Nikolas Dimitrio Badani Gasdaglis 20092
Juan Angel Carrera Soto 20593
Data Science
Sección 10

Laboratorio 4 : Minería de Textos

Cálculos de Frecuencia

```
> table(train$keyword)

               ablaze              accident            aftershock
                   36                    35                    34
  airplane%20accident             ambulance           annihilated
                   35                    38                    34
        annihilation            apocalypse            armageddon
                   29                    32                    42
                army                 arson               arsonist
                   34                    32                    34
              attack              attacked             avalanche
                   36                    35                    30
              battle             bioterror          bioterrorism
                   26                    37                    30
               blaze               blazing              bleeding
                   38                    34                    35
           blew%20up               blight              blizzard
                   33                    32                    37
               blood               bloody            blown%20up
                   35                    35                    33
          body%20bag        body%20bagging           body%20bags
                   33                    33                    41
                bomb               bombed               bombing
                   34                    38                    29
   bridge%20collapse   buildings%20burning buildings%20on%20fire
                   35                    35                    33
             burned               burning   burning%20buildings
                   33                    34                    37
       bush%20fires            casualties              casualty
                   25                    35                    34
        catastrophe          catastrophic  chemical%20emergency
                   36                    30                    33
       cliff%20fall              collapse             collapsed
                   36                    34                    35
            collide             collided             collision
                   34                    40                    39
              crash               crashed                 crush
                   33                    34                    37
```

crushed	curfew	cyclone
31	37	32
damage	danger	dead
41	36	30
death	deaths	debris
36	38	37
deluge	deluged	demolish
42	27	34
demolished	demolition	derail
28	35	35
derailed	derailment	desolate
38	39	29
desolation	destroy	destroyed
36	37	32
destruction	detonate	detonation
34	36	32
devastated	devastation	disaster
31	36	35
displaced	drought	drown
36	35	32
drowned	drowning	dust%20storm
38	34	36
earthquake	electrocute	electrocuted
39	32	34
emergency	emergency%20plan	emergency%20services
37	35	33
engulfed	epicentre	evacuate
36	12	40
evacuated	evacuation	explode
36	36	38
exploded	explosion	eyewitness
33	39	32
famine	fatal	fatalities
39	38	45
fatality	fear	fire
37	40	38
fire%20truck	first%20responders	flames
33	29	39

flattened	flood	flooding
34	35	38
floods	forest%20fire	forest%20fires
36	19	32
hail	hailstorm	harm
35	32	41
hazard	hazardous	heat%20wave
34	35	34
hellfire	hijack	hijacker
39	33	35
hijacking	hostage	hostages
32	31	37
hurricane	injured	injuries
38	35	33
injury	inundated	inundation
38	35	10
landslide	lava	lightning
33	34	33
loud%20bang	mass%20murder	mass%20murderer
34	33	32
massacre	mayhem	meltdown
36	30	33
military	mudslide	natural%20disaster
34	37	34
nuclear%20disaster	nuclear%20reactor	obliterate
34	36	31
obliterated	obliteration	oil%20spill
31	29	38
outbreak	pandemonium	panic
40	37	37
panicking	police	quarantine
33	37	34
quarantined	radiation%20emergency	rainstorm
37	9	34
razed	refugees	rescue
35	36	22
rescued	rescuers	riot
35	35	34

```
            rioting                rubble                  ruin
                 35                    28                    37
          sandstorm              screamed             screaming
                 37                    34                    36
            screams              seismic              sinkhole
                 35                    29                    39
            sinking                siren               sirens
                 41                    40                    29
             smoke             snowstorm                storm
                 34                    29                    35
          stretcher  structural%20failure       suicide%20bomb
                 33                    35                    35
   suicide%20bomber     suicide%20bombing                 sunk
                 31                    33                    39
           survive              survived             survivors
                 32                    33                    30
         terrorism             terrorist                threat
                 34                    31                    11
           thunder          thunderstorm               tornado
                 38                    33                    35
           tragedy               trapped                trauma
                 36                    32                    31
       traumatised              trouble               tsunami
                 35                    33                    34
           twister               typhoon              upheaval
                 40                    38                    38
    violent%20storm               volcano            war%20zone
                 33                    27                    24
            weapon              weapons             whirlwind
                 39                    39                    39
      wild%20fires              wildfire             windstorm
                 31                    33                    40
           wounded               wounds                 wreck
                 37                    33                    37
          wreckage              wrecked
                 39                    39
>
```

Como se puede observar, el análisis realizado permitió encontrar que las palabras clave en los tweets tiene una relación directa con el peligro al estar relacionados con desastres naturales, situaciones de emergencia, equipos de rescate, elementos que se forman o se encuentran presentes en un accidente o desastre natural. También se pudo determinar que la frecuencia en la que todas estas palabras son utilizadas en la lista de tweets del data set es bastante grande. Esto se debe a que todas las palabras poseen una frecuencia que ronda entre los valores del 9 hasta el 42.

