



**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN INGENIERÍA INFORMÁTICA E INGENIERÍA DEL  
SOFTWARE**

**Curso Académico 2020/2021**

**Trabajo Fin de Grado**

**EXPERIMENTOS CON SPARK NLP EN EL DOMINIO DE  
LAS REDES SOCIALES**

**Autor:** MARTÍNEZ SÁNCHEZ, NOELIA

**Directores:** SERRANO HIDALGO, JUAN MANUEL

Resumen

Palabras clave

## Índice

## 1. Introducción

El *Big Data* (cuya traducción es “macrodatos”) es un término que hace referencia a los conjuntos de datos que crecen tanto que alcanzan un tamaño y una complejidad que hacen inviable su tratamiento a nivel de almacenamiento, gestión, búsqueda, análisis o visualización en un tiempo razonable. Para hacer esto posible, se han desarrollado diferentes tecnologías que facilitan el escalado, distribución, diversidad y gestión veloz de los datos. Por ello, se suele decir que el *Big Data* tiene tres características: volumen, velocidad y variedad (las tres V's) [1].

Una de las tecnologías que lidera el avance en *Big Data* es MapReduce. MapReduce es un modelo de Programación Paralela inspirado en las funciones de Programación Funcional *Map* y *Reduce*. El usuario programa una función *Map* que procesa un par clave-valor y genera un conjunto de pares clave-valor que después se combinan con la función *Reduce*, también definida por el usuario. Estos programas se paralelizan y ejecutan automáticamente en un clúster de máquinas distribuidas [2].

En este contexto, surge Apache Spark, un motor de procesamiento distribuido, cuyo objetivo es mantener los beneficios e ideas de MapReduce a la vez que mejorar su rendimiento. Además de mejorar la latencia de los programas, Spark incluye diferentes módulos que ofrecen servicios como análisis interactivo de datos SQL con Spark SQL, creación de pipelines de Aprendizaje Automático con MLlib o procesamiento en tiempo real con Spark Streaming [3].

El *Big Data* ha tenido hasta ahora múltiples aplicaciones en todo tipo de sectores, siendo destacables las relativas a la provisión de información útil para la toma de decisiones de cualquier empresa de cualquier sector, por ejemplo, analizando datos de clientes que puedan revelar sus comportamientos o preferencias.

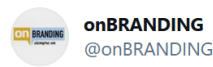
Como no puede ser de otra manera, Twitter, la popular red social de microblogueo, es una de las principales fuentes de datos de las empresas a la hora de analizar las conductas y tendencias que siguen sus clientes potenciales. Tanto es así, que Twitter proporciona una API para que, tanto empresas como individuos, puedan extraer los tuits y la información que deseen (más información en el Anexo X).

Pero no solo basta con el *Big Data* para obtener análisis valiosos, sino que se necesita otra tecnología que aporte la lógica necesaria para producir dichos resultados: la Inteligencia Artificial. Por esta razón, *Big Data* e Inteligencia Artificial son dos términos que suelen ir de la mano, y se combinan para crear productos de gran valor para empresas y usuarios.

La Inteligencia Artificial, y en concreto, el Procesamiento de Lenguaje Natural (PLN), es el área en el que se investigan y desarrollan mecanismos para lograr una comunicación entre los humanos y la computadora a través del lenguaje natural [4]. En estos últimos años, han aparecido algoritmos avanzados que han servido, entre otras cosas, para la Traducción Automática, la Recuperación de Información, el Reconocimiento de Voz o el Modelado de Temas.

Nosotros vamos a aplicar algoritmos del último tipo, Modelado de Temas (o *Topic Modelling*, término en inglés habitual), a un conjunto de tuits para llevar a cabo diferentes experimentos, haciendo uso de Spark y de su librería Spark NLP, especializada en Procesamiento de Lenguaje Natural.

## 2. Objetivos



España sufre 40.000 ciberataques diarios:  
administraciones y pymes, entre los objetivos más  
vulnerables [bit.ly/35CH6aG](https://bit.ly/35CH6aG)  
#ciberseguridad  
#Tendencias  
#Actualidad

Figura 1. Ejemplo de tuit haciendo uso de diferentes hashtags

Los *hashtag* de Twitter, al final, son un mecanismo con el que los usuarios etiquetan la temática (el *topic*) de su tuit (Anexo X). El problema que se plantea consiste en realizar experimentos con la técnica ALD de *Topic Modelling* (Anexo X) y estudiar en qué medida los resultados obtenidos a partir de ella se corresponden con los marcados por los usuarios, es decir, comprobar que los *topic* que produce el modelo concuerdan con los *hashtag* originales elegidos por los propios autores.

El procedimiento a seguir será el siguiente: se va a seleccionar una serie de *hashtag*, se va a extraer un determinado número de tuits que utilicen dichos *hashtag*, y se va a elaborar un *dataset* con todos ellos, eliminando convenientemente del texto de cada tuit el *hashtag* que contiene y que se quiere inferir. Después, se procederá a preprocesar todos estos datos y alimentar el algoritmo ALD con ellos, creando así un modelo que devuelva como salida una distribución de *topics*. Finalmente, se procederá a analizar dichos resultados.

El procedimiento se aplicará a los siguientes dos experimentos:

- En primer lugar, se extraerán masivamente datos relativos a un número determinado de *hashtags* elegidos a priori y se comprobará que efectivamente el modelo de *Topic Modelling* es capaz de distinguir los *topic* presentes.
- En segundo lugar, se extraerán masivamente datos relativos a un solo *hashtag*, también elegido a priori. A continuación, se analizará qué otros *hashtags* aparecen junto a este *hashtag* anterior y se filtrarán los tuits para obtener solo los que contengan los *hashtags* que más aparecen. Por último, se comprobará que el modelo es capaz de distinguir estos temas.

Además, se pretende estudiar en qué medida afectan a los resultados variables como el número de tuits extraídos o el de *hashtags* escogidos.

En concreto, los objetivos que se quiere cumplir son los siguientes:

- Extracción de datos masivos de Twitter y construcción de un *dataset* en base a ellos.
- Entrenamiento y creación de modelos de *Topic Modelling* que infieran la distribución deseada de *topics* a partir del conjunto de datos.
- Análisis e interpretación efectiva de los resultados obtenidos por los modelos.

### 3. Descripción informática

#### 3.1. Preparación del entorno de desarrollo

En este apartado se va a explicar detalladamente cómo se ha procedido para preparar el entorno de desarrollo necesario para desarrollar y ejecutar aplicaciones de Apache Spark escritas en Scala.

##### 3.1.1. Instalación de Java SE Development Kit 8

Al ser Scala un lenguaje JVM que necesita la Java Virtual Machine para correr, debemos descargar e instalar el Java Development Kit (JDK) 8 o superior disponible en la página oficial de Oracle [4].

##### 3.1.2. Configuración de IntelliJ IDEA

- Descarga e instalación

Desde la página oficial de Scala [5], se nos recomienda IntelliJ IDEA como IDE para programar en Scala, debido a su amplia compatibilidad con este lenguaje, si bien es cierto que otros IDE como VS Code, Vim o Sublime Text son también aptos para el desarrollo en Scala.

Nosotros vamos a descargar e instalar IntelliJ IDEA 2020.3.3 *Ultimate Edition* [6], pero la versión *Community Edition* es suficiente.

- Creación de un proyecto Scala

A continuación, creamos un nuevo proyecto de Scala usando Apache Maven, siguiendo los pasos:

1. Seleccionamos “New Project” para abrir la ventana de creación de proyectos.
2. Seleccionamos “Maven” en el panel de la izquierda de dicha ventana.
3. Marcamos la opción “Create from prototype”.
4. Seleccionamos la opción “org.scala-tools.archetypes:scala-archetypes-simple” como se muestra en la Figura 2. Selección del archetype.
5. En la siguiente pantalla, definimos el nombre del proyecto y su localización.
6. Finalizamos.

Los *archetypes* de Maven son plantillas para crear proyectos Maven con una estructura predeterminada. Son especialmente útiles a la hora de descargar ciertas dependencias necesarias para nuestro proyecto y para seguir una estructura estandarizada fácilmente comprensible por el resto de desarrolladores.

