Ejercicio Procesamiento Distribuido

José Julián Barbosa Ayala
Michael Hernández Vera
Camilo Vega Ramírez

Contenido

Introducción

Para la actividad de contenido de palabras, se seleccionó el trabajo de grado titulado "Apoyo Tecnológico en la Inversión en Renta Variable", escrito por Román Zamora Carreras como parte de la obtención de su doble titulación en Ingeniería Informática y Administración en la Universidad Politécnica de Madrid. El artículo en cuestión consta de un total de 71 páginas y se adjunta como documento anexo al presente estudio.

El ejercicio debe ser desarrollado utilizando una de las herramientas de procesamiento distribuido, como Hadoop o Spark. A continuación, se enumeran algunas de las características de estas herramientas:

Hadoop:

- Arquitectura: Hadoop se basa en el modelo de programación MapReduce. Divide las tareas en etapas de map y reduce, donde los datos se dividen en bloques y se procesan de forma distribuida en un clúster de nodos.
- Escalabilidad: Hadoop es altamente escalable y puede manejar grandes conjuntos de datos mediante la distribución del procesamiento en múltiples nodos en un clúster.
- Tolerancia a fallos: Hadoop está diseñado para ser tolerante a fallos, lo que significa que puede recuperarse automáticamente de errores o fallos en los nodos individuales del clúster.
- Uso de almacenamiento distribuido: Hadoop utiliza Hadoop Distributed File System (HDFS) para almacenar datos de manera distribuida en los nodos del clúster.
- Caso de uso recomendado: Hadoop es adecuado para procesar grandes volúmenes de datos sin un tiempo de respuesta en tiempo real crítico. Es útil cuando se requiere un procesamiento en lotes de datos, como análisis de registros o procesamiento de datos históricos.

Spark:

- Arquitectura: Spark se basa en el modelo de programación de cómputo en memoria, lo que significa que puede realizar operaciones en memoria, lo que resulta en un procesamiento más rápido en comparación con Hadoop.
- Velocidad: Spark es conocido por su velocidad y capacidad de procesamiento en tiempo real. Puede realizar operaciones de procesamiento y análisis de datos de manera más rápida que Hadoop.
- Flexibilidad: Spark admite varios lenguajes de programación, como Java, Scala, Python y R, lo que brinda más flexibilidad a los desarrolladores para trabajar en el lenguaje de su elección.
- Uso de almacenamiento distribuido: Spark también puede utilizar HDFS para almacenar datos distribuidos, al igual que Hadoop, o puede acceder a otros sistemas de almacenamiento, como bases de datos o sistemas de archivos en la nube.
- Caso de uso recomendado: Spark es ideal cuando se necesita un procesamiento de datos en tiempo real o cerca del tiempo real. Es adecuado para casos de uso como análisis de streaming, procesamiento de eventos en tiempo real y procesamiento interactivo de datos.

Respecto al uso de Hadoop o Spark para llevar acabo conteo de palabras o analisis de texto en general podemos decir que, Si el texto es de gran tamaño o el procesamiento se realiza en lotes de datos sin la necesidad de un tiempo de respuesta en tiempo real, Hadoop puede ser una opción adecuada. La escalabilidad y tolerancia a fallos de Hadoop pueden ser beneficiosas para manejar grandes conjuntos de datos. Por otro lado, si se requiere un procesamiento más rápido y en tiempo real, especialmente si el texto es de tamaño moderado o pequeño, Spark puede ser una opción más adecuada debido a su velocidad y capacidad de procesamiento en memoria.

Para el desarrollo de este ejercicio, se ha elegido la herramienta Spark debido a su flexibilidad. Spark contiene funciones integradas para el análisis de texto, lo cual facilitará y agilizará nuestro análisis en comparación con Hadoop MapReduce, donde tendríamos que escribir nuestras propias funciones. Además, el tamaño del texto en el que realizaremos el conteo no es lo suficientemente grande como para tener en cuenta las consideraciones mencionadas anteriormente.

