Procesamiento de Texto

Juan Diego Estrada Pérez
jestra15@eafit.edu.co
Andrés Franco Zapata
afranco8@eafit.edu.co
Liceth Cristina Mosquera Galvis
lcmosquerg@eafit.edu.co
Alejandro Palacio Vásquez
apalac19@eafit.edu.co
Johan Steward Rios Naranjo
jriosna1@eafit.edu.co

Maestría en Ciencia de los Datos y Analítica, Universidad EAFIT, 2019

***Resumen*—En la actualidad existen muchas librerías en programación para procesamiento de texto. Lo que hacen estas librerías es facilitar las implementaciones para aplicaciones de minería de texto. Consisten en trabajar con datos no estructurados, como lo son los documentos, y aplicar un proceso para darles estructura y así poder realizar diferentes aproximaciones y metodologías que permitan cumplir con un objetivo en específico. A este proceso se le conoce como crear una bolsa de palabras o *bag of words* por su traducción al inglés.**

En este artículo se busca profundizar sobre la necesidad de crear *bag of words* en minería de texto, las implementaciones de la misma en un índice invertido y como usar este para generar un rank de documentos en dónde se encuentre una palabra (o varias) buscada. Además, se pretende mostrar un ejemplo de cómo se implementa y algunos resultados obtenidos usando el lenguaje de programación python. URL Github:

https://github.com/franco18/text-mining-applied-project

Palabras Clave: Procesamiento de texto, datos no estructurados, minería de texto, bag of words, índice invertido, rank, python

I. Introducción

El procesamiento de texto, minería de texto o simplemente analítica de texto es un conjunto de técnicas estadísticas computacionales, que nace como respuesta a las nuevas tendencias y permite analizar, de una manera estructurada, datos almacenados de manera no estructurada como lo son los textos.

Lo anterior no quiere decir que la manera de representar e interpretar esta información no se haya estudiado en el pasado, de hecho, los primeros estudios relacionados al tema se referencian en la década de 1950 [1] sólo que dichas tendencias obedecen a los cambios que han generado los avances tecnológicos, que han permitido capturar información estructurada y no estructurada de manera novedosa y masiva. Las nuevas bases de datos que se generan están compuestas por ambos tipos de datosm las cuales requieren el desarrollo de nuevas técnicas para ser analizados e interpretados.

Hoy en día, como derivado de la hiper conectividad, las empresas y las personas están cada vez más interesadas en generar contenido web, contenido que se materializa a través de redes sociales, mensajes de texto, blogs, entre otras y dicho interés ha generado una conexión natural por el campo y por estas técnicas, buscando extraer información que logre apalancar a quienes toman decisiones [**?**].

El objetivo general del procesamiento de texto es entonces lograr que la información no estructurada almacenada como texto se transforme en información que pueda ser analizable para tomar decisiones, para esto, existen varias etapas de trabajo que se aprovechan de técnicas estadísticas y computacionales robustas, estas son:

1. Recuperación de la información
2. Procesamiento de lenguajes naturales (NLP) por sus siglas en inglés
3. Extracción de la información
4. Minería de datos

Los primeros dos son el centro de este trabajo, profundizando en cómo se debe realizar la identificación de los documentos relevantes dada una búsqueda o consulta hecha por el usuario.

II. Marco Teórico

Citando el libro "Deep Text"de Tom Reamy, se pretende mostrar la importancia del porqué hacer analítica de texto. ."El análisis de texto puede ahorrarle decenas de millones de dólares, abrir nuevas dimensiones de inteligencia de clientes y comunicación y, de hecho, permitirle utilizar una pila gigante de lo que actualmente se considera en su mayoría cosas inútiles: texto no estructurado."[2].

Es por esta importancia que muchas personas hablan acerca de qué es la analítica de texto y cómo usarla. Sin embargo, no comienzan por el porqué es importante, y son los mismos usuarios los que deben encontrarle una utilidad y un sentido. Según el artículo [3] se puede definir la analítica de texto como una extensión de la minería de datos, cuyo propósito es encontrar patrones de texto en grandes fuentes no estructuradas de información.

La utilidad de la analítica de texto y el procesamiento del lenguaje natural a menudo se presentan como funciones de

ciencia de datos muy difíciles de aprender y usar y que sólo pueden ser entendidas y manejadas por científicos de datos entrenados en el tema. Sin embargo, los principios teóricos sobre los que se fundamenta la analítica de texto son fáciles de entender y en general, es lo que pasa con muchos problemas de analítica en ciencia de datos. Por ejemplo, un motor de análisis de texto debe dividir las oraciones y las frases antes de poder entrar a analizar cualquier cosa, dividiendo documentos de texto no estructurados en sus principales componentes. Este es el primer paso en casi todas las funciones de procesamiento del lenguaje natural, que incluye la recomendación de textos [4], la extracción de temáticas [5], entre otras.

