Reporte Técnico: Proyecto integrador procesamiento de texto

Diana Catalina Velásquez 201919006228;
Jairo Loaiza 201919005228; Juan Sevastián Moreno 201917508228;
Julián Castelblanco 201910040228; Mariana Arboleda 201910015114

Abril 12 2019

1 Resumen de logros

Teniendo en cuenta que el proyecto se está desarrollando basándose en la metodología CRISP-DM, los logros en avance de esta primera etapa se enmarcan en las primeras tres etapas de la metodología: Comprensión del negocio, Comprensión de los datos y **Preparación de los datos**.

Para las dos primeras fases se analizó el caso de estudio y se revisaron los datasets. Entendiendo que, si bien ya las bases de datos están dadas y, metodológicamente no estamos ejecutando la fase de *Recolección de datos iniciales*, es necesario comprender la información, los formatos en los que fue entregada y la cantidad de la que disponemos. Se obtuvo que en total se deberán procesar **980** archivos. En la tercera fase, se logró preparar la información para ser utilizada en las futuras aplicaciones, siguiendo algunos de los pasos de Indexación de textos (Text indexing): tokenización, eliminación de stopwords, stemming y lemmatization descritos en [6], [7], [8] y [9] y que se explicarán en detalle más adelante. Se obtuvo como resultado final, el análisis de frecuencias de una gran lista de palabras únicas, que se refieren (casi en su totalidad) a sustantivos, adjetivos o verbos, cuyos significados difieren unos de otros y aportan al entendimiento de los documentos a los que pertenecen, tal como se puede observar en el repositorio creado para el proyecto:Demo Jupyter Notebook.

2 Introducción y Marco teórico

Dado que el proyecto se define como Procesamiento de Texto, su realización estará enmarcada en las técnicas de Procesamiento de Lenguaje Natural (o NLP por sus siglas en inglés) que Cady [6] define como una colección de técnicas para trabajar con el lenguaje humano en actividades como encontrar documentos de texto que tratan temas similares.
Para lograrlo, lo primero que se debe hacer es preprocesar la información, por medio de una serie de pasos que se describen a continuación y cuyos nombres, por frecuencia de uso, se conocen en inglés:

2.1 Tokenization

El primer paso para la transformación del texto consiste en *tokenizar* o dividir los textos en cada una de las palabras que los componen, a partir de los separadores como los espacios en blanco, signos de puntuación y caracteres especiales; eliminar los caracteres especiales y transformar su escritura a letras minúsculas [8], [9]. Por lo general se ha considerado que cada palabra está compuesta mínimamente por tres caracteres alfanuméricos y que se encuentra delimitada por espacios en blanco [7], sin embargo, con esta definición se pierde suficiente información como para que sea necesaria la revisión de estas características para cada aplicación.
La lista que se genera, es la materia prima para los pasos siguientes, que pueden ser implementados en diferente orden, según la arquitectura que define cada usuario.

2.2 Stopwords removal

Los autores coinciden en que cada idioma posee palabras que no aportan información cuando se encuentran por sí solas, como las preposiciones y los artículos (en, entre, la, los) y que estas pueden ser eliminadas para mejorar el procesamiento del resto de la información. Por lo general se encuentran en listas a las que se acceden durante el procesamiento y, cada vez que se identifica una en el texto, se detiene el procesamiento y se elimina dicha palabra, de ahí su nombre *Stopword* [7].

2.3 Stemming y Lemmatization

Son algoritmos usados para reducir la cantidad de inflexiones y variaciones de las palabras que se refieren a un mismo significado. Estos métodos reducen las palabras o tokens, a su raíz, que dependiendo del algoritmo se refiere a un **lemma** que tiene en cuenta el contexto de la palabra para hallar la base apropiada o simplemente a la parte de la palabra que no varía según su inflexió [6].
Algunos de los stemmers más populares son:

- **Porter**: Es el algoritmo más usado y el que se considera menos estricto en cuanto a eliminación de palabras. Consiste en una serie de 5 pasos para detectar y eliminar los sufijos de cada palabra y mantener una raiz razonable [4].

