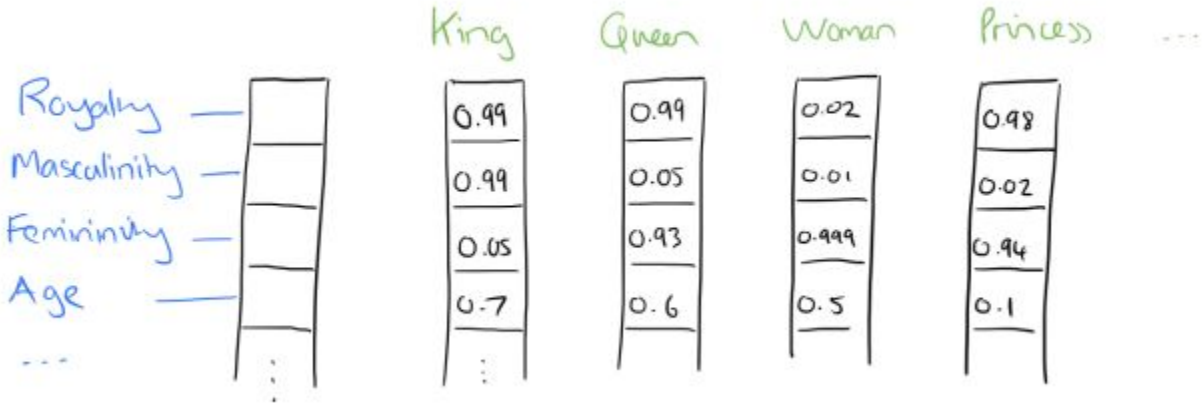
Procesamiento de lenguaje natural II

Curso de inteligencia artificial-Intermedio

Objetivo

Profundizar en los métodos de NLP y su implementación en Python.



Bag of Words

Introducción

Bag of Words (BoW) es una técnica fundamental en procesamiento de lenguaje natural (NLP) para representar texto de manera numérica y poder utilizarlo en modelos de Machine Learning (ML) y Deep Learning (DL).

Ejemplo

Supongamos que tienes dos frases:

1. "El gato duerme"
2. "Duerme el gato"

Si estas dos frases forman tu conjunto de datos, el vocabulario sería: `["el", "gato", "duerme"]`.

Luego, las representaciones BoW de cada frase serían:

- "El gato duerme" → [1, 1, 1] (la palabra "el", "gato" y "duerme" aparecen una vez cada una).
- "Duerme el gato" → [1, 1, 1] (aunque el orden es diferente, BoW ignora esto).

Por lo tanto, **BoW no captura la estructura de la frase**.

Ventajas

- **Simpleza**: Es fácil de implementar y comprender. Es uno de los primeros pasos más comunes en tareas de NLP.
- **Escalable**: Funciona bien incluso en grandes conjuntos de datos.
- **Representación numérica clara**: Convierte el texto en vectores que pueden ser procesados por modelos de ML/DL.

Desventajas

- **No captura el orden**: Como en el ejemplo anterior, "El gato duerme" y "Duerme el gato" son indistinguibles para BoW. Esto puede ser problemático en casos donde el orden importa.
- **Pierde contexto**: Palabras que tienen múltiples significados, como "banco" (para sentarse o una institución financiera), se tratarán igual, sin tener en cuenta el contexto en que se usan.
- **Vocabulario grande**: Si tienes un vocabulario muy amplio, los vectores se vuelven de alta dimensión, lo que puede hacer el procesamiento menos eficiente.
- **Sparsidad**: La mayoría de las posiciones en los vectores suelen ser ceros, porque un documento individual solo usa un pequeño subconjunto del vocabulario completo.

TF-IDF

Introducción

TF-IDF (Term Frequency-Inverse Document Frequency) es una técnica avanzada en procesamiento de lenguaje natural (NLP) para convertir texto en una representación numérica, similar a Bag of Words (BoW), pero con una diferencia clave: **TF-IDF no solo tiene en cuenta la frecuencia de las palabras en un documento, sino también la importancia relativa de esas palabras en el conjunto de documentos**. Esto permite que TF-IDF reduzca la influencia de palabras comunes (como "el", "es", etc.) y resalte aquellas palabras que son más representativas de un documento específico.

TF

1. **TF (Frecuencia de Término, o Term Frequency)**: Mide cuántas veces aparece una palabra en un documento. La fórmula básica de TF es:

$$TF(t,d) = \frac{\text{Número de veces que el término } t \text{ aparece en el documento } d}{\text{Número total de términos en el documento } d}$$

Cuanto más frecuente sea una palabra en un documento, mayor será su TF.

IDF

2. **IDF (Frecuencia Inversa de Documentos, o Inverse Document Frequency)**: Mide la importancia de una palabra en relación con todos los documentos del conjunto de datos. Si una palabra aparece en muchos documentos, su IDF será bajo, ya que se considera menos relevante. La fórmula de IDF es:

$$IDF(t, D) = \log\left(\frac{N}{1 + \text{Número de documentos que contienen el término } t}\right)$$

Donde:

- N es el número total de documentos en el corpus.
- El 1 en el denominador se agrega para evitar divisiones por cero.