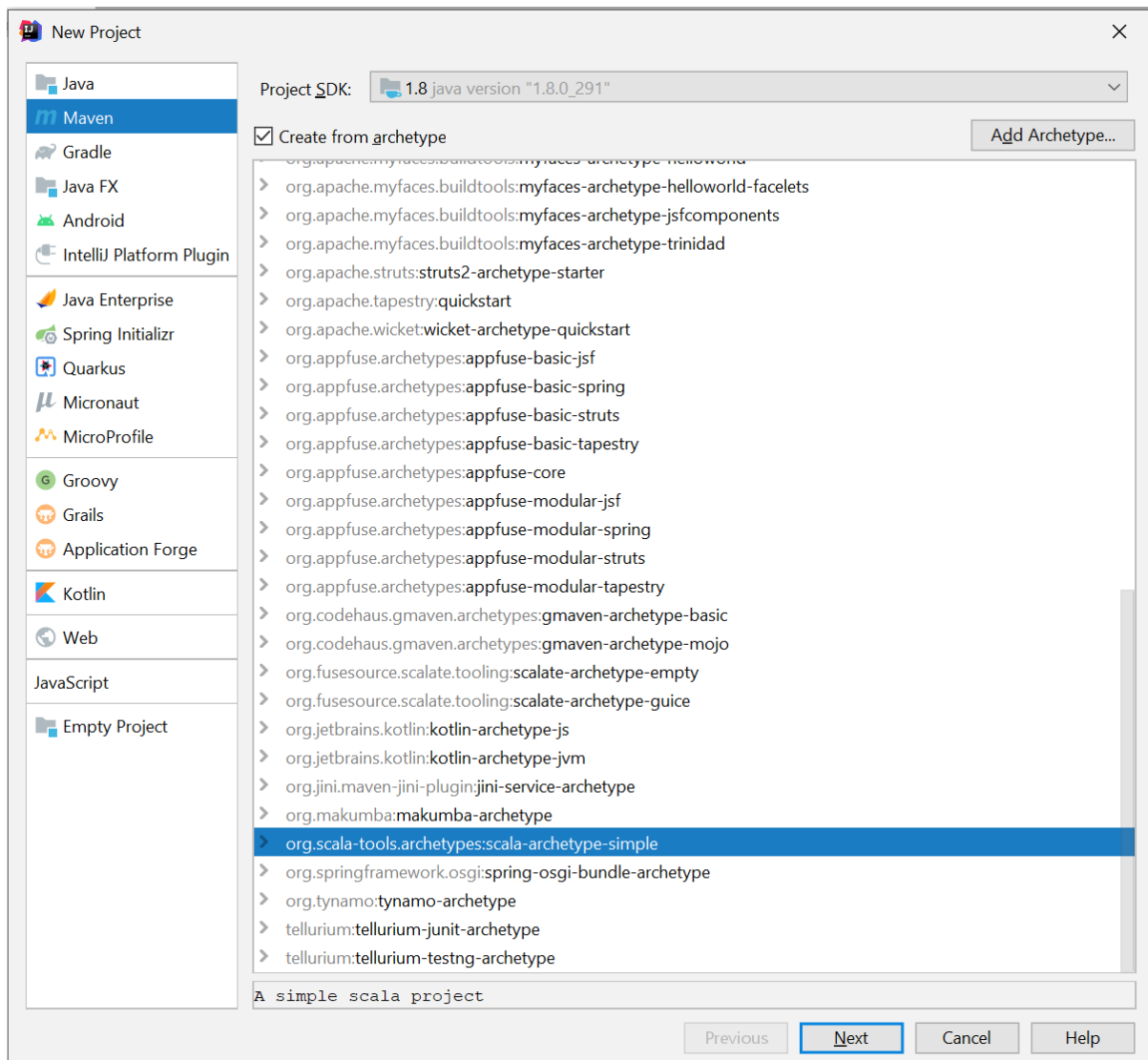


Figura 2. Selección del archetype

- Instalación del plugin de Scala

Instalamos el plugin de Scala disponible en el gestor de plugins al que podemos acceder en “File > Settings > Plugins” tal y como se muestra en la Figura 3. Instalación del plugin de Scala.

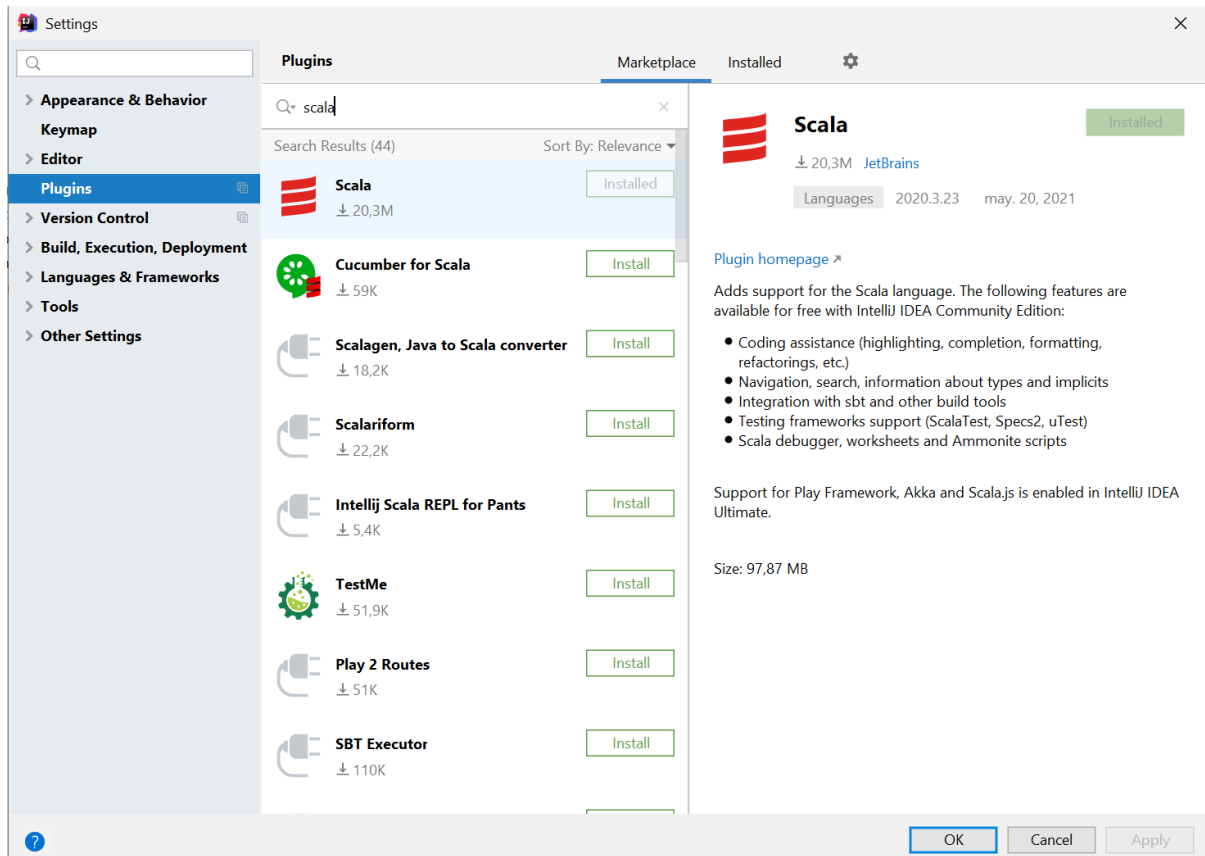


Figura 3. Instalación del plugin de Scala

- Descarga e instalación de Scala SDK

Al instalar el plugin de Scala, nos aparecerá un mensaje como el de la Figura 4. Mensaje "No Scala SDK in module" avisándonos de la necesidad de establecer un Scala SDK para nuestro proyecto.

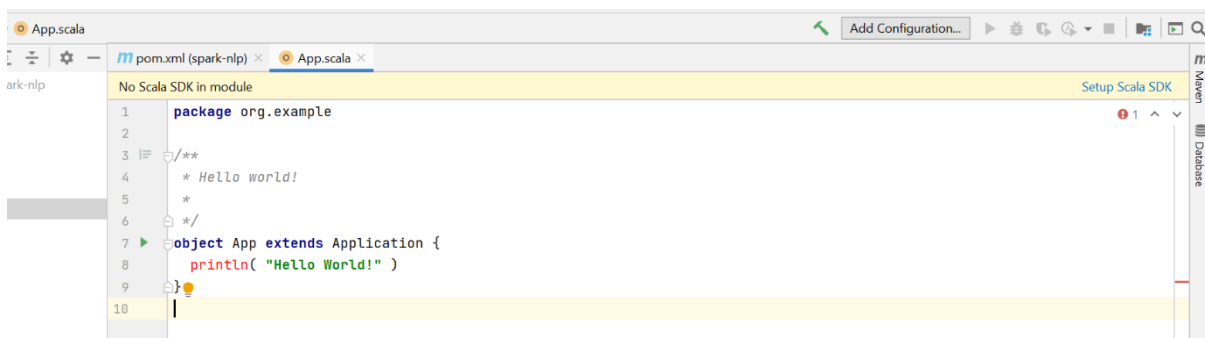
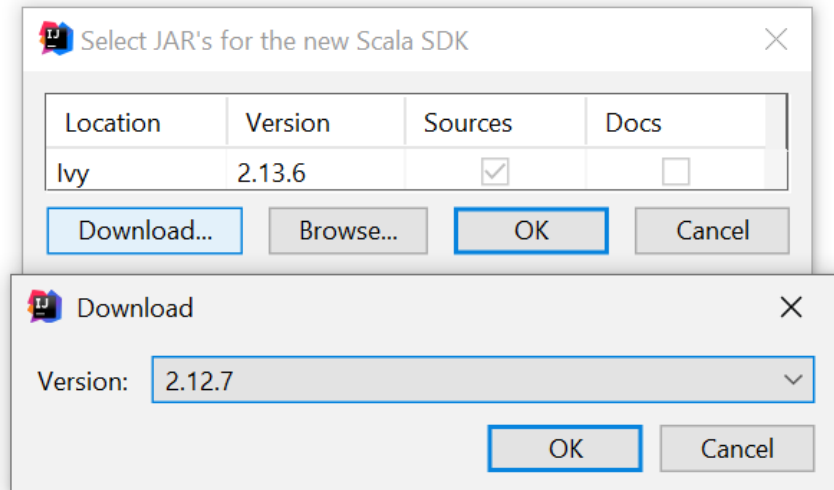


Figura 4. Mensaje "No Scala SDK in module"



Tras pulsar en “Setup Scala SDK”, un gestor de descargas nos permitirá descargar el Scala SDK que queramos, nosotros seleccionamos la versión 2.12.7 como se muestra en la Figura 5. Descarga del Scala SDK. Es especialmente importante descargar e instalar esta versión concretamente, si se quieren evitar errores por conflictos con el resto de dependencias.



*Figura 5. Descarga del Scala SDK*

La descarga se iniciará automáticamente tras darle a “OK”.

- Borrar archivos innecesarios

Debido a que la carpeta “test” puede generar errores por incompatibilidades de versiones, como de momento resulta innecesaria, procedemos a eliminarla.

- Cambios en el POM

En el *pom.xml*, vamos a indicar la versión de nuestro Scala SDK descargado anteriormente. La nuestra es la 2.12.7, que especificamos de esta manera:

```
<properties>
  <scala.version>2.12.7</scala.version>
</properties>
```

Tras realizar estos pasos, ya podremos ejecutar el “Hello World” del proyecto.