En [6] se puede encontrar una revisión de diferentes técnicas utilizadas con ayuda de minería de texto, para diferentes campos de investigación y los problemas que pueden surgir al momento de implementar alguna de estas metodologías.

Es importante entender el concepto que existe detrás del uso de la metodología y el propósito para el cual se quiere usar. Es así como más adelante en este artículo se enfrentan dos enfoques para la creación de la bolsa de palabras que contienen información similar pero se utilizan para enfoques distintos. Lo que se busca lograr con eso es una mejora en la eficiencia de la tarea específica para la que se está diseñando. Una visión más detallada de la importancia de la aclaración del propósito viene dada por [7] donde se entiende la creación de la bolsa de palabras como un proceso complementario a la implementación de un modelo y no sólo como un prerrequisito para poder lograrlo.

El obejtivo, indiferente de la elección de la forma de generar la bolsa de palabras, reside en generar un índice invertido el cual es una estructura de datos que reune en un solo lugar, palbras, documentos, frecuencia de palabra en el documento, posición en el documento, y en general cualquier información que se considere relevante para el rank del documento [8]. Para este rank existen muchas formas de calcularlo, pero para este caso se usa el Okapi BM25, el cual es una función de ranking utilizada en recuperación de información para la asignación de relevancia a los documentos en un buscador, dicho de otra forma, es una función que nos permite ordenar por relevancia los documentos que contienen las palabras que el usuario ha introducido en la caja de búsqueda de un buscador [9].

Diferentes usos como el análisis de opiniones [10] o el análisis de sentimientos [11] son utilizados con gran medida para direccionar campañas publicitarias en las empresas, tomando como fuente de datos diferentes medios como las redes sociales o los blogs de internet. Se busca reconocer patrones en palabras clave como los numerales (#) o el arroba (@) y captar las temáticas tendencia del momento así como los adjetivos subyacentes a dichas temáticas con el fin de generar una respuesta que mejore o corrija lo que se encontró en dicho análisis.

III. Descripción del Problema

Se tiene una gran cantidad de documentos con una gran cantidad de texto no estructurado y se debe desarrollar un motor de búsqueda, que es un programa que ante una consulta realizada por un usuario, regresa unos documentos ordenados de acuerdo con una ponderación o un sistema de recomendación más específicos, que ayude a los usuarios a obtener acceso a documentos relevantes.

En esta etapa del proyecto se requiere hacer la limpieza y preparación de datos, lo cual demanda un esfuerzo considerable de tiempo, además de ser este el proceso responsable del éxito o fracaso del modelo. Los archivos están en formato PDF, los cuales contienen imágenes, fórmulas e información que no se reconoce como texto, por lo que se transforman en archivos en formato txt, quedando con una gran cantidad de datos no estructurados que combinan información relevante con secuencia de caracteres sin sentido en el lenguaje humano. Mientras el lenguaje humano está diseñado para hacer la comunicación humana más eficiente, en el lenguaje de programación está diseñado para facilitar la comprensión informática, por lo que, en Procesamiento de Lenguaje Natural se omite conocimiento de sentido común y se dejan ambigüedades que el usuario debe entender [4].

El problema entonces reside en construir una bolsa de palabras con la información de los textos que a su vez es un insumo para construir un índice invertido, el cual debe usarse para generar una clasificación de documentos. Para todo esto se necesita un preprocesamiento de la información, se debe pensar que información es relevante para incluir en el índice invertido y adicionalmente elegir un método que permita obtener la mejor clasificación posible para la obtención de documentos en base a una consulta.

Existe un último reto, el cual consiste en validar que los documentos obtenidos como consecuencia de la búsqueda sean documentos relevantes para la persona que realiza la consulta.

IV. Descripción de los Datasets

Como se mencionó en la sección anterior, se cuenta con dos tipos de fuente de datos, una que corresponde a los documentos (arhivos PDF) y otra que corresponde a la metadata de esos documentos (archivo XML).

IV-A. Documentos

Los datos que acompañan el problema descrito son un conjunto de documentos o de datos no estructurados compuestos por palabras. Se cuenta con 980 documentos en diferentes formatos de texto como lo son: .txt, .pdf y .dc, dichos documentos pertenecen al mismo dominio de interés, por ejemplo: ingeniería, medicina, o astronomía, entre muchos otros campos de conocimiento o aplicación.