N-Grama

```
> train_bigrama <-
+   train %>%
+   unnest_tokens(input = "keyword", output = "bigrama", token = "ngrams", n = 2)
> View(train_bigrama)
> train_bigrama %>%
+   count(bigrama, sort = T)
# A tibble: 38 × 2
   bigrama                 n
   <chr>               <int>
 1 NA                   6448
 2 body 20bags            41
 3 oil 20spill            38
 4 burning 20buildings    37
 5 cliff 20fall           36
 6 dust 20storm           36
 7 nuclear 20reactor      36
 8 airplane 20accident    35
 9 bridge 20collapse      35
10 buildings 20burning    35
# ℹ 28 more rows
# ℹ Use `print(n = ...)` to see more rows
>
```

Según el análisis realizado con el n-grama, se puede observar la frecuencia y la cantidad en que las palabras clave de los tweets están siendo utilizadas. Esto no solo permite tener una idea más clara sobre qué tanto se han utilizado estas palabras clave, sino también permite identificar sobre qué tipo de emergencia o desastre está hablando el tweet.

```
# Importing necessary libraries
import pandas as pd

# Loading the dataset to take a quick look at the first few rows
file_path = './train.csv'
df = pd.read_csv(file_path)

# Displaying the first few rows of the dataset to get an overview
df.head()
```

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

Razonamineto Preprocesamiento

1. Convertir el texto a minúsculas:

Razón: Esto se hace para garantizar que la misma palabra con diferentes casos no se trate como palabras distintas. Por ejemplo, "Hola" y "hola" se tratarían como la misma palabra.

2. Quitar URLs:

Razón: Las URLs generalmente no añaden información significativa para el análisis de texto y pueden ser una fuente de ruido, especialmente en tareas como la clasificación de texto.

3. Quitar caracteres especiales como #, @, y apóstrofes:

Razón: Estos caracteres suelen ser ruidosos y no aportan mucho a la semántica del texto. Sin embargo, en algunos casos, como el análisis de sentimientos en tweets, podrían ser útiles.

4. Quitar emoticones:

Razón: Los emoticones pueden ser útiles para algunas tareas específicas como el análisis de sentimientos, pero en general pueden considerarse como ruido en el texto.

5. Quitar signos de puntuación:

Razón: Los signos de puntuación raramente aportan valor en tareas de análisis de texto y generalmente se consideran ruido.

6. Quitar palabras vacías (stopwords):

Razón: Palabras como "y", "o", "el", "la", etc., son muy comunes pero no aportan información significativa para muchas tareas de análisis de texto.

7. Quitar números:

Razón: Los números pueden ser útiles para ciertas tareas, pero en muchos casos son irrelevantes. Por ejemplo, en el análisis de sentimientos, los números raramente aportan algún sentimiento.

```
# Importing necessary libraries for text preprocessing
import re
import string
from nltk.corpus import stopwords

# Function to preprocess text
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
```

```
    # Remove special characters like #, @, and apostrophes
    text = re.sub(r'\@\w+|\#\w+|\'[\w\d\s]+|[\w\d\s]+\'', '', text)

    # Remove emoticons
    text = text.encode('ascii', 'ignore').decode('ascii')

    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    text = ' '.join([word for word in text.split() if word not in stop_words])

    # Remove numbers (we can customize this further based on domain-specific needs)
    text = re.sub(r'\d+', '', text)

    return text

# Applying the preprocessing function to the 'text' column
df['preprocessed_text'] = df['text'].apply(preprocess_text)

# Displaying the first few rows to see the changes
df[['text', 'preprocessed_text']].head()
```

