Aprendizaje Automático para la Gestión de Datos Masivos

Dr. Paulo López Meyer

Análisis de texto

Itzel Rubi Alcala Gil 2930147

Embedings

En el siguiente reporte, voy a explicar tres códigos de análisis de texto, utilizando tres técnicas diferentes de representación de texto: Bag of Words (BoW), Word2Vec y Doc2Vec.

A continuación, desgloso el funcionamiento de cada sección del código y su propósito.

```python
from sklearn.feature_extraction.text import CountVectorizer

# Ejemplo de documentos
documents = [
    "El perro ladra",
    "El gato maúlla",
    "El perro y el gato son amigos",
    "El perro ladra y el gato maúlla"
]

# Crear el modelo BoW
vectorizer = CountVectorizer()

# Ajustar y transformar los documentos en vectores
X = vectorizer.fit_transform(documents)

# Mostrar el vocabulario
print("Vocabulario:", vectorizer.vocabulary_)

# Mostrar la representación BoW de los documentos
print("Representación BoW:\n", X.toarray())
```

```
Vocabulario: {'el': 1, 'perro': 5, 'ladra': 3, 'gato': 2, 'maúlla': 4, 'son': 6, 'amigos': 0}
Representación BoW:
 [[0 1 0 1 0 1 0]
 [0 1 1 0 1 0 0]
 [1 2 1 0 0 1 1]
 [0 2 1 1 1 1 0]]
```

Primero, se utiliza la técnica Bag of Words (BoW) para convertir una colección de documentos en una matriz de conteos de términos. En esta técnica, cada documento es representado como una bolsa (conjunto) de sus palabras, sin tener en cuenta el orden de las mismas. Se utiliza CountVectorizer de sklearn.feature_extraction.text para crear el modelo BoW.

El código ajusta y transforma una lista de documentos en una matriz BoW y luego imprime tanto el vocabulario como la representación BoW de los documentos. El vocabulario es un diccionario que asigna un índice a cada palabra única en el

corpus, y la representación BoW es una matriz en la que cada fila representa un documento y cada columna representa la frecuencia de un término específico en ese documento.

```python
import gensim
from gensim.models import Word2Vec

# Ejemplo de corpus de documentos
sentences = [
    ['el', 'perro', 'ladra'],
    ['el', 'gato', 'maúlla'],
    ['el', 'perro', 'y', 'el', 'gato', 'son', 'amigos'],
    ['el', 'perro', 'ladra', 'y', 'el', 'gato', 'maúlla']
]

# Entrenar el modelo Word2Vec
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)

# Obtener el vector de la palabra 'perro'
vector = model.wv['perro']
print( "Vector de 'perro':\n", vector, "de Tamaño ", vector.shape[0])


# Encontrar palabras similares a 'perro'
similar_words = model.wv.most_similar('perro')
print("Palabras similares a 'perro':\n", similar_words)
```

```
Vector de 'perro':
 [ 9.4563962e-05  3.0773198e-03 -6.8126451e-03 -1.3754654e-03
  7.6685809e-03  7.3464094e-03 -3.6732971e-03  2.6427018e-03
 -8.3171297e-03  6.2054861e-03 -4.6373224e-03 -3.1641065e-03
  9.3113566e-03  8.7338570e-04  7.4907029e-03 -6.0740625e-03
  5.1605068e-03  9.9228229e-03 -8.4573915e-03 -5.1356913e-03
 -7.0648370e-03 -4.8626517e-03 -3.7785638e-03 -8.5361991e-03
  7.9556061e-03 -4.8439382e-03  8.4236134e-03  5.2625705e-03
 -6.5500261e-03  3.9578713e-03  5.4701497e-03 -7.4265362e-03
 -7.4057197e-03 -2.4752307e-03 -8.6257253e-03 -1.5815723e-03
 -4.0343284e-04  3.2996845e-03  1.4418805e-03 -8.8142155e-04
 -5.5940580e-03  1.7303658e-03 -8.9737179e-04  6.7936908e-03
  3.9735902e-03  4.5294715e-03  1.4343059e-03 -2.6998555e-03
 -4.3668128e-03 -1.0320747e-03  1.4370275e-03 -2.6460087e-03
 -7.0737829e-03 -7.8053069e-03 -9.1217868e-03 -5.9351693e-03
 -1.8474245e-03 -4.3238713e-03 -6.4606704e-03 -3.7173224e-03
  4.2891586e-03 -3.7390434e-03  8.3781751e-03  1.5339935e-03
 -7.2423196e-03  9.4337985e-03  7.6312125e-03  5.4932819e-03
 -6.8488456e-03  5.8226790e-03  4.0090932e-03  5.1853694e-03
  4.2559016e-03  1.9397545e-03 -3.1701624e-03  8.3538452e-03
  9.6121803e-03  3.7926030e-03 -2.8369951e-03  7.1275235e-06
  1.2188185e-03 -8.4583247e-03 -8.2239453e-03 -2.3101569e-04
  1.2372875e-03 -5.7433806e-03 -4.7252737e-03 -7.3460746e-03
  8.3286157e-03  1.2129784e-04 -4.5093987e-03  5.7017053e-03
  9.1800150e-03 -4.0998720e-03  7.9646818e-03  5.3754342e-03
  5.8791232e-03  5.1259040e-04  8.2130842e-03 -7.0190406e-03] de Tamaño  100
Palabras similares a 'perro':
 [('maúlla', 0.17018885910511017), ('amigos', 0.145950585603714), ('ladra', 0.06408977508544922), ('son', -0.002754019573330879), ('y', -0.013514922931790352)
```