No obstante, se realizó un análisis descriptivo de la información para conocer en detalle con qué se estaba trabajando. Se cuenta con alrededor de 980 documentos que, en total, tienen alrededor de trece millones de palabras. La siguiente tabla expone la cantidad de palabras que tienen los documentos sin aplicarles ninguna depuración:

Min	685
Max	311,961
Mean	12,900
Median	10,286

Se encontró que el menor número de palabras que puede tener un documento son 685, mientras que, el número máximo

de palabras que puede tener un documento ronda las 312,000. En promedio, los documentos tienen alrededor de trece mil palabras.

IV-B. Metadata

IV-B1. Descripción: Para entender y analizar la meta data disponible, se realizaron dos ejercicios. El primero fue generar una nube de palabras, en donde se resaltan las mas relevantes del Corpus o conjunto de documentos, teniendo en cuenta solamente la información de los títulos y temas de los artículos. En el segundo ejercicio, tratando de clasificar un poco los artículos y basados también en esta misma información se aplica el modelo Latent Dirichlet Allocation (LDA) [12] y como resultado se generaron 10 tópicos y cada documento clasifica dentro alguno de estos.

IV-B2. Proceso: Se logra procesar sin problema la información gracias a la librería $xmltodict$ la cual transforma un documento en formato XML a un diccionario, una estructura de datos que proporciona operaciones de gran utilidad sobre sus datos. Esta, permitió extraer el insumo para generar la nube de palabras del Corpus y el bag of words para la categorización de documentos mediante tópicos.

IV-B3. Nube de Palabras: Una vez se tiene la información de títulos y temas a nivel de documento, se realiza el mismo pre procesamiento de los datos para normalizar las palabras como si fuera un bag of words, y se concatenan para crear un único documento. Éste documento es el insumo para crear la nube de palabras utilizando una librería llamada $wordcloud$, que indica las palabras mas relevantes en el documento.

IV-B4. Categorización de Documentos: Basándonos en el artículo Topic Modeling and Latent Dirichlet Allocation (LDA) in Python [5] donde se desarrolla un modelo de tópicos que permite clasificar cada documento según su conjunto de palabras dentro de 10 categorías, se extrajeron los título y temas de los documentos, se pre-procesan y se generó una matriz dispersa donde las columnas (individuos) son los documentos y las palabras (observaciones) son las filas. Para cada posición i, j de la matriz se almacena la métrica TF-IDF, para la palabra en la posición i y documento en j. Luego con esta matriz se generan los tópicos correspondientes.

En la siguiente imagen se ilustra como para el documento en la posición 900 clasifica en el tópico 0 con un scoring del 0.84 el cual es dado por el conjunto de palabras de su metadata, que ocurren en éste tópico y su peso relativo:

Toda la preparación y procesamiento de información para entender los temas y categorías de los artículos, se puede encontrar el notebook llamado MetaDataBagOfWords.

V. Arquitectura

Para esta etapa inicial se piensa en una arquitectura modular que permita realizar el proceso de ingesta de datos, su preparación, la ejecución y ajuste de dos modelos [13]: el primero que es el de categorización de documentos utilizado sobre la meta data y el segundo una aproximación del modelo de búsqueda. Por último, se tendrá un modulo de despliegue en el cual se pueden visualizar los datos procesados, para este caso los documentos ordenados por relevancia y las categorías o tópicos de los documentos.