	text	preprocessed_text
0	Our Deeds are the Reason of this #earthquake M...	deeds reason may allah forgive us
1	Forest fire near La Ronge Sask. Canada	forest fire near la ronge sask canada
2	All residents asked to 'shelter in place' are ...	notified officers evacuation shelter place ord...
3	13,000 people receive #wildfires evacuation or...	people receive evacuation orders california
4	Just got sent this photo from Ruby #Alaska as ...	got sent photo ruby smoke pours school

```
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader, TensorDataset
from collections import Counter
import torch

# Tokenize the text into words
tokenized_text = [text.split() for text in df['preprocessed_text']]

# Count the occurrences of each word to create a vocabulary
flat_list = [word for sublist in tokenized_text for word in sublist]
word_counter = Counter(flat_list)
vocab = {word: i+1 for i, (word, _) in enumerate(word_counter.most_common())}

# Convert the tokenized text to numerical format based on the vocabulary
numericalized_text = [[vocab[word] for word in text] for text in tokenized_text]

# Padding: Make all sequences have the same length
# First, find the sequence with maximum length
max_len = max(len(seq) for seq in numericalized_text)

# Pad sequences with zeros at the end
padded_sequences = [seq + [0]*(max_len - len(seq)) for seq in numericalized_text]

# Convert to PyTorch tensors
tensor_sequences = torch.tensor(padded_sequences, dtype=torch.long)
tensor_labels = torch.tensor(df['target'].values, dtype=torch.float32)

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(tensor_sequences, tensor_labels, test_size=0.2, random_state=42)

# Create DataLoader objects
train_data = TensorDataset(X_train, y_train)
train_loader = DataLoader(train_data, shuffle=True, batch_size=64)

test_data = TensorDataset(X_test, y_test)
test_loader = DataLoader(test_data, shuffle=False, batch_size=64)

vocab_size = len(vocab) + 1  # Adding 1 for padding (0)
tensor_sequences.shape, vocab_size
```

```
(torch.Size([7613, 23]), 12789)
```

Model

LSTM

```
import torch.nn as nn

class TweetClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(TweetClassifier, self).__init__()

        # Embedding Layer
        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        # LSTM Layer
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)

        # Fully Connected Layer
        self.fc = nn.Linear(hidden_dim, 1)

        # Sigmoid Activation
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.embedding(x)
        lstm_out, _ = self.lstm(x)
        lstm_out = lstm_out[:, -1, :]
        out = self.fc(lstm_out)
        out = self.sigmoid(out)
        return out
```