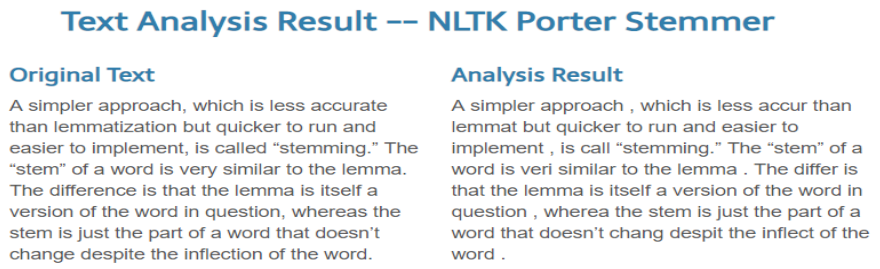
- **Snowball**:Es una versión mejorada del algoritmo de Porter, en el que se evidencian cambios como: menor frecuencia de cambio de la terminación y por i, mayor cantidad de sufijos que se eliminan como *ly*, unificación de algunos pasos de procesamiento, entre otros [2].

- **Lancaster**: Se considera uno de los algoritmos más agresivos en cuanto a selección de los stemms, itera a través de 120 reglas de eliminación o reemplazo de la última letra de la palabra o el stem remanente [3].

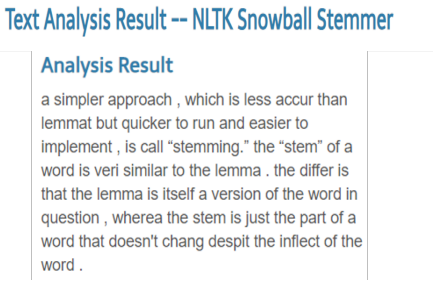
En la figura 1 se observa una comparación de los resultados obtenidos.

3 Problema

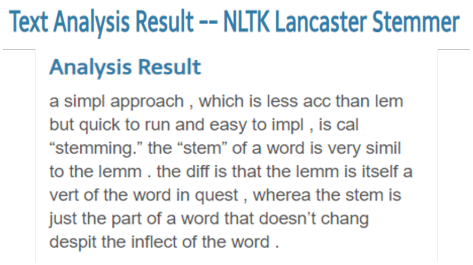
Se tiene un conjunto muy grande (big data) de documentos tipo texto (txt, html o pdf) y los metadatos sobre dichos documentos. Es necesario realizar un sistema para la ingesta, almacenamiento, indexación, búsqueda, recuperación, navegación y visualización de estos documentos. Estos documentos pertenecen al mismo dominio de interés, por ejemplo: ingeniería, medicina, o astronomía, entre muchos otros campos de conocimiento o aplicación. El tamaño del conjunto, que como se mencionó anteriormente, consta de 980 artículos, dificulta cada una de las actividades que se desean ejecutar. Más aún cuando se presentan como textos separados, sin ninguna estructura. Específicamente en este avance, se comprendió el problema como la necesidad de cargar los archivos, almacenarlos e indexarlos.



(a) Porter



(b) Snowball



(c) Lancaster

Figure 1: Comparación entre métodos de stemming usando un apartado del texto de [6]

Para esto, se tuvo en cuenta que, al contar tanto con los artículos en txt como en pdf, resulta más sencillo obtener la información directamente desde los archivos txt ya que pdf es un formato binario, complejo de tratar en python y que requiere procesamiento adicional para extraer el texto [5]. Los resultados obtenidos, siguen la problemática inicial y se prevé lograr el objetivo final.