Performance evaluation by classifying sample document using LDA TF-IDF model.

```
In [33]: for index, score in sorted(lda_model_tfidf[bow_corpus[900]], key=lambda tup: -1*tup[1]):
             print("\nScore: {}\t \nTopic: {}".format(score, lda_model_tfidf.print_topic(index, 10)))

Score: 0.8499809503555298
Topic: 0.061*"comput" + 0.059*"geometri" + 0.057*"quantum" + 0.056*"algorithm" + 0.047*"data" + 0.046*"structur" + 0.044*"mathemat" + 0.031*"algebra" + 0.029*"cluster" + 0.029*"discret"

Score: 0.016673041507601738
Topic: 0.094*"complex" + 0.056*"mathemat" + 0.055*"comput" + 0.055*"probabl" + 0.044*"secur" + 0.040*"cryptographi" + 0.029*"theori" + 0.029*"discret" + 0.029*"problem" + 0.029*"combinator"

Score: 0.016669208188388462
Topic: 0.074*"control" + 0.067*"optim" + 0.046*"system" + 0.038*"inform" + 0.036*"theori" + 0.035*"physic" + 0.030*"mathemat" + 0.029*"bound" + 0.026*"societi" + 0.023*"commun"

Score: 0.016668755561113358
Topic: 0.043*"cod" + 0.039*"inform" + 0.037*"artifici" + 0.037*"intellig" + 0.030*"detect" + 0.027*"base" + 0.024*"network" + 0.024*"gener" + 0.024*"recognit" + 0.023*"financ"

Score: 0.016668524593114853
Topic: 0.087*"inform" + 0.082*"theori" + 0.066*"channel" + 0.039*"secondari" + 0.039*"primari" + 0.036*"system" + 0.034*"physic" + 0.033*"game" + 0.031*"function" + 0.029*"cod"

Score: 0.016668450087308884
Topic: 0.069*"network" + 0.040*"artifici" + 0.040*"intellig" + 0.039*"inform" + 0.038*"architectur" + 0.038*"internet" + 0.033*"social" + 0.033*"learn" + 0.032*"machin" + 0.030*"physic"

Score: 0.016668079420924187
Topic: 0.111*"logic" + 0.097*"learn" + 0.088*"machin" + 0.051*"statist" + 0.035*"mathemat" + 0.034*"program" + 0.027*"comput" + 0.023*"analysi" + 0.022*"numer" + 0.020*"model"

Score: 0.016667750904595852
Topic: 0.066*"recognit" + 0.065*"pattern" + 0.064*"vision" + 0.062*"automata" + 0.056*"game" + 0.053*"languag" + 0.050*"theori" + 0.047*"logic" + 0.045*"formal" + 0.024*"algorithm"

Score: 0.016667632380608837
Topic: 0.082*"algorithm" + 0.073*"data" + 0.071*"structur" + 0.033*"theori" + 0.029*"time" + 0.028*"inform" + 0.027*"method" + 0.026*"group" + 0.025*"comput" + 0.023*"statist"

Score: 0.016667623072862625
Topic: 0.076*"mathemat" + 0.064*"discret" + 0.064*"combinator" + 0.044*"graph" + 0.043*"analysi" + 0.039*"numer" + 0.037*"softwar" + 0.032*"engin" + 0.031*"theori" + 0.026*"game"
```

Figura 1. Categorías de documentos

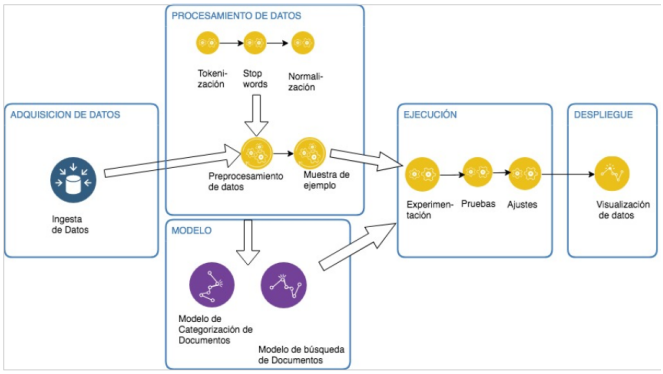

Figura 2. Arquitectura de la solución

Para cada proceso de la arquitectura se hará una pequeña descripción.

V-A. Adquisición de datos

Este componente de la arquitectura es importante ya que para la minería de datos se necesita un gran volumen de datos desde diferentes fuentes, tales como un sistema de archivos, Data-Warehouse, o datos obtenidos por medio dispositivos de IoT. Es aquí entonces donde serán implementadas las diferentes integraciones con dichas fuentes de datos, la cual para esta fase inicial del proyecto la única integración necesaria será al sistema de archivos de un sistema operativo Linux, que se logra fácilmente utilizando una funcionalidad llamada open de una librería estándar de Python para cargar los archivos en memoria como objetos.

V-B. Procesamiento de datos

Es el lugar donde los datos son reenviados, para unos pasos de procesamiento y preparar los datos para la ejecución de los modelos. Estos pasos incluyen: encoding, remover caracteres extraños, stop-words, tokenizar, normalizar las palabras con diferentes técnicas y finalmente la transformación de los datos en una estructura que sea fácil de manipular por los modelos.

Se aplica el siguiente proceso para obtener las palabras que aportan valor a los documentos: Inicialmente, se eliminan los elementos que no sean alfanuméricos, luego se segmenta el texto por espacios para la obtención las palabras, eso es conocido como tokenización. Después de esto se aplica el stemming, que toma la base común como la raíz de la palabra sin tener en cuenta el afijo derivativo y también se aplica lematización, que vuelve las palabras a la base o en forma de diccionario y sólo remueve finales inflexibles. Posterior a esto se eliminan las palabras que producen ruido o son irrelevantes como artículos, conjunciones, preposiciones, también conocidas como *stop-words*, se eliminan los caracteres especiales, las palabras coincidentes y las palabras de un solo dígito. Por último, las palabras obtenidas se filtran en un diccionario, en este caso de inglés.

