



3 “R” para ciencia de datos.

3.1 Análisis de sentimientos en “R”

- 3.1.1 Un paseo por el software “R”.
- 3.1.2 Paquetería para Ciencia de Datos.
- 3.1.3 Caso práctico: El estado de ánimo de mi libro o la noticia del día.
- 3.1.4 Codificación en “R” v. Corpus del documento.
- 3.1.5 Palabras de opinión negativa.

3.2 Minería de datos en Twitter para sentimiento análisis.

- 3.2.1 Creación de cuenta para desarrollador, Application Programming Interface (API).
- 3.2.2 Paquetería para Twitter Mining.
- 3.2.3 Caso práctico: El estado de ánimo de los tuiteros respecto al “NAFTA”, “Trump”.

R es un lenguaje de programación y un entorno de software diseñado específicamente para el análisis estadístico y la visualización de datos. Su versatilidad radica en su capacidad para manejar una amplia gama de tareas relacionadas con la Ciencia de Datos, desde la importación y manipulación de datos hasta el desarrollo de modelos estadísticos complejos.

Manipulación de Datos

Una de las fortalezas de R es su capacidad para manejar datos en una variedad de formatos, incluidos CSV, Excel, bases de datos SQL, JSON y más. Los paquetes como dplyr y tidyverse proporcionan herramientas poderosas para filtrar, transformar y unir conjuntos de datos de manera eficiente, permitiendo a los usuarios limpiar y preparar sus datos para el análisis.



Visualización de Datos

R ofrece una amplia gama de opciones para visualizar datos de manera efectiva. La librería ggplot2, por ejemplo, permite crear gráficos de alta calidad y altamente personalizables con una sintaxis simple y elegante. Desde diagramas de dispersión hasta gráficos de barras y mapas de calor, R proporciona las herramientas necesarias para explorar y comunicar patrones y tendencias en los datos de manera efectiva.

Modelado Estadístico

Con paquetes como stats y lme4, R permite a los científicos de datos desarrollar una amplia variedad de modelos estadísticos, desde regresiones lineales simples hasta análisis de series temporales y modelos de efectos mixtos. Además, la integración con herramientas como R Markdown facilita la generación de informes y documentos reproducibles que documentan todo el proceso de análisis, desde la carga inicial de los datos hasta la interpretación final de los resultados del modelo.

Comunidad y Ecosistema

Una de las razones fundamentales detrás del éxito de R en el campo de la Ciencia de Datos es su próspera comunidad de usuarios y desarrolladores. La comunidad R es conocida por su colaboración, innovación y apoyo mutuo, lo que se refleja en la amplia variedad de paquetes y recursos disponibles. Desde foros en línea y grupos de usuarios locales hasta conferencias y tutoriales en línea, los usuarios de R tienen acceso a una red global de expertos y recursos que pueden ayudarlos a resolver problemas y mejorar sus habilidades.

Además, el ecosistema de R se beneficia de la contribución activa de desarrolladores de software y académicos de todo el mundo, lo que garantiza que el lenguaje y sus herramientas sigan evolucionando para satisfacer las necesidades cambiantes de la comunidad de Ciencia de Datos.



En R, hay varias bibliotecas de código (llamadas "paquetes") que están orientadas a la ciencia de datos. Algunas de las más populares incluyen:

- **tidyverse:** No es solo una biblioteca, sino un conjunto de paquetes diseñados para trabajar de manera coherente y eficiente en tareas de manipulación, visualización y modelado de datos. Incluye paquetes como ggplot2 para visualización, dplyr para manipulación de datos, tidyr para reestructuración de datos, entre otros.
- **caret:** Es una biblioteca para entrenamiento y evaluación de modelos de aprendizaje automático. Proporciona una interfaz unificada para una variedad de algoritmos de aprendizaje automático y herramientas para la selección de modelos, validación cruzada y preprocesamiento de datos.
- **ggplot2:** Es una biblioteca para la creación de gráficos estadísticos basada en la gramática de gráficos. Permite crear visualizaciones complejas y atractivas con una sintaxis simple y flexible.
- **dplyr:** Es una biblioteca para la manipulación de datos tabulares. Proporciona funciones intuitivas y eficientes para realizar operaciones comunes como filtrado, selección, agrupación y resumen de datos.
- **tidyr:** Es una biblioteca para la reestructuración de datos. Permite convertir datos entre formatos "anchos" y "largos", lo que facilita el análisis y la visualización de datos.
- **rpart:** Implementa el algoritmo de árboles de decisión recursivos para análisis predictivo. Es útil para la construcción de modelos predictivos simples y fácilmente interpretables.
- **randomForest:** Es una implementación del algoritmo de bosques aleatorios para aprendizaje automático. Es útil para la construcción de modelos predictivos robustos y precisos.
- **glmnet:** Es una biblioteca para la regularización de modelos de regresión lineal y logística. Proporciona métodos eficientes para ajustar modelos de regresión con penalizaciones L1 (lasso) y L2 (ridge).

```
#Instalar los paquetes necesarios si aún no los tienes:  
install.packages("tidyverse")  
install.packages("IRkernel")  
IRkernel::installspec()
```

```
#Cargar el paquete  
library(tidyverse)
```

```
#Carga tus datos. Puedes usar un conjunto de datos de muestra, como el conjunto de datos iris que está incluido en R:  
data(iris)
```

```
#Explora tus datos utilizando funciones del tidyverse. Por ejemplo, puedes usar ggplot2 para crear un gráfico:  
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +  
  geom_point() +  
  labs(title = "Relación entre longitud y ancho de sépalos, estructura floral, por especie",  
       x = "Longitud del sépalo",  
       y = "Ancho del sépalo")
```

```
#Esto te dará un gráfico de dispersión de longitud de sépalo vs. ancho de sépalo, coloreado por especie.
```

