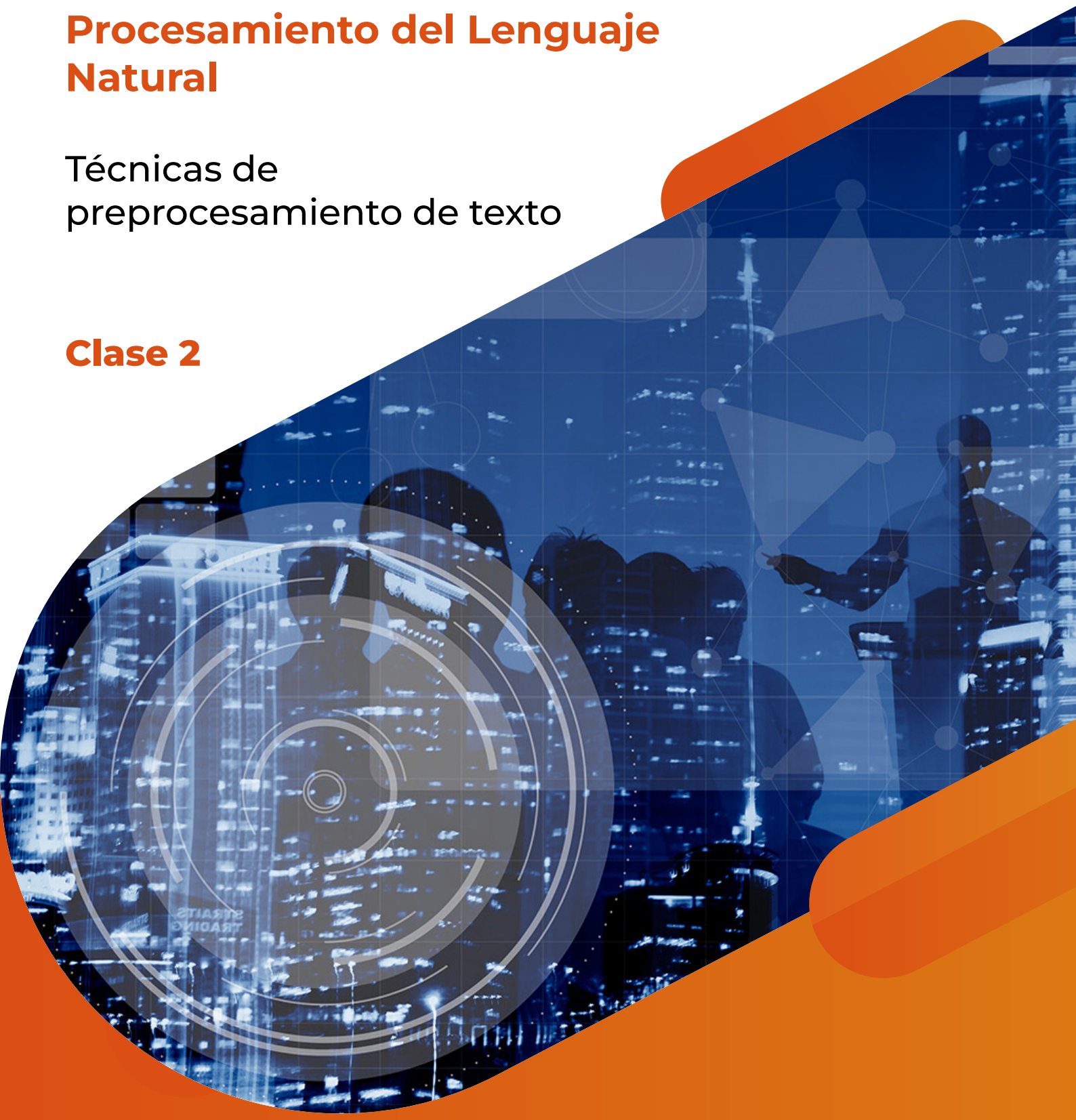
PUCE Virtual
EDUCACIÓN FLEXIBLE

Procesamiento del Lenguaje Natural

Técnicas de preprocesamiento de texto

Clase 2



Maestria en Educacion en Inteligencia Artificial y Entornos Virtuales

La excelencia no se improvisa

Introducción

En esta clase se aborda técnicas fundamentales de preprocesamiento de texto en el Procesamiento de Lenguaje Natural (PLN), destacando estrategias como la normalización de mayúsculas y minúsculas, la eliminación de caracteres especiales y palabras vacías, así como el despalillado y la lematización. Además, se introduce modelos de lenguaje como Bag of Words y TF-IDF, explicando su funcionamiento, aplicaciones y limitaciones. Estas técnicas buscan mejorar la coherencia y calidad de los datos textuales antes de su análisis. Se presentan implementaciones prácticas para cada una de estas técnicas, resaltando su utilidad y limitaciones en distintas aplicaciones del PLN, como el análisis de sentimientos y la recuperación de información.