Finalmente, con las palabras obtenidas después del proceso de limpieza tanto en los archivos de texto como en la meta data (que podría ser utilizada más delante de otra manera) se crea una matriz que tiene las palabras relacionadas con los documentos en los que aparecen, llamada "Bag of Words" (BoW). El valor en cada elemento de la matriz puede ser desde un booleano que indique si la palabra existe o no en el documento, un número que indique la frecuencia con la que aparece la palabra en el documento, hasta el resultado de una fórmula que ayude a ponderar mejor la relevancia de aquella palabra y el documento, entre otros.

La forma de trabajar con la bolsa de palabras es a través de un índice invertido, el cual es simplementa una forma distinta de representar la matriz previamente dicha. El problema del índice invertido es que todo se almacena en memoria, lo que significa que entre más datos existan, más recursos consume y más difícil y demorado resulta el procesar la información. Una alternativa a esto es el uso de posting, el cual es una metodología que permite ganar velocidad en el tiempo de indexación. Para este trabajo no se requiere el uso de la metodología dado que la cantidad de información a procesar no representa un gran problema para los computadores actuales. Más sobre el posting puede ser consultado en el libro Ïntroduction to information retrieval"[14].

V-C. Modelo

Es la porción de la arquitectura donde los algoritmos son seleccionados y adaptados para abordar el problema que será examinado en la fase de ejecución.

La idea es tomar una consulta realizada por el usuario y devolver en orden de coincidencia unos documentos, para esto la consulta debe pasar por el procesamiento de los textos y mediante una función de similitud, Okapi BM25, ubicar los documentos que cumplen con la consulta.

V-D. Ejecución

Es el lugar donde se cargan los datos, se corren los modelos y se prueban, con el fin de ajustarlos, para mejorar resultados obtenidos. Para esta parte se decide utilizar un wrapper de python con c++, en el cual se utiliza una librería llamada Meta en c++ y en python metapy. Meta es una librería que lleva desarrollada por mucho tiempo en la cual se ha validado que los métodos utilizados por la misma están correctamente implementados y retornan los documentos precisos. Es por esta razón que se utiliza la libreria metapy de python como una especie de validador de los métodos utilizados en este trabajo.

Como medidas de validación se tiene precision y recall.

V-E. Despliegue

La salida de los modelos es similar a cualquier salida de una aplicación de software, y puede ser persistida en alguna base de datos, archivo o Tableros. Para este caso se retorna, dado un query, los documentos mas relevantes, y dado un documento a cual de las 10 categorías pertenece.

VI. Modelos

Dentro de los métodos para recuperar información de los textos, existen varios tipos de modelos. Por el momento, los modelos que están implementados para la recuperación de información son:

- Term Frequency (TF): Es la técnica más simple para reconocer la relevancia de un término dentro de un texto. Básicamente realiza el conteo de la palabra en el texto y mientras más grande sea este número más relevante es
- Relative Term Frequency (RTF): Está técnica vuelve relativo al número de palabras total el conteo anterior, representando entonces cuánto porcentaje del texto está explicado por esa palabra
- T-Term Frequency (T-TF): Para eliminar riesgos de modelo cuando se realiza el conteo lineal de la frecuencia de los términos, se propone trabajar con una transformación del conteo: T-TF = $1 + log(x)$ de esta manera no se benefician aquellas palabras que aparecen muchas veces en un documento, pues no necesariamente son más relevantes que las demás
- Inverse Document Frequency (IDF): Este modelo está basado en el principio de que mientras menor sea la frecuencia de la palabra en el documento, más relevante y más información puede tener IDF = $log(totaldocumentos/documentosdondeexistepalabra)$
- TF-IDF: Al tener los modelos ya cuantificados, tanto el TF (recomendable trabajar con la transformación) y el IDF, la multiplicación de ambos entrega información valiosa de cara a la similaridad de la búsqueda o query con el documento.
- Okapi BM25: $\sum_{i=1}^{n} IDF(q_i).\frac{f(q_i,D).(k_1+1)}{f(q_i,D).k_1.(1-b+b.\frac{\|D\|}{avgdl})}$

donde $f(q_i, D)$ es la frecuencia de aparición en el documento D de los términos que aparecen en la consulta Q, $\|D\|$ es la longitud del documento (en número de palabras), $avgdl$ es la longitud media de los documentos en la colección sobre la cual estamos realizando la búsqueda, k_1 y b son parámetros que permiten ajustar la función a las características concretas de la colección con la que estamos trabajando. Aunque estos

parámetros suelen depender de las características concretas de cada colección normalmente se asignan los valores $k_1 = 2{,}0$ ó $k_1 = 1{,}2$ y $b = 0{,}75$, los cuales se han establecido a partir de los experimentos que durante años se han realizado en las conferencias TREC.