tidyverse: Conjunto de paquetes diseñados para trabajar de manera coherente y eficiente en tareas de manipulación, visualización y modelado de datos. Incluye paquetes como ggplot2 para visualización, dplyr para manipulación de datos, tidyr para reestructuración de datos, entre otros.



```
# Instalar e importar el paquete caret
install.packages("caret")
library(caret)

# Cargar un conjunto de datos de ejemplo (por ejemplo, el conjunto de datos iris)
data(iris)

# Dividir el conjunto de datos en entrenamiento y prueba
set.seed(123)
trainIndex <- createDataPartition(iris$Species, p = .8,
                                 list = FALSE,
                                 times = 1)
data_train <- iris[trainIndex,]
data_test <- iris[-trainIndex,]

# Crear un modelo de clasificación (por ejemplo, usando el método de clasificación de árboles de decisión)
model <- train(Species ~ ., data = data_train, method = "rpart")

# Hacer predicciones en el conjunto de prueba
predictions <- predict(model, newdata = data_test)

# Evaluar el rendimiento del modelo
confusionMatrix(predictions, data_test$Species)
```

```
install.packages("ggplot2")
install.packages("IRdisplay")

# Carga de librerías
library(ggplot2)
library(IRdisplay)

# Creación de datos de ejemplo
datos <- data.frame(
  Sexo = c("Masculino", "Femenino", "Masculino", "Femenino"),
  Edad = c(25, 30, 35, 40),
  Puntuación = c(70, 85, 80, 75)
)

# Crear un gráfico de dispersión
scatter_plot <- ggplot(datos, aes(x = Edad, y = Puntuación, color = Sexo)) +
  geom_point(size = 3) +
  labs(title = "Puntuación por Edad y Sexo",
       x = "Edad",
       y = "Puntuación",
       color = "Sexo") +
  theme_minimal()

# Mostrar el gráfico
print(scatter_plot)

# Para mostrar el gráfico en el Jupyter Notebook
display_png(print(scatter_plot))
```

caret: Es una biblioteca para entrenamiento y evaluación de modelos de aprendizaje automático. Proporciona una interfaz unificada para una variedad de algoritmos de aprendizaje automático y herramientas para la selección de modelos, validación cruzada y preprocesamiento de datos.

ggplot2: Es una biblioteca para la creación de gráficos estadísticos basada en la gramática de gráficos. Permite crear visualizaciones complejas y atractivas con una sintaxis simple y flexible.



```
install.packages("tidyverse")
install.packages("IRkernel")
IRkernel::installspec(user = FALSE)

library(dplyr)

# Creamos un data frame de ejemplo
datos <- data.frame(
  ID = c(1, 2, 3, 4, 5),
  Nombre = c("Juan", "María", "Pedro", "Ana", "Luisa"),
  Edad = c(25, 30, 35, 40, 45),
  Puntuacion = c(80, 90, 75, 85, 95)
)

# Filtramos las filas donde la edad es mayor que 30
datos_filtrados <- datos %>%
  filter(Edad > 30)

# Mostramos el resultado
print(datos_filtrados)
```

```
# Instalación de paquetes (si no están instalados)
if(!requireNamespace("tidyverse", quietly = TRUE)) {
  install.packages("tidyverse")
}

# Carga del paquete tidyverse
library(tidyverse)

# Ejemplo de datos
datos <- data.frame(
  id = c(1, 2, 3),
  nombre = c("Juan", "María", "Pedro"),
  telefono = c("123-456-7890", "456-789-0123", "789-012-3456"),
  direccion = c("123 Calle A", "456 Calle B", "789 Calle C")
)

# Visualización de los datos originales
print("Datos originales:")
print(datos)

# Separación del número de teléfono en tres columnas: codigo_area, primer_dígito y resto
datos_separados <- separate(datos, telefono, into = c("codigo_area", "primer_dígito", "resto"), sep = "-")

# Visualización de los datos después de la separación
print("Datos después de la separación del número de teléfono:")
print(datos_separados)
```

dplyr: Es una biblioteca para la manipulación de datos tabulares. Proporciona funciones intuitivas y eficientes para realizar operaciones comunes como filtrado, selección, agrupación y resumen de datos.

Este código crea un data frame con información de personas, luego filtra las filas donde la edad es mayor que 30 utilizando la función filter() de dplyr, y finalmente muestra el resultado.

Permite agregar más operaciones como agrupar, resumir datos, entre otras, utilizando las funciones proporcionadas por dplyr.

tidyverse: Es una biblioteca para la reestructuración de datos. Permite convertir datos entre formatos "anchos" y "largos", lo que facilita el análisis y la visualización de datos.

Este código muestra un ejemplo básico de cómo utilizar el paquete tidyverse en R para separar una columna de datos en múltiples columnas.



```
# Instala el paquete si no lo tienes instalado
if (!require("rpart")) {
  install.packages("rpart")
}

# Carga la librería
library(rpart)

# Cargamos un conjunto de datos de ejemplo (por ejemplo, el conjunto de datos iris)
data(iris)

# Creamos un árbol de decisión utilizando rpart
modelo_arbol <- rpart(Species ~ ., data = iris, method = "class")

# Visualizamos el árbol de decisión
plot(modelo_arbol)
text(modelo_arbol, pretty = 0)

# Evaluamos la precisión del modelo
predicciones <- predict(modelo_arbol, newdata = iris, type = "class")
precision <- mean(predicciones == iris$Species)
cat("Precisión del modelo:", precision)

# Instalar e importar el paquete randomForest
install.packages("randomForest")
library(randomForest)

# Cargar conjunto de datos de ejemplo (por ejemplo, el dataset iris)
data(iris)

# Dividir los datos en conjunto de entrenamiento y conjunto de prueba
set.seed(123)
indices <- sample(2, nrow(iris), replace = TRUE, prob = c(0.7, 0.3))
train_data <- iris[indices == 1, ]
test_data <- iris[indices == 2, ]

# Entrenar el modelo de bosque aleatorio
rf_model <- randomForest(Species ~ ., data = train_data)

# Realizar predicciones en el conjunto de prueba
predictions <- predict(rf_model, test_data)

# Calcular la precisión de las predicciones
accuracy <- sum(predictions == test_data$Species) / nrow(test_data)
print(paste("Precisión del modelo:", round(accuracy * 100, 2), "%"))

# Visualizar la importancia de las variables
importance <- importance(rf_model)
varImpPlot(rf_model)
```

rpart: Implementa el algoritmo de árboles de decisión recursivos para análisis predictivo. Es útil para la construcción de modelos predictivos simples y fácilmente interpretables.