Los resultados de aprendizaje incluyen analizar los conceptos y técnicas de PLN en ambientes educativos, la capacidad de comprender y aplicar estrategias de preprocesamiento para mejorar la calidad de los datos textuales y evaluar la pertinencia de cada técnica según el contexto, proporcionando así una base sólida para el procesamiento y análisis de texto en entornos computacionales.

Clase 2

3. Técnicas de preprocesamiento de texto

El procesamiento de texto en el Procesamiento de Lenguaje Natural (PLN), transforma datos textuales en representaciones estructuradas para su análisis por algoritmos de IA. Incluye limpieza, normalización, tokenización y vectorización, mejorando la eficiencia en tareas como análisis de sentimientos, traducción automática y chatbots. Su aplicación optimiza la comprensión y generación del lenguaje natural.

3.1. Convertir cadena a minúsculas o mayúsculas

La conversión de cadenas de texto a minúsculas o mayúsculas es una técnica fundamental en el preprocesamiento de texto para el PLN. Esta práctica, conocida como normalización de texto, busca reducir la variabilidad causada por las diferencias de capitalización, permitiendo que palabras como "Casa" y "casa" sean tratadas de manera uniforme por los algoritmos. Al unificar la representación de las palabras, se mejora la consistencia y se facilita el análisis posterior (Manning et al., 2008).

3.1.1. Importancia de la normalización de mayúsculas y minúsculas

En los textos, las palabras pueden aparecer en diferentes formas debido a la capitalización, lo que puede llevar a que los modelos de PLN las interpreten como entidades distintas. Por ejemplo, "Python" y "python" podrían ser consideradas palabras diferentes si no se realiza una normalización adecuada. Al convertir todo el texto a minúsculas o mayúsculas, se reduce esta ambigüedad, lo que resulta en una representación más coherente y una mejora en el rendimiento de los modelos de PLN (Jurafsky y Martin, 2021).

3.1.2. Implementación práctica

La conversión de texto a minúsculas o mayúsculas es una operación sencilla que se puede realizar en diversos lenguajes de programación. En la Figura 1, se presenta un ejemplo en Python: los métodos “lower()” y “upper()” se utilizan para convertir las cadenas de texto a minúsculas y mayúsculas, respectivamente. Es importante tener en cuenta que estas funciones no modifican la cadena original, sino que devuelven una nueva cadena con la transformación aplicada.

Figura 1. Transformar texto en mayúsculas y minúsculas

```
texto = "Hola Mundo"
texto_minusculas = texto.lower()
texto_mayusculas = texto.upper()
print(texto_minusculas)  # Output: "hola mundo"
print(texto_mayusculas)  # Output: "HOLA MUNDO"
```

Nota: Creación propia Diego Ordoñez (2025)

3.1.3. Consideraciones adicionales

Aunque la normalización de mayúsculas y minúsculas es una práctica común, es crucial evaluar su aplicabilidad según el contexto. En algunos casos, la capitalización puede contener información significativa. Por ejemplo, en el reconocimiento de entidades nombradas, distinguir entre "Apple" (la empresa) y "apple" (la fruta) es esencial. Por lo tanto, la decisión de aplicar esta técnica debe basarse en la naturaleza de la tarea y el dominio del texto (Bird et al., 2009).

La conversión de cadenas a minúsculas o mayúsculas es una técnica esencial en el preprocesamiento de texto que contribuye a la uniformidad y coherencia de los datos textuales. Su aplicación adecuada puede mejorar significativamente el rendimiento de los modelos de PLN, aunque siempre debe considerarse el contexto específico para determinar su pertinencia.

