Procesamiento del Lenguaje Natural

Rodrigo S. Cortez Madrigal

Tarea

Modificar el código de la siguiente implementación para utilizar frecuencias en lugar de PMI.

PMI

La PMI (Pointwise Mutual Information) es una medida de asociación entre dos eventos, en este caso, entre dos palabras. Se calcula como:

$$PMI(x, y) = \log \frac{P(x, y)}{P(x)P(y)}$$

Donde:

- (P(x, y)) es la probabilidad conjunta de que ocurran (x) y (y).
- (P(x)) es la probabilidad de que ocurra (x).
- (P(y)) es la probabilidad de que ocurra (y).
- (P(x, y)) se puede estimar como el número de veces que ocurren (x) y (y) juntos dividido por el total de palabras en el corpus.
- (P(x)) se puede estimar como el número de veces que ocurre (x) dividido por el total de palabras en el corpus.
- (P(y)) se puede estimar como el número de veces que ocurre (y) dividido por el total de palabras en el corpus.

La PMI es útil para identificar asociaciones entre palabras en un corpus de texto. Sin embargo, puede ser sensible a la frecuencia de las palabras y puede dar lugar a valores negativos si las palabras no están asociadas.

Bigramas

Los bigramas son pares de palabras que aparecen juntas en un texto. Por ejemplo, en la frase "El perro ladra", los bigramas son "El perro" y "perro ladra". Los bigramas se utilizan a menudo en el procesamiento del lenguaje natural para modelar la relación entre palabras y mejorar la comprensión del contexto.

Trigramas

Los trigramas son secuencias de tres palabras que aparecen juntas en un texto. Por ejemplo, en la frase "El perro ladra fuerte", los trigramas son "El perro ladra" y "perro ladra fuerte". Los trigramas se utilizan para capturar relaciones más complejas entre palabras y mejorar la comprensión del contexto en el procesamiento del lenguaje natural.

In [8]:
```
import nltk
from nltk import collocations as col
```

In [12]:
```
# En este caso, estamos usando el corpus de Genesis en inglés.
# El corpus de Genesis es un conjunto de textos bíblicos en inglés.

nltk.download('genesis')
```

```
[nltk_data] Downloading package genesis to /Users/roicort/nltk_data...
[nltk_data]   Package genesis is already up-to-date!
```

Out[12]: True

In [19]:
```
bigram_measures = col.BigramAssocMeasures() # Medidas de bigramas
trigram_measures = col.TrigramAssocMeasures() # Medidas de trigramas

# La función `from_words` crea un objeto `BigramCollocationFinder` a partir de una lista de palabras.
finderBi = col.BigramCollocationFinder.from_words(nltk.corpus.genesis.words('english-web.txt'))
finderTri = col.TrigramCollocationFinder.from_words(nltk.corpus.genesis.words('english-web.txt'))

# Luego, usamos el método `nbest` para encontrar las 10 mejores
# combinaciones de palabras (bigrams) según la medida PMI.

print(" Bigrams " + "-" * 50)
print (finderBi.nbest(bigram_measures.pmi, 10))
print(" Trigrams " + "-" * 50)
print (finderTri.nbest(trigram_measures.pmi, 10))
print("-" * 50)
```

```
 Bigrams --------------------------------------------------
[('Allon', 'Bacuth'), ('Ashteroth', 'Karnaim'), ('Ben', 'Ammi'), ('En', 'Mishpat'), ('Jegar', 'Sahadutha'), ('Salt', 'Sea'), ('Whoever', 'sheds'), ('appoint', 'overseers'), ('aro
matic', 'resin'), ('cutting', 'instrument')]
 Trigrams -------------------------------------------------
[('olive', 'leaf', 'plucked'), ('rider', 'falls', 'backward'), ('sewed', 'fig', 'leaves'), ('yield', 'royal', 'dainties'), ('during', 'mating', 'season'), ('Salt', 'Sea', ').'),
('Sea', ').', 'Twelve'), ('Their', 'hearts', 'failed'), ('Valley', ').', 'Melchizedek'), ('doing', 'forced', 'labor')]
--------------------------------------------------
```