randomForest: Es una implementación del algoritmo de bosques aleatorios para aprendizaje automático. Es útil para la construcción de modelos predictivos robustos y precisos.



```
# Instalar el paquete glmnet si aún no está instalado
if(!require(glmnet)){
  install.packages("glmnet")
}

# Cargar la biblioteca glmnet
library(glmnet)

# Generar datos de ejemplo
set.seed(123)
n <- 100 # Número de observaciones
p <- 10 # Número de predictores
x <- matrix(rnorm(n * p), nrow = n, ncol = p) # Matriz de predictores
beta <- rnorm(p) # Verdadero coeficiente beta
y <- x %*% beta + rnorm(n) # Variable de respuesta

# Ajustar modelo de regresión LASSO usando glmnet
lasso_model <- glmnet(x, y, alpha = 1) # alpha = 1 para LASSO

# Graficar el camino de regularización
plot(lasso_model, xvar = "lambda", label = TRUE)

# Seleccionar el valor óptimo de lambda usando validación cruzada
cv_model <- cv.glmnet(x, y, alpha = 1) # alpha = 1 para LASSO
best_lambda <- cv_model$lambda.min
best_lambda

# Ajustar el modelo final usando el valor óptimo de lambda
lasso_model_final <- glmnet(x, y, alpha = 1, lambda = best_lambda)

# Resumen del modelo final
summary(lasso_model_final)

# Predicciones usando el modelo final
predictions <- predict(lasso_model_final, newx = x)

# Calcular el error cuadrático medio
mse <- mean((y - predictions)^2)
mse
```

glmnet: Es una biblioteca para la regularización de modelos de regresión lineal y logística. Proporciona métodos eficientes para ajustar modelos de regresión con penalizaciones L1 (lasso) y L2 (ridge).

El análisis de sentimientos en "R" es un proceso mediante el cual se utiliza el lenguaje de programación R y sus herramientas para analizar textos y extraer información sobre las emociones, opiniones o actitudes expresadas en ellos. Este análisis se realiza generalmente en texto escrito, como comentarios en redes sociales, reseñas de productos, opiniones en blogs, entre otras fuentes de información.

Preprocesamiento de texto: Este paso implica limpiar y preparar el texto para el análisis. Esto puede incluir la eliminación de signos de puntuación, palabras vacías (stopwords), lematización o derivación de palabras, etc.

Creación de un diccionario de sentimientos: Se utiliza un diccionario que asocia palabras con su polaridad de sentimiento (positivo, negativo o neutro). Estos diccionarios pueden ser elaborados manualmente o estar disponibles en paquetes de análisis de sentimientos en R.

Asignación de polaridad de sentimiento: Se asigna a cada palabra del texto una polaridad de sentimiento según el diccionario utilizado. Por ejemplo, palabras como "bueno", "feliz" tendrían una polaridad positiva, mientras que palabras como "malo", "triste" tendrían una polaridad negativa.



Sumarización de polaridad: Se suma la polaridad de todas las palabras en el texto para obtener una medida general del sentimiento expresado en el texto.

Análisis y visualización de resultados: Se analizan los resultados para entender las tendencias de sentimiento en el texto y se pueden visualizar utilizando gráficos, como histogramas o series de tiempo.

Existen varios paquetes en R que facilitan el análisis de sentimientos, como tidytext, sentimentr, syuzhet, entre otros. Estos paquetes ofrecen funciones y herramientas específicas para llevar a cabo cada uno de los pasos mencionados anteriormente.

```
# Instala los paquetes necesarios si aún no los tienes instalados
install.packages("tidyverse")
install.packages("jupyter")

# Carga los paquetes necesarios
library(tidyverse)
library(tidytext)

# Carga el conjunto de datos de reseñas de películas (este es solo un ejemplo, puedes usar tus propios datos)
data("imdb_reviews")

# Visualiza las primeras filas del conjunto de datos
head(imdb_reviews)

# Carga el diccionario de palabras con polaridades de sentimiento
data("sentiments")

# Realiza el análisis de sentimientos
imdb_sentiments <- imdb_reviews %>%
  unnest_tokens(word, review) %>%
  inner_join(sentiments) %>%
  count(title, index = 1) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment_score = positive - negative)

# Visualiza los resultados
head(imdb_sentiments)

# Grafica los resultados
ggplot(imdb_sentiments, aes(x = reorder(title, sentiment_score), y = sentiment_score, fill = sentiment_score > 0)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Análisis de Sentimiento en Reseñas de Películas",
      x = "Película",
      y = "Puntuación de Sentimiento",
      fill = "Sentimiento Positivo")
```

Tidytext: tidytext es un paquete de R que se utiliza para realizar análisis de texto de manera eficiente y estructurada. Este paquete está diseñado para trabajar con datos de texto en formato tidy, que es un formato en el que cada palabra o término está en una fila individual del conjunto de datos. tidytext proporciona una serie de funciones que facilitan el procesamiento de texto, la exploración y visualización de datos de texto, así como la realización de análisis de sentimientos y otros análisis de texto avanzados. Es particularmente útil para trabajar con texto en combinación con el paquete dplyr para manipulación de datos y ggplot2 para visualización.

En este ejemplo, estamos utilizando un conjunto de datos de reseñas de películas proporcionado por el paquete tidytext. Luego, utilizamos la función unnest_tokens para dividir cada reseña en palabras individuales y asociarlas con su respectiva película. Después, unimos estas palabras con el diccionario de sentimientos (sentiments) y contamos la frecuencia de palabras positivas y negativas para cada película. Luego, calculamos una puntuación de sentimiento restando las palabras negativas de las positivas. Finalmente, visualizamos los resultados con un gráfico de barras que muestra la puntuación de sentimiento para cada película.



Sentimentr: sentimentr es otro paquete de R que se utiliza específicamente para realizar análisis de sentimientos en textos. A diferencia de algunos otros enfoques de análisis de sentimientos que se basan en diccionarios predefinidos de palabras asociadas con polaridades de sentimientos, sentimentr utiliza un enfoque basado en algoritmos para determinar la polaridad de los textos. El paquete proporciona funciones para analizar el sentimiento en textos en inglés, considerando tanto la polaridad de las palabras como la negación y la intensidad de las emociones expresadas.