De esta manera, palabras que aparecen en casi todos los documentos (como "el", "es") tendrán un valor de IDF cercano a cero, mientras que palabras raras en el corpus tendrán un valor de IDF alto.

TF-IDF

3. **TF-IDF**: Es el producto de estos dos valores:

$$TF\text{-}IDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

Así, el TF-IDF para una palabra en un documento es alto si la palabra aparece muchas veces en ese documento (alto TF) pero en pocos documentos (alto IDF). Esto permite destacar palabras importantes y eliminar el "ruido" de palabras comunes.

Ejemplo

Ejemplo:

Imagina un corpus con tres documentos:

1. "Me gusta el helado"
2. "El helado es delicioso"
3. "El gato duerme"

El vocabulario completo sería: `["me", "gusta", "el", "helado", "es", "delicioso", "gato", "duerme"]`

Ejemplo

Ahora, calculemos el **TF** para "helado" en el segundo documento:

$$TF(\text{helado}, d2) = \frac{1}{5} = 0.2$$

El **IDF** de "helado" en el corpus es:

$$IDF(\text{helado}, D) = \log\left(\frac{3}{2}\right) \approx 0.18$$

Entonces, el valor **TF-IDF** para "helado" en el segundo documento es:

$$TF\text{-}IDF(\text{helado}, d2, D) = 0.2 \times 0.18 = 0.036$$

Ejemplo

Ahora, calculemos el **TF** para "helado" en el segundo documento:

$$TF(\text{helado}, d2) = \frac{1}{5} = 0.2$$

El **IDF** de "helado" en el corpus es:

$$IDF(\text{helado}, D) = \log\left(\frac{3}{2}\right) \approx 0.18$$

Entonces, el valor **TF-IDF** para "helado" en el segundo documento es:

$$TF\text{-}IDF(\text{helado}, d2, D) = 0.2 \times 0.18 = 0.036$$

Ventajas de TF-IDF

Capta la relevancia: A diferencia de BoW, TF-IDF asigna mayor peso a palabras que son importantes para un documento, pero que no aparecen en muchos documentos del corpus.

Desempeño en recuperación de información: TF-IDF es ampliamente usado en motores de búsqueda, ya que permite identificar documentos relevantes para una consulta, priorizando aquellos que contengan términos distintivos.

Más eficiente que BoW: Al reducir el peso de las palabras comunes, TF-IDF genera vectores menos redundantes, lo que mejora la precisión de los modelos de Machine Learning.

Desventajas

No captura el contexto: Al igual que BoW, TF-IDF sigue siendo una representación basada únicamente en la frecuencia de las palabras, sin tener en cuenta el significado, el contexto o el orden de las mismas.

Dimensionalidad alta: Si el vocabulario es extenso, los vectores generados por TF-IDF aún pueden ser grandes y dispersos (sparse), lo que puede afectar la eficiencia en grandes corpus.

Sensibilidad a datos poco representativos: En conjuntos de datos pequeños o cuando el número de documentos es limitado, TF-IDF puede ser menos efectivo para capturar la importancia real de las palabras.

Desventajas

No captura el contexto: Al igual que BoW, TF-IDF sigue siendo una representación basada únicamente en la frecuencia de las palabras, sin tener en cuenta el significado, el contexto o el orden de las mismas.

Dimensionalidad alta: Si el vocabulario es extenso, los vectores generados por TF-IDF aún pueden ser grandes y dispersos (sparse), lo que puede afectar la eficiencia en grandes corpus.

Sensibilidad a datos poco representativos: En conjuntos de datos pequeños o cuando el número de documentos es limitado, TF-IDF puede ser menos efectivo para capturar la importancia real de las palabras.

NLTK

NLTK

NLTK (Natural Language Toolkit) es una de las bibliotecas más populares para el procesamiento de lenguaje natural (NLP) en Python. Proporciona herramientas y recursos para trabajar con texto, permitiendo desde tareas simples como tokenización hasta análisis lingüísticos más complejos como clasificación de texto y análisis sintáctico.

¿Qué es NLTK?

NLTK es una plataforma integral que facilita el trabajo con texto mediante una serie de funciones que ayudan a procesarlo, analizarlo y modelarlo para tareas de NLP. Incluye una amplia variedad de algoritmos y métodos para tareas de procesamiento de lenguaje, así como acceso a conjuntos de datos de texto precompilados, recursos lingüísticos y herramientas de visualización.