Training

```
import torch.optim as optim

# Parámetros del modelo
embedding_dim = 100
hidden_dim = 256

# Instancia del modelo
model = TweetClassifier(vocab_size, embedding_dim, hidden_dim)

# Función de Pérdida y Optimizador
loss_function = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
import torch.optim as optim
import torch.utils.data as data

# Asumiendo que 'TweetClassifier' es tu modelo LSTM personalizado para este problema

# Inicialización del modelo, optimizador y función de pérdida
model = TweetClassifier(vocab_size, embedding_dim, hidden_dim)
optimizer = optim.Adam(model.parameters())
loss_fn = nn.BCELoss()

# DataLoader
loader = data.DataLoader(data.TensorDataset(X_train, y_train), shuffle=True, batch_size=64)

# Número de épocas
n_epochs = 100

# Listas para almacenar las métricas
loss_train = []
loss_test = []

# Entrenamiento y evaluación
for epoch in range(n_epochs):
```

```
    model.train()
    for X_batch, y_batch in loader:
        y_pred = model(X_batch).squeeze()
        loss = loss_fn(y_pred, y_batch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Validación cada 100 épocas
    if epoch % 10 != 0 and epoch != n_epochs - 1:
        continue

    model.eval()
    with torch.no_grad():
        # Cálculo de métricas para el conjunto de entrenamiento
        y_pred_train = model(X_train).squeeze()
        train_loss = loss_fn(y_pred_train, y_train)

        # Cálculo de métricas para el conjunto de prueba
        y_pred_test = model(X_test).squeeze()
        test_loss = loss_fn(y_pred_test, y_test)

        # Aquí puedes añadir el cálculo de otras métricas como precisión, recall, etc.

    print(f"Epoch {epoch}: train loss {train_loss:.4f}, test loss {test_loss:.4f}")
```

```
 Epoch 0: train loss 0.6837, test loss 0.6827
 Epoch 10: train loss 0.6834, test loss 0.6822
 Epoch 20: train loss 0.0611, test loss 1.0010
 Epoch 30: train loss 0.0379, test loss 1.5286
 Epoch 40: train loss 0.0343, test loss 1.9039
 Epoch 50: train loss 0.0456, test loss 1.5087
 Epoch 60: train loss 0.0336, test loss 1.8403
 Epoch 70: train loss 0.0335, test loss 1.9411
 Epoch 80: train loss 0.0335, test loss 1.9126
 Epoch 90: train loss 0.0334, test loss 2.0605
 Epoch 99: train loss 0.0360, test loss 1.3992
```

```
# Lista para almacenar las etiquetas verdaderas
test_labels = []

# No es necesario calcular gradientes
with torch.no_grad():
    for batch in test_loader:
        # Obtener datos y etiquetas del batch
        _, batch_labels = batch

        # Almacenar las etiquetas verdaderas
        test_labels.extend(batch_labels.tolist())

# Convertir las etiquetas verdaderas a una lista de enteros (0 o 1)
test_labels = [int(label) for label in test_labels]

# Cambiar el modelo a modo de evaluación
model.eval()

# Lista para almacenar las predicciones
test_preds = []

# No calcular gradientes para acelerar la computación
with torch.no_grad():
    for batch in test_loader:
        # Obtener datos y etiquetas del batch
        batch_data, _ = batch

        # Hacer predicciones
        predictions = model(batch_data).squeeze()

        # Almacenar las predicciones
        test_preds.extend(predictions.tolist())

# Convertir las predicciones a etiquetas binarias (0 o 1)
rounded_preds = [round(pred) for pred in test_preds]
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Redondear las predicciones para obtener etiquetas binarias
rounded_preds = [round(pred) for pred in test_preds]

# Calcular métricas
accuracy = accuracy_score(test_labels, rounded_preds)
precision = precision_score(test_labels, rounded_preds)
recall = recall_score(test_labels, rounded_preds)
f1 = f1_score(test_labels, rounded_preds)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print
```

```
Accuracy: 0.7360472751149048
Precision: 0.6963434022257552
Recall: 0.674884437596302
<function print(*args, sep=' ', end='\n', file=None, flush=False)>
```

```
def classify_tweet(new_tweet, model, vocab, max_len):
    # Preprocesar el tweet
    processed_tweet = preprocess_text(new_tweet)

    # Tokenizar y aplicar padding
    tokenized = [vocab[word] for word in processed_tweet.split() if word in vocab]
    padded = tokenized + [0] * (max_len - len(tokenized))

    # Convertir a tensor y pasar por el modelo
    tweet_tensor = torch.tensor([padded], dtype=torch.long)
    model.eval()
    with torch.no_grad():
        prediction = model(tweet_tensor).item()

    # Interpretar la salida del modelo
    if prediction >= 0.5:
        return "\33[91m Este tweet probablemente se trata de un desastre natural real. \33[0m"
    else:
        return "\33[92m Este tweet probablemente no se trata de un desastre natural real. \33[0m"
```

```
# Leer input del usuario
new_tweet = input("Escribe un tweet: ")
result = classify_tweet(new_tweet, model, vocab, max_len)
print(result)
```

```
Este tweet probablemente no se trata de un desastre natural real.
```

Descripcion del algormio utilizado:

1. Preprocesamiento de Datos

Convertimos todo el texto a minúsculas para unificar el formato. Eliminamos URLs, caracteres especiales, emoticones y signos de puntuación para reducir el ruido. Eliminamos palabras vacías (stopwords) y números para centrarnos en las palabras más significativas.

2. Tokenización y Padding

Dividimos el texto en palabras (tokens) y creamos un vocabulario que asigna un número único a cada palabra. Convertimos las palabras en cada tweet a su número correspondiente según el vocabulario. Rellenamos las secuencias con ceros para que todas tengan la misma longitud.

3. División del Conjunto de Datos

Dividimos el conjunto de datos en conjuntos de entrenamiento y prueba.

4. Preparación del DataLoader

Utilizamos el DataLoader de PyTorch para manejar el muestreo de batches durante el entrenamiento.

5. Construcción del Modelo

Creamos un modelo de red neuronal utilizando PyTorch, que incluye una capa de embedding, una capa LSTM y una capa totalmente conectada.

6. Compilación del Modelo

Utilizamos el optimizador Adam y la función de pérdida de entropía cruzada binaria (BCELoss).

7. Entrenamiento del Modelo

Entrenamos el modelo a través de múltiples épocas, utilizando backpropagation y optimización de Adam. En cada época, calculamos y almacenamos la pérdida en los conjuntos de entrenamiento y prueba.

8. Evaluación del Modelo

Utilizamos el modelo entrenado para hacer predicciones en el conjunto de prueba. Calculamos métricas como precisión, recall y F1-score para evaluar el rendimiento del modelo. Este algoritmo proporciona un enfoque sólido para clasificar tweets en categorías relacionadas con desastres naturales. El uso de una arquitectura LSTM permite al modelo capturar dependencias temporales en los datos de texto, lo que es crucial para entender el contexto y hacer predicciones más precisas.