3.2. Eliminar caracteres especiales

La eliminación de caracteres especiales es una etapa crucial en el preprocesamiento de texto para el PLN. Este proceso implica la remoción de símbolos no alfanuméricos, como signos de puntuación, emojis y otros caracteres que pueden introducir ruido en el análisis textual. Al limpiar el texto de estos elementos, se mejora la calidad de los datos y se facilita su posterior procesamiento por algoritmos de PLN.

3.2.1. Importancia de eliminar caracteres especiales

Los caracteres especiales pueden distorsionar el análisis de texto al ser interpretados incorrectamente por los modelos de PLN. Por ejemplo, signos de puntuación o emojis pueden no aportar valor semántico en ciertas aplicaciones y, por lo tanto, es recomendable eliminarlos para reducir la dimensionalidad y el ruido en los datos (Manning et al., 2008).

3.2.2. Implementación práctica

La eliminación de caracteres especiales se puede realizar utilizando expresiones regulares en lenguajes de programación como Python. En la Figura 2, se presenta un ejemplo de cómo hacerlo: la función “eliminar_caracteres_especiales” utiliza una expresión regular para sustituir cualquier carácter que no sea una letra, número o espacio en blanco por una cadena vacía, eliminando así los caracteres especiales del texto.

Figura 2. Eliminar caracteres especiales

```python
import re

def eliminar_caracteres_especiales(texto):
    # Eliminar caracteres especiales
    texto_limpio = re.sub(r'[^a-zA-Z0-9\s]', '', texto)
    return texto_limpio

texto = "¡Hola, mundo! ¿Cómo estás? :) #bienvenido"
texto_limpio = eliminar_caracteres_especiales(texto)
print(texto_limpio)  # Output: "Hola mundo Como estas bienvenido"
```

Nota: Creación propia Diego Ordoñez (2025)

3.3. Eliminación de palabras vacías (stop words)

La eliminación de palabras vacías, conocidas en inglés como *stop words*, es una técnica fundamental en el preprocesamiento de texto para el PLN. Estas palabras, que incluyen artículos, preposiciones, conjunciones y otros términos de alta frecuencia, suelen aportar poco valor semántico en tareas específicas y pueden introducir ruido en el análisis. Por lo tanto, su eliminación puede mejorar la eficiencia y precisión de los modelos de PLN.

3.3.1. Importancia de la eliminación de palabras vacías

Las palabras vacías representan una proporción significativa del texto en la mayoría de los idiomas. Aunque son esenciales para la estructura gramatical y la fluidez del lenguaje, en muchas aplicaciones de PLN, como la recuperación de información y el análisis de sentimientos, estas palabras no contribuyen significativamente al significado del contenido. Al eliminarlas, se reduce la dimensionalidad de los datos y se mejora la relación señal-ruido, lo que facilita la identificación de patrones relevantes en el texto (Manning et al., 2008). Este es un tema particularmente importante, y el siguiente video proporciona un mayor detalle con un enfoque práctico: https://youtu.be/vUPAOU2NPls?si=QZ7VWahcYs0k8MOm.

3.3.2. Implementación práctica

La eliminación de palabras vacías se puede implementar utilizando bibliotecas especializadas en diversos lenguajes de programación. En la Figura 3, se presenta un ejemplo en Python utilizando la biblioteca NLTK: el texto se tokeniza en palabras individuales, y luego se filtran aquellas que aparecen en la lista de palabras vacías en español proporcionada por NLTK.Es importante tener en cuenta que la lista de palabras vacías puede variar según el dominio y la aplicación, por lo que es recomendable personalizarla según las necesidades específicas del proyecto.

Figura 3. Eliminar Stop Words

```python
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Descargar recursos necesarios
nltk.download('punkt')
nltk.download('stopwords')

# Texto de ejemplo
texto = "El rápido zorro marrón salta sobre el perro perezoso."

# Tokenización
palabras = word_tokenize(texto)

# Lista de palabras vacías en español
stop_words = set(stopwords.words('spanish'))

# Eliminación de palabras vacías
palabras_filtradas = [palabra for palabra in palabras if palabra.lower() not in stop_words]

print(palabras_filtradas)
# Salida: ['rápido', 'zorro', 'marrón', 'salta', 'perro', 'perezoso', '.']
```

Nota: Creación propia Diego Ordoñez (2025)

3.4 Despalillado (Stemming)

El *stemming*, o despalillado, es una técnica esencial en el preprocesamiento de texto dentro del ámbito del PLN (MathWorks, 2025). Consiste en reducir las palabras a su raíz o base, eliminando afijos como sufijos y prefijos, con el objetivo de agrupar variantes morfológicas de una misma palabra y así simplificar el análisis textual. Por ejemplo, las palabras "corriendo", "corrió" y "correrá" se reducen a la raíz "corr".