### 3.1.3. Instalación de Spark

Para incluir Spark en nuestro proyecto, añadimos las siguientes dependencias Maven [7] en el POM:

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-mllib_2.12</artifactId>
  <version>3.0.0</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.12</artifactId>
  <version>3.0.0</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.12</artifactId>
  <version>3.0.0</version>
  <scope>compile</scope>
</dependency>
```

### 3.1.4. Instalación de Spark NLP

A su vez, para incluir la librería Spark NLP en nuestro proyecto, debemos añadir las siguientes dependencias Maven [8] el POM:

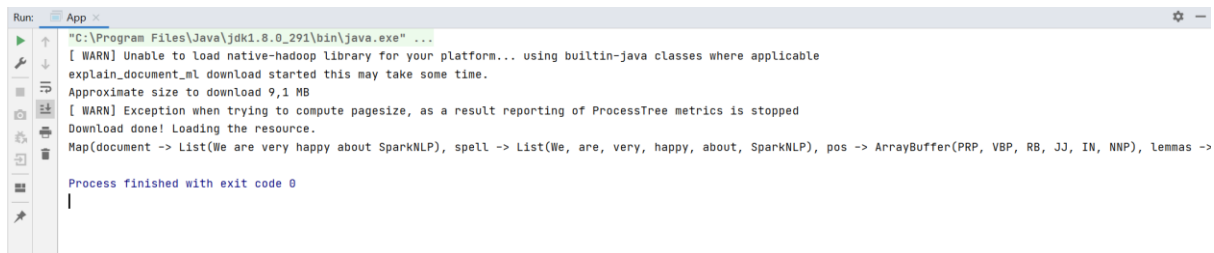
```
<dependency>
  <groupId>com.johnsnowlabs.nlp</groupId>
  <artifactId>spark-nlp_2.12</artifactId>
  <version>3.0.3</version>
</dependency>
<dependency>
  <groupId>com.johnsnowlabs.nlp</groupId>
  <artifactId>spark-nlp-gpu_2.12</artifactId>
  <version>3.0.3</version>
</dependency>
</dependencies>
```

### 3.1.5. Ejecución del caso base de ejemplo de Spark NLP

Para comprobar que se ha completado la preparación del entorno de desarrollo satisfactoriamente, vamos a ejecutar el “Hello World” [9] que nos proporciona John Snow, creadores de Spark NLP, como ejemplo básico de uso de su librería:

```
1. object HelloWorld extends App {
2.   import com.johnsnowlabs.nlp.pretrained.PretrainedPipeline
3.   val explainDocumentPipeline = PretrainedPipeline("explain_document_ml")
4.   val annotations = explainDocumentPipeline.annotate("We are very happy about SparkNLP")
5.   println(annotations)
6. }
7.
```

Tras ejecutar la aplicación con “Run > Run ‘HelloWorld’” la salida que debería salir por consola debería ser similar a la que se muestra en la Figura 6. Salida de "HelloWorld"



```
Run: App x
"C:\Program Files\Java\jdk1.8.0_291\bin\java.exe" ...
[ WARN] Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
explain_document_ml download started this may take some time.
Approximate size to download 9,1 MB
[ WARN] Exception when trying to compute pagesize, as a result reporting of ProcessTree metrics is stopped
Download done! Loading the resource.
Map(document -> List(We are very happy about SparkNLP), spell -> List(We, are, very, happy, about, SparkNLP), pos -> ArrayBuffer(PRP, VBP, RB, JJ, IN, NNP), lemmas ->
Process finished with exit code 0
```

Figura 6. Salida de "HelloWorld"

El “map” que se imprime por pantalla es el siguiente:

```
1. Map(
2.   stem -> List(we, ar, veri, happi, about, sparknlp),
3.   checked -> List(We, are, very, happy, about, SparkNLP),
4.   lemma -> List(We, be, very, happy, about, SparkNLP),
5.   document -> List(We are very happy about SparkNLP),
6.   pos -> ArrayBuffer(PRP, VBP, RB, JJ, IN, NNP),
7.   token -> List(We, are, very, happy, about, SparkNLP),
8.   sentence -> List(We are very happy about SparkNLP)
9. )
10.
```

En esta pequeña aplicación, se ha importado un *pipeline* preentrenado y se ha aplicado a un documento (en este caso, un *string* con una pequeña oración, “We are very happy about Spark NLP”). El resultado es un mapa con información sobre el documento, como sus token, lemmas y stem.

Con esto, quedaría concluida la preparación del entorno de desarrollo.

### 3.2. Acceso y extracción de los datos

#### 3.2.1. Acceso a datos de Twitter con su API

Twitter proporciona a las empresas, desarrolladores y usuarios un mecanismo de acceso y extracción de sus datos: la API de Twitter. A partir de estas interfaces, se puede crear software que se integre con Twitter y acceda a sus datos públicos. Entre estos datos, se encuentran [10]:

- Cuentas y usuarios: se puede automatizar la gestión de cualquier perfil siempre que se tenga la autorización para ello. Por ejemplo, se pueden bloquear o administrar seguidores desde la API.
- Tweets y respuestas: se pueden buscar tuits y respuestas a tuits por filtros como palabras clave, fecha, geolocalización. Se puede analizar sus métricas: número de likes, respuestas, “retuits”, etc.

La API de Twitter cuenta con una nueva versión lanzada recientemente, Twitter API v2. Esta nueva versión de la API incorpora nuevas características, como nuevos y más detallados campos en los objetos, métricas avanzadas sobre el rendimiento de los tuits, mejoras en el lenguaje de consulta, y muchas otras más.

El grado de acceso a estos datos lo determina el “tier” del usuario, es decir, el tipo de cuenta del usuario que va a utilizar la API. Hay tres tipos de cuenta: “estándar”, “académica” y “empresa”. Nosotros hemos podido acceder a la API desde la cuenta académica. Esta tiene un acceso amplio a los recursos de la API, y un límite de peticiones que se ajusta a las necesidades de este trabajo.

#### 3.2.2. Descarga masiva de datos a través de la herramienta TwitterAPI

Para facilitar y acelerar la extracción de datos masiva a través de consultas a la API, se ha utilizado una herramienta para automatizar este proceso: TwitterAPI. Esta herramienta, desarrollada por Habla Computing [11], realiza peticiones al *endpoint* de “tweets recientes” y almacena los tuits descargados en un formato adecuado para ser procesado.

La herramienta TwitterAPI es capaz de realizar peticiones de grupos de 10 a 100 tuits, a una tasa de 450 grupos cada 15 minutos (estos límites los proporciona la propia API de Twitter). Este software es capaz de lanzar peticiones de manera concurrente y gestionar la descarga de tuits hasta que el límite de peticiones mensuales disponibles para el usuario se haya alcanzado. Por último, TwitterAPI almacena la información extraída en un directorio establecido por el usuario, en formato JSON Line.

Antes de ejecutar la aplicación, hay que configurarla adecuadamente indicando los datos de autorización de nuestra cuenta en la API de Twitter (el *bearer token* proporcionado por la página web) en el archivo de configuración de la herramienta. Además, hay que establecer los términos (que pueden ser palabras u oraciones, en nuestro caso, *hashtags*) sobre los que se va a realizar la búsqueda de tuits.

Tras la ejecución, los datos extraídos quedarán almacenados en la carpeta “data” (o cualquier otra que se especifique) en formato JSON Line. Además, se generará un archivo denominado *twitterv2.log* donde podremos consultar los *logs* producidos durante la ejecución.

### 3.2.3. Ejecución de TwitterAPI

Para ejecutar TwitterAPI, hay que seguir procedimiento descrito a continuación.

- Requisitos y contenido del paquete

Es necesario asegurarse de que se cumplen con los requisitos para ejecutar la aplicación: el paquete TwitterAPI v2, la versión Java 1.8 o superior y la capacidad para ejecutar scripts de Shell.

El contenido del paquete TwitterAPI v2 es el siguiente:

- Archivo README
- `twitterv2-jar`, una aplicación de Java
- `setup.sh`, el script de instalación
- `launch.sh`, el script de ejecución
- Instalación

Primeramente, se procederá a instalar la aplicación ejecutando el script `setup.sh` a través del siguiente comando:

```
1. ./setup.sh --target-dir
```

`Target-dir` será la ruta absoluta del directorio donde se descargarán los datos y donde se almacenará la configuración.

- Configuración

Tras terminar el anterior paso, se habrán creado en el directorio especificado los archivos `application.conf` y `logback.xml`. En el archivo `application.conf` podremos configurar los parámetros necesarios para la correcta descarga de tuits:



```
1  twitterV2.recent-queries {
2    bearer-token = "
3    max-total = 1000,
4    max-result = 100,
5    queries = [
6      # TÉRMINOS GENERALES
7      { "term" = "#COVID19 -has:hashtags" }
8      { "term" = "#masterchef -has:hashtags" }
9      { "term" = "#RolandGarros -has:hashtags" }
10     { "term" = "#yoveosalvame -has:hashtags" }
11     { "term" = "#EBAU -has:hashtags" }
12     { "term" = "#E32021 -has:hashtags" }
13   ]
14 }
15
16 akka {
17   loggers = ["akka.event.slf4j.Slf4jLogger"]
18   logging-filter = "akka.event.slf4j.Slf4jLoggingFilter"
19
20   http.host-connection-pool.max-open-requests = 512
21 }
--
```

Figura 7. Archivo `application.conf`

- **bearer-token**: Token proporcionado por la API de Twitter para realizar la autorización.
- **max-total**: Número máximo de tuits a descargar.
- **max-result**: Número máximo de tuits que se devolverá en cada respuesta a una petición.
- **queries**: Consultas a realizar a la API. En nuestro caso, dado que queremos obtener los tuits recientes que contienen un determinado hashtag, la *query* utilizada es la siguiente:

```
1. "#hashtag -has:hashtags"
```

- Descarga de datos

Después de configurar los parámetros de `application.conf`, procedemos a ejecutar la aplicación con el script `launch.sh`, con el siguiente comando Shell:

```
1. ./launch.sh --target-dir
```

La descarga de datos puede demorarse en función de la cantidad de tuits a descargar.

- Resultados obtenidos

Al terminar de ejecutarse el script, podremos comprobar que en la carpeta `data` generada en nuestro directorio de instalación, se encuentran los archivos JSON con los tuits. Habrá un archivo por cada *query*. A continuación, se muestra un ejemplo de una de las líneas (formateada para su correcta comprensión) de uno de los archivos generados tras la ejecución.