Para interactuar con Spark, utilizaremos R, en particular la librería sparklyr, que permite trabajar con Spark a través de SparkSQL utilizando una interfaz similar a los paquetes utilizados por el tidyverse, una popular herramienta de R para el procesamiento de datos. Además, sparklyr ofrece una forma rápida de instalar y configurar Spark en computadoras Windows, tanto para trabajar con clústeres locales como para conectarse a clústeres remotos de Hadoop YARN, Mesos, Kubernetes, Databricks, entre otros.

A continuación se presentan las librerías de R que se utilizarán a lo largo del estudio:

```
# Cargar paquetes necesarios
library(tidyverse)    # Proporciona una colección de paquetes para manipulación de datos
library(sparklyr)     # Proporciona una interfaz para trabajar con Apache Spark
library(pdftools)     # Permite trabajar con archivos PDF en R
library(tidytext)     # Proporciona herramientas para el análisis de texto
library(wordcloud)    # Genera nubes de palabras a partir de un texto
```

Procesamiento texto del articulo.

Dado que el artículo está en formato PDF, utilizaremos la biblioteca 'pdftools' para extraer su texto. Posteriormente, almacenaremos dicho texto en el objeto llamado 'texto'

```
# Extraer el texto de un archivo PDF
texto <- pdf_text("TFG_ROMAN_ZAMORA_CARRERAS.pdf")
```

Vamos a examinar la clase del objeto y su extensión para obtener más detalles al respecto.

```
# Devuelve la clase de la variable "texto", en este caso
class(texto)
```

[1] "character"

```
# Devuelve la longitud de la variable "texto"
length(texto)
```

[1] 71

Observamos que se ha obtenido un vector de tipo "caracter" con una longitud de 71 elementos, lo cual indica que hay un registro por cada página del artículo.

Limpieza de texto

Nuestro primer paso en la limpieza de datos será eliminar la página 5, que contiene el resumen del artículo en inglés. Dado que nuestro objetivo es contar las palabras en español, consideramos pertinente excluir esta página. Además, procederemos a eliminar las páginas 63 a 71, que contienen las referencias bibliográficas. Estas páginas se eliminarán debido a que la mayoría de ellas están en inglés y fueron consultadas en páginas web. Cabe mencionar que, en R, a diferencia de la mayoría de los lenguajes de programación, los índices de los elementos comienzan en 1 en lugar de 0.

```
# Eliminar páginas específicas del texto extraído del PDF
texto <- texto[-c(5,63:71)]

# Devuelve la longitud de la variable "texto"
length(texto)
```

[1] 61

Al eliminar las páginas mencionadas, la longitud de nuestro vector se reduce a 61 observaciones. Ahora, examinemos las dos primeras observaciones para tener una idea del contenido.

```
# Mostrar las primeras dos páginas del texto extraído del PDF
cat(head(texto,2))
```

```
         Universidad Politécnica
               de Madrid
           Escuela Técnica Superior de
             Ingenieros Informáticos
```

Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas

```
           Trabajo Fin de Grado
```

Apoyo Tecnológico en la Inversión en Renta Variable:

```
Indicadores financieros y finanzas sociales
```

Autor: Román Zamora Carreras Tutora: Bárbara Soriano

```
                 Madrid, mayo 2021
```

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas Título: Apoyo Tecnológico en la Inversión en Renta Variable Mayo 2021

Autor: Román Zamora Carreras

Tutor: Bárbara Soriano Centro para la gestión de riesgos agrarios y medioambientales ETSI Informáticos Universidad Politécnica de Madrid

---

Entonces, parece necesario realizar una limpieza adicional, que consiste en eliminar los saltos de línea, números, puntuaciones y caracteres especiales. El objetivo es conservar únicamente el texto que será utilizado por Spark. Utilizaremos la librería “stringr” que forma parte de “tidyverse” para llevar a cabo esta limpieza, siguiendo el siguiente orden:

- Pasar todo el texto a minuscula
- Quitar los saltos de linea
- Dejar solo letras, incluyendo la ñ y todas las vocales con tilde.
- Quitar los dobles espacios.
- Para cada observación (pagína) quitar de existir los espacios al princiío y fin del texto.