Este código calculará el sentimiento del texto utilizando la función sentiment() del paquete sentimentr y mostrará los resultados en el notebook. Los resultados incluirán la puntuación de sentimiento para cada oración, así como la puntuación agregada para todo el texto.

```
# Instala el paquete sentimentr si aún no lo has hecho
install.packages("sentimentr")

# Carga la librería
library(sentimentr)

# Texto de ejemplo
texto <- "Me encanta este producto, es increíblemente útil y fácil de usar. Sin embargo, el servicio al cliente deja mucho que desear."

# Analiza el sentimiento del texto
sentimientos <- sentiment(texto)

# Muestra los resultados
print(sentimientos)
```

```
install.packages("syuzhet")
install.packages("ggplot2")

library(syuzhet) library(ggplot2)

# Ejemplo de texto
texto <- "Me encanta este producto, es increíblemente útil y fácil de usar. ¡Altamente recomendado!"

# Tokenizar el texto en oraciones
oraciones <- get_sentences(texto)

# Mostrar las oraciones
oraciones

# Obtener los puntajes de sentimiento
sentimientos <- get_sentiment(oraciones, method = "syuzhet")

# Mostrar los puntajes de sentimiento
sentimientos

# Crear un data frame con los puntajes de sentimiento
df <- data.frame(Oración = oraciones, Sentimiento = sentimientos)

# Graficar los puntajes de sentimiento
ggplot(df, aes(x = reorder(Oración, Sentimiento), y = Sentimiento)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  coord_flip() +
  labs(title = "Puntajes de Sentimiento por Oración",
       x = "Oración",
       y = "Sentimiento")
```

Syuzhet: syuzhet es un paquete de R que se utiliza para extraer la polaridad de los textos utilizando diferentes diccionarios de sentimientos. Este paquete proporciona funciones para analizar la subjetividad y la polaridad de textos en inglés utilizando diferentes enfoques y diccionarios predefinidos. syuzhet puede ser útil para analizar sentimientos en textos literarios, discursos, redes sociales y otros tipos de texto en inglés. El paquete incluye varios diccionarios de sentimientos que pueden ser utilizados para determinar la polaridad de las palabras en los textos.



```
# Instala los paquetes necesarios si no los tienes instalados
install.packages("tm")
install.packages("SentimentAnalysis")

# Carga las librerías
library(tm)
library(SentimentAnalysis)
```

```
# Carga el conjunto de datos de opiniones de películas (en este caso, voy a cargar un conjunto de datos aleatorio)
data("acq")
```

```
# Crea un corpus de texto
corpus <- Corpus(VectorSource(acq))
```

```
# Preprocesamiento del texto
corpus <- tm_map(corpus, content_transformer(tolower)) # Convierte el texto a minúsculas
corpus <- tm_map(corpus, removePunctuation) # Elimina la puntuación
corpus <- tm_map(corpus, removeNumbers) # Elimina los números
corpus <- tm_map(corpus, removeWords, stopwords("en")) # Elimina las stopwords en inglés
```

```
# Realiza el análisis de sentimiento
sentiment <- analyzeSentiment(corpus)
```

```
# Muestra los resultados
head(sentiment)
```

tm: Este paquete es útil para la minería de textos y proporciona herramientas para preprocesar datos de texto, lo que puede ser útil en el análisis de sentimientos.

Este es un ejemplo básico de cómo realizar un análisis de sentimiento en R utilizando el paquete tm. Recuerda que la precisión del análisis puede variar según el conjunto de datos y la complejidad del problema.

```
# Instalar y cargar el paquete quanteda si aún no lo tienes instalado
install.packages("quanteda")
library(quanteda)

# Crear un corpus con algunos textos de ejemplo
texto <- c("Me encanta este producto, es excelente.",
          "No estoy seguro de cómo me siento acerca de esto.",
          "Estoy muy decepcionado con el servicio al cliente.",
          "Qué gran experiencia! Definitivamente recomendaría este lugar.",
          "No puedo soportar esta aplicación, es terrible.")

corpus <- corpus(texto)

# Crear un diccionario de palabras con sus polaridades de sentimiento
diccionario_sentimiento <- dictionary(list(
  positivo = c("encanta", "excelente", "gran", "recomendaría"),
  negativo = c("decepcionado", "terrible", "no puedo soportar"))
))
```

```
# Realizar el análisis de sentimiento
sentimiento <- textstat_sentiment(corpus, dictionary = diccionario_sentimiento)
```

```
# Mostrar los resultados
print(sentimiento)
```

quanteda: Es una poderosa herramienta para el análisis de texto que permite realizar análisis de sentimientos, entre otras técnicas de procesamiento de lenguaje natural (NLP).

Este código crea un corpus con una serie de textos de ejemplo y luego realiza un análisis de sentimiento utilizando un diccionario de palabras positivas y negativas. Finalmente, muestra los resultados del análisis de sentimiento. Recuerda que los resultados dependerán de la calidad y el tamaño del diccionario utilizado, así como de la complejidad y el contexto de los textos analizados.



```
# Instalar y cargar las librerías necesarias
install.packages("sentimentr")
library(sentimentr)
library(dplyr)
```

```
# Texto de ejemplo
text <- c("Me siento muy feliz hoy, todo va bien.",
       "Este es un día terrible, me siento muy triste.",
       "La película fue genial, me encantó cada momento.",
       "Odio tener que trabajar los fines de semana.")
```

```
# Crear un data frame con el texto
df <- data.frame(text)
```

```
# Realizar el análisis de sentimientos
sentiments <- get_sentiment(df$text)
```

```
# Unir los resultados al data frame original
df <- cbind(df, sentiments)
```

```
# Ver el resultado
print(df)
```

text2vec: Proporciona herramientas para el análisis de texto a gran escala, lo que puede ser útil para analizar grandes volúmenes de datos en busca de sentimientos.

```
# Instalar el paquete lexicon si aún no lo has hecho
# install.packages("lexicon")
```

```
# Cargar el paquete
library(lexicon)
```

```
# Texto de ejemplo
texto <- "El nuevo producto es excelente. Estoy muy contento con mi compra."
```

```
# Realizar análisis de sentimiento utilizando el léxico "bing" incluido en el paquete
resultado <- get_sentiment(texto, method = "bing")
```

```
# Mostrar el resultado
print(resultado)
```

lexicon: Contiene varios lexícones de sentimientos que se pueden utilizar para analizar el sentimiento en texto.