```
{ "attachments": {
  "media_keys": [
    "16_1402409554965667842"
  ],
},
"author_id": "1188863666663374854",
"context_annotations": [
  {
    "domain": {
      "description": "Television shows from around the world",
      "id": "3",
      "name": "TV Shows"
    },
    "entity": {
      "description": "The entertaining amateur cookery contest, where chefs attempt to create the most delicious dishes.",
      "id": "10029072721",
      "name": "MasterChef"
    }
  },
],
"conversation_id": "1402409563291369474",
"created_at": "2021-06-08T23:37:50.000Z",
"entities": {
  "hashtags": [
    {
      "end": 96,
      "start": 85,
      "tag": "MasterChef"
    }
  ],
},
"urls": [
```

```
{
  "display_url": "pic.twitter.com/DPRvldelMJ",
```

```
    "end": 186,
    "expanded_url": "https://twitter.com/foxtuits/status/1402409563291369474/photo/1",
    "start": 163,
    "url": "https://t.co/DPRvldelMJ"      }
  ]
},
"id": "1402409563291369474",
"lang": "es",
"possibly_sensitive": false,
"public_metrics": {
  "like_count": 4,
  "quote_count": 0,
  "reply_count": 0,
  "retweet_count": 0  },
"reply_settings": "everyone",
"source": "Twitter Web App",
"text": "Buenas Noches! Voy a poner una FOCACCIA en el horno para cenar, y prepareme para ver
#MasterChef \nQue tengan una muy buena noche de Martes!\nHasta Mañana! 🌟👁️🌟🌟🌟🌟🌟🌟
https://t.co/DPRvldelMJ"
},
```

Este archivo se generó al buscar tuits con el *hashtag* #Masterchef. Como vemos, se almacenan múltiples campos de cada tuit, como por ejemplo: id del autor, anotaciones sobre su contexto (donde incluso podemos consultar información sobre el dominio en el que se enmarca el tuit, en este caso el dominio “TV Shows”), fecha de creación, idioma, métricas de interés generado, dispositivo con el que se tuiteó y, finalmente, el texto del tuit.

### 3.3. Construcción del *dataset*

Para construir el *dataset* a partir de los ficheros obtenidos con twitterAPI, se decidió crear una pequeña aplicación de Python que automatizase el proceso de limpieza de datos y combinación de todos ellos en un solo archivo CSV.

#### 3.3.1. Estructura de la aplicación

La aplicación está compuesta por los siguientes módulos:

- **Directorio data:** Contendrá los archivos JSON Lite generados por TwitterAPI y un script *to\_json.bat*
- **build-dataset.ipynb:** Notebook de Jupyter<sup>1</sup> con utilidades para visualizar el *dataset* y los pasos seguidos para su elaboración.
- **config.yaml:** Fichero de configuración donde se especifican los hashtags y los ficheros que se quiere incluir en el *dataset* final.
- **main.py:** Aplicación. Contiene el código empleado para la limpieza y construcción del *dataset*.

---

<sup>1</sup> Un Jupyter Notebook es un documento ejecutable por el entorno Jupyter en el que los usuarios pueden desarrollar y ejecutar fragmentos de código de hasta 40 lenguajes de programación diferentes.

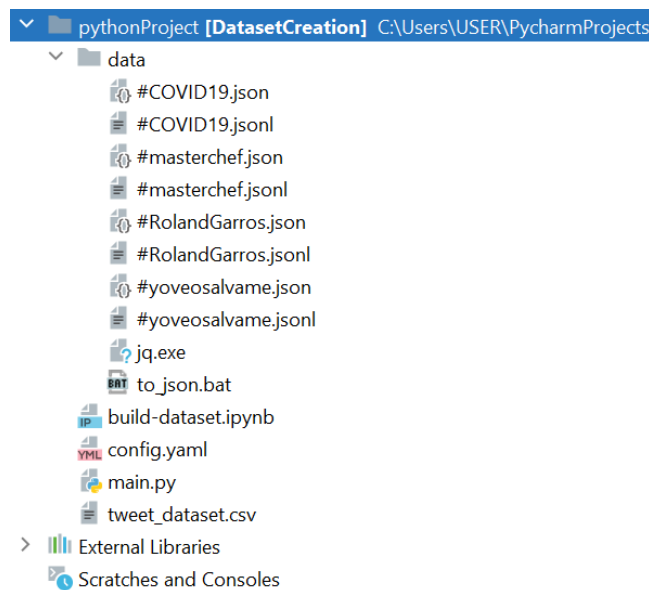


Figura 8. Estructura de directorios

### 3.3.2. Conversión de formato JSON Line a JSON

Los archivos generados por TwitterAPI eran de formato JSON Line, un formato que Python, por defecto, no es capaz de procesar. Resultaba necesario, entonces, convertir estos ficheros a un formato JSON, que sí es procesable fácilmente por Python. Para ello, se hizo uso de JQ, un procesador ligero de JSON [12].

El script `to_json.bat` automatiza el proceso de conversión entre formatos haciendo uso de JQ. Tan solo habría que especificar los nombres de los archivos de entrada y de salida. Este es un ejemplo de comando que se utilizaría para crear un archivo `#masterchef.json` en formato JSON a partir del archivo `#masterchef.jsonl` en formato JSON Line:

```
1. jq.exe --slurp . < #masterchef.jsonl > #masterchef.json
```

### 3.3.3. Limpieza del texto del tuit

Para que el rendimiento cualquier modelo de *Topic Modelling* sea óptimo, los datos de entrada deben ser preferiblemente de texto plano; sin embargo, un tuit puede contener desde emoticonos hasta URLs, menciones a otros usuarios y demás elementos que contaminan el texto del tuit y generan ruido que probablemente de lugar a resultados finales pobres. Por ello, se elaboró un programa que limpiase el texto correctamente y generase un *dataset* con los datos procesados. El procedimiento que sigue es el siguiente:

- Seleccionar la columna `text`. De entre todos los campos de información almacenada de cada tuit, solo nos interesa el campo relativo al texto del propio tuit, el campo `text`.
- Sustituir las vocales con tilde por las vocales normales. Algunos algoritmos no procesan correctamente estos caracteres.
- Eliminar las URLs.
- Crear un campo con los *hashtags* encontrados en cada tuit. Esto es necesario de cara a analizar resultados más adelante.



- e) Eliminar las menciones a otros usuarios.
- f) Eliminar el *hashtag* original y que se quiere que el modelo infiera.
- g) Eliminar la almohadilla “#” al inicio de las palabras. Con el resto de hashtags (que no son el que se quiere estudiar) que puede contener un tuit, se procederá a eliminar tan solo la almohadilla del principio de la palabra. Esto se hace porque, aunque nos interese borrar el hashtag principal del tuit, el resto de hashtags que no estamos estudiando sí que pueden contener palabras representativas e importantes a la hora de identificar el *topic* de un tuit.
- h) Quitar los caracteres que no sean letras del abecedario español, incluyendo números.

Este código se implementó haciendo uso en gran medida de expresiones regulares. Por ejemplo, el siguiente código implementa una función que, haciendo uso de la librería de expresiones regulares de Python, elimina las URL encontradas en un tuit (sustituyéndolas por un string vacío):

```
1. def quitarURLs(text):
2.     text = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', '', text)
3.     return text
```

Por otra parte, la siguiente función es la que se encarga de eliminar caracteres que no se encuentren en el conjunto  $[a - zA - Zñ]$ :

```
1. def quitarCaracteresRaros(text):
2.     text = re.sub('[^a-zA-Zñ ]', ' ', text)
3.     return text
```

### 3.3.4. Combinación de *datasets* en uno solo

Se pretendía crear un solo archivo CSV que almacenase el *dataset* que combinaba todos los *datasets* creados a partir de cada archivo JSON.

Esto se implementó iterando la lista de archivos JSON, llamando a la función encargada de construir un *dataset* tal y como hemos mencionado en el apartado anterior, y almacenando el resultado en una lista de *datasets*. Finalmente, se concatenaban todos ellos haciendo uso de la función `concat()` de la famosa librería Pandas de Python.

### 3.3.5. Exportación del *dataset*

Por último, se exporta el *dataset* creado a un formato CSV, haciendo uso de la función `to_csv()` también de Pandas. El resultado es un dataset formado por tres columnas (text, hashtags y preprocessed) y tantas filas como tuits se descargaron, como el del ejemplo que aparece en Figura 9. Ejemplo de dataset generado con este procedimiento.

	C1	C2	C3
1	text	hashtags	preprocessed
2	●#EnVivo   Sigue la conferencia ...	['EnVivo', 'COVID19', 'JuntosSald...	Sigue la conferencia de la sobre la...
3	#EnVivo l Sigue en @FortunayPoder la transmisión de la conferencia de prensa de @SSalud_mx sobre el reporte técnico		
4	■ Conoce el reporte que ofrece @...	['COVID19', 'OnceNoticias', 'App']	Conoce el reporte que ofrece este mart...
5	INFO=#pmrkemaman #MalaysiaPrihati...	['pmrkemaman', 'MalaysiaPrihatin'...	INFO pmrkemaman MalaysiaPrihatin MomenNeg...
6	#24HorasQRoo@   #EnVivo ● Confere...	['24HorasQRoo', 'EnVivo', 'COVID1...	HorasQRoo Conferencia de prensa CO...
7	El Salvador mejora de categoria e...	['COVID19']	El Salvador mejora de categoria en la lis...
8	●#EnVivo   Sigue la conferencia ...	['EnVivo', 'COVID19', 'JuntosSald...	Sigue la conferencia de la sobre la...
9	#Comunicado #Oficial=Llama salud ...	['Comunicado', 'Oficial', 'COVID1...	Comunicado Oficial Llama salud a reforzar...
10	El lado B del #COVID19 : el costo...	['COVID19', 'SaludMental']	El lado B del COVID el costo de la Sa...
11	Evita ingerir alimentos mientras ...	['Entorno', 'COVID19']	Evita ingerir alimentos mientras viajas ...
12	INFO=#pmrkemaman #MalaysiaPrihati...	['pmrkemaman', 'MalaysiaPrihatin'...	INFO pmrkemaman MalaysiaPrihatin MomenNeg...
13	★Nos despedimos por hoy. No olvi...	['COVID19']	Nos despedimos por hoy No olvides mante...
14	INFO=#pmrkemaman #MalaysiaPrihati...	['pmrkemaman', 'MalaysiaPrihatin'...	INFO pmrkemaman MalaysiaPrihatin MomenNeg...
15	¡El último The HJL Radio Hits Dai...	['coronavirus', 'covid19']	El ultimo The HJL Radio Hits Daily Gra...
16	INFO=#pmrkemaman #MalaysiaPrihati...	['pmrkemaman', 'MalaysiaPrihatin'...	INFO pmrkemaman MalaysiaPrihatin MomenNeg...
17	★ Conferencia de prensa número 4...	['COVID19', 'GraciasPorCuidarnos'...	Conferencia de prensa numero para a...
18	☺ #ALMomento    Autoridades del ...	['ALMomento', 'COVID19', 'Quédate...	ALMomento Autoridades del sector sal...

Figura 9. Ejemplo de dataset generado con este procedimiento

### 3.4. Preprocesamiento

En el ámbito del Machine Learning, el preprocesamiento de datos consiste en transformar los datos en bruto a un formato adecuado para ser suministrados a un modelo de Machine Learning determinado.

En nuestro caso, se va a crear un pipeline que transforme los tuits limpios en tokens válidos sobre los que entrenar el modelo de Topic Modelling. Este pipeline estará conformado por un conjunto de elementos que aplicarán diferentes transformaciones a sus entradas, produciendo una salida que sirva como entrada al siguiente componente del *pipeline*.

El pipeline es el siguiente:

