65	1064.88	1022.06
MEDIA	32026.29	26608.79	20826.26	22609.90	1411.10	1334.64	1081.39	1024.94

A.4. Anexo: Resultados pruebas paralelas con NIPS usando Undersampling.

Los siguientes resultados corresponden a diversas pruebas realizadas con el algoritmo CLDA sobre el corpus de NIPS. En cada oportunidad se corrieron 20 pruebas donde se varía el factor de undersampling del corpus y se ejecuta la implementación con asignándole 1, 2, 4 u 8 cores de procesamiento. Se registra la *perplexity* y el tiempo de ejecución. Cada fila de las tablas corresponde a un experimento diferente.

Cuadro A.10: Valores de *perplexity* y tiempos de ejecución para las pruebas de undersampling (datos completos), con diferentes cantidades de cores asociados al procesamiento del dataset NIPS. Cada línea representa una prueba del algoritmo CLDA..

Factor: 100 %	Perplexity				Tiempo [s]			
N° cores	1	2	4	8	1	2	4	8
1	4568.66	4652.70	4131.09	6516.92	23.77	21.16	19.58	21.45
2	5297.84	4573.85	4486.62	5350.19	22.64	23.60	21.85	19.50
3	7054.16	5241.01	4846.41	6095.70	23.77	22.89	20.17	19.57
4	5243.15	4895.31	4807.44	4508.96	24.75	23.53	19.61	20.48
5	4900.46	4799.25	5397.79	4027.15	21.74	23.36	20.55	20.44
MAX	7054.16	5241.01	5397.79	6516.92	24.75	23.60	21.85	21.45
MIN	4568.66	4573.85	4131.09	4027.15	21.74	21.16	19.58	29.50
MEDIA	5412.86	4832.42	4733.87	5299.78	23.34	22.91	20.35	20.29

Cuadro A.11: Valores de *perplexity* y tiempos de ejecución para las pruebas de undersampling (mitad de los datos), con diferentes cantidades de cores asociados al procesamiento del dataset NIPS. Cada línea representa una prueba del algoritmo CLDA.

Factor: 50 %	Perplexity				Tiempo [s]			
N° cores	1	2	4	8	1	2	4	8
1	3880.08	2929.37	4519.24	4536.14	16.26	17.16	19.58	15.45
2	6856.50	4088.42	5829.58	4610.58	16.30	13.60	11.85	17.50
3	4525.95	6791.25	5129.88	4633.40	16.28	22.89	17.17	15.57
4	4627.21	5175.31	5829.58	3830.09	16.36	13.53	19.61	15.48
5	4305.10	4374.14	5013.85	5338.58	16.22	13.36	11.55	14.44
MAX	6856.50	6791.25	5829.58	5338.58	16.36	22.89	19.61	17.50
MIN	3880.08	2929.37	4519.24	3830.09	16.22	13.36	11.55	14.44
MEDIA	4838.97	4671.70	5264.43	4589.76	16.28	16.11	15.95	15.69

Cuadro A.12: Valores de *perplexity* y tiempos de ejecución para las pruebas de undersampling (10 % de los datos, con diferentes cantidades de cores asociados al procesamiento del dataset NIPS. Cada línea representa una prueba del algoritmo CLDA.

Factor: 10 %	Perplexity				Tiempo [s]			
N° cores	1	2	4	8	1	2	4	8
1	5311.31	10452.81	7043.14	8700.75	6.06	6.06	5.79	6.07
2	6386.63	5823.46	5940.49	4999.98	6.13	6.04	5.74	6.10
3	7343.06	4990.52	5581.90	5720.85	5.99	6.17	5.80	6.06
4	6068.12	8744.44	7518.26	7844.84	6.03	6.15	5.81	6.08
5	5014.44	4996.40	6917.56	4257.97	6.03	6.16	5.76	6.16
MAX	7343.06	10452.81	7518.26	8700.75	6.13	6.17	5.81	6.16
MIN	5014.44	4990.52	5581.90	4257.97	5.99	6.04	5.74	6.06
MEDIA	6024.71	7001.53	6600.27	6304.88	6.05	6.11	5.78	6.09

A.5. Anexo: Resultados pruebas paralelas con Reclamos.cl usando Undersampling.