A continuación, se utiliza la técnica Word2Vec, que es un modelo basado en redes neuronales que aprende representaciones vectoriales de palabras a partir de un corpus de texto.

En el código, se utiliza la biblioteca gensim para entrenar un modelo Word2Vec con un pequeño conjunto de oraciones. Una vez entrenado el modelo, se obtiene el

vector correspondiente a la palabra 'perro' y se imprimen sus valores, que son un conjunto de números que representan las características de la palabra en un espacio vectorial de alta dimensión.

Además, el modelo se utiliza para encontrar palabras similares a 'perro' basándose en la proximidad de sus vectores en el espacio vectorial, y se imprimen estas palabras similares.

```python
from gensim.models import Doc2Vec
from gensim.models.doc2vec import TaggedDocument

# Ejemplo de corpus de documentos
documents = [
    TaggedDocument(words=['el', 'perro', 'ladra'], tags=['doc1']),
    TaggedDocument(words=['el', 'gato', 'maúlla'], tags=['doc2']),
    TaggedDocument(words=['el', 'perro', 'y', 'el', 'gato', 'son', 'amigos'], tags=['doc3']),
    TaggedDocument(words=['el', 'perro', 'ladra', 'y', 'el', 'gato', 'maúlla'], tags=['doc4'])
]

# Entrenar el modelo Doc2Vec
model = Doc2Vec(documents, vector_size=100, window=5, min_count=1, workers=4)

# Obtener el vector del documento 'doc1'
vector = model.dv['doc1']
print("Vector de 'doc1':\n", vector, "de Tamaño ", vector.shape[0])

# Encontrar documentos similares a 'doc1'
similar_docs = model.dv.most_similar('doc1')
print("Documentos similares a 'doc1':\n", similar_docs)
```

```
Vector de 'doc1':
 [-0.00523712 -0.00598646 -0.00987788  0.00855222  0.00357316  0.00026141
 -0.00987699 -0.00517173 -0.00971827  0.00201409  0.00282551  0.00464802
 -0.00430254 -0.00314885 -0.00307666 -0.00871805  0.00217124  0.00922598
 -0.00950408 -0.00345932 -0.00377703  0.00260302 -0.00568931  0.00262589
  0.00580102 -0.00810679 -0.00833266 -0.00995493  0.00493033 -0.00912276
  0.0058403   0.00679778 -0.00650261 -0.00452599 -0.00125707  0.00164952
 -0.00148704 -0.00854769 -0.00360297  0.00172834 -0.00205181 -0.00722613
  0.00419036 -0.00857468  0.00270903 -0.00461518  0.00064432 -0.00205395
  0.00541016 -0.00801009 -0.00212653 -0.00010093 -0.00664452 -0.00652926
 -0.00193493  0.00880966 -0.00126062  0.00354016 -0.00575008  0.00881831
  0.0029191   0.00928757  0.00435432 -0.00419932  0.00224115 -0.00441026
  0.00578298  0.00183784 -0.00227519 -0.00587963 -0.00803539 -0.00085109
 -0.00894284 -0.00922349 -0.00793784  0.00216816 -0.00650811 -0.00779177
  0.00212847  0.00205082  0.00834689  0.00467235 -0.00941201 -0.00033874
  0.00785296  0.00267828  0.00268141 -0.00488653  0.00646293  0.00165103
 -0.00760886  0.00686273 -0.0097776  -0.00815981 -0.00487008  0.0099439
  0.00311186 -0.00201327  0.00890355  0.00234753] de Tamaño  100
Documentos similares a 'doc1':
 [('doc2', 0.16251668334007263), ('doc3', 0.0014203854370862246), ('doc4', -0.016057299450039864)]
```