```
1. val pipeline = new Pipeline().
2.   setStages(Array(
3.     documentAssembler,
4.     tokenizer,
5.     stopWordsCleaner,
6.     lemmatizer,
7.     finisher
8.   ))
```

Sus componentes se describirán en las siguientes secciones.

#### 3.4.1. DocumentAssembler

El DocumentAssembler [13] es el punto de partida de cada pipeline de Spark NLP, ya que se encarga de preparar los datos en un formato que sea procesable por la librería.

```
1. val documentAssembler = new DocumentAssembler()
2.   .setInputCol("preprocessed")
3.   .setOutputCol("document")
4.   .setCleanupMode("shrink")
```

### 3.4.2. Tokenizer

La denominada como “tokenización” es la tarea de dividir un documento de entrada en partes denominadas “tokens”, que pueden entenderse como unidades de información (en la sección 3.4.6 se puede visualizar qué representa un token en nuestra aplicación en concreto). El `Tokenizer` [14] convierte texto en bruto en una secuencia de tokens, utilizando para ello reglas de tokenización de estándares abiertos.

```
1. val tokenizer = new Tokenizer()  
2.   .setInputCols(Array("document"))  
3.   .setOutputCol("token")
```

### 3.4.3. StopWordsCleaner

“Stop words” es un término utilizado en inglés para referirse a todas esas palabras presentes en el discurso que no aportan significado al mismo, sino que tan solo sirven para articularlo, y por lo tanto, aparecen con extraordinaria frecuencia en la gran mayoría de textos.

Este elemento se incorporó al pipeline después de ejecutar las primeras pruebas y comprobar que la distribución de palabras de cada topic estaba claramente influenciada por la presencia de estas palabras, tal y como se muestra en la Figura 10. Distribución de palabras sesgada por la presencia de “stop words”.

```
WrappedArray(Some(de), Some(el), Some(en), Some(lo), Some(a), Some(y), Some(del), Some(ser), Some(djokovic), Some(nadal))  
WrappedArray(Some(de), Some(lo), Some(en), Some(y), Some(por), Some(caso), Some(los), Some(nuevo), Some(el), Some(a))  
WrappedArray(Some(de), Some(lo), Some(o), Some(y), Some(a), Some(que), Some(el), Some(en), Some(an), Some(ser))  
WrappedArray(Some(de), Some(lo), Some(a), Some(en), Some(que), Some(el), Some(y), Some(no), Some(haber), Some(uno))  
WrappedArray(Some(que), Some(el), Some(lo), Some(de), Some(ser), Some(y), Some(no), Some(a), Some(en), Some(se))  
WrappedArray(Some(j), Some(rafa), Some(orgullo), Some(apoyorocio), Some(siempre), Some(grande), Some([]), Some(xbox), Some(summergamefest), Some(pride))
```

*Figura 10. Distribución de palabras sesgada por la presencia de “stop words”*

El elemento `StopWordsCleaner` [15] hace uso de un diccionario de “stop words” y los elimina, dando como salida el documento de entrada sin dichos términos. Dado que el diccionario que incluye Spark NLP por defecto no sirve, pues sus términos están en inglés, y nuestros textos provienen del castellano, se ha utilizado un diccionario propio construido a partir de la recopilación de stop words de diferentes fuentes y de elaboración propia.

```
1. val stopWordsCleaner = new StopWordsCleaner()  
2.   .setInputCols("token")  
3.   .setOutputCol("cleanTokens")  
4.   .setStopWords(stopwords) // Lista de “stop words” a eliminar  
5.   .setCaseSensitive(false)
```

### 3.4.4. Lemmatizer

En lingüística, un lema es un conjunto de caracteres que forma una unidad semántica [16] y que probablemente sea la forma base que aparezca en un diccionario. Por ejemplo, “programar” es el lema de “programas”, “programan” o “programé”.

Haciendo uso del `Lemmatizer` [17] entrenado en el diccionario de lemas español, es posible convertir cada token a su forma “lematizada”, algo especialmente útil a la hora de unificar los token sobre los que se entrenará nuestro modelo.

```

1. val lemmatizer = LemmatizerModel.pretrained("lemma", "es")
2.   .setInputCols(Array("cleanTokens"))
3.   .setOutputCol("lemma")

```

### 3.4.5. Finisher

La clase `Finisher` [17] convierte las entradas a un formato de salida fácil de interpretar y de utilizar en la fase posterior de obtención de *features*.

```

1. val finisher = new Finisher()
2.   .setInputCols(Array("lemma"))
3.   .setOutputCols(Array("tokens"))
4.   .setOutputAsArray(true)
5.   .setCleanAnnotations(false)

```

### 3.4.6. Resultado del pipeline y de cada componente

En la Figura 11. Resultado de aplicar el pipeline a los datos de entrada se puede ver una muestra (las primeras 20 filas) del resultado obtenido a la salida del pipeline: un `DataFrame` con una columna por cada componente del pipeline. Además, se puede comprobar que a la salida de `DocumentAssembler` tenemos `document`, a diferencia de la salida del `Tokenizer`, el `StopWordsCleaner` o el `Lemmatizer`, que devuelven `token`, mientras que el `Finisher` devuelve una lista de strings.

preprocessed hashtag	document	token	cleanTokens	lemma	tokens
informe diario pa...	covid19 [[document, 0, 68...	[[token, 0, 6, in...	[[token, 0, 6, in...	[[token, 0, 6, in...	informar, diario...
el coronavirus lo...	covid19 [[document, 0, 25...	[[token, 0, 1, el...	[[token, 3, 13, c...	[[token, 3, 13, c...	coronavirus, aca...
desde venezuela s...	covid19 [[document, 0, 14...	[[token, 0, 4, de...	[[token, 6, 14, v...	[[token, 6, 14, v...	venezuela, infor...
eps sanitas santa...	covid19 [[document, 0, 22...	[[token, 0, 2, ep...	[[token, 0, 2, ep...	[[token, 0, 2, ep...	eps, sanitas, sa...
sobre el uso de l...	covid19 [[document, 0, 80...	[[token, 0, 4, so...	[[token, 19, 30, ...	[[token, 19, 30, ...	desametaxona, le...
the latest an ora...	covid19 [[document, 0, 48...	[[token, 0, 2, th...	[[token, 0, 2, th...	[[token, 0, 2, th...	the, latest, ora...
la no es pretexto...	covid19 [[document, 0, 14...	[[token, 0, 1, la...	[[token, 9, 16, p...	[[token, 9, 16, p...	pretextar, ceder...
estendencia campo...	covid19 [[document, 0, 22...	[[token, 0, 10, e...	[[token, 0, 10, e...	[[token, 0, 10, e...	estendencia, cam...
fortalecimiento d...	covid19 [[document, 0, 14...	[[token, 0, 14, f...	[[token, 0, 14, f...	[[token, 0, 14, f...	fortalecimiento,...
gt gt reporte dia...	covid19 [[document, 0, 19...	[[token, 0, 1, gt...	[[token, 0, 1, gt...	[[token, 0, 1, gt...	gt, gt, reportar...
parte del persona...	covid19 [[document, 0, 19...	[[token, 0, 4, pa...	[[token, 10, 17, ...	[[token, 10, 17, ...	personal, univer...
heroesdeverdad la...	covid19 [[document, 0, 20...	[[token, 0, 13, h...	[[token, 0, 13, h...	[[token, 0, 13, h...	heroesdeverdad, ...
noticiero ministe...	covid19 [[document, 0, 17...	[[token, 0, 8, no...	[[token, 0, 8, no...	[[token, 0, 8, no...	noticiero, minis...
metropolitana val...	covid19 [[document, 0, 14...	[[token, 0, 12, m...	[[token, 0, 12, m...	[[token, 0, 12, m...	metropolitano, v...
gracias a nuestro...	covid19 [[document, 0, 11...	[[token, 0, 6, gr...	[[token, 0, 6, gr...	[[token, 0, 6, gr...	gracia, gobernar...
para la marcha pa...	covid19 [[document, 0, 60...	[[token, 0, 3, pa...	[[token, 8, 13, m...	[[token, 8, 13, m...	marchar, defenes...
casos nuevos de p...	covid19 [[document, 0, 17...	[[token, 0, 4, ca...	[[token, 0, 4, ca...	[[token, 0, 4, ca...	caso, region, me...
seguimos en estad...	covid19 [[document, 0, 18...	[[token, 0, 7, se...	[[token, 0, 7, se...	[[token, 0, 7, se...	seguir, alertar,...
continua la vacun...	covid19 [[document, 0, 17...	[[token, 0, 7, co...	[[token, 0, 7, co...	[[token, 0, 7, co...	continuo, vacuna...
apoya ayuntamient...	covid19 [[document, 0, 75...	[[token, 0, 4, ap...	[[token, 0, 4, ap...	[[token, 0, 4, ap...	apoyar, ayuntami...

Figura 11. Resultado de aplicar el pipeline a los datos de entrada

Si por ejemplo quisiéramos inspeccionar más detalladamente la salida de `StopWordsCleaner`, la columna `cleanTokens`, veríamos algo similar a lo que aparece en Figura 12. Salida del componente `StopWordsCleaner`, donde se puede que la salida en cada fila la constituye una lista de listas con metadatos y un *string*. Esto es lo que podríamos entender como “token” en nuestro caso en concreto.