Y para medida de validación de la precisión se tiene:

- Precision: $\frac{T_P}{T_P + F_P}$
- Recall: $\frac{T_P}{T_P + F_N}$

VII. Evaluación y Resultados

Como se planteó en la descripción del problema, existen tres objetivos que se quieren cumplir. Búsqueda de documentos según palabras, agregrar documentos nuevos y hacer recomendaciones de documentos. En muchos artículos se encuentra una creación directa del bag of words con el objetivo de aplicar modelos de clasificación para documentos, estos modelos, basados en vectores, sugieren una implementación matricial del bag of words, el cual es un acercamiento común y altamente implementado en librerias ya existentes de los lenguajes de programación. Sin embargo, siendo conscientes de la importancia de la representación matricial para entrenar modelos, no se busca dejar de lado los dos primeros objetivos anteriormente descritos y para los cuales una representación matricial funciona pero no es la forma más efectiva si se entienden los problemas como complementarios y no como consecutivos. En pocas palabras, lo que se busca es encontrar una estructura de datos que sirviera para cada objetivo y no para los tres objetivos al mismo tiempo; esto basado en el hecho que las tres operaciones jamás se ejecutarán al mismo tiempo y que las tres tienen metodologías diferentes para ser realizadas. Una discusión acerca del tema puede ser encontrada en el siguiente link https://stats.stackexchange.com/questions/31060/bag-of-words-vs-vector-space-model.

Ambas aproximaciones a la bolsa de palabras pueden encontrarse en link a github de la sección IX.

VII-A. Utilizando diccionarios

Para los dos primeros objetivos se elige una representación en forma de diccionario en el lenguaje python por su alta eficiencia al momento de la búsqueda directa de palabras por documento y por la facilidad de agregar nuevas palabras al bag of words ya existente. Mientras en una representación matricial se debe recorrer la lista para cada documento y verificar si estas palabras existen, en un diccionario basta solo con preguntar si las palabras pertenecen directamente. Adicionalmente, por su estructura de clave:valor, es fácil asignarle significados o propiedades a los elementos, por lo cual se evita crear nuevas listas o ampliar las existentes con el fin de tener más información para las palabras. En el artículo [7] se hacen algunas aclaraciones sobre el significado de un bag of words como estructura y la necesidad de una estructura que permita entrenar un modelo de clasificación.

VII-B. Representación vectorial del bag of words

Se realiza la preparación de los datos siguiendo pasos ya descritos, y se genera una matriz dispersa donde las columnas (individuos) son los documentos y las palabras (observaciones) son las filas. Y para cada posición i, j de la matriz se almacena la frecuencia de la palabra para cada documento.

Este procedimiento utiliza la librería sklearn y se basa en el articulo [15]. En donde claramente se ilustra, dado unas palabras pre procesadas; utilizando el objeto CountVectorizer y su funcion fitTransform se logra construir un vocabulario para todos los documentos y la frecuencia de cada palabra por documento, todo esto representado en una matriz. El notebook asociado a este procedimiento se llama BagOfWordsSecondApproach.

Algunos de los resultados obtenidos se observan gráficamente a continuación.