Este es un ejemplo básico que utiliza el léxico "bing" incluido en el paquete lexicon. El resultado mostrará el número de palabras positivas, negativas y neutras encontradas en el texto.



```
# Instalar y cargar paquetes necesarios
install.packages("tidyverse")
install.packages("tidytext")
install.packages("syuzhet")
library(tidyverse)
library(tidytext)
library(syuzhet)

# Ejemplo de datos de texto
texto <- c("Me siento muy feliz hoy",
         "Estoy bastante triste por la noticia",
         "La comida en ese restaurante es deliciosa",
         "No me gusta este clima lluvioso")

# Crear un data frame con el texto
df <- data.frame(texto = texto)

# Tokenizar el texto
df_tokens <- df %>%
  mutate(texto = tolower(texto)) %>%
  unnest_tokens(word, texto)

# Cargar diccionario de palabras positivas y negativas
palabras_afinn <- get_sentiments("afinn")

# Realizar el análisis de sentimiento
sentimientos <- df_tokens %>%
  inner_join(palabras_afinn) %>%
  group_by(texto) %>%
  summarize(sentimiento = sum(value)) %>%
  ungroup()

# Mostrar resultados
print(sentimientos)
```

textblob: Ofrece una API simple para el análisis de sentimientos, incluyendo polaridad, subjetividad y detección de emociones.

```
# Instalar y cargar los paquetes necesarios
install.packages("NLP")
install.packages("tm")
install.packages("SentimentAnalysis")

library(NLP)
library(tm)
library(SentimentAnalysis)

# Texto de ejemplo
texto <- c("Me encanta este producto, es increíblemente útil.",
          "No estoy muy contento con el servicio al cliente.",
          "La calidad de este producto es decepcionante.")

# Crear un corpus
corpus <- Corpus(VectorSource(texto))

# Preprocesamiento del texto
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, removeWords, stopwords("en"))
corpus <- tm_map(corpus, stripWhitespace)

# Crear una matriz de términos de documento (DTM)
dtm <- DocumentTermMatrix(corpus)

# Calcular el sentimiento
sentimientos <- classify_polarity(dtm)

# Mostrar resultados
print(sentimientos)
```

NLP: Contiene funciones para el análisis de lenguaje natural, como la detección de entidades nombradas y la extracción de relaciones.

Este es un ejemplo básico para ilustrar cómo usar el paquete NLP en R para análisis de sentimientos. Puedes ajustar y expandir este código según tus necesidades específicas. Asegúrate de instalar los paquetes tm y SentimentAnalysis si no los tienes instalados.



```
# Instalar los paquetes, si no los tienes instalados
# install.packages("tidytext")
# install.packages("dplyr")
# install.packages("ggplot2")

# Carga las bibliotecas necesarias
library(tidytext)
library(dplyr)
library(ggplot2)

# Carga el conjunto de datos de opiniones de películas (descargarás tener este conjunto de datos)
# Agregar argumentos un conjunto de datos llamado "opiniones_peliculas.csv" con dos columnas: "Opinion" y "Sentimiento"
# opiniones <- read.csv("opiniones_peliculas.csv", stringsAsFactors = FALSE)

# Supongamos que tienes un conjunto de datos de opiniones como este
opiniones <- tribble(
  ~Opinion, ~Sentimiento,
  "Me encantó esta película, la recomendaría a todos mis amigos",
  "La actuación fue mediocre, no volvería a verla",
  "Excelente trama y personajes, una obra maestra",
  "No me acordé de nada, fue una pérdida de tiempo",
  stringsAsFactors = FALSE
)

# Tokenización y análisis de sentimientos
opiniones_sentimiento <- opinions %>%
  unnest_tokens(word, Opinion) %>%
  unnest_tokens(Sentimiento, by = "word")

# Sumarizar el sentimiento de cada opinión
sentimiento_promedio <- opinions_sentimiento %>%
  group_by(Opinion) %>%
  summarise(Sentimiento = sum(Sentimiento))

# Visualización del sentimiento
ggplot(sentimiento_promedio, aes(x = Opinion, y = Sentimiento, fill = Sentimiento > 0)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  labs(
    x = "Opinión",
    y = "Sentimiento",
    title = "Sentimiento de opiniones de películas"
  )
```



```
# Cargar el paquete dplyr
library(dplyr)

# Crear un conjunto de datos ficticio de opiniones de películas
opiniones_peliculas <- data.frame(
  película = c("Titanic", "Titanic", "Matrix", "Matrix", "Matrix"),
  opinión = c("Me encantó", "No me gustó", "Interesante", "Increíble!", "Aburrida")
)

# Definir una función simple para asignar puntuajes de sentimiento a las opiniones
puntaje_sentimiento <- function(opinión) {
  # Definir palabras clave para sentimientos positivos y negativos
  palabras_positivas <- c("encantado", "increíble", "interesante")
  palabras_negativas <- c("aburrido", "no me gustó")

  # Calcular el puntaje de sentimiento
  puntaje <- ifelse(tolower(opinión) %in% palabras_positivas, 1,
    ifelse(tolower(opinión) %in% palabras_negativas, -1, 0))
  return(puntaje)
}

# Aplicar la función puntaje_sentimiento a cada opinión y crear una nueva columna
opiniones_peliculas <- opinions_peliculas %>%
  mutate(sentimiento = sapply(opinión, puntaje_sentimiento))

# Calcular el sentimiento promedio por película utilizando dplyr
sentimiento_promedio <- opinions_peliculas %>%
  group_by(película) %>%
  summarise(sentimiento_promedio = mean(sentimiento))

# Imprimir los resultados
print(sentimiento_promedio)
```

bigmemory: Permite el manejo eficiente de datos textuales grandes en memoria.

Este es un ejemplo básico de cómo realizar análisis de sentimientos en R utilizando el paquete tidytext. Para un análisis más detallado o para manejar grandes volúmenes de datos, podrías necesitar ajustar el código y potencialmente utilizar paquetes adicionales como bigmemory para gestionar los datos de manera más eficiente en la memoria.

dplyr: Proporciona una sintaxis elegante para la manipulación de datos, útil para la limpieza y preprocesamiento.