Finalmente, se presenta la técnica Doc2Vec, que es una extensión de Word2Vec diseñada para obtener representaciones vectoriales de documentos completos en lugar de solo palabras.

En el código, se utiliza Doc2Vec de gensim.models para entrenar un modelo con una colección de documentos etiquetados. Cada documento es representado como una lista de palabras y una etiqueta única. Después de entrenar el modelo, se obtiene el vector que representa uno de los documentos ('doc1') y se imprimen sus valores.

Además, se utiliza el modelo para encontrar documentos similares a 'doc1' y se imprimen estos documentos similares basados en la proximidad de sus vectores en el espacio vectorial.

En resumen, este código demuestra cómo transformar texto en representaciones vectoriales utilizando tres técnicas diferentes: Bag of Words (BoW), Word2Vec y Doc2Vec. Estas representaciones vectoriales son fundamentales en muchas aplicaciones de NLP, como la clasificación de texto, la búsqueda de similitud y el análisis de sentimientos, permitiendo a los algoritmos de aprendizaje automático trabajar eficazmente con datos de texto.

TF-IDF

Ahora voy a realizar un análisis utilizando la técnica de Term Frequency-Inverse Document Frequency (TF-IDF) para convertir una colección de documentos en una matriz TF-IDF. Esta técnica es muy utilizada en el campo del procesamiento del lenguaje natural (NLP) para evaluar la importancia de una palabra en un documento en relación con una colección de documentos (corpus).

A continuación, se detalla cada parte del código y su propósito.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# Definir el conjunto de documentos
documents = [
    "Data science is an interdisciplinary field",
    "Machine learning is a part of data science",
    "Data science involves statistics"
]

# Crear el vectorizador TF-IDF
vectorizer = TfidfVectorizer()

# Ajustar y transformar los documentos
tfidf_matrix = vectorizer.fit_transform(documents)

# Crear un DataFrame para visualizar los resultados
df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.get_feature_names_out())
print(df)
```