Como nota adicional, somos conscientes de que Spark proporciona funciones de limpieza similares. Sin embargo, al intentar utilizar estas funciones, obtuvimos resultados inesperados debido a la forma en que Spark interpreta la codificación del idioma español. Por esta razón, optamos por utilizar las herramientas de R, donde pudimos especificar el encoding específico para este idioma.

```r
# Preprocesamiento del texto
texto <- texto |>
  # Convertir el texto a minúsculas
  str_to_lower() |>
  # Reemplazar saltos de línea por espacios en blanco
  str_replace_all("\n", " ") |>
  # Reemplazar caracteres que no sean letras o blancos por espacios en blanco
  str_replace_all("[^a-záéíóúñ\\s]"," ") |>
  # Reemplazar múltiples espacios en blanco por uno solo
  str_replace_all("\\s+", " ") |>
  # Eliminar espacios en blanco al principio y al final del texto
  str_trim()
```

Veamos nuevamente el resultado con el texto ya limpio.

```r
# Mostrar las primeras dos páginas del texto extraído del PDF
cat(head(texto,2))
```

universidad politécnica de madrid escuela técnica superior de ingenieros informáticos doble grado en ingeniería informática y administración y dirección de empresas trabajo fin de grado apoyo tecnológico en la inversión en renta variable indicadores financieros y finanzas sociales autor román zamora carreras tutora bárbara soriano madrid mayo este trabajo fin de grado se ha depositado en la etsi informáticos de la universidad politécnica de madrid para su defensa trabajo fin de grado doble grado en ingeniería informática y administración y dirección de empresas título apoyo tecnológico en la inversión en renta variable mayo autor román zamora carreras tutor bárbara soriano centro para la gestión de riesgos agrarios y medioambientales etsi informáticos universidad politécnica de madrid

---

Ahora que el texto se encuentra en el formato adecuado para el análisis en Spark, procederemos a unir las 61 páginas en una sola observación para su procesamiento.

```r
# Concatenar el texto en una sola cadena
texto <- paste0(texto, collapse = " ")

# Devuelve la clase de la variable "texto"
class(texto)
```

[1] “character”

```r
# Devuelve la longitud de la cadena de texto en "texto"
length(texto)
```

[1] 1

Análisis en Spark

Llevando los datos a Spark

Como primer paso, vamos a convertir nuestro texto en un DataFrame para simular la estructura que utiliza Spark para el análisis de datos.

```r
# Crear un tibble con la variable "texto"
texto_df <- tibble(texto = texto)

# Obtener la clase del tibble "texto_df"
class(texto_df)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

Ahora nos conectaremos a un clúster local donde procesaremos el texto utilizando Spark.

```r
# Establecer la variable de entorno JAVA_HOME
Sys.setenv(JAVA_HOME="C:/Program Files/Java/jre-1.8")

# Conectar con Spark localmente
sc <- spark_connect(master = "local")
```

En este momento, nuestro clúster local se encuentra vacío, como se puede observar en la Figura 1.

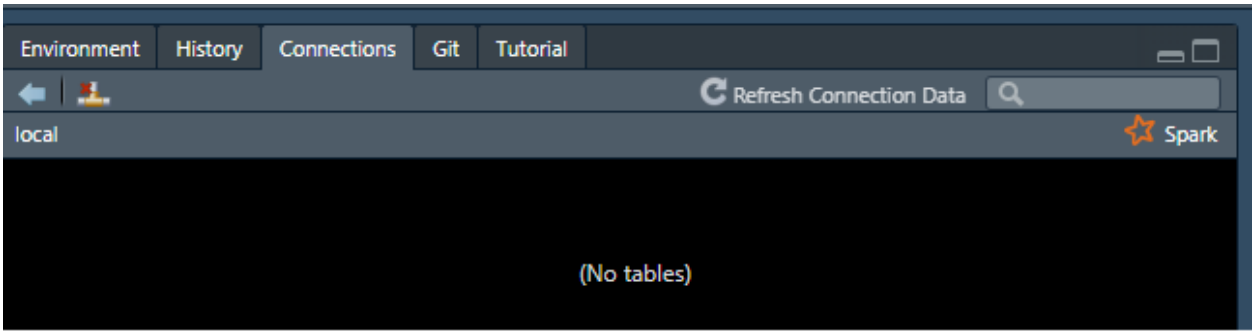

Figure 1: Cluster Vacio

Copiaremos nuestro DataFrame de R a Spark utilizando la función copy_to de sparklyr.