4 Datasets

Se cuentan con tres (3) conjuntos de datos:

1. **papers-pdf.zip (980 papers):** Contiene una colección de artículos de carácter científico en ciencias de la computación, escritos en inglés. Se presentan en formato pdf y contienen información de los autores, título de la investigación, y abstract, así como el desarrollo del tema. Se con-

sidera información **no** estructurada, que además posee características adicionales como encabezados y diferentes tipografías que no aportan al procesamiento ni comprensión de los textos.

2. **papers-txt.zip (980 papers):** Contiene la misma colección de artículos de carácter científico del dataset anterior, transformados a formato txt. Esto facilita el procesamiento de las palabras, para su posterior tratamiento y uso. Sin embargo, se considera información **no** estructurada.

3. **papers-dc.xml:** contiene los metadatos de los artículos científicos en formato XML en estándar Dublin Core Metadata Initiative (DCMI) [1]. Lo que indica que se tiene información **semi** estructurada y etiquetada acerca de los artículos, tales como el título, creadores, temas de los que trata, descripción, número de paginas, cantidad de figuras, fecha de publicación y códigos de identificación.

5 Arquitectura

5.1 Arquitectura de infraestructura

Para el desarrollo de este proyecto se tiene la siguiente plataforma: -

- Un servidor Ubuntu Linux con:
  - 72 nucleos
  - 512 GB de RAM
  - 252 GB de Disco SSD
  - 2 Teslas 480
- Plataforma Jupyter para ejecutar código Python y MarkDown

5.2 Arquitectura del proyecto Python

Para esta aplicación se utilizaron las estructuras de datos que se detallan a continuación:

1. LISTA doc_id : Una Lista python, la cual contiene un Diccionario python.
   El Diccionario tiene:

   - Clave: doc_id : Un id que se le asigna a cada documento
   - Valor: doc_length : nro de caracteres del documento original

   Un ejemplo de esta lista es:

   $$doc\_id = [\{'doc\_id' : 1,' doc\_length' : 1424\}\ \{'doc\_id' : 2,' doc\_length' : 1740\}\ \{'doc\_id' : 3,' doc\_length' : 1522\}\ \{'doc\_id' : 4,' doc\_length' : 988\}\ \{'doc\_id' : 5,' doc\_length' : 4022\}]$$

2. LISTA freqDict_list : Una Lista python, la cual contiene un Diccionario python.
   El Diccionario tiene:

   - Clave: doc_id : id del doc
   - Valor: freq_dict : este es otro Diccionario, el cual contiene a su vez:
     - Clave: Una palabra
     - Valor: Frecuencia de la palabra (número de veces que esta aparece en el documento).

   La lista tiene tantos registros como documentos leídos. El diccionario de cada documento tiene tantas claves (palabras) como palabras diferentes tenga el documento (ya limpio). Un ejemplo de un registro de la lista; es decir, de la información de un documento es:

   freqDict_list=[{'doc_id': 1, 'freq_dict': {'entropi': 112, 'earli': 3, 'function': 13}}]

3. LISTA TF_scores : Una Lista python, la cual contiene un Diccionario python. El Diccionario tiene:

   - Clave: doc_id : id del doc
   - Valor: TF_score : este es otro diccionario, el cual contiene:
     - Clave: una palabra

- Valor: Frecuencia relativa de la palabra

La lista tiene tantos registros como documentos se hayan leído. El diccionario de cada documento tiene tantas claves (palabras) como palabras diferentes tenga el documento (ya limpio). Un ejemplo de un registro de la lista; es decir, de la información de un documento es:

TF_scores=[{'doc_id': 1, 'TF_score': {'entropi': 0.07865168539325842, 'earli': 0.002106741573033708}}]

4. LISTA IDF_scores : Una lista python, la cual contiene un diccionario python. El diccionario tiene:

   - Clave: doc_id : id del doc
   - Valor: IDF_score : este es otro Diccionario, el cual contiene:
     - Clave: una palabra
     - Valor: índice inverso de la palabra en el documento