Este código primero define un conjunto de datos ficticio de opiniones de películas y luego define una función `puntaje_sentimiento` que asigna puntuajes de sentimiento a cada opinión en función de palabras clave predefinidas. Luego, utiliza dplyr para calcular el sentimiento promedio por película y muestra los resultados.



```
# Cargar paquetes
library(data.table)

# Crear un data.table con el diccionario de palabras y sus puntajes de sentimiento
sentiment_dictionary <- data.table(
  word = c("happy", "sad", "good", "bad", "awesome"),
  sentiment_score = c(1, -1, 1, -1, 2)
)
```

```
# Crear un data.table con frases
phrases <- data.table(
  id = 1:5,
  text = c("I'm so happy today!", "I feel sad about this", "This is good news", "I had a bad day", "This is awesome!")
)
```

```
# Función para calcular el sentimiento promedio de una frase
calculate_sentiment <- function(sentence, sentiment_dict) {
  words <- unlist(strsplit(sentence, "(\s+|)"))
  matched_words <- words[words %in% sentiment_dict$word]
  if (length(matched_words) == 0) {
    return(0) # Si no se encuentran palabras en el diccionario, retornar 0
  } else {
    return(mean(sentiment_dict$sentiment_dict[word %in% matched_words, sentiment_score]))
  }
}
```

```
# Calcular el sentimiento promedio de cada frase
phrases[, sentiment := sapply(text, calculate_sentiment, sentiment_dict = sentiment_dictionary)]
```

```
# Mostrar el resultado
print(phrases)
```

```
# Instalar y cargar paquete Spark
install.packages("SparkR")
library(SparkR)
```

```
# Iniciar sesión de Spark
sparkR.session()
```

```
# Cargar datos de un archivo de texto
datos <- read.df("ruta/a/tus/datos.txt", "text")
```

```
# Preprocesar texto (por ejemplo, eliminar signos de puntuación y palabras vacías)
datos <- withColumn(datos, "texto_limpio", regexp_replace(datos$texto, "[[:punct:]]", ""))
datos <- withColumn(datos, "texto_limpio", regexp_replace(datos$texto_limpio, "\\b\\w{1,2}\\b", ""))
datos <- withColumn(datos, "texto_limpio", regexp_replace(datos$texto_limpio, "(\\s+).", ""))
datos <- withColumn(datos, "texto_limpio", trim(datos$texto_limpio))
datos <- withColumn(datos, "texto_limpio", lower(datos$texto_limpio))
```

```
# Realizar análisis de sentimiento utilizando VADER
# Asumiendo que tienes el paquete 'text' instalado
install.packages("text")
library(text)

datos$sentimiento <- sapply(as.character(datos$texto_limpio), function(x) sentiment(x)$sentiment)

# Mostrar resultados
head(select(datos, "texto", "sentimiento"))

# Cerrar sesión de Spark
sparkR.session.stop()
```

data.table: Ofrece una estructura de datos eficiente para el almacenamiento y procesamiento de grandes cantidades de datos.

Este código crea un diccionario de sentimientos con algunas palabras y sus puntajes respectivos. Luego, tiene un conjunto de datos con algunas frases. Utilizando la función calculate_sentiment, calcula el sentimiento promedio para cada frase en función de las palabras en el diccionario de sentimientos. Finalmente, añade una columna al data.table original con los sentimientos calculados y muestra el resultado.

SparkR: Integra R con Apache Spark para el procesamiento paralelo de grandes conjuntos de datos.

Validar el tener instalado Apache Spark en tu sistema y configurarlo correctamente antes de ejecutar este código. Además, puedes necesitar instalar e importar la biblioteca text para usar la función sentiment() de VADER. Este ejemplo carga datos de un archivo de texto, limpia el texto eliminando signos de puntuación y palabras vacías, y luego aplica el análisis de sentimiento a cada línea de texto utilizando VADER.



```
# Instalar y cargar el paquete
install.packages("opinionminer")
library(opinionminer)

# Supongamos que tienes un gran volumen de datos en un dataframe llamado 'datos' con una columna llamada 'texto'

# Dividir el dataframe en lotes más pequeños para procesamiento eficiente
batch_size <- 1000
num_batches <- ceiling(nrow(datos) / batch_size)

# Función para analizar el sentimiento de un lote de texto
analyze_batch_sentiment <- function(batch_text) {
  return(analyze_sentiment(batch_text)$sentiment)
}

# Procesamiento paralelo de lotes de texto
library(parallel)
cl <- makeCluster(detectCores())
clusterExport(cl, "analyze_batch_sentiment") # Exporta la función al clúster

sentiments <- parLapply(cl, split(datos$texto, seq_len(nrow(datos)) - 1) %% batch_size + 1), analyze_batch_sentiment)

stopCluster(cl)

# Combina los resultados de los lotes en un único vector
all_sentiments <- unlist(sentiments)

# Agrega la columna de sentimiento al dataframe original
datos$Sentimiento <- all_sentiments

# Ver los resultados
print(datos)
```

opinionminer: Se enfoca en la extracción de opiniones y la identificación de entidades relevantes en el análisis de sentimientos.

```
library(senticnet)

# Cargar el conjunto de datos (asumiendo que tienes un conjunto de datos llamado "datos.csv" con una columna de texto llamada "texto")
datos <- read.csv("datos.csv")

# Función para calcular el sentimiento de una frase
calcular_sentimiento <- function(frase) {
  sentimientos <- c()
  palabras <- unlist(strsplit(as.character(frase), " "))

  for (palabra in palabras) {
    sentimiento <- senticnet(palabra)
    if (!is.null(sentimiento)) {
      sentimientos <- c(sentimientos, sentimiento$sentiment$pleasantness)
    }
  }

  if (length(sentimientos) > 0) {
    sentimiento_medio <- mean(sentimientos)
    return(sentimiento_medio)
  } else {
    return(NA)
  }
}

# Calcular el sentimiento para cada frase en el conjunto de datos
datos$Sentimiento <- sapply(datos$texto, calcular_sentimiento)

# Imprimir los resultados
print(datos)
```

senticnet: Utiliza una red semántica para el análisis de sentimientos, permitiendo una comprensión más profunda del significado del texto.