```r
# Copiar el tibble al entorno Spark
texto_df_spk <- copy_to(sc, texto_df,"texto_df_spk", overwrite = TRUE)

# Obtener la clase del tibble Spark
class(texto_df_spk)
```

```
## [1] "tbl_spark" "tbl_sql"   "tbl_lazy"  "tbl"
```

En la Figura 2, podemos observar que nuestro DataFrame texto_df_spk de Spark ya se encuentra dentro del clúster.

Tokenización del texto

Una vez que tenemos nuestro texto en Spark, procederemos primero a separar cada una de las palabras en una lista utilizando la función ft_tokenizer (que utiliza internamente la función tokenizer de Spark). Luego, convertiremos cada elemento de esta lista en una fila de un DataFrame utilizando la función explode de Spark. El resultado se guardará en el DataFrame palabras_df_spk dentro de nuestro clúster de Spark.

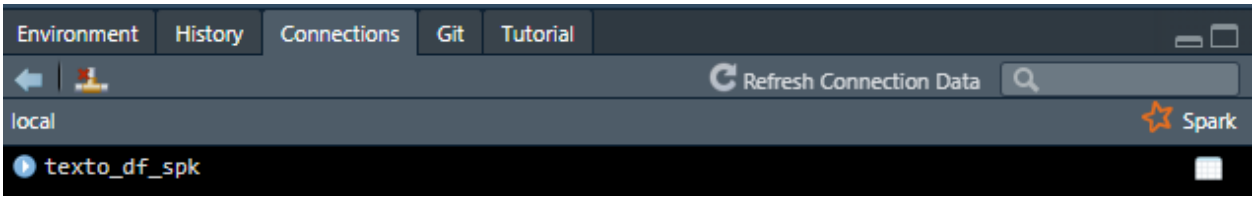

Figure 2: Adición de DF texto_df_spk al cluster

```
# Tokenizar el texto en palabras utilizando sparklyr
palabras_df_spk <- texto_df_spk %>%
  ft_tokenizer(
    # texto a tokenizar
    input_col = "texto", #texto a tokenizar
    # Columna de salida que contendrá la lista de palabras tokenizadas
    output_col = "lista_palabras"
  ) |>
  # Expandir la lista de palabras en filas individuales
  mutate(palabra = explode(lista_palabras)) |>
  # Seleccionar solo la columna 'palabra'
  select(palabra) |>
  # Realizar el cómputo y generar un nuevo tibble Spark llamado 'palabras_df_spk'
  compute("palabras_df_spk")
```

En la Figura 3 se puede apreciar la inclusión del DataFrame palabras_df_spk en nuestro clúster de Spark.

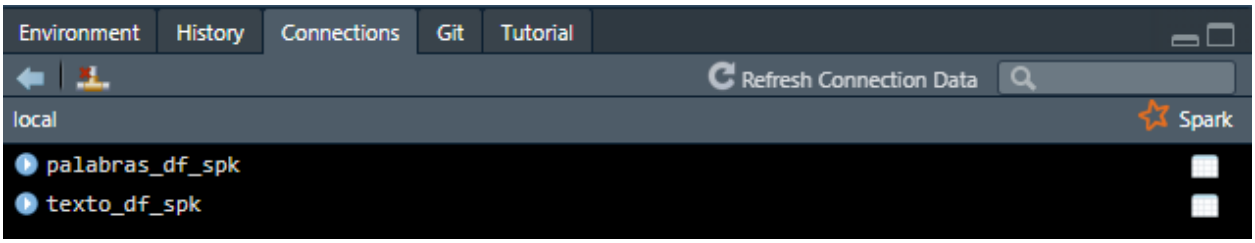

Figure 3: Adición de DF palabras_df_spk al cluster

Vamos a revisar las primeras 10 filas del DataFrame palabras_df_spk.