3.4.1. Importancia del *stemming*

La aplicación del *stemming* es fundamental en tareas como la recuperación de información y la minería de textos, ya que permite mejorar la eficiencia de los modelos al reducir la dimensionalidad de los datos. Al unificar las variantes de una palabra en una sola forma raíz, se facilita la identificación de temas y patrones en grandes volúmenes de texto (IBM, 2023).

3.4.2. Algoritmos de *stemming*

Existen diversos algoritmos de *stemming*, entre los cuales destacan:

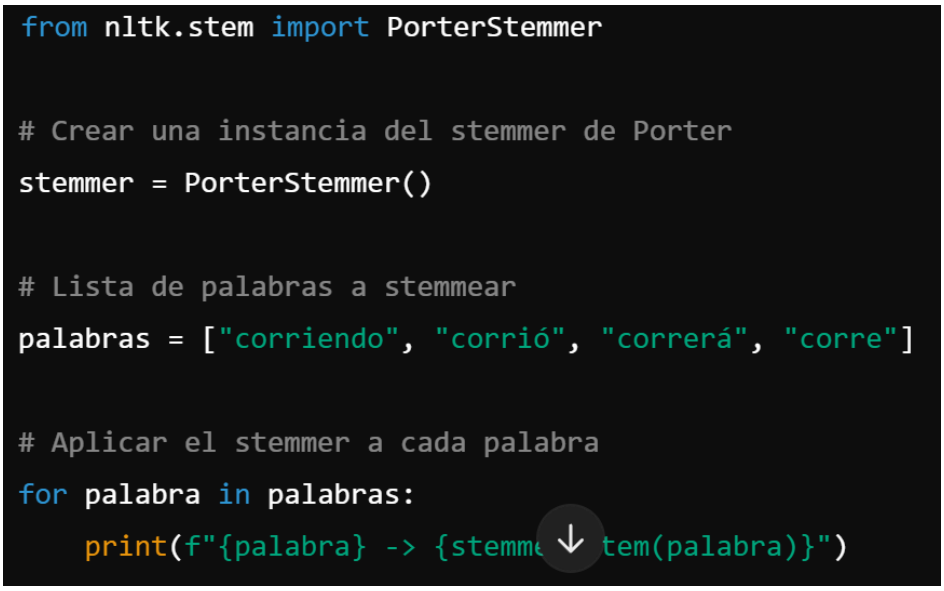
- Algoritmo de Porter: Desarrollado por Martin Porter en 1980, es uno de los más utilizados para el idioma inglés. Se basa en una serie de reglas para eliminar sufijos comunes y reducir las palabras a su forma raíz.

- Algoritmo de Lovins: Propuesto por Julie Beth Lovins en 1968, fue uno de los primeros algoritmos de *stemming*. Utiliza una lista de sufijos y reglas de transformación para reducir las palabras a sus raíces.

3.4.3. Implementación práctica

La implementación del *stemming* se puede realizar en diversos lenguajes de programación. En la Figura 4, se muestra un ejemplo en Python utilizando la biblioteca NLTK.

Figura 4. Stemming



Nota: Creación propia Diego Ordoñez (2025)

Aunque el stemming es una herramienta poderosa, presenta ciertas limitaciones. Uno de los desafíos es que la raíz obtenida no siempre es una palabra válida, lo que puede dificultar la interpretación. Además, el stemming puede no capturar diferencias semánticas entre palabras que comparten la misma raíz, pero tienen significados distintos. Por ejemplo, "organizar" y "organismo" se reducirían a "organ", aunque tienen significados diferentes.

Para superar algunas de estas limitaciones, se utiliza la lematización, que, a diferencia del stemming, considera el contexto y la categoría gramatical de las palabras para reducirlas a su forma base o "lema". Sin embargo, la lematización requiere recursos lingüísticos más complejos y es computacionalmente más costosa.