```

+-----+
|cleanTokens
+-----+
|[[token, 0, 6, informe, [sentence -> 0], []], [token, 8, 13, diario, [sentence -> 0], []], [token, 15, 21, partido, [sentence -> 0], []], [token, 26, 32, pehuajo,
|[[token, 3, 13, coronavirus, [sentence -> 0], []], [token, 26, 31, acabar, [sentence -> 0], []], [token, 33, 38, usando, [sentence -> 0], []], [token, 40, 48, taps
|[[token, 6, 14, venezuela, [sentence -> 0], []], [token, 19, 25, informa, [sentence -> 0], []], [token, 37, 43, jugador, [sentence -> 0], []], [token, 48, 51, gelf
|[[token, 0, 2, eps, [sentence -> 0], []], [token, 4, 10, sanitas, [sentence -> 0], []], [token, 12, 16, santa, [sentence -> 0], []], [token, 18, 22, marta, [sente
|[[token, 19, 30, desametaxona, [sentence -> 0], []], [token, 32, 35, lean, [sentence -> 0], []], [token, 46, 52, exponen, [sentence -> 0], []], [token, 58, 70, aut
|[[token, 0, 2, the, [sentence -> 0], []], [token, 4, 9, latest, [sentence -> 0], []], [token, 14, 20, oranzas, [sentence -> 0], []], [token, 28, 33, tierra, [sente
|[[token, 9, 16, pretexto, [sentence -> 0], []], [token, 23, 27, ceder, [sentence -> 0], []], [token, 38, 42, datos, [sentence -> 0], []], [token, 46, 54, renunciar
|[[token, 0, 10, estendencia, [sentence -> 0], []], [token, 12, 16, campo, [sentence -> 0], []], [token, 21, 25, marte, [sentence -> 0], []], [token, 42, 49, empeze
|[[token, 0, 14, fortalecimiento, [sentence -> 0], []], [token, 23, 33, capacidades, [sentence -> 0], []], [token, 38, 46, respuesta, [sentence -> 0], []], [token,
|[[token, 0, 1, gt, [sentence -> 0], []], [token, 3, 4, gt, [sentence -> 0], []], [token, 6, 12, reporte, [sentence -> 0], []], [token, 14, 19, diario, [sentence ->
|[[token, 10, 17, personal, [sentence -> 0], []], [token, 25, 35, universidad, [sentence -> 0], []], [token, 40, 46, oriente, [sentence -> 0], []], [token, 51, 54,
|[[token, 0, 13, heroesdeverdad, [sentence -> 0], []], [token, 18, 24, batalla, [sentence -> 0], []], [token, 39, 42, dias, [sentence -> 0], []], [token, 56, 61, pu
|[[token, 0, 8, noticiero, [sentence -> 0], []], [token, 10, 19, ministerio, [sentence -> 0], []], [token, 24, 28, salud, [sentence -> 0], []], [token, 30, 37, conti
|[[token, 0, 12, metropolitana, [sentence -> 0], []], [token, 14, 23, valparaiso, [sentence -> 0], []], [token, 25, 30, biobio, [sentence -> 0], []], [token, 32, 34
|[[token, 0, 6, gracias, [sentence -> 0], []], [token, 23, 30, gobierno, [sentence -> 0], []], [token, 42, 49, personas, [sentence -> 0], []], [token, 51, 57, cuent
|[[token, 8, 13, marcha, [sentence -> 0], []], [token, 20, 30, defenestrar, [sentence -> 0], []], [token, 34, 39, meribo, [sentence -> 0], []], [token, 50, 60, dese
|[[token, 0, 4, casos, [sentence -> 0], []], [token, 20, 25, region, [sentence -> 0], []], [token, 27, 39, metropolitana, [sentence -> 0], []], [token, 41, 50, valp
|[[token, 0, 7, seguimos, [sentence -> 0], []], [token, 22, 27, alerta, [sentence -> 0], []], [token, 32, 38, materia, [sentence -> 0], []], [token, 43, 51, infecta
|[[token, 0, 7, continua, [sentence -> 0], []], [token, 12, 21, vacunacion, [sentence -> 0], []], [token, 35, 42, personas, [sentence -> 0], []], [token, 58, 69, cc
|[[token, 0, 4, apoya, [sentence -> 0], []], [token, 6, 17, ayuntamiento, [sentence -> 0], []], [token, 21, 30, habitantes, [sentence -> 0], []], [token, 37, 45, ve
+-----+
only showing top 20 rows

```

Figura 12. Salida del componente StopWordsCleaner. Tokens.

### 3.5. Construcción del diccionario y obtención de *features*

```

1. val cv = new
   CountVectorizer().setInputCol("tokens").setOutputCol("features").setMinDF(100)
2. val cv_model = cv.fit(tokens_df)
3. val vectorized_tokens = cv_model.transform(tokens_df)

```

Tokenization result

```

+-----+
|          tokens|          features|
+-----+
|[informar, diario...|(877,[11,41,84,32...|
|[coronavirus, aca...|(877,[2,25,26,51,...|
|[venezuela, infor...|(877,[17,77,84,18...|
|[eps, sanitas, sa...|(877,[51,110,229,...|
|[desametaxona, le...|(877,[193],[1.0])|
|[the, latest, ora...|(877,[314,439],[1...|
|[pretextar, ceder...|(877,[31,266,336]...|
|[estendencia, cam...|(877,[15,33,42,50...|
|[fortalecimiento,...|(877,[51,672,686]...|
|[gt, gt, reportar...|(877,[64,154,165,...|
|[personal, univer...|(877,[64,127,169,...|
|[heroesdeverdad, ...|(877,[33,44,63,13...|
|[noticiero, minis...|(877,[50,51,402,6...|
|[metropolitano, v...|(877,[],[])|
|[gracia, gobernar...|(877,[10,27,66,18...|
|[marchar, defenes...|(877,[],[])|
|[caso, region, me...|(877,[13,591],[1...|
|[seguir, alertar,...|(877,[2,26,40,223...|
|[continuo, vacuna...|(877,[5,23,27,50,...|
|[apoyar, ayuntami...|(877,[131,775],[1...|
+-----+
only showing top 20 rows

```

### 3.6. Entrenamiento

```
1. val lda = new LDA().setK(num_topics).setMaxIter(10)
2. val model = lda.fit(vectorized_tokens)
```

### 3.7. Exhibición de resultados