NLTK

Características principales

- **Tokenización**: NLTK permite dividir el texto en unidades menores como palabras (word tokenization) o frases (sentence tokenization).
- **Etiquetado gramatical (POS tagging)**: Permite asignar una etiqueta a cada palabra para identificar su categoría gramatical, como sustantivo, verbo, adjetivo, etc.
- **Stemming y Lemmatization**: Estas técnicas permiten reducir las palabras a su raíz o forma base, respectivamente, lo que es útil para simplificar el procesamiento del lenguaje.

NLTK

Características principales

- **Stopwords**: NLTK ofrece listas de palabras vacías ("stopwords") para eliminar palabras comunes, como "el", "la", "de", que no añaden significado sustancial al análisis.
- **Frecuencia de palabras y análisis de n-grams**: NLTK permite analizar la frecuencia de palabras o frases en un texto, así como crear n-grams, que son secuencias de palabras.
- **Corpora y recursos**: NLTK incluye varios corpora preconstruidos (colecciones de texto) y recursos como WordNet, que es un diccionario léxico para inglés, pero adaptable para otros idiomas.Puedes acceder a textos famosos como el de Shakespeare, noticias de Reuters, etc.

NLTK

Características principales

- **Análisis de sintaxis**: NLTK proporciona herramientas para el análisis sintáctico de oraciones, como análisis de dependencias y parseo.
- **Clasificación de texto**: NLTK tiene implementaciones para algoritmos de Machine Learning como Naive Bayes y SVM para tareas de clasificación de texto.

Implementación en Python

Nuevas librerías

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

Sequential: Tipo de modelo de red neuronal secuencial (donde las capas se apilan una tras otra).

SimpleRNN: Capa de red neuronal recurrente simple, utilizada para procesar secuencias.

Dense: Capa totalmente conectada (o capa "densa") para generar la salida.

Creación dataset artificial

```python
def create_dataset(data, time_steps):

    X, y = [], []

    for i in range(len(data) - time_steps):

        X.append(data[i:i + time_steps])

        y.append(data[i + time_steps])

    return np.array(X), np.array(y)
```

`create_dataset`: Esta función transforma una secuencia de datos en pares de entrada (X) y salida (y).

a. Toma `time_steps` valores de la secuencia como la entrada `X`.
b. El valor inmediatamente después de estos `time_steps` se toma como la salida `y` correspondiente.
c. Así, para una secuencia de longitud `n`, la función genera múltiples pares de `X` e `y`.

- Ejemplo: Si la secuencia es `[1, 2, 3, 4, 5, 6]` y `time_steps = 2`, entonces:
  a. Entrada `X`: `[[1, 2], [2, 3], [3, 4], [4, 5]]`
  b. Salida `y`: `[3, 4, 5, 6]`

Creación dataset artificial

```
np.random.seed(0)

time_steps = 10

data = np.sin(np.linspace(0, 100, 500))  # 500 puntos de una onda senoidal
```

a. **`np.random.seed(0)`**: Establece una semilla para generar números aleatorios para que los resultados sean reproducibles.
b. **`time_steps = 10`**: Define que vamos a tomar 10 pasos de tiempo para predecir el siguiente valor.
c. **`np.sin(np.linspace(0, 100, 500))`**: Genera una secuencia de 500 puntos basados en la función seno entre 0 y 100, lo que crea una serie de datos periódica y continua.

Creación RNN

```
model = Sequential()

model.add(SimpleRNN(50, activation='tanh', input_shape=(time_steps, 1)))

model.add(Dense(1))  # Salida de una dimensión (predicción de un valor)
```

`Sequential`: Crea un modelo secuencial, donde las capas se añaden una tras otra.

`SimpleRNN(50)`: Añade una capa de red recurrente simple con 50 neuronas. Esta capa tomará secuencias de longitud `time_steps=10` y producirá una salida recurrente de 50 unidades.

- **`tanh`**: Función de activación utilizada para limitar los valores de salida de las neuronas entre -1 y 1.

`Dense(1)`: Añade una capa totalmente conectada con 1 neurona de salida, ya que queremos predecir un único valor numérico (el siguiente valor en la secuencia).

Creación RNN

```
model = Sequential()

model.add(SimpleRNN(50, activation='tanh', input_shape=(time_steps, 1)))

model.add(Dense(1))  # Salida de una dimensión (predicción de un valor)
```

input_shape: Define la forma de los datos de entrada que esta capa espera recibir. Es muy importante especificar correctamente la forma de entrada, especialmente en redes recurrentes.

- **time_steps**: Indica el número de pasos de tiempo (o longitud de la secuencia) que la red recibirá como entrada en cada muestra. En tu caso, es 10 (lo que significa que la red recibirá 10 valores consecutivos de la serie temporal en cada muestra de entrada).
- **1**: Esta parte define el número de características en cada paso de tiempo. En este caso, es 1 porque cada paso de tiempo tiene un solo valor (el valor de la serie temporal en ese punto).

En resumen:

- **input_shape=(10, 1)** significa que la red recibirá una secuencia de 10 pasos de tiempo, donde cada paso de tiempo contiene solo una característica.