3.5. Lemmatización

La lematización es una técnica avanzada de preprocesamiento en el ámbito del PLN que consiste en reducir las palabras a su forma base o "lema". A diferencia del stemming, que corta afijos para obtener una raíz que puede no ser una palabra válida, la lematización tiene en cuenta el contexto y la morfología de las palabras para garantizar que la forma resultante sea una palabra reconocida en el idioma (IBM, 2023).

3.5.1. Importancia de la lematización

La lematización es esencial para normalizar el texto y reducir la variabilidad lingüística, lo que mejora la eficacia de los modelos de PLN. Al convertir diferentes formas flexionadas de una palabra en su

lema, se facilita la agrupación de términos relacionados y se mejora la precisión en tareas como la recuperación de información, el análisis de sentimientos y la traducción automática.

3.5.2. Diferencias entre lematización y stemming

Aunque tanto la lematización como el stemming buscan reducir las palabras a una forma base, difieren en su enfoque y precisión. El stemming aplica reglas heurísticas para eliminar afijos, lo que puede resultar en raíces no reconocidas. Por ejemplo, "correrá", "corriendo" y "corrió" podrían reducirse a "corr". En contraste, la lematización analiza la estructura morfológica y el contexto gramatical para devolver el lema correcto, en este caso, "correr".

3.5.3. Implementación práctica

La lematización se implementa comúnmente utilizando bibliotecas especializadas en lenguajes de programación como Python. En la Figura 5, se muestra un ejemplo utilizando la biblioteca NLTK: En este ejemplo, el texto se tokeniza y se etiqueta gramaticalmente para determinar la categoría de cada palabra. Luego, el lematizador utiliza esta información para convertir las palabras a sus lemas correspondientes, considerando su función sintáctica en la oración (DataCamp, 2023).

Figura 5. Lematización

```python
import nltk
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

# Descargar recursos necesarios
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

# Función para obtener la etiqueta de WordNet
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

# Texto de ejemplo
texto = "Los niños corrieron rápidamente hacia el parque."

# Tokenización
palabras = word_tokenize(texto)

# Etiquetado gramatical
pos_tags = nltk.pos_tag(palabras)

# Inicializar lematizador
lemmatizer = WordNetLemmatizer()

# Lematización con consideración de la categoría gramatical
lemas = []
for token, tag in pos_tags:
    wn_tag = get_wordnet_pos(tag)
    if wn_tag:
        lemas.append(lemmatizer.lemmatize(token, pos=wn_tag))
    else:
        lemas.append(lemmatizer.lemmatize(token))

print(lemas)
# Salida: ['Los', 'niño', 'correr', 'rápidamente', 'hacia', 'el', 'parque', '.']
```

Nota: Creación propia Diego Ordoñez (2025)

3.6. Etiquetado de la parte del discurso

El etiquetado de la parte del discurso, conocido en inglés como *Part-of-Speech Tagging* (POS tagging), es una técnica fundamental en el PLN, que consiste en asignar a cada palabra de un texto su categoría gramatical correspondiente, como sustantivo, verbo, adjetivo, entre otros. Esta tarea es esencial para comprender la estructura sintáctica de las oraciones y facilita el análisis semántico del lenguaje. (Cunningham et al., 2011; Jurafsky y Martin, 2009).

3.6.1. Importancia del etiquetado de la parte del discurso

El etiquetado POS es crucial en diversas aplicaciones del PLN, incluyendo:

- Análisis sintáctico: Proporciona información sobre la estructura de las oraciones, lo que es esencial para la comprensión gramatical.
- Desambiguación semántica: Ayuda a determinar el significado correcto de una palabra según su categoría gramatical y contexto.
- Reconocimiento de entidades nombradas: Facilita la identificación de nombres propios, lugares y organizaciones en un texto.
- Traducción automática: Mejora la precisión al identificar las funciones gramaticales de las palabras en diferentes idiomas.