```
1. val ll = model.logLikelihood(vectorized_tokens)
2. val lp = model.logPerplexity(vectorized_tokens)
3. println("The lower bound on the log likelihood of the entire corpus: " + ll)
4. println("The upper bound on perplexity: " + lp)
5.
6. val vocab = cv_model.vocabulary val topics = model.describeTopics()
7. val topics_rdd = topics.rdd topics.show()
8.
9. val topics_words = topics_rdd.map(row =>
  row.getAs[mutable.WrappedArray[Int]]("termIndices"))
  .map(idx_list => idx_list.map(idx => vocab.lift(idx)))
10.
11. topics_words.foreach(array_token => println(array_token))
```

## 4. Experimentos

## 5. Conclusiones



## 6. Anexos

### Procesamiento de Lenguaje Natural

El Procesamiento de Lenguaje Natural (PLN) es un área fundamental de la Inteligencia Artificial que consiste en la investigación y desarrollo de mecanismos para lograr una comunicación entre los humanos y la computadora a través de lenguaje natural [18].

Entre las múltiples aplicaciones del PLN están:

- Traducción automática
- Recuperación de la Información
- Extracción de Información y resúmenes
- Tutores inteligentes
- Reconocimiento de Voz

Aunque se considera que el PLN comienza a mediados del siglo XX, con el llamado experimento de Georgetown [19], el progreso en este ámbito ha sido muy lento durante todos estos años. Esto no se debe a que no se estén dedicando recursos a esta disciplina, sino más bien a que el desarrollo presenta varias dificultades, entre las que, sin duda, la principal es la de la ambigüedad. Esto se puede ejemplificar con el caso expuesto por [20], en el que la palabra inglesa *hit* da lugar a múltiples traducciones en función de su contexto:

1. *He hit the nail with the hammer* (Él golpeó el clavo con el martillo). => “golpear” o “martillar”
2. *The car swerved and hit the tree* (El coche se desvió bruscamente y chocó contra el árbol). => “chocar”
3. *The soldier fired and hit his target* (El soldado hizo fuego y dio en el blanco) => “acertar”

Aun cuando para nosotros los humanos resulte sencillo distinguir el significado entre palabras a partir de su contexto, resulta muy complejo trasladar este conocimiento a una máquina. Muchas investigaciones han logrado avances en esta tarea, con métodos como correlaciones estadísticas, bases de conocimiento o gramáticas; sin embargo, todavía queda mucho trabajo por delante si queremos tener modelos óptimos de lenguaje natural capaces de realizar sus tareas con precisión.

Uno de los avances recientes más destacados en este campo llegó con la introducción de algoritmos de aprendizaje automático. El presente estudio va a seguir esta línea y tratar de resolver tareas de PLN con estos algoritmos.

#### 1.1. Aprendizaje no supervisado

Dentro del campo de la Inteligencia Artificial y, en concreto, del Aprendizaje Automático (conocido como *Machine Learning* en inglés), hay dos tipos de algoritmos, principalmente: algoritmos de Aprendizaje Supervisado (*supervised learning*, en inglés) y algoritmos de Aprendizaje No Supervisado (*unsupervised learning*). La diferencia fundamental entre ambos reside en que los algoritmos de Aprendizaje Supervisado hacen uso de datos etiquetados previamente mientras que los del otro tipo no.

De esta forma, mientras que los algoritmos de Aprendizaje Supervisado infieren una función de clasificación o predicción a partir de datos ya etiquetados, los algoritmos de Aprendizaje No Supervisado tienen como objetivo analizar y descubrir patrones partiendo de una colección de datos no etiquetada y sobre la que no se tiene conocimiento “a priori”.

## 1.2. Topic Modelling

En el contexto del Procesamiento de Lenguaje Natural, el *Topic Modelling* es una técnica de Aprendizaje No Supervisado que tiene como objetivo analizar la estructura semántica subyacente de una colección de documentos [21].

Esta técnica surge de la necesidad de crear herramientas para explorar el contenido de grandes colecciones de documentos. Muchas veces, la búsqueda simple<sup>2</sup> es insuficiente para que las organizaciones sean capaces de conocer y gestionar la estructura interna de sus documentos. Por ello, se necesitan mecanismos para procesar grandes volúmenes de datos y conocer qué *topics* están presentes y cómo conectan los documentos.

Por ejemplo, la revista *Journal of Refugee Studies* realizó un estudio [22] basándose en técnicas de *Topic Modelling* para analizar las diferentes narrativas que había mantenido la prensa europea sobre la crisis de los refugiados. El conjunto de datos que utilizaron para realizar esta investigación consistió en artículos que contuviesen la palabra “refugiado” publicados en periódicos de cinco países europeos. Algunos ejemplos de *topics* que encontraron se recogen en esta tabla:

Nombre del Topic <sup>3</sup>	Palabras asociadas
Economía	porcentaje, Euro, economía, compañía
Ayuda humanitaria	Internacional, ministro de asuntos exteriores, organizaciones, humanitaria
Campamentos de refugiados	emigrante, campamento, barco, detención
Movimiento de los refugiados	huir, llegar, escapar, isla, Siria
Guerra	Militar, guerra, régimen, fuerzas, paz

Tabla 1. Ejemplos de *topics* en documentos sobre la crisis europea de refugiados

<sup>2</sup> Algoritmos de búsqueda que funcionan por coincidencia de términos en ciertos campos del documento, como su título o autor.

<sup>3</sup> El algoritmo de *Topic Modelling* aplicado no tiene como salida el nombre del topic, tan sólo la lista de las palabras que lo definen. Son los propios investigadores los que han clasificado y nombrado esta lista de palabras tras interpretar los resultados.

Como vemos, gracias al *Topic Modelling*, han podido extraer satisfactoriamente *topics* subyacentes al cuerpo de documentos. Estos *topics* están definidos por un conjunto de palabras que aparecen recurrentemente y siguiendo ciertos patrones (en la sección 1.5 se profundizará más en este concepto).

En la investigación mencionada anteriormente, se ha utilizado el algoritmo LDA para producir los *topics*, sin embargo, hay múltiples algoritmos disponibles. El primero de ellos surgió en 1990, cuando Landauer y Dumais propusieron el algoritmo denominado como *Latent Semantic Analysis* [23]. Tanto este como muchos de los algoritmos que aparecieron a partir de entonces se basan en el modelo algebraico conocido como Modelo de Espacio Vectorial (*Vector Space Model*, *VSM*, en inglés), que tiene como premisa básica que la semántica del documento puede describirse a partir de los términos que lo constituyen [24]. Así, VSM representa documentos mediante el uso de vectores, de esta manera:

$$d_j = (w_{1j}, w_{2j}, \dots, w_{mj})$$

Donde  $m$  es la cardinalidad del diccionario (conjunto de términos distintos que aparecen en el cuerpo de documentos) y  $0 \leq w_{ij} \leq 1$  representa la contribución del término  $t_i$  para la representación semántica del documento  $d_j$ .

Los vectores calculados a partir de los documentos pueden recogerse en una matriz llamada matriz término-documento.

No hay un consenso sobre cómo debe ponderarse la contribución de dichos términos en cada vector, por lo que las múltiples investigaciones en este sentido han dado lugar a diferentes métodos para realizar dicho cálculo, aunque generalmente esta ponderación depende del número de ocurrencias del término en un documento.

Aunque el Modelo de Espacio Vectorial tiene sus limitaciones (como, por ejemplo, la sensibilidad semántica: dos documentos con contextos similares, pero con diferente vocabulario, no serán asociados), sí que está demostrado que es un modelo eficiente para representar explícitamente la estructura interna de una serie de documentos, por lo que constituye un buen punto de partida para los nuevos algoritmos de *topic modelling*.

Estos algoritmos se pueden clasificar [21] en función de si se diseñaron desde una perspectiva algebraica de factorización de matrices o si fue desde un enfoque probabilístico.

Los enfoques no probabilísticos, como el LSA o el NNMF, utilizan técnicas de factorización de matrices, como la conocida Descomposición en Valores Singulares (*Singular Value Decomposition*, *SVD*, en inglés), para reducir la dimensionalidad de la matriz término-documento y conseguir una representación eficiente del cuerpo de documentos, en la que la similitud entre dos documentos pueda ser fácilmente calculable con medidas de similitud entre vectores. Los enfoques probabilísticos, entre los que se encuentran LDA y PLSA, trataron de utilizar métodos probabilísticos para mejorar la efectividad de los anteriores enfoques. En la siguiente sección se profundizará en el método LDA, por ser el que se utilizó en el experimento.

### 1.3. Asignación Latente de Dirichlet

En aprendizaje supervisado, la Asignación Latente de Dirichlet (ALD) (*Latent Dirichlet Allocation*, LDA, en inglés) es un modelo generativo que permite que grupos de observaciones

puedan ser explicados a partir de variables latentes<sup>4</sup>. En este caso, nuestras observaciones son palabras organizadas como documentos, y la presencia de cada palabra viene explicada por las variables latentes, en particular, los *topics* del documento.

En este modelo, se asume que los documentos surgen a partir de ciertos *topics*. En concreto, se asume que  $K$  *topics* se asocian con la colección de documentos, y que cada documento contiene cada *topic* en diferente proporción. Esto tiene sentido y resulta natural si entendemos que cada documento es heterogéneo y combina diferentes ideas, temáticas y puntos de vista. El objetivo sería aprender estos *topics* a partir de los datos de los que disponemos.

La descripción del algoritmo se puede encontrar en [25], pero como involucra conceptos y mecanismos complejos de matemáticas y estadística, fuera del alcance del presente estudio, se describirá tan solo una intuición [26] del mismo.

#### 1.3.1. Definiciones y notaciones

$K$  : Número de topics presentes en la colección de documentos

$V$  : Número de palabras en el vocabulario

$M$  : Número de documentos

$N$  : Número de palabras en cada documento

$w$  : Palabra representada por un vector de tamaño  $V$

$W$  : Documento formado por  $N$  palabras

$D$  : Corpus, colección de  $M$  documentos

$z$  : Uno de los  $K$  topics, definido como una distribución de palabras

$\theta(i, j)$  : Matriz que contiene la probabilidad de que el documento  $i$  contenga palabras pertenecientes al topic  $j$

$\beta(i, j)$  : Matriz que representa la probabilidad de que el topic  $i$  contenga a la palabra  $j$ .

$\alpha$  : Parámetro que define la distribución de  $\theta$

$\eta$  : Parámetro que define la distribución de  $\beta$