```
# Mostrar las primeras 10 filas del tibble 'palabras_df_spk'
head(palabras_df_spk, 10)
```

```
## # Source: spark<?> [?? x 1]
##    palabra
##    <chr>
##  1 universidad
##  2 politécnica
##  3 de
##  4 madrid
##  5 escuela
##  6 técnica
##  7 superior
##  8 de
```

```
##  9 ingenieros
## 10 informáticos
```

Conteo de palabras

Nuestro siguiente paso consiste en generar una agregación a partir de este DataFrame, el cual contiene en cada fila una palabra del artículo en orden. Queremos contar el número de veces que aparece cada una de estas palabras en el artículo. Para lograr esto, utilizaremos la función count de Spark sobre la columna palabra. El resultado lo almacenaremos en un nuevo DataFrame llamado conteo_palabras, el cual estará disponible en el clúster.

```
# Contar la frecuencia de las palabras en 'palabras_df_spk'
conteo_palabras <- palabras_df_spk |>
  # Contar la frecuencia de cada palabra en la columna 'palabra'
  count(palabra) |>
  # Desagrupar el tibble después de contar las palabras
  ungroup() |>
  # Realizar el cálculo y generar un nuevo tibble Spark llamado 'conteo_palabras'
  compute("conteo_palabras")
```

En la Figura 4 se puede observar la inclusión del DataFrame conteo_palabras en nuestro clúster.

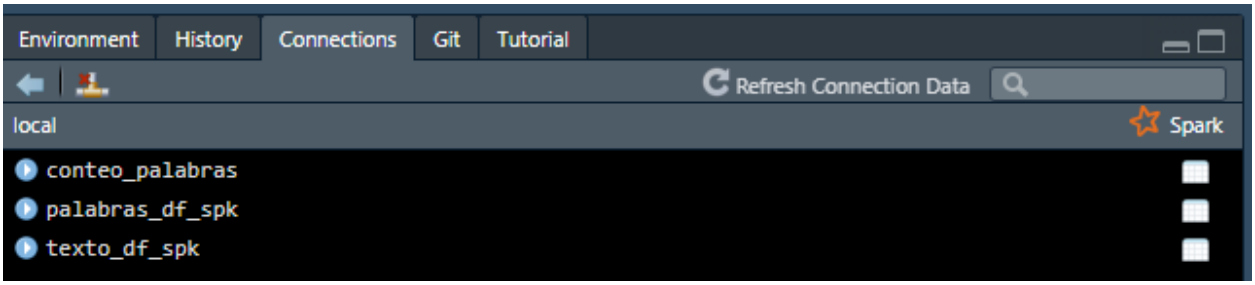

Figure 4: Adición de DF conteo_palabras al cluster

Vamos a revisar el top 10 de las palabras más utilizadas en el artículo.

```
# Mostrar las 10 palabras más frecuentes en 'conteo_palabras'
conteo_palabras |>
  # Ordenar el tibble por la columna 'n' en orden descendente
  arrange(-n) |>
  # Mostrar las primeras 10 filas del tibble resultante
  head(10)
```

```
## # Source:     spark<?> [?? x 2]
## # Ordered by: -n
##    palabra     n
##    <chr>   <dbl>
##  1 de       1413
##  2 la        739
##  3 el        572
##  4 en        539
##  5 a         392
##  6 y         359
```