Los siguientes resultados corresponden a diversas pruebas realizadas con el algoritmo CLDA sobre el corpus de Reclamos.cl. En cada oportunidad se corrieron 20 pruebas donde se varía el factor de undersampling del corpus y se ejecuta la implementación con asignándole 1, 2, 4 u 8 cores de procesamiento. Se registra la *perplexity* y el tiempo de ejecución. Cada fila de las tablas corresponde a un experimento diferente.

Cuadro A.13: Valores de *perplexity* y tiempos de ejecución para las pruebas de undersampling, con diferentes cantidades de threads asociados al procesamiento del dataset de Reclamos.cl. Factor de undersampling de 100 %.

Factor:100 %	Perplexity				Tiempo [s]			
N° Cores	1	2	4	8	1	2	4	8
1	39518.13	20057.42	28206.12	19556.61	424.98	417.77	370.37	328.52
2	37145.67	21195.41	12781.66	22858.77	424.21	412.64	377.75	329.48
3	41767.50	44298.72	14648.78	18374.07	436.06	408.81	373.15	328.28
4	23353.67	43410.61	16064.50	20207.72	421.11	411.18	376.26	328.73
5	19424.85	9817.86	21938.36	15776.55	419.61	410.68	368.98	329.07
MAX	41767.50	44298.72	28206.12	22858.77	436.06	417.77	377.75	329.48
MIN	19424.85	9817.86	12781.66	15776.55	419.61	408.81	368.98	328.28
MEDIA	32241.96	27756.00	18727.88	19354.74	425.19	412.22	373.30	328.82

Cuadro A.14: Valores de *perplexity* y tiempos de ejecución para las pruebas de undersampling, con diferentes cantidades de threads asociados al procesamiento del dataset de Reclamos.cl. Factor de undersampling de 50 %.

Factor: 50 %	Perplexity				Tiempo [s]			
N° cores	1	2	4	8	1	2	4	8
1	38570.06	12833.71	15375.26	15531.65	253.22	233.43	233.33	205.48
2	24320.47	20199.21	28825.09	29138.74	247.13	241.26	222.18	203.90
3	14642.46	36215.15	13594.76	17400.83	249.62	235.13	222.96	205.38
4	30136.74	19218.27	18961.19	19656.23	246.05	248.79	223.36	205.32
5	17936.88	28390.87	29781.76	29421.37	248.64	240.52	223.52	302.94
MAX	38570.06	36215.15	29781.76	29421.37	253.22	248.79	233.33	302.94
MIN	14642.46	12833.71	13594.76	15531.65	246.05	233.43	222.18	203.90
MEDIA	25121.32	23371.44	21307.61	22229.76	248.93	239.83	225.07	224.60

Cuadro A.15: Valores de perplexity y tiempos de ejecución para las pruebas de undersampling, con diferentes cantidades de threads asociados al procesamiento del dataset de Reclamos.cl. Factor de undersampling de 10 %.

Factor: 10 %	Perplexity				Tiempo [s]			
N° cores	1	2	4	8	1	2	4	8
1	14429.65	28683.13	19536.36	23182.38	119.670	102.23	110.22	106.54
2	25882.52	26036.59	28250.77	20117.07	114.23	114.60	110.21	106.63
3	26668.96	14822.63	26931.92	28616.82	114.43	113.88	111.27	106.39
4	29601.39	18647.11	20800.26	25693.82	113.97	112.46	110.48	107.97
5	15431.26	19536.36	9826.99	29196.45	113.150	112.74	110.07	106.50
MAX	29601.39	28683.13	28250.77	29196.45	119.67	114.60	111.27	107.97
MIN	14429.65	14822.63	9826.99	20117.07	113.15	102.23	110.07	106.39
MEDIA	22402.76	21545.16	21069.26	25361.31	115.09	111.18	110.45	106.81

Bibliografía

[1] David M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, April 2012.