#### 1.3.2. Asunción de ALD respecto a cómo los documentos son generados

Pero antes de explicar cómo se entrena este algoritmo, vamos a ver cómo un modelo ALD ya entrenado generaría un documento. Es decir, vamos a ver cuál es el proceso generativo que asume ALD de cada documento de la colección.

---

<sup>4</sup> En estadística, una variable latente es aquella que no se observa directamente en el conjunto de datos, sino que se infiere de otras variables que sí se observan.

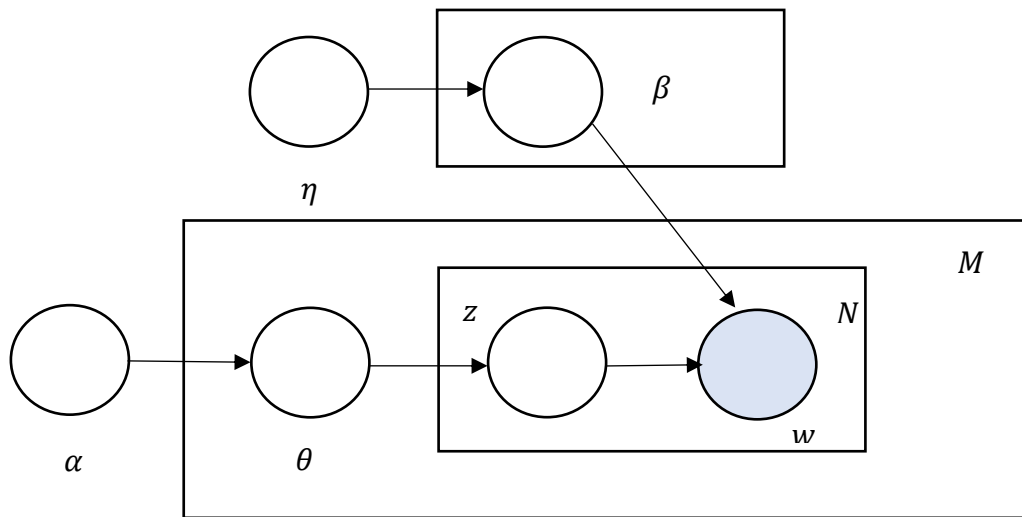


Figura 1. Representación gráfica del modelo LDA

El modelo LDA se representa gráficamente en la Figura 1 [25]. En él, se puede ver fácilmente la relación entre las diferentes variables que cobran papel en el modelo. Como vemos, tenemos un valor  $\alpha$  que determina la distribución que sigue la matriz  $\theta$ , que representa la distribución de *topics* en cada uno de los  $M$  documentos. Cada documento tiene  $N$  palabras y cada palabra  $w$  la genera un *topic*  $z$ . Además, tenemos que, dependiendo de  $\eta$ , la matriz  $\beta$  presentará una distribución que será la que defina las palabras de cada *topic*. Ambas matrices  $\beta$  y  $\sigma$  siguen una distribución Dirichlet.

De esta manera, para generarse un documento, habría que escoger aleatoriamente  $N$  *topics* (uno por cada palabra del documento) y, basándonos en  $\beta$ , escoger una palabra para cada *topic* del conjunto.

### 1.3.3. Entrenamiento del modelo LDA

Desde un punto de vista matemático, lo que se quiere calcular es la siguiente función de distribución de probabilidad a posteriori (*posterior probability distribution*):

$$P(\theta_{1:M}, z_{1:M,1:N}, \beta_{1:K} \mid D; \alpha, \eta)$$

Esta función puede verse como la “función inversa” a la función de generación de documentos descrita en el apartado anterior: dado un corpus de documentos observado  $D$  y los parámetros  $\alpha$  y  $\eta$ , esta función representa la distribución de las variables latentes que lo generaron,  $\theta$ ,  $z_{1:M,1:N}$  y  $\beta$ , y que son precisamente las que nos interesa conocer.

El cálculo de esta función es inabordable computacionalmente [27]. Sin embargo, existen aproximaciones, pero su descripción queda fuera del alcance de este estudio.

También es importante tener en cuenta que la salida del modelo ALD es una distribución de *topics*, que a su vez son distribuciones de palabras; el modelo no va a dar como salida una lista con los nombres de los *topics*, esos nombres los tiene que atribuir el propio investigador tras interpretar qué concepto puede asociar a cada distribución de palabras.

#### 1.4.Topics

El *output* de los *topic models* es un conjunto de *topics*, pero ¿qué es exactamente un *topic*?

En concreto, un *topic* es un conjunto de palabras que coinciden en ciertos documentos siguiendo patrones determinados (por ejemplo, “doctor”, “paciente”, “hospital” en noticias o documentos sobre “salud”). Debido a esto, cada *topic* tiene consistencia interna, y es el propio investigador el que debe determinar si esa consistencia tiene sentido y es útil para él. Es decir, el investigador, tras revisar los *topics* generados a la salida de su *topic model*, debe ser quien interprete si dichos *topics* siguen una consistencia interna coherente y los *topics* obtenidos son esperables y útiles para sus objetivos. Por ejemplo, si un investigador está buscando *topics* dentro de una colección de documentos sobre armas nucleares en EEUU, como es el caso de la investigación [28], probablemente considere que los *topics* “Armas” (definido por palabras como “explosión”, “misil”, “barco de caza”, “bomba”) o “Política Estadounidense” (definido por palabras como “guerra”, “presidente”, “política”, “Reagan”) son de utilidad, mientras que el *topic* “Películas y Música” (definido por palabras como “teatro”, “show”, “domingo”) es irrelevante y es mejor suprimirlo.

Es importante notar que en multitud de ocasiones el vocabulario que coincide en un mismo documento viene determinado por el punto de vista del autor, por lo que muchas veces, los *topics* recogen patrones de estilo de escritura, sentimientos, perspectiva e influencias de los autores. Es por esto, que entender un *topic* como simplemente un conjunto de palabras que suelen aparecer juntas es incorrecto, y quizá estemos ante un concepto mucho más complejo y abstracto; pero, en cualquier caso, qué representa exactamente un *topic* es una cuestión que se escapa del alcance de este estudio.

#### 1.5.Twitter: lo que está pasando ahora

Twitter es una red social creada en 2006 en California por Jack Dorsey y que tiene como principal servicio proporcionar a los usuarios la posibilidad de vivir y comentar las últimas tendencias, *lo que está pasando ahora* (tal y como la empresa presume en su página web [29]).

Puede entenderse también como un servicio de microblogueo, en el que los usuarios pueden difundir mensajes de hasta 280 caracteres llamados “tweets”, que se mostrarán en su perfil de manera pública. Los usuarios pueden tuitear sobre cualquier cosa siempre que no infrinjan las normas de Twitter; si lo desean, pueden utilizar los “hashtag” (palabras precedidas por “#”, por ejemplo, #Eurovisión), que agrupan mensajes sobre un mismo tema.

La popularidad de Twitter no ha hecho más que ir en aumento y actualmente se estima que cuenta con más de 330 millones de usuarios activos [30]. Esto lo convierte en la plataforma ideal para que las empresas puedan anunciarse y conocer a sus usuarios, sus gustos y sus necesidades.

## 7. Referencias

- [1] N. Elgendy y A. Elragal, «Big Data Analytics: A Literature Review Paper,» *Lecture Notes in Computer Science*, vol. 8557, pp. 214-227, 2014.
- [2] J. Dean y S. Ghemawat, «MapReduce: simplified data processing on large clusters,» *Communications of the ACM*, vol. 51, nº 1, p. 107–113, 2008.
- [3] Apache, « Apache Spark™ - Unified Analytics Engine for Big Data,» [En línea]. Available: <https://spark.apache.org/>.
- [4] Oracle, «Java SE Development Kit 8 Downloads,» [En línea]. Available: <https://www.oracle.com/es/java/technologies/javase-jdk8-downloads.html>.
- [5] École Polytechnique Fédérale , «The Scala Programming Language,» [En línea]. Available: <https://www.scala-lang.org/>.
- [6] Jet Brains, «Descargar IntelliJ IDEA: el IDE de Java eficaz y ergonómico de JetBrains,» [En línea]. Available: <https://www.jetbrains.com/es-es/idea/download/>.
- [7] MVN Repository, «Maven Repository: org.apache.spark,» [En línea]. Available: <https://mvnrepository.com/artifact/org.apache.spark>.
- [8] John Snow LABS, «Installation - Spark NLP,» [En línea]. Available: <https://nlp.johnsnowlabs.com/docs/en/install>.
- [9] John Snow LABS, «General Concepts - Spark NLP,» [En línea]. Available: <https://nlp.johnsnowlabs.com/docs/en/concepts>.
- [10] Twitter, «API reference index | Docs,» [En línea]. Available: <https://developer.twitter.com/en/docs/twitter-api/api-reference-index>.
- [11] hablapps, «TwitterAPI | GitHub Repository,» 2021. [En línea]. Available: <https://github.com/hablapps/twitterapi>.
- [12] jq, «jq,» 2013. [En línea]. Available: <https://stedolan.github.io/jq/>.
- [13] John Snow LABS, «Spark NLP 3.1.0 ScalaDoc - com.johnsnowlabs.nlp.DocumentAssembler,» [En línea]. Available: <https://nlp.johnsnowlabs.com/api/com/johnsnowlabs/nlp/DocumentAssembler.html>.
- [14] John Snow LABS, «Spark NLP 3.1.0 ScalaDoc com.johnsnowlabs.nlp.annotators.Tokenizer,» [En línea]. Available: <https://nlp.johnsnowlabs.com/api/com/johnsnowlabs/nlp/annotators/Tokenizer.html>.
- [15] John Snow LABS, «Spark NLP 3.1.0 ScalaDoc - com.johnsnowlabs.nlp.annotators.StopWordsCleaner,» [En línea]. Available:



<https://nlp.johnsnowlabs.com/api/com/johnsnowlabs/nlp/annotators/StopWordsCleaner.html>.

- [16 DBpedia, «About: Lema (lingüística),» [En línea]. Available:  
] [https://es.dbpedia.org/page/Lema\\_\(ling%C3%BC%C3%ADstica\)](https://es.dbpedia.org/page/Lema_(ling%C3%BC%C3%ADstica)).
- [17 John Snow Labs, «Spark NLP 3.1.0 ScalaDoc - com.johnsnowlabs.nlp.Finisher,» [En  
] línea]. Available:  
<https://nlp.johnsnowlabs.com/api/com/johnsnowlabs/nlp/Finisher.html>.
- [18 A. Cortez Vásquez, H. Vega Huerta, J. Pairona Quispe y A. Maria Huayna,  
] «Procesamiento de lenguaje natural,» *Revista de investigación de Sistemas e Informática*,  
vol. 6, nº 2, p. 48, 2009.
- [19 IBM Press, «IBM archives: 701 Translator,» 1 Enero 1954. [En línea]. Available:  
] [https://www.ibm.com/ibm/history/exhibits/701/701\\_translator.html](https://www.ibm.com/ibm/history/exhibits/701/701_translator.html).
- [20 J. Carbonell, «El procesamiento del lenguaje natural, tecnología en transición,» de  
] *Congreso de la Lengua Española*, Sevilla, 1992.
- [21 P. Kherwa y P. Bansal, «Topic Modeling: A Comprehensive Review,» *EAI Endorsed  
] Transactions on Scalable Information Systems*, vol. 7, nº 24, 2018.
- [22 «Media Framing Dynamics of the ‘European Refugee Crisis’: A Comparative Topic  
] Modelling Approach,» *Journal of Refugee Studies*, vol. 32, nº 1, p. 172–182, 2019.
- [23 S. Deerwester, S. T. Dumais y G. W. T. K. & H. R. Furnas, «Journal of the American  
] Society for Information Science,» pp. 391-407.
- [24 C. A. Kumar, M. Radvansky y J. Annapurna, «Analysis of a Vector Space Model, Latent  
] Semantic Indexing and Formal Concept Analysis for Information Retrieval,» *Cybernetics  
and Information Technologies*, vol. 12, nº 1, pp. 34-48, 2013.
- [25 D. M. Blei, A. Y. Ng y M. I. Jordan, «Latent Dirichlet Allocation,» *Journal of Machine  
] Learning Research*, vol. 3, nº January, pp. 993-1022, 2003.
- [26 T. Ganegedara, «Intuitive Guide to Latent Dirichlet Allocation,» 23 Agosto 2018. [En  
] línea]. Available: <https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-latent-dirichlet-allocation-437c81220158>.
- [27 D. M. Blei y J. D. Lafferty, *Topic Models*, Carnegie Mellon University, 2009.  
]
- [28 C. Jacobi, W. van Atteveldt y K. Welbers, «Quantitative analysis of large amounts  
] of journalistic texts using topic modelling,» *Digital Journalism*, vol. 4, nº 1, pp. 89-106,  
2016.
- [29 Twitter, «About Twitter | Our company and priorities,» [En línea]. Available:  
] <https://about.twitter.com/>.

- [30] M. Ahlgren, «Más de 50 estadísticas y datos de Twitter para 2021 que debe conocer,»  
] Marzo 2021. [En línea]. Available: <https://www.websitehostingrating.com/es/twitter-statistics/>.
- [31] John Snow LABS, «Spark NLP 3.1.0 ScalaDoc -  
] com.johnsnowlabs.nlp.annotators.Lemmatizer,» [En línea]. Available:  
<https://nlp.johnsnowlabs.com/api/com/johnsnowlabs/nlp/annotators/Lemmatizer.html>.