Este código carga un conjunto de datos desde un archivo CSV, luego define una función calcular_sentimiento para calcular el sentimiento de cada frase utilizando el diccionario de sentimientos de SenticNet.



```
# Instala y carga el paquete NRC
install.packages("NRC")
library(NRC)

# Carga el lexicon de emociones
data(emotion_lexicon)

# Carga tus datos (en este caso, asumiremos que tienes un dataframe llamado 'datos' con una columna llamada 'texto')
# Puedes adaptar esto para cargar tus datos de la manera que sea necesaria para tu caso específico
datos <- read.csv("tus_datos.csv", stringsAsFactors = FALSE)

# Tokenización y limpieza de texto (puedes adaptar esto según tus necesidades)
library(tm)
library(tokenizers)
library(tm)
corpus <- Corpus(VectorSource(datos$Texto))
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, removeWords, stopwords("english"))
corpus <- tm_map(corpus, stripWhitespace)
tokens <- unlist(tokenize_words(corpus))

# Calcular el conteo de emociones para cada palabra en tus datos
emociones_por_palabra <- as.data.frame(table(tokens, includeNA = FALSE))

# Unir el conteo de emociones con el lexicon de emociones
emociones_por_palabra <- merge(emociones_por_palabra, emotion_lexicon, by.x = "tokens", by.y = "word", all.x = TRUE)

# Calcular el total de emociones por categoría para tus datos
emociones_por_categoria <- aggregate(~ category, data = emociones_por_palabra, FUN = sum)

# Ver los resultados
print(emociones_por_categoria)
```

NRC Emotion Lexicon: Diccionario de emociones con puntuaciones para un análisis de sentimientos más granular.

En este ejemplo, primero instalamos y cargamos el paquete NRC. Luego cargamos el lexicon de emociones proporcionado por el paquete. Después, cargamos tus datos y los limpiamos y tokenizamos para prepararlos para el análisis. Luego, calculamos el conteo de cada emoción para cada palabra en tus datos y unimos este conteo con el lexicon de emociones. Finalmente, calculamos el total de emociones por categoría y mostramos los resultados.

```
# Instalar y cargar paquete
install.packages("sentimentr")
library(sentimentr)
```

Bing Liu Lexicon: Diccionario con puntuaciones de polaridad para palabras y frases.

```
# Cargar datos fuentes
datos <- read.csv("datos_fuente.csv", stringsAsFactors = FALSE)
```

```
# Análisis de sentimiento con Bing Liu Lexicon
sentimiento <- sentiment_by(datos$texto, by = "sentence")
```

```
# Mostrar resultados
head(sentimiento)
```

Este código supone que tienes un archivo CSV llamado "datos_tweets.csv" que contiene una columna llamada "texto" con los textos de los tweets. Asegúrate de reemplazar "datos_tweets.csv" con el nombre de tu propio archivo CSV y "texto" con el nombre de la columna que contiene los textos de tus datos. El paquete sentiment proporciona funciones convenientes para realizar análisis de sentimiento en grandes volúmenes de datos utilizando el léxico de Bing Liu. La función sentiment_by divide el texto en oraciones y luego calcula el sentimiento de cada oración usando el léxico de Bing Liu.



```
library(tidyverse)
library(tidytext)
library(textdata)

# Cargar el texto del libro
texto_libro <- readLines("libro.txt")

# Convertir el texto a un data frame
df_texto <- tibble(texto = texto_libro)

# Tokenización del texto
df_tokenizado <- df_texto %>%
  unnest_tokens(word, texto)

# Cargar el lexicon de sentimientos
data("bing")

# Unirse al lexicon de sentimientos
df_sentimientos <- df_tokenizado %>%
  inner_join(bing, by = "word")

# Contar los sentimientos positivos y negativos
sentimientos_contados <- df_sentimientos %>%
  count(sentiment) %>%
  spread(sentiment, n, fill = 0)

# Calcular el estado de ánimo general
sentimientos_contados$estado_animo <- ifelse(sentimientos_contados$positive > sentimientos_contados$negative, "Positivo",
                                               ifelse(sentimientos_contados$positive < sentimientos_contados$negative, "Negativo", "Neutral"))

# Imprimir el resultado
print(sentimientos_contados)
```

3.1.3 Caso práctico: El estado de ánimo de mi libro o la noticia del día

Este código cargará el texto del libro "seleccionado", analizará los sentimientos de cada palabra utilizando el lexicon de sentimientos bing, contará el número de palabras positivas y negativas, y determinará si el estado de ánimo general del texto es positivo, negativo o neutral.

Características de un corpus big data:

- Volumen masivo: Un corpus big data típicamente contiene una cantidad enorme de datos de texto. Esto puede ser desde millones hasta miles de millones de documentos o más.
- Variedad de fuentes: Los datos en un corpus big data pueden provenir de diversas fuentes, como redes sociales, sitios web, documentos gubernamentales, foros en línea, noticias, entre otros. Esta variedad de fuentes puede enriquecer el corpus con una amplia gama de lenguaje y estilos de escritura.
- Velocidad de adquisición: La recopilación de datos en un corpus big data puede ser continua y en tiempo real, lo que implica una alta velocidad de adquisición de datos para mantenerse al día con la información que se genera constantemente en línea.
- Complejidad de procesamiento: Dado el tamaño masivo del corpus, el procesamiento de datos y el análisis lingüístico pueden requerir técnicas y herramientas específicas de big data para manejar eficientemente la carga de trabajo.
- Diversidad lingüística: Debido a la variedad de fuentes de datos, un corpus big data puede contener textos en varios idiomas y dialectos, lo que lo hace útil para análisis multilingües y estudios comparativos.
- Desafíos de almacenamiento y procesamiento: El almacenamiento y procesamiento de un corpus big data pueden ser desafiantes debido a la necesidad de infraestructura de almacenamiento y computación escalable.



