Text Analyzer

Lo primero que hay que tener en cuenta es que tendríamos que extraer información y, por lo tanto, quedarnos solo aquello que nos interesa, es decir, solo aquellas palabras que han sido filtradas.

Supongamos entonces que, nuestras fuentes de datos de diverso ámbito, ya sean bases de datos o, ficheros, nos proporcionan de frases de diversa índole que tenemos que previamente tratar y guardar para nuestro posterior análisis.

1.1 Esquemas

Hay que tener claro el esquema que debemos seguir para este preprocesamiento (Figura 1.1), el cual no es más que teniendo una serie de fuentes de datos, extraer todo lo posible y después tratarlo para su posterior almacenamiento en una base de datos estructurada (ETL).

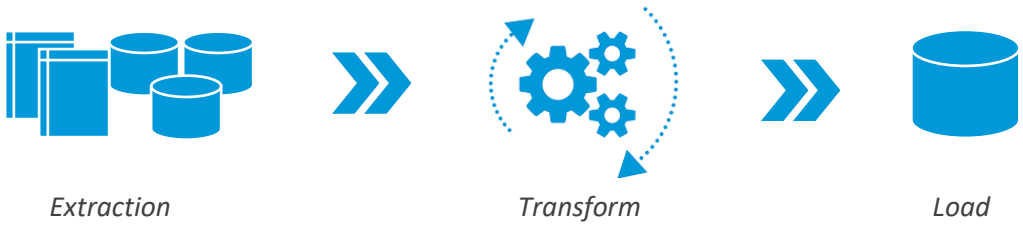

Figura 1 - Esquema de preprocesamiento (ETL)

La base de datos a utilizar será una base de datos SQL, la cual nos proporcionará la capacidad de guardar información sobre las palabras (Lemmas) extraídas, sus posibles categorías y, los análisis realizados.

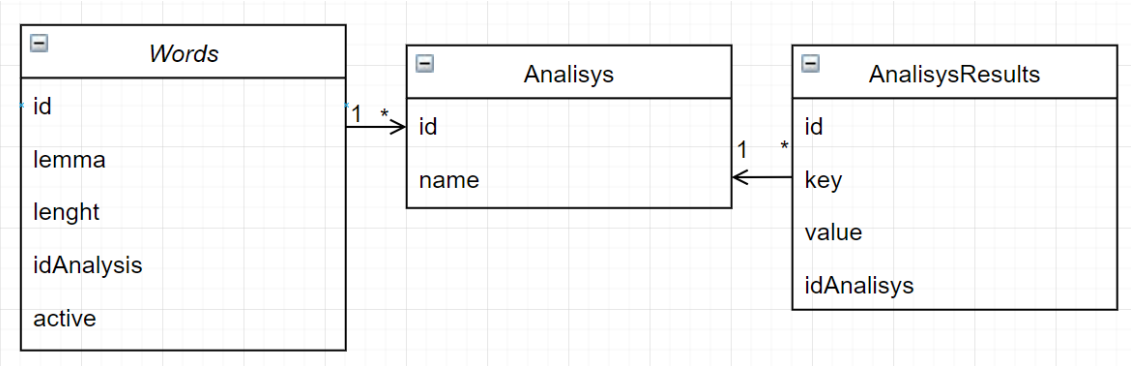

Figura 2 - Diagrama UML para la Base de Datos destino

La base de datos de destino, está contenida por una tabla Words, que guarda las palabras junto con su longitud y el análisis aplicado, además de un booleano active para saber si esa palabra sigue en posteriores procesos.

Una tabla Analisys, que contiene los tipos de análisis a las palabras, y que irá unida a una tabla AnalisysResults, que contendrá todos los resultados de los análisis, ya sean estadísticos o no, en formato (key,value).

Por ejemplo, podríamos extraer, una vez realizado el análisis, aquellas palabras que cumplan un criterio específico, como puede ser que quisiéramos ver la importancia de ciertas palabras y, por ello, querríamos saber todas estas palabras que aparezcan más de X veces (aunque realmente en este primer análisis se requieren saber todas), y para ello filtraríamos por aquellos resultados de una palabra que contenga un count (key) mayor a X (value).

1.2 Tratamiento

Para este tratamiento o transformación de datos en información útil, se propone utilizar Spark NLP, ya que es una biblioteca de procesamiento de lenguaje natural (NLP), integrado además con MLLib para el procesamiento de grandes cantidades de datos y, que podremos utilizar en Python para hacer una primera aproximación del modelo mediante scripts sencillos, y posteriormente una versión de producción en Java, cuyo lenguaje es considerablemente más rápido y, cuya versión gracias a la mayor rapidez podría ser utilizada con fines comerciales. Además, y como aspecto más importante, nos proporciona una gran escalabilidad y rapidez, acorde a las necesidades del proyecto y con numerosos métodos de transformación y análisis ya implementados, por lo que implementar nuevas fuentes y/o algoritmos será bastante más sencillo que con una librería común.

Es en este momento, donde toda la información de las fuentes será transformada, mediante los diferentes métodos de Spark NLP. En concreto, y como primer análisis:

- Tokenizer. Que nos troceará las frases en palabras.
- Normalizer. Que limpiará de palabras información no útil, es decir, aquellas palabras que no aportan nada y que, para nuestro caso, cuyo primer análisis es la extracción de Lemmas, facilitar al siguiente método la extracción de los mismos. Eliminaremos pro lo tanto aquellas palabras como: a, un, por para, … O palabras en inglés como: a, about, after, …
- Lemmatizer. Que recuperará de las palabras que nos queden aquellos Lemmas con el objetivo de devolver una palabra base del diccionario.