```
         an      data     field  interdisciplinary  involves        is  \
0  0.483591  0.285617  0.483591           0.483591  0.000000  0.367784
1  0.000000  0.257129  0.000000           0.000000  0.000000  0.331100
2  0.000000  0.359594  0.000000           0.000000  0.608845  0.000000

   learning   machine        of      part   science  statistics
0  0.000000  0.000000  0.000000  0.000000  0.285617    0.000000
1  0.435357  0.435357  0.435357  0.435357  0.257129    0.000000
2  0.000000  0.000000  0.000000  0.000000  0.359594    0.608845
```

Primero, se importa la biblioteca pandas para manejar estructuras de datos y TfidfVectorizer de sklearn.feature_extraction.text para calcular la matriz TF-IDF.

La lista documents contiene tres documentos de texto en inglés relacionados con el campo de la ciencia de datos.

Se crea una instancia de TfidfVectorizer y se ajusta y transforma la lista de documentos en una matriz TF-IDF. La técnica TF-IDF asigna a cada término un peso

que es proporcional a su frecuencia en el documento y inversamente proporcional a su frecuencia en el corpus. Esto ayuda a resaltar términos importantes y a disminuir la importancia de términos muy frecuentes, pero poco informativos.

Finalmente, se convierte la matriz TF-IDF en un DataFrame de pandas para visualizar los resultados de manera más clara. El DataFrame muestra los términos (características) como columnas y los documentos como filas, con cada celda representando el peso TF-IDF de un término en un documento específico. El DataFrame se imprime para mostrar estos valores.

Interpretación de resultados

1. **Filas (Documentos)**:
    - Cada fila representa uno de los documentos en el corpus original.
    - Por ejemplo, la primera fila corresponde al documento "Data science is an interdisciplinary field".
2. **Columnas (Términos)**:
    - Cada columna representa un término único en el corpus.
    - Por ejemplo, la columna 'data' corresponde a la palabra "data".
3. **Valores (Pesos TF-IDF)**:
    - Los valores en las celdas representan el peso TF-IDF del término en el documento correspondiente.
    - Un valor más alto indica que el término es más relevante para ese documento en comparación con otros documentos en el corpus.
    - Por ejemplo, el valor 0.48 en la celda (0, 'field') indica que la palabra "field" tiene una alta importancia en el primer documento.

- **Documento 1 ("Data science is an interdisciplinary field")**:
    - "interdisciplinary" tiene un valor TF-IDF de 0.4835, lo que indica una alta relevancia en este documento.
    - "field" también tiene un valor alto de 0.4835, sugiriendo que es una palabra importante en este documento.
- **Documento 2 ("Machine learning is a part of data science")**:

    - "learning" , "machine" y “part” tienen altos valores de 0.4353, indicando que estas palabras son particularmente relevantes en este documento.

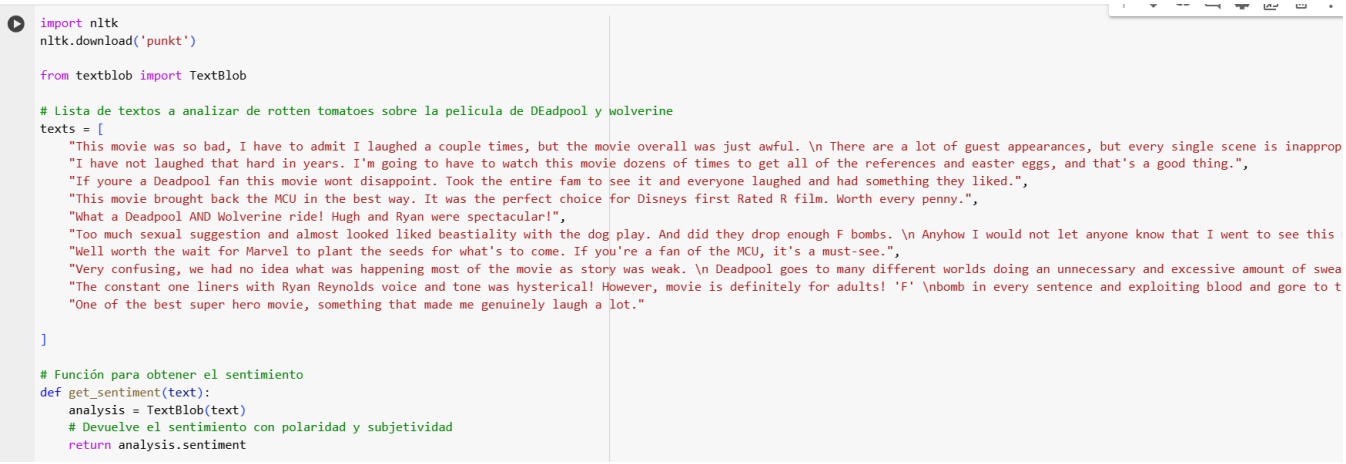
- **Documento 3 ("Data science involves statistics")**:
    - "involves" y "statistics" tienen valores altos de 0.6088, indicando alta relevancia.

En resumen, este código muestra cómo transformar texto en una matriz TF-IDF utilizando TfidfVectorizer de sklearn. La matriz TF-IDF resultante es útil en diversas tareas de NLP, como la clasificación de texto, la búsqueda de similitud y el análisis de sentimientos, ya que permite a los algoritmos de aprendizaje automático trabajar eficazmente con datos de texto al destacar los términos más informativos y relevantes.

Análisis de sentimientos

A continuación, voy a realizar un análisis de sentimiento en una lista de textos utilizando las bibliotecas nltk y TextBlob.

El análisis de sentimiento es una técnica en el campo del procesamiento del lenguaje natural (NLP) que permite determinar las opiniones y emociones expresadas en un texto.