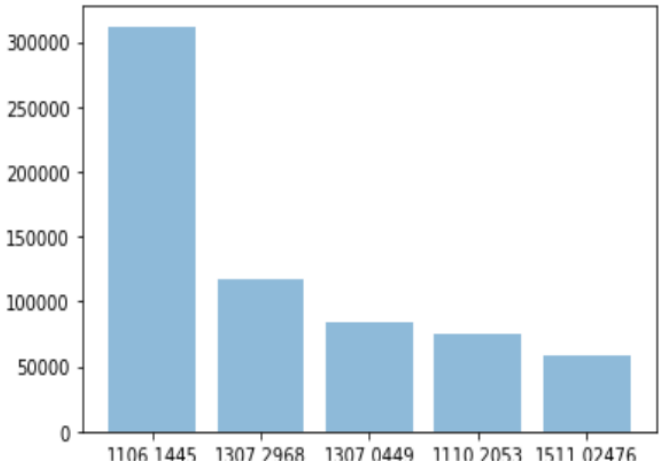

Figura 3. Documento con mayor número de palabras

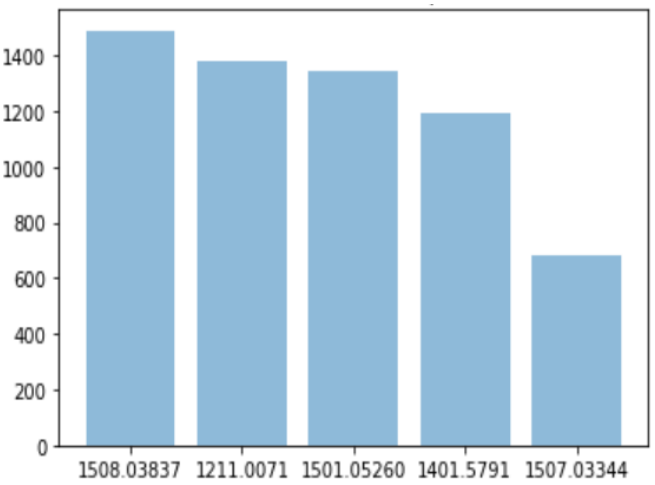

Figura 4. Documento con menor número de palabras

Al realizar una gráfica de barras, complementada con un box plot, se observa que existen documentos que están muy desviados en cantidad de palabras respecto a la normalidad de la colección, sin embargo, estos documentos no deben ser eliminados simplemente por ser outlieres puesto que pueden

contener información relevante de cara a la consulta que realice el usuario.

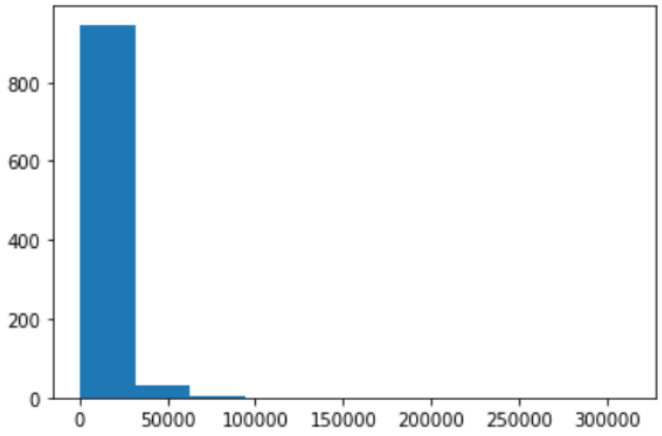

Figura 5. Cantidad de palabras por documento

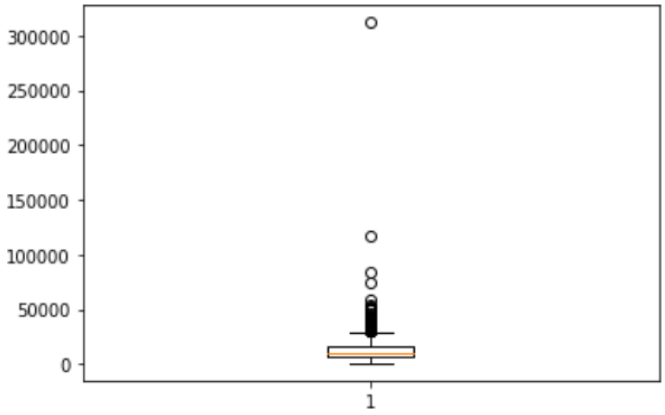

Figura 6. Desviaciones y outliers

VIII. Conclusiones y Trabajo Futuro

[h!] Este trabajo presenta un entendimiento general de qué es el procesamiento de texto y la relevancia que ha adquirido derivado de la manifestación de las mega tendencias tecnológicas. Las capacidades que se han generado y que permiten capturar de diversas fuentes de información grandes volúmenes de información no estructurada, presentan un desafío para que, quienes toman decisiones, logren tratar dichas bases de datos y extraer la mayor cantidad posible de información valiosa.

De cara al problema específico presentando en la Sección III, se abordó la globalidad, entendiendo las etapas que componen el mismo relacionadas con la búsqueda y recuperación de documentos. Se analizaron los datos y sus metadatos respectivos, analizando qué tipo de información se está tratando, a qué dominio de interés pertenecen, de cuántas palabras está compuesto y se categorizaron los documentos por tópicos.

Dichos documentos fueron procesados para obtener la información más relevante eliminando los elementos que no eran alfanuméricos, segmentando el texto por medio de una Tokenización, aplicando el stemming, eliminando las *stop-words* y filtrando por medio de un diccionario de inglés.