   La lista tiene tantos registros como documentos se hayan leído. El diccionario de cada documento tiene tantas claves (palabras) como palabras diferentes tenga el documento (ya limpio). Un ejemplo de un registro de la lista; es decir, de la información de un documento es:

   IDF_scores=[{'doc_id': 1, 'IDF_score': {'entropi': 1.9972034434428638, 'earli': 1.5452183196998066}}]

5. LISTA TFIDF_scores : Una lista python, la cual contiene un diccionario python. El diccionario tiene:

   - Clave: doc_id : id del doc
   - Valor: TFIDF_score : este es otro diccionario, el cual contiene:
     - Clave: una palabra.
     - Valor: Este valor se calcula multiplicando el TF_Score * IDF_Score de cada palabra.

La lista tiene tantos registros como documentos se hayan leído El diccionario de cada documento tiene tantas claves(palabras) como palabras diferentes tenga el documento (ya limpio) Un ejemplo de un registro de la lista; es decir, de la información de un documento es:

TFIDF_scores=[{'doc_id': 1, 'TFIDF_score': {'entropi': 0.15708341690000052, 'earli': 0.0032553756735248735}}]

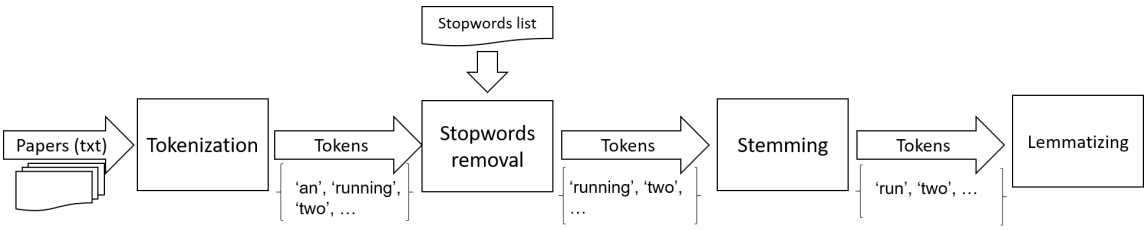


Figure 2: Arquitectura seguida

5.3 Arquitectura de los datos

Los datos del proyecto constan de 980 archivos tipo texto que se encuentran en un file system del servidor jupyter de la universidad EAFIT (hpcdis). De allí se leen dichos archivos y se procesan

6 Modelos

Como se puede ver en el procedimiento: Demo Jupyter Notebook. Para este primer avance se utilizaron los modelos disponibles en la librería nltk para python:

- PorterStemmer
- WordNetLemmatizer

7 Evaluación y Resultados

Se pasó de un dataset de 980 artículos científicos a un listado de 13900 palabras únicas con sus frecuencias de aparición en los textos, reducidas a sus raíces para obtener la menor cantidad de ambigüedades.

8 Trabajo Futuro y conclusiones

Ahora que se cuenta con el listado de palabras únicas que aportan significado en los textos, a futuro se deberá generar un vector numérico que represente las palabras y sobre el cual se podrán implementar los modelos estadísticos que permitirán obtener información de los datasets. Se deberán explorar métodos de indexación del texto, representación de documentos y sus metadatos, implementación de modelos de búsqueda, mecanismos de ranking o priorización de los resultados, entre otros elementos.

References

[1] Dcmi metadata terms.

[2] The english (porter2) stemming algorithm.

[3] Stemming and lemmatization in python.

[4] Stemming and lemmatization. standford.edu.

[5] Tim Arnold. Manipulating pdfs with python.

[6] F. Cady. *The Data Science Handbook*. John Wiley & Sons, Incorporated, 2017.

[7] W Croft, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*. 01 2009.

[8] Ronen Feldman and James Sanger. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, 2006.

[9] T. Jo. *Text Mining: Concepts, Implementation, and Big Data Challenge*. Studies in Big Data. Springer International Publishing, 2018.