```python
import nltk
nltk.download('punkt')

from textblob import TextBlob

# Lista de textos a analizar de rotten tomatoes sobre la pelicula de DEadpool y wolverine
texts = [
    "This movie was so bad, I have to admit I laughed a couple times, but the movie overall was just awful. \n There are a lot of guest appearances, but every single scene is inapprop
    "I have not laughed that hard in years. I'm going to have to watch this movie dozens of times to get all of the references and easter eggs, and that's a good thing.",
    "If youre a Deadpool fan this movie wont disappoint. Took the entire fam to see it and everyone laughed and had something they liked.",
    "This movie brought back the MCU in the best way. It was the perfect choice for Disneys first Rated R film. Worth every penny.",
    "What a Deadpool AND Wolverine ride! Hugh and Ryan were spectacular!",
    "Too much sexual suggestion and almost looked liked beastiality with the dog play. And did they drop enough F bombs. \n Anyhow I would not let anyone know that I went to see this
    "Well worth the wait for Marvel to plant the seeds for what's to come. If you're a fan of the MCU, it's a must-see.",
    "Very confusing, we had no idea what was happening most of the movie as story was weak. \n Deadpool goes to many different worlds doing an unnecessary and excessive amount of swea
    "The constant one liners with Ryan Reynolds voice and tone was hysterical! However, movie is definitely for adults! 'F' \nbomb in every sentence and exploiting blood and gore to t
    "One of the best super hero movie, something that made me genuinely laugh a lot."

]

# Función para obtener el sentimiento
def get_sentiment(text):
    analysis = TextBlob(text)
    # Devuelve el sentimiento con polaridad y subjetividad
    return analysis.sentiment
```