```
##  7 los       329
##  8 que       314
##  9 se        272
## 10 un        265
```

Es comprensible que el top 10 actual no sea muy revelador, ya que contiene palabras comunes del idioma español. En este caso, es necesario realizar una limpieza adicional para eliminar las palabras que se utilizan con mayor frecuencia y que no aportan información relevante al análisis.

Remoción de Stop Words

Las "stop words" son palabras comunes y frecuentes en un idioma que se suelen excluir o filtrar durante el procesamiento o análisis de texto. Estas palabras se consideran irrelevantes en el análisis de texto debido a su alta frecuencia de aparición y su bajo valor semántico o informativo.

Por lo tanto, es necesario eliminar estas "stop words" en nuestro análisis para que el top de palabras de nuestro artículo tenga un sentido más claro y relevante.

Para llevar a cabo esta tarea, utilizaremos la lista de "stop words" en español proporcionada por el paquete de R llamado tidytext. Esta lista incluye las 308 palabras más utilizadas en el idioma español, según el Lexicon Snowball, generado por el algoritmo homónimo desarrollado por el Dr. Martin Porter. Si deseas obtener más información acerca de este lexicon, puedes visitar el sitio web oficial en https://snowballstem.org/.

En el siguiente código, almacenaremos estas 308 palabras en el objeto

```
# Obtener las palabras de parada en español
stop_words_es <- get_stopwords("es")
```

Vamos a revisar las primeras 10 palabras de este objeto.

```
# Mostrar las primeras 10 palabras de parada en español
head(stop_words_es,10)
```

```
## # A tibble: 10 x 2
##    word  lexicon
##    <chr> <chr>
##  1 de    snowball
##  2 la    snowball
##  3 que   snowball
##  4 el    snowball
##  5 en    snowball
##  6 y     snowball
##  7 a     snowball
##  8 los   snowball
##  9 del   snowball
## 10 se    snowball
```

Observamos que varias de las 10 primeras palabras de esta lista coinciden con el top 10 de las palabras más utilizadas en el artículo. Por lo tanto, tiene sentido eliminar estas palabras del artículo para realizar un análisis adecuado.

Nuestro siguiente paso será convertir esta lista en un DataFrame llamado stop_words_es_spk dentro de nuestro clúster.

```
# Copiar el conjunto de palabras de parada en español al entorno Spark
stop_words_es_spk <- copy_to(sc, stop_words_es,"stop_words_es_spk", overwrite = TRUE)
```

En la Figura 5 se muestra la inclusión del DataFrame stop_words_es_spk en nuestro clúster.

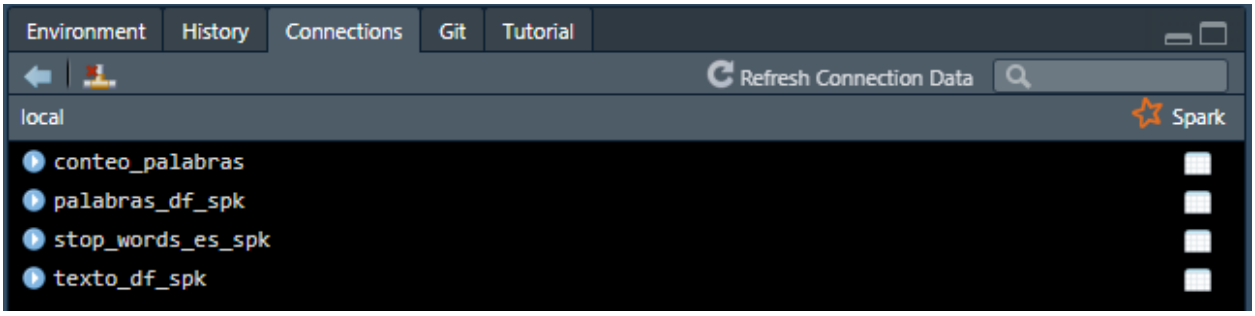

Figure 5: Adición de DF stop_words_es_spk al cluster

Nuestro siguiente paso será crear el DataFrame conteo_palabras_real, en el cual eliminaremos las palabras contenidas en stop_words_es_spk del DataFrame conteo_palabras. Para lograr esto, utilizaremos la función anti_join de sparklyr, la cual utiliza internamente la funcionalidad de join de tipo "anti" en Spark.