In [20]:
```
# Podemos usar un filtro para eliminar bigramas que aparecen menos de un número determinado de veces.

finderBi.apply_freq_filter(3) # Menos de 3 veces
finderTri.apply_freq_filter(3) # Menos de 3 veces
# Ahora, volvemos a encontrar las 10 mejores combinaciones de palabras según la medida PMI.
# Esto eliminará los bigramas que no cumplen con el filtro de frecuencia.

print(" Bigrams " + "-" * 50)
print (finderBi.nbest(bigram_measures.pmi, 10))
print(" Trigrams " + "-" * 50)
print (finderTri.nbest(trigram_measures.pmi, 10))
```

```
 Bigrams --------------------------------------------------
[('Beer', 'Lahai'), ('Lahai', 'Roi'), ('gray', 'hairs'), ('ewe', 'lambs'), ('Most', 'High'), ('many', 'colors'), ('burnt', 'offering'), ('Paddan', 'Aram'), ('east', 'wind'), ('li
ving', 'creature')]
 Trigrams -------------------------------------------------
[('Beer', 'Lahai', 'Roi'), ('seven', 'ewe', 'lambs'), ('God', 'Most', 'High'), ('built', 'an', 'altar'), ('every', 'living', 'creature'), ('an', 'everlasting', 'covenant'), ('eve
ry', 'creeping', 'thing'), ('sixty', '-', 'five'), ('soul', 'may', 'bless'), ('after', 'its', 'kind')]
```

Usando la frecuencua bruta

In [21]:
```
bigram_measures = col.BigramAssocMeasures() # Medidas de bigramas
trigram_measures = col.TrigramAssocMeasures() # Medidas de trigramas

# La función `from_words` crea un objeto `BigramCollocationFinder` a partir de una lista de palabras.
finderBi = col.BigramCollocationFinder.from_words(nltk.corpus.genesis.words('english-web.txt'))
finderTri = col.TrigramCollocationFinder.from_words(nltk.corpus.genesis.words('english-web.txt'))

print(" Bigrams " + "-" * 50)
print (finderBi.nbest(bigram_measures.raw_freq, 10))
print(" Trigrams " + "-" * 50)
print (finderTri.nbest(trigram_measures.raw_freq, 10))
```

```
 Bigrams --------------------------------------------------
[(',', 'and'), (',', '"'), ('of', 'the'), ("'", 's'), ('in', 'the'), ('said', ','), ('said', 'to'), ('.', 'He'), ('the', 'land'), ('.', 'The')]
 Trigrams -------------------------------------------------
[('said', ',', '"'), ('the', 'land', 'of'), ('the', 'father', 'of'), (',', 'and', 'the'), (',', 'and', 'said'), (',', 'saying', ','), ('in', 'the', 'land'), ('He', 'said', ','),
('became', 'the', 'father'), (',', 'and', 'he')]
```

In [22]:
```
# Habría entonces que quitar las puntuaciones y los números.

# Para ello, podemos usar el método `apply_word_filter` para filtrar palabras específicas.
# En este caso, estamos filtrando palabras que son puntuaciones o números.
finderBi.apply_word_filter(lambda w: len(w) < 3 or w.isalpha() == False)
finderTri.apply_word_filter(lambda w: len(w) < 3 or w.isalpha() == False)

print(" Bigrams " + "-" * 50)
print (finderBi.nbest(bigram_measures.raw_freq, 10))
print(" Trigrams " + "-" * 50)
print (finderTri.nbest(trigram_measures.raw_freq, 10))
```

```
 Bigrams --------------------------------------------------
[('the', 'land'), ('and', 'the'), ('the', 'earth'), ('all', 'the'), ('the', 'father'), ('and', 'said'), ('became', 'the'), ('his', 'father'), ('from', 'the'), ('with', 'him')]
 Trigrams -------------------------------------------------
[('became', 'the', 'father'), ('and', 'became', 'the'), ('These', 'are', 'the'), ('are', 'the', 'sons'), ('sons', 'and', 'daughters'), ('all', 'the', 'land'), ('that', 'you', 'ha
ve'), ('after', 'their', 'kind'), ('and', 'all', 'the'), ('called', 'the', 'name')]
```

Comparación

La diferencia entre utilizar la PMI y la frecuencia bruta es que la PMI mide la asociación entre dos palabras en función de su probabilidad conjunta, mientras que la frecuencia bruta simplemente cuenta cuántas veces ocurren juntas. La PMI puede ser más útil para identificar asociaciones significativas entre palabras, mientras que la frecuencia bruta puede ser más útil para identificar patrones generales en el texto.