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, vol 3:993–1022, March 2003.

[3] Charu C. Aggarwal and Cheng Xiang Zhai. *Mining Text Data*. Springer Publishing Company, Incorporated, 2012.

[4] Thomas Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, pages 289–296. Morgan Kaufmann Publishers Inc., 1999.

[5] Han Xiao and Thomas Stibor. Efficient collapsed gibbs sampling for latent dirichlet allocation. In *Proceedings of 2nd Asian Conference on Machine Learning*, volume 13 of *Proceedings of Machine Learning Research*, pages 63–78, Tokyo, Japan, 08–10 Nov 2010.

[6] Tom Griffiths. Gibbs sampling in the generative model of latent dirichlet allocation. 518, 01 2002.

[7] Yi Wang, Hongjie Bai, Matt Stanton, Wen-Yen Chen, and Edward Y. Chang. Plda: Parallel latent dirichlet allocation for large-scale applications. In *Algorithmic Aspects in Information and Management*, pages 301–314, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[8] David Newman, Arthur Asuncion, Padhraic Smyth, and Max Welling. Distributed algorithms for topic models. *J. Mach. Learn. Res.*, 10:1801–1828, December 2009.

[9] Zhiyuan Liu, Yuzhou Zhang, Edward Y. Chang, and Maosong Sun. Plda+: Parallel latent dirichlet allocation with data placement and pipeline processing. *ACM Trans. Intell. Syst. Technol.*, 2(3):26:1–26:18, May 2011.

[10] David M. Blei and John D. Lafferty. Dynamic topic models. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 113–120, New York, NY, USA, 2006. ACM.

[11] Xuerui Wang and Andrew McCallum. Topics over time: A non-markov continuous-time model of topical trends. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 424–433, New York, NY, USA, 2006. ACM.

[12] David M Blei and John D Lafferty. A correlated topic model of science. *The Annals of Applied Statistics*, pages 17–35, 2007.

[13] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.

[14] Arnab Bhadury, Jianfei Chen, Jun Zhu, and Shixia Liu. Scaling up dynamic topic models. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 381–390, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.

[15] Chris Gropp, Alexander Herzog, Ilya Safro, Paul W. Wilson, and Amy W. Apon. Scalable dynamic topic modeling with clustered latent dirichlet allocation (CLDA). *CoRR*, abs/1610.07703, 2016.

[16] Wei keng Liao. Parallel k-means data clustering. `http://www.ece.northwestern.edu/~wkliao/Kmeans/`, 2013.

[17] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[18] David M W Powers. Applications and explanations of zipf's law. In *Association for Computational Linguistics*, page 151–160, 1998.

[19] Sergey I Nikolenko, Sergey Koltsov, and Olessia Koltsova. Measuring topic quality in latent dirichlet allocation. In *Proceedings of the Philosophy, Mathematics, Linguistics: Aspects of Interaction 2014 Conference*, pages 149–157. The Euler International Mathematical Institute, 2014.

[20] Raluca Budiu, Christiaan Royer, and Peter Pirolli. Modeling information scent: A comparison of lsa, pmi and glsa similarity measures on common tests and corpora. In *Large scale semantic access to content (text, image, video, and sound)*, pages 314–332, 2007.

[21] Jonathan Chang, Sean Gerrish, Chong Wang, Jordan L Boyd-Graber, and David M Blei. Reading tea leaves: How humans interpret topic models. In *Advances in neural information processing systems*, pages 288–296, 2009.

[22] Chong Wang, David Blei, and David Heckerman. Continuous time dynamic topic models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, UAI'08, pages 579–586, Arlington, Virginia, United States, 2008. AUAI Press.