3.6.2. Métodos de etiquetado POS

Existen varios enfoques para realizar el etiquetado de la parte del discurso:

1. Basados en reglas: Utilizan un conjunto de reglas lingüísticas predefinidas para asignar etiquetas gramaticales. Aunque pueden ser precisos en dominios específicos, su desarrollo y mantenimiento son laboriosos.
2. Basados en aprendizaje automático: Emplean algoritmos que aprenden patrones a partir de datos etiquetados. Modelos como los de cadenas de Markov ocultas (HMM) y los clasificadores de máxima entropía son comunes en este enfoque.
3. Enfoques híbridos: Combinan reglas lingüísticas con técnicas de aprendizaje automático para mejorar la precisión y adaptabilidad.

3.6.3. Implementación práctica

En Python, bibliotecas como NLTK y spaCy ofrecen herramientas para el etiquetado POS. A continuación, se muestra un ejemplo utilizando NLTK: este script tokeniza el texto y asigna etiquetas gramaticales a cada palabra. Es importante destacar que, aunque NLTK es una herramienta poderosa, su soporte para el español es limitado. Para un etiquetado POS en español más preciso, se recomienda utilizar bibliotecas especializadas o modelos entrenados específicamente para este idioma (Bird et al., 2009).

Figura 6. Part-of-Speech Tagging

```python
from nltk.tokenize import word_tokenize

# Descargar recursos necesarios
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# Texto de ejemplo
texto = "El rápido zorro marrón salta sobre el perro perezoso."

# Tokenización
palabras = word_tokenize(texto)

# Etiquetado POS
etiquetas = nltk.pos_tag(palabras, lang='spa')

print(etiquetas)
```

Nota: Creación propia Diego Ordoñez (2025)

3.6.4. Retos y consideraciones adicionales

El etiquetado POS enfrenta desafíos como:

- Ambigüedad léxica: Muchas palabras pueden pertenecer a diferentes categorías gramaticales según el contexto. Por ejemplo, "banco" puede ser un sustantivo ("Me senté en el banco") o un verbo ("Yo banco esa idea").
- Variaciones lingüísticas: Las diferencias dialectales y estilísticas pueden afectar la precisión del etiquetado.
- Palabras desconocidas: La aparición de neologismos o términos técnicos no presentes en el corpus de entrenamiento puede dificultar la asignación correcta de etiquetas.

El etiquetado de la parte del discurso es una técnica esencial en el preprocesamiento de texto que facilita la comprensión y análisis del lenguaje natural. Su correcta aplicación mejora significativamente el rendimiento de diversas aplicaciones en PLN, desde el análisis sintáctico hasta la traducción automática.

En esta sección se ha tratado la temática de preprocesamiento de texto, con varias técnicas de PLN. En el siguiente video se encuentra una visión general del tema: https://youtu.be/I173TmCTxpk?si=MXJJXJi7vGHilWG6.

4. Modelos de Lenguaje

En el PLN, los modelos de lenguaje son fundamentales para convertir texto en representaciones numéricas que puedan ser procesadas por algoritmos de machine learning. Dos de las técnicas más utilizadas para esta tarea son Bag of Words (BoW) y TF-IDF (Term Frequency-Inverse Document

Frequency). Esta sección explora estos métodos, sus aplicaciones, ventajas y limitaciones, con ejemplos prácticos y referencias académicas.

4.1. Bag of Words

El modelo Bag of Words (BoW) es una de las técnicas más simples y ampliamente utilizadas para representar texto en PLN. Este método se basa en la idea de contar la frecuencia de las palabras en un documento, ignorando el orden y la estructura gramatical. Aunque parece rudimentario, BoW ha demostrado ser efectivo en tareas como clasificación de texto y análisis de sentimientos.

4.1.1. Funcionamiento de BoW

El proceso de BoW consta de tres pasos principales:

1. Tokenización: Dividir el texto en palabras individuales (tokens).

2. Construcción del vocabulario: Crear un diccionario de todas las palabras únicas en el corpus.

3. Vectorización: Representar cada documento como un vector de frecuencias de palabras.

Ejemplo: Supongamos que tenemos los siguientes dos documentos:

- Documento 1: "El gato come pescado"

- Documento 2: "El perro come carne"

El vocabulario sería: ["el", "gato", "come", "pescado", "perro", "carne"]. Los vectores BoW para los documentos serían:

- Documento 1: [1, 1, 1, 1, 0, 0]

- Documento 2: [1, 0, 1, 0, 1, 1]

4.1.2. Aplicaciones de BoW

BoW es ampliamente utilizado en tareas de clasificación de texto, como la detección de spam o la categorización de noticias. Por ejemplo, en un clasificador de spam, las palabras como "oferta" o "gratis" pueden tener una alta frecuencia en correos no deseados, lo que permite identificarlos (Manning y Schütze, 1999).