Ejemplo

```
# Cargar el paquete 'tm' para el análisis de texto
library(tm)

# Crear un vector con las rutas a los archivos del corpus
archivos <- list.files("ruta/al/corpus", full.names = TRUE)

# Función para leer un archivo de texto y convertirlo a un corpus
leer_corpus <- function(archivo)
{ texto <- readLines(archivo)
corpus(VectorSource(texto)) }

# Crear un corpus a partir de los archivos
corpus <- lapply(archivos, leer_corpus)

# Combinar todos los documentos en un solo corpus
corpus <- tm_corpus(corpus)

# Convertir todo el texto a minúsculas
corpus <- tm_map(corpus, content_transformer(tolower))

# Eliminar puntuación y símbolos especiales
corpus <- tm_map(corpus, removePunctuation)

# Eliminar palabras vacías (stopwords)
corpus <- tm_map(corpus, removeWords, stopwords("es"))

# Lemmatizar las palabras (opcional)
#corpus <- tm_map(corpus, stemDocument)
```



```
# Obtener la frecuencia de términos
frecuencias <- tm_term_matrix(corpus)

# Visualizar las 10 palabras más frecuentes
top_palabras <- sort(colSums(frecuencias), decreasing = TRUE)[1:10]
print(top_palabras)

# Crear un gráfico de nube de palabras
wordcloud(corpus, size = sqrt(colSums(frecuencias)), min.freq = 5)

# Calcular la distancia entre documentos
distancia <- dist(t(frecuencias))

# Visualizar la distancia entre documentos mediante un MDS
mds <- cmdscale(distancia)
plot(mds, labels = names(corpus))

# Crear un modelo LDA
modelo_lda <- LDA(corpus, k = 5)

# Visualizar los tópicos
print(topics(modelo_lda, 5))

# Asignar cada documento a un tema
doc_topic <- classify(modelo_lda, corpus)

# Visualizar la distribución de temas por documento
barplot(table(doc_topic))
```



```
# Ejemplo en R para administrar Palabras de opinión negativa de grandes volúmenes de datos

library(tm)
library(SnowballC)
library(wordcloud)
library(tidyverse)

# Ejemplo de datos
corpus <- Corpus(VectorSource(c("Este producto es terrible", "El servicio es pésimo", "No lo recomiendo", "Muy mala experiencia")))

corpus <- tm_map(corpus, content_transformer(tolower)) # Convertir a minúsculas
corpus <- tm_map(corpus, removePunctuation) # Eliminar puntuación
corpus <- tm_map(corpus, removeWords, stopwords("es")) # Eliminar palabras vacías
corpus <- tm_map(corpus, stemDocument) # Lematización

# Convertir a matriz de términos-documento
dtm <- DocumentTermMatrix(corpus)

# Frecuencia de palabras
freq <- sort(colSums(as.matrix(dtm)), decreasing = TRUE)

# Nube de palabras
wordcloud(names(freq)[1:20], freq[1:20], min.freq = 5)

# Diccionario de palabras negativas
negatives <- c("terrible", "pésimo", "malo", "deficiente", "insatisfactorio")

# Identificar palabras negativas en el corpus
corpus_neg <- tm_map(corpus, function(x) sum(grepl(negatives, x)) > 0)

# Proporción de documentos con palabras negativas
prop_neg <- mean(corpus_neg)

# Filtrar documentos con palabras negativas
corpus_negativo <- corpus[corpus_neg]
```



```
# Gráfico de barras con la frecuencia de palabras negativas  
barplot(table(corpus_negativo))  
  
# Exportar frecuencia de palabras  
write.csv(freq, "frecuencia_palabras.csv")  
  
# Exportar documentos con palabras negativas  
write.csv(corpus_negativo, "documentos_negativos.csv")
```

E-books

- Bouso Freijo, J. (2018). El paquete estadístico R: (2 ed.). CIS - Centro de Investigaciones Sociológicas. <https://elibro.net/es/lc/ucags/titulos/105698>
- Royé, D. & Serrano Notivoli, R. (2019). Introducción a los SIG con R: (ed.). Prensas de la Universidad de Zaragoza. <https://elibro.net/es/lc/ucags/titulos/122173>
- Mas Elias, J. (2020). Análisis de datos con R en estudios internacionales: (ed.). Editorial UOC. <https://elibro.net/es/lc/ucags/titulos/167261>
- Alonso, J. C. & Largo, M. F. (2022). Empezando a visualizar datos con R y ggplot2: (1 ed.). Editorial Universidad Icesi. <https://elibro.net/es/lc/ucags/titulos/225846>
- Pujo Jover, M. & Pujo Jover, M. (2017). Análisis cuantitativo con R: matemáticas, estadística y econometría: (ed.). Editorial UOC. <https://elibro.net/es/lc/ucags/titulos/58652>
- Cabrero Ortega, M. Y. & García Pérez, A. (2022). Análisis estadístico de datos espaciales con QGIS y R: (1 ed.). UNED - Universidad Nacional de Educación a Distancia. <https://elibro.net/es/lc/ucags/titulos/218566>
- Gil Pascual, J. A. (2021). Minería de texto con R: aplicaciones y técnicas estadísticas de apoyo: (ed.). UNED - Universidad Nacional de Educación a Distancia. <https://elibro.net/es/lc/ucags/titulos/188719>
- Iryopogu, J. (2021). Análisis de datos con Power BI, R-RStudio y Knime: curso práctico: (1 ed.). RA-MA Editorial. <https://elibro.net/es/lc/ucags/titulos/222665>



Referencias

- Shalabh, Shalabh. (2023). The Big R-Book: From Data Science to Learning Machines and Big Data. Journal of the Royal Statistical Society Series A: Statistics in Society. 186. 896-897. 10.1093/jrsssa/qnad029.
- Balazka, Dominik & Rodighiero, Dario. (2020). Big Data and the Little Big Bang: An Epistemological (R)evolution. Frontiers in Big Data. 3. 1-13. 10.3389/fdata.2020.00031.
- Hodeghatta, U.R. & Nayak, U.. (2016). Business analytics using R-A practical approach. 10.1007/978-1-4842-2514-1.
- Tripathi, Subhashini. (2016). Learn Business Analytics in Six Steps Using SAS and R. 10.1007/978-1-4842-1001-7.
- Ohri, Ajay. (2013). R for Business Analytics. 10.1007/978-1-4614-4343-8.