```
# Obtener el conteo de palabras sin las palabras de parada en español
conteo_palabras_real <- conteo_palabras |>
  # Realizar una anti-unión entre 'conteo_palabras' y 'stop_words_es_spk'
  # por las columnas 'palabra' y 'word'
  anti_join(stop_words_es_spk,
            by = c("palabra" = "word")) |>
  # Realizar el cálculo y generar un nuevo tibble Spark llamado 'conteo_palabras_real'
  compute("conteo_palabras_real")
```

En la Figura 6 se muestra la inclusión del DataFrame stop_words_es_spk en nuestro clúster.

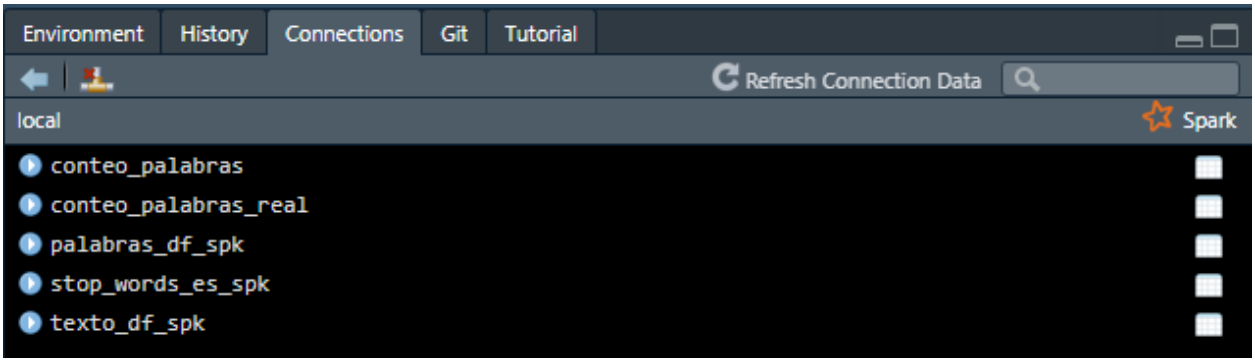

Figure 6: Adición de DF stop_words_es_spk al cluster

Vamos a revisar el nuevo top 10 de las palabras más utilizadas en el artículo una vez que se han eliminado las palabras más comunes del español.

```
# Mostrar las 10 palabras más frecuentes sin las palabras de parada en español
conteo_palabras_real |>
  # Ordenar el tibble por la columna 'n' en orden descendente
```

```
  arrange(-n) |>
  # Mostrar las primeras 10 filas del tibble resultante
  head(10)
```

```
## # Source:     spark<?> [?? x 2]
## # Ordered by: -n
##    palabra         n
##    <chr>       <dbl>
##  1 mercado       116
##  2 inversión     107
##  3 inversor      103
##  4 empresa        91
##  5 activo         83
##  6 activos        77
##  7 análisis       72
##  8 puede          60
##  9 indicadores    59
## 10 empresas       55
```

Nuestro nuevo top 10 tiene más sentido y, con solo esas primeras 10 palabras, podemos obtener una idea de la temática del artículo sin haberlo leído previamente.

Como observación, notamos que dentro del top 10 hay palabras que tienen un origen similar, como "inversión", "inversor" y "activo", "activos". Esto se debe a que en la limpieza del texto no se realizó un proceso de lematización de las palabras. La lematización requeriría un análisis más profundo y metodologías avanzadas que están fuera del alcance del presente estudio.

WordClod

Para finalizar nuestro estudio, visualizaremos nuestros resultados mediante un WordCloud. Un WordCloud es una representación visual de un conjunto de palabras, donde el tamaño de cada palabra está relacionado con su frecuencia o importancia en un texto o conjunto de datos.

En primer lugar, extraeremos el DataFrame conteo_palabras_real de nuestro clúster de Spark y lo llevaremos a R en el DataFrame conteo_palabras_real_R. Utilizaremos esta herramienta para realizar la visualización mediante el WordCloud.

```
# Recopilar el tibble Spark 'conteo_palabras_real' en R
conteo_palabras_real_R <- collect(conteo_palabras_real)
```

Vamos a revisar el tamaño de nuestro DataFrame conteo_palabras_real_R.