```python
# Analizar los textos
for text in texts:
    sentiment = get_sentiment(text)
    print(f'Text: "{text}"')
    print(f'Polarity: {sentiment.polarity}, Subjectivity: {sentiment.subjectivity}')
    print()
```

Primero, se importa la biblioteca nltk y se descarga el paquete 'punkt', que es un modelo de tokenización de oraciones y palabras. La tokenización es el proceso de dividir el texto en unidades más pequeñas, como oraciones o palabras.

Luego, se importa TextBlob de la biblioteca textblob, que es una herramienta simple para el procesamiento de texto. TextBlob proporciona una API sencilla para tareas comunes de NLP, incluyendo la clasificación de texto, la traducción y el análisis de sentimientos.

Se define una lista de textos que contiene diversas opiniones y experiencias, tanto positivas como negativas. La función get_sentiment toma un texto como entrada, crea un objeto TextBlob a partir del texto y devuelve el sentimiento del texto. El sentimiento se mide en términos de polaridad y subjetividad. La polaridad es un valor que oscila entre -1 y 1, donde -1 indica un sentimiento negativo, 0 indica

neutralidad y 1 indica un sentimiento positivo. La subjetividad es un valor que oscila entre 0 y 1, donde 0 indica objetividad y 1 indica subjetividad.

Finalmente, se itera sobre la lista de textos, se aplica la función get_sentiment a cada texto y se imprime el texto junto con su polaridad y subjetividad. Este análisis permite entender mejor las emociones y opiniones expresadas en los textos.

```
Text: "This movie was so bad, I have to admit I laughed a couple times, but the movie overall was just awful.
 There are a lot of guest appearances, but every single scene is inappropriate, all the fighting scenes are sexualized, and it is very blasphemous."
Polarity: -0.14523809523809525, Subjectivity: 0.3968253968253968

Text: "I have not laughed that hard in years. I'm going to have to watch this movie dozens of times to get all of the references and easter eggs, and that's a good thing."
Polarity: 0.019444444444444448, Subjectivity: 0.44722222222222224

Text: "If youre a Deadpool fan this movie wont disappoint. Took the entire fam to see it and everyone laughed and had something they liked."
Polarity: 0.4333333333333333, Subjectivity: 0.5416666666666666

Text: "This movie brought back the MCU in the best way. It was the perfect choice for Disneys first Rated R film. Worth every penny."
Polarity: 0.51, Subjectivity: 0.3466666666666667

Text: "What a Deadpool AND Wolverine ride! Hugh and Ryan were spectacular!"
Polarity: 0.75, Subjectivity: 0.9

Text: "Too much sexual suggestion and almost looked liked beastiality with the dog play. And did they drop enough F bombs.
 Anyhow I would not let anyone know that I went to see this movie. Even though it was rated R it was way over the top."
Polarity: 0.4, Subjectivity: 0.6583333333333333

Text: "Well worth the wait for Marvel to plant the seeds for what's to come. If you're a fan of the MCU, it's a must-see."
Polarity: 0.3, Subjectivity: 0.1

Text: "Very confusing, we had no idea what was happening most of the movie as story was weak.
 Deadpool goes to many different worlds doing an unnecessary and excessive amount of swearing looking for a Wolverine there.
 Movie was nothing more than Deadpool killing off everyone in his way and then eventually Wolverine and Deadpool
 fighting each other in numerous bloody battles over and over since they dont die.
 We could hardly wait for it to end, very disappointing!!"
Polarity: -0.163974358974359, Subjectivity: 0.643974358974359

Text: "The constant one liners with Ryan Reynolds voice and tone was hysterical! However, movie is definitely for adults! 'F'
bomb in every sentence and exploiting blood and gore to the max that was unnecessary. Some nasty jokes."
Polarity: -0.48, Subjectivity: 0.7466666666666667

Text: "One of the best super hero movie, something that made me genuinely laugh a lot."
Polarity: 0.5444444444444444, Subjectivity: 0.35555555555555557

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Nota: Nuestros datos de reviews de la película de deadpool y wolverine los obtuve de rottentomatoes.

De nuestro modelo y viendo los reviews y los valores que nos está arrojando, podemos decir que nuestro modelo arrojó valores errados en algunos reviews y en otros si fue más certero, podemos decir que otro modelo podría ser más certero.

En resumen, este código realiza un análisis de sentimiento en una lista de textos utilizando TextBlob. Este análisis es útil en diversas aplicaciones, como la minería de opiniones, el análisis de redes sociales y la retroalimentación de clientes, ya que permite a los algoritmos de aprendizaje automático identificar y cuantificar las emociones y opiniones expresadas en el texto.