4.1.3. Limitaciones de BoW

A pesar de su simplicidad, BoW tiene varias limitaciones:

- Ignora el orden de las palabras: Frases como "el perro muerde al hombre" y "el hombre muerde al perro" tendrían la misma representación.

- No captura el significado: Palabras como "banco" (institución financiera) y "banco" (asiento) se tratan igual.

- Dimensionalidad alta: En corpus grandes, el vocabulario puede volverse extremadamente grande, lo que aumenta la complejidad computacional.

4.2. TF-IDF

El modelo TF-IDF es una mejora sobre BoW que no solo considera la frecuencia de las palabras en un documento, sino también su importancia en el corpus. TF-IDF asigna un peso a cada palabra en función de su frecuencia en el documento (TF) y su rareza en el corpus (IDF).

4.2.1. Funcionamiento de TF-IDF

TF-IDF se calcula en dos partes:

1. Term Frequency (TF): Mide la frecuencia de una palabra en un documento (Figura 7).

Figura 7. Term Frequency

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

Nota: Creación propia Diego Ordoñez (2025)

2. Inverse Document Frequency (IDF): Mide la importancia de una palabra en el corpus (Figura 8).

Figura 8. Inverse Document Frequency

$$IDF(t,D) = \log\left(\frac{\text{Número total de documentos en } D}{\text{Número de documentos que contienen } t}\right)$$

Nota: Creación propia Diego Ordoñez (2025)

El valor TF-IDF se obtiene multiplicando TF por IDF (Figura 9).

Figura 9. TF-IDF

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

Nota: Creación propia Diego Ordoñez (2025)

Ejemplo: Supongamos que tenemos los siguientes documentos:

- Documento 1: "El gato come pescado"

- Documento 2: "El perro come carne"

- Documento 3: "El gato y el perro juegan"

El cálculo de TF-IDF para la palabra "gato" en el Documento 1 sería:

- TF("gato", Documento 1) = 1/4 = 0.25

- IDF("gato", Corpus) = log(3/2) ≈ 0.176

- TF-IDF("gato", Documento 1) = 0.25 × 0.176 ≈ 0.044

4.2.2. Aplicaciones de TF-IDF

TF-IDF es ampliamente utilizado en sistemas de recuperación de información, como motores de búsqueda. Por ejemplo, en Google Search, TF-IDF ayuda a identificar documentos relevantes para una consulta basándose en la importancia de las palabras clave (Manning et al., 2008).

También se utiliza en tareas de minería de texto, como la extracción de temas principales de un conjunto de documentos. Por ejemplo, en un corpus de reseñas de películas, palabras con un alto TF-IDF como "acción" o "comedia" pueden indicar los géneros predominantes.

4.2.3. Ventajas de TF-IDF

- Reduce el peso de palabras comunes: Palabras como "el" o "de" tienen un IDF bajo, lo que reduce su importancia en la representación.

- Captura la relevancia de las palabras: Las palabras raras pero significativas reciben un peso más alto.

4.2.4. Limitaciones de TF-IDF

- No captura relaciones semánticas: Al igual que BoW, TF-IDF no entiende el significado de las palabras.

- Dependencia del corpus: El IDF depende del corpus, por lo que puede variar en diferentes conjuntos de datos.

4.3. Comparación entre BoW y TF-IDF

Ambas técnicas son fundamentales en PLN, pero tienen diferencias clave:

- Simplicidad: BoW es más simple y rápido de calcular, mientras que TF-IDF requiere cálculos adicionales.

- Precisión: TF-IDF es más preciso al capturar la importancia de las palabras, especialmente en tareas de recuperación de información.

- Uso de recursos: BoW puede generar vectores de alta dimensionalidad, mientras que TF-IDF reduce este problema al ponderar las palabras.

Ejemplo: En un clasificador de noticias, BoW podría tratar palabras como "fútbol" y "deporte" como independientes, mientras que TF-IDF podría dar más peso a "fútbol" si aparece con menos frecuencia en el corpus, lo que ayudaría a identificar noticias específicas sobre fútbol.