```
# Obtener las dimensiones del tibble 'conteo_palabras_real_R'
dim(conteo_palabras_real_R)
```

```
## [1] 3092    2
```

Observamos que el DataFrame conteo_palabras_real_R contiene un total de 3092 palabras. Para nuestra visualización, utilizaremos únicamente las primeras 100 palabras, las cuales creemos que capturan la esencia del artículo. Asignaremos un tono verde a las palabras más utilizadas, seguido de azul, naranja y finalmente gris para representar las palabras menos utilizadas dentro de estas primeras 100 palabras más frecuentes.

```r
# Generar un wordcloud con las 100 palabras más frecuentes en 'conteo_palabras_real_R'
with(conteo_palabras_real_R |> # Se utiliza el tibble en R 'conteo_palabras_real_R'
       # Ordenar el tibble por la columna 'n' en orden descendente
       arrange(-n) |>
       # Seleccionar las primeras 100 filas del tibble resultante
       head(100),
     wordcloud(
       # Columna 'palabra' como el texto a mostrar en el wordcloud
       palabra,
       # Columna 'n' como la frecuencia de las palabras
       n,
       # Escala del tamaño de las palabras en el wordcloud
       scale = c(2.5,.5),
       # Colores utilizados en el wordcloud
       colors = c("#999999", "#E69F00", "#56B4E9", "#5DAE68")
     ))
```

En nuestra visualización, podemos observar que se hace hincapié en palabras como “inversión” y “empresas”, seguidas por “activos”, “análisis” e “indicadores”. Una vez más, confirmamos que este conteo de palabras nos ayuda a comprender la temática del artículo sin necesidad de leerlo. Esto demuestra el poder que puede tener el análisis de texto en el campo de la ciencia de datos, ya que con un ejercicio básico como el presente, pudimos extraer la temática de un artículo sin necesidad de leerlo.

Como paso final, nos desconectaremos de nuestro clúster para evitar seguir consumiendo recursos. Es importante realizar esta desconexión para garantizar un uso eficiente de los recursos disponibles.

```
# Desconectar de Spark
spark_disconnect(sc)
```

Reflexión: La importancia de Hadoop y Spark en la gestión de datos en la Ciencia de Datos.

La gestión y análisis de datos en el ámbito de la Ciencia de Datos se ha vuelto cada vez más crucial en nuestra era digital. Con el crecimiento exponencial de la cantidad de datos generados, surge la necesidad de herramientas eficientes y escalables que permitan procesar y extraer información valiosa de estos vastos conjuntos de datos.

En este contexto, herramientas como Hadoop y Spark desempeñan un papel fundamental al ofrecer capacidades de procesamiento distribuido y manejo de grandes volúmenes de datos. Estas herramientas permiten a los científicos de datos abordar desafíos complejos en términos de escala, velocidad y variedad de datos.

Hadoop, basado en el modelo de programación MapReduce, se destaca por su escalabilidad y capacidad de manejar grandes conjuntos de datos en forma de lotes. Es especialmente útil para análisis retrospectivos y tareas en las que no se requiere una respuesta en tiempo real. Al dividir el procesamiento en etapas de map y reduce distribuidas en un clúster, Hadoop ofrece una solución robusta y tolerante a fallos.

Por otro lado, Spark se basa en el modelo de cómputo en memoria y es conocido por su velocidad y capacidad de procesamiento en tiempo real. Esto hace que Spark sea una opción valiosa cuando se necesita un análisis interactivo y en tiempo real de datos. Además, la flexibilidad de Spark al admitir múltiples lenguajes de programación y su capacidad para acceder a diversos sistemas de almacenamiento distribuido le brindan a los científicos de datos más opciones y flexibilidad en su trabajo.

La elección de utilizar Hadoop o Spark depende de los requisitos y características específicas de cada casi. Si el análisis se realiza en lotes de datos y no se necesita una respuesta en tiempo real, Hadoop ofrece escalabilidad y tolerancia a fallos. Por otro lado, si se requiere un análisis interactivo y en tiempo real, Spark se destaca por su velocidad y capacidad de procesamiento en memoria.