Se construyó el Bag of Words por dos caminos, el primero a través de diccionarios por su alta eficiencia al momento de la búsqueda directa de palabras por documento y por la facilidad de agregar nuevas palabras, el segundo por medio de una representación vectorial, que tiene como fin volver homogéneos los documentos de cara a las métricas o modelos de recuperación que buscan hallar documentos similares.

Para los resultados se realizan 100 búsquedas, se calcula el precission y recall como se explica en la sección VI, obteniendo los siguientes resultados para las primeras 10 palabras.

Palabras	Precision	Recall
biology	1	1
activity	1	0.8
machine learning	1	0.8
machine	1	1
math	1	0.7
magazine	1	0.9
mahalanobis distance	1	1
kruskal algorithm	1	0.8
mathematician	1	0.9
norm	1	0.8

Figura 7. Tabla de Precision y Recall

Para dar un poco de contexto de como fueron calculado los resultados, se proporcionan las siguientes definiciones:

Se define falso negativo como un resultado generado por el meta y omitido por el calculado en este trabajo.

Se define un falso positivo como el resultado generado en este trabajo y que no se encuentre dentro del dominio del meta.

Luego se procede a realizar la validación tomando los primeros 20 artículos retornados por el meta y compararlos (sin tener en cuenta el orden) con los 20 retornados por implementación del Okapi BM25. Como se puede observar la precisión siempre dará 1 dado que se toma a meta como la lista de documentos ideal y por tanto nunca se encontrará un falso positivo.

A continuación, se utiliza el Okapi BM25 y se valida con lo arrojado por metapy obteniendo resultados buenos para las 85 o mas comparaciones entre meta y este trabajo. Más del $80\,\%$ de los resultados arroja un recall de 0.7 o más.

Como trabajo a futuro se va a implementar una validación teniendo en cuenta el orden de los artículos.

IX. Github

Este artículo, los códigos y la documentación pueden ser encontrados en el siguiente enlace https://github.com/franco18/text-mining-applied-project.

Referencias

[1] A. Turing, “Mind a quarterly review of psychology and philosophy,” 1950.

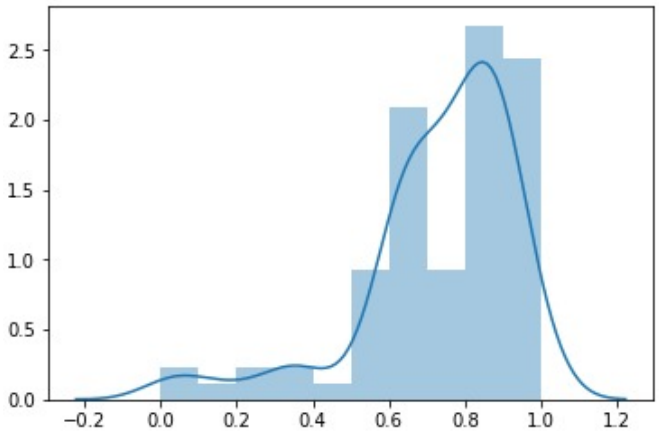

Figura 8. Frecuencia de Recall

[2] T. Reamy, *Deep text*. 2016.
[3] T. R. Antonio Moreno, "Text analytics: the convergence of big data and artificial intelligence," 2016.
[4] S. M. ChengXiang Zhai, *Text Data Managment and Analysis*. 2016.
[5] S. Li, "Topic modeling and latent dirichlet allocation (lda) in python," May 2018.
[6] S. A. Ramzan Talib, Muhammad Kashif Hanif† and F. Fatima, "Text mining: Techniques, applications and issues," 2016.
[7] J. H. M. Daniel Jurafsky, *Speech and Language Processing*. 2018.
[8] S. B. Ajit Kumar Mahapatra, "Text mining: An introduction to theory and some applications," 2011.
[9] H. Z. Stephen Robertson, *The Probabilistic Relevance Framework: BM25 and Beyond*. Editorial Board, 2009.
[10] B. Liu, "Opinion mining," 2010.
[11] J. G. Carlos Henríquez Miranda, "Una revisión sobre el análisis de sentimientos en español," 2015.
[12] Wikipedia, "Latent dirichlet allocation," April 2019.
[13] C. E. Sapp, *Preparing and Architecting for MachineLearning*. 2017.
[14] H. S. Christopher D. Manning, Prabhakar Raghavan, "Introduction to information retrieval," 2008.
[15] S. Li, "Multi-class text classification with scikit-learn," February 2019.