Todo ello debería de ir en un pipeline para que, si quisiéramos ejecutar otro método, no perdamos el tiempo de ejecución anterior (siempre y cuando nuestras fuentes no hayan cambiado).

1.3 Escalado del script

Será necesario normalizar nuestras fuentes de datos, para evitar posibles problemas de conexión con las mismas.

Tras esta normalización, deberíamos de definir que tipos de análisis se requieren a lo largo de la vida del script (o al menos hasta un punto donde se imagine), pues es posible que se requieran análisis no implementados en Spark NLP, y cuyo desarrollo necesitaría de un tiempo considerable aparte, pues se tratarían de algoritmos MapReduce, para utilizar la máxima potencia de Spark.

1.4 Desarrollo

Suponiendo que somos 3 personas a tiempo completo, pero sin conocminetos sobre este tipo de sistemas y sobre la libería Spark NLP, y empezando el proyecto el día 21/01/2021, utilizando metodología SCRUM, dividida en al menos 2 sprints de 2 semanas cada uno, tendríamos una versión inicial del proyecto el día 16/01/2021, según el siguiente diagrama de Gantt:

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
Toma de conocimiento del equipo	3 días	lun 21/06/21	mié 23/06/21	
Implementación de scripts sencillos sobre python en SPARK NLP para entender los procesos de forma sencilla	3 días	jue 24/06/21	lun 28/06/21	1
Desarrollo de la base de datos destino	2 días	mar 29/06/21	mié 30/06/21	2
Desarrollo de la autenticación con las fuentes	2 días	mar 29/06/21	mié 30/06/21	2
Instalacion de las dependencias del proyecto	1 día	mar 29/06/21	mar 29/06/21	2
Desarrollo de un primer análisis de pruebas	2 días	jue 01/07/21	vie 02/07/21	3;4;5
Fin del primer sprint	0 días	vie 02/07/21	vie 02/07/21	
Desarrollo de algoritmos para análisis no contemplados en Spark NLP	5 días	lun 05/07/21	vie 09/07/21	7
Prueba y correcciones de los algoritmos de análisis	2 días	lun 12/07/21	mar 13/07/21	8
Pruebas y correcciones de robustez del sistema	2 días	mié 14/07/21	jue 15/07/21	9
Entrega del sistema en version 0.1	0 días	vie 16/07/21	vie 16/07/21	10

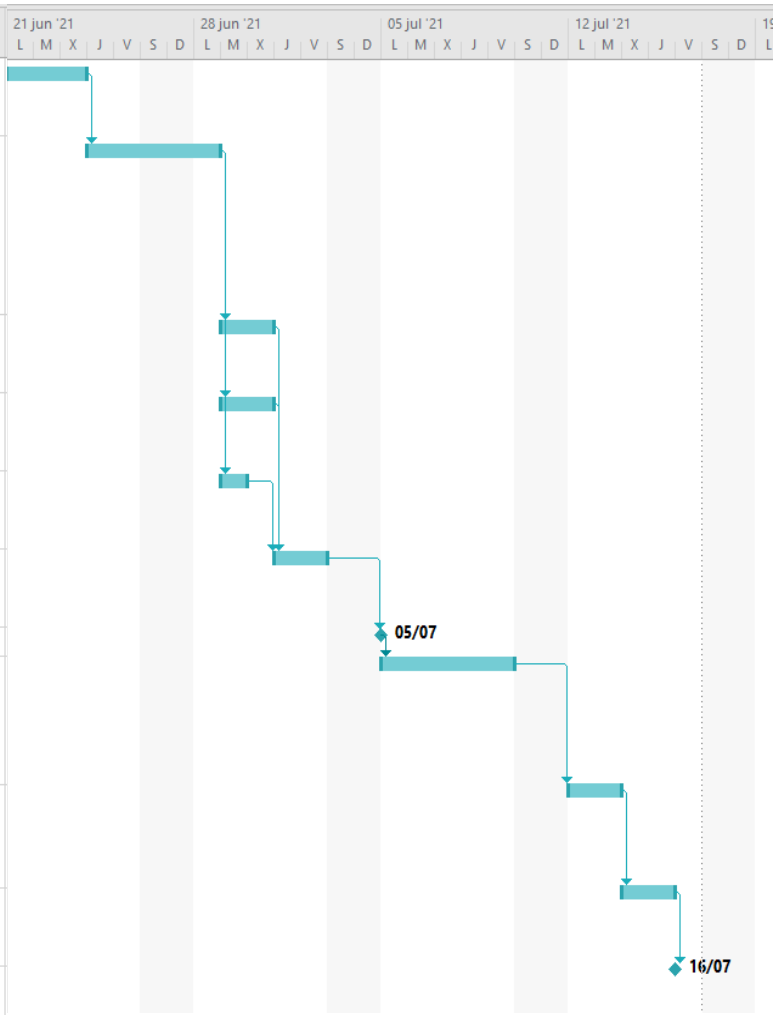

Figura 3 - Diagrama de Gantt