Tanto Bag of Words como TF-IDF son técnicas esenciales en PLN para convertir texto en representaciones numéricas. BoW es simple y efectivo para tareas básicas, mientras que TF-IDF ofrece una mejora al considerar la importancia de las palabras en el corpus. Aunque ambas tienen limitaciones, su combinación con técnicas más avanzadas, como embeddings y modelos de lenguaje basados en redes neuronales, ha permitido avances significativos en el campo del PLN.

Referencias

Bird, S., Klein, E., y Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.

Cunningham, H., Tablan, V., Roberts, A., y Bontcheva, K. (2011). *Getting More Out of Biomedical Documents with GATE's Full Lifecycle Open Source Text Analytics*. PLoS Computational Biology, 7(12), e1002243.

DataCamp. (2023). *Stemming y lematización en Python*. Recuperado de https://www.datacamp.com/es/tutorial/stemming-lemmatization-python

IBM. (2023*). ¿Qué son el stemming y la lematización?* Recuperado de https://www.ibm.com/es-es/topics/stemming-lemmatization

Jurafsky, D., y Martin, J. H. (2009*). Speech and Language Processing (2nd ed.).* Pearson.

Jurafsky, D., y Martin, J. H. (2021). *Speech and Language Processing (3rd ed.).* Draft available at https://web.stanford.edu/~jurafsky/slp3/

Lovins, J. B. (1968). *Development of a Stemming Algorithm,* Mechanical Translation and Computational Linguistics, Vol. 11, No. 1-2, 1968, pp. 22-31.

Manning, C. D., y Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press.

Manning, C. D., Raghavan, P., y Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

MathWorks. (2025). *Stemming - MATLAB y Simulink*. Recuperado de https://la.mathworks.com/discovery/stemming.html

Porter, M. (1980). *The Porter Stemming Algorithm*. https://tartarus.org/martin/PorterStemmer/index-old.html

Salton, G., y Buckley, C. (1988). *Term-weighting approaches in automatic text retrieval*. Information Processing y Management, 24(5), 513–523.

Términos para definición

Normalización

La normalización en el Procesamiento de Lenguaje Natural (PLN) es el proceso de transformar y estandarizar los textos para reducir su variabilidad y mejorar su interpretación por parte de los algoritmos. Esto es clave en tareas como la clasificación, recuperación de información y análisis semántico. Para lograrlo, se emplean técnicas como la conversión del texto a minúsculas, evitando diferencias innecesarias entre palabras con distintas capitalizaciones. También se eliminan caracteres especiales y signos de puntuación cuando no aportan valor al análisis.

Además, la normalización incluye la expansión de contracciones para hacer los textos más comprensibles para los modelos de PLN, así como la corrección ortográfica, que ayuda a evitar errores tipográficos que podrían distorsionar el análisis. Técnicas más avanzadas, como el stemming y la lematización, reducen las palabras a su forma base para unificar términos con el mismo significado. Estos pasos garantizan que los textos procesados sean más estructurados y precisos, facilitando su análisis y optimizando el rendimiento de los modelos de PLN.

Expresión Regular (Regex)

Las expresiones regulares (regex) son secuencias de caracteres que permiten definir patrones de búsqueda en cadenas de texto, facilitando su manipulación en tareas como el Procesamiento de Lenguaje Natural (PLN), la validación de datos y la extracción de información. Gracias a su capacidad para identificar patrones específicos, las regex son ampliamente utilizadas para procesar grandes volúmenes de texto de manera eficiente, permitiendo realizar búsquedas avanzadas y modificaciones precisas dentro de los datos textuales.

Las expresiones regulares combinan caracteres literales y metacaracteres especiales para definir patrones de coincidencia. Entre los elementos más comunes se encuentran los metacaracteres, como . (cualquier carácter), \d (dígitos) y \w (alfanuméricos), así como los cuantificadores, que establecen la cantidad de repeticiones de un patrón (+, *, {n,m}). Además, las anclas como ^ (inicio de línea) y $ (final de línea) permiten delimitar la ubicación exacta de la búsqueda dentro del texto. Estas herramientas son esenciales en programación y se implementan en lenguajes como Python y Java facilitando tareas de búsqueda y manipulación de datos con precisión y rapidez.

PUCE Virtual

EDUCACIÓN FLEXIBLE

La excelencia no se improvisa

síguenos