



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

Instituto Tecnológico de Apizaco

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE APIZACO

**SISTEMA DE TUTORÍA INTELIGENTE CONVERSACIONAL PARA
MEJORAR LA COMPETENCIA EN LA LENGUA NÁHUATL DE
HABLANTES NO NATIVOS A TRAVÉS DEL ANÁLISIS DE
TRANSCRIPCIONES (FASE 1)**

TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL

**PARA OBTENER EL TÍTULO PROFESIONAL DE:
INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y
COMUNICACIONES**

PRESENTA:

TORRES MARAVILLA JOSE ALONSO

ASESOR:

DR. JOSÉ CRISPÍN HERNÁNDEZ HERNÁNDEZ

APIZACO, TLAX., OCTUBRE 2025



Instituto Tecnológico de Apizaco
Sistemas y Computación

Apizaco, Tlaxcala, 29/agosto/2025

Oficio No. S.C./OPV/0508/25

ASUNTO: Liberación de proyecto
para titulación integral

ISRAEL JORGE SÁNCHEZ SÁNCHEZ
JEFE DEL DEPTO. DE DIVISIÓN DE ESTUDIOS PROFESIONALES
P R E S E N T E.

POR ESTE MEDIO LE INFORMO QUE HA SIDO LIBERADO EL SIGUIENTE PROYECTO PARA LA TITULACIÓN INTEGRAL.

NOMBRE DEL EGRESADO:	TORRES MARAVILLA JOSÉ ALONSO
CARRERA:	INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES
NO. DE CONTROL:	20370973
NOMBRE DEL PROYECTO:	Sistema de Tutoría Inteligente Conversacional para Mejorar la Competencia en la lengua náhuatl de Hablantes No Nativos a Través del Análisis de Transcripciones (fase 1)
PRODUCTO U OPCIÓN:	TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL

AGRADEZCO DE ANTEMANO SU VALIOSO APOYO EN ESTA IMPORTANTE ACTIVIDAD PARA LA FORMACIÓN PROFESIONAL DE NUESTROS EGRESADOS.

ATENTAMENTE

*Excelencia en Educación Tecnológica.
Pensar para Servir, Servir para Triunfar®*

MIQUELINA SÁNCHEZ PULIDO
JEFA DEL DEPTO. DE SISTEMAS Y COMPUTACIÓN

ccp. Coordinación de carrera
Expediente
MSP/mrol*



SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL
DE MÉXICO
INSTITUTO TECNOLÓGICO DE APIZACO
DEPARTAMENTO DE
SISTEMAS Y COMPUTACIÓN

JOSÉ CRISPÍN HERNÁNDEZ HERNÁNDEZ

ASESOR INTERNO



2025
Año de
La Mujer
Indígena

Av. Instituto Tecnológico No. 418, San Andrés Ahuashuatepec,
Municipio de Tzompantepec, Tlaxcala, Mex. C.P. 90491
Tel. (241) 4172010 Ext. 115, e-mail: sistemas@apizaco.tecnm.mx
tecnm.mx | apizaco.tecnm.mx





Educación
Secretaría de Educación Pública



TECNOLÓGICO
NACIONAL DE MÉXICO



Instituto Tecnológico de Apizaco
División de Estudios Profesionales

Tzompantepec, Tlaxcala **09/octubre/2025**

TORRES MARAVILLA JOSÉ ALONSO
EGRESADO(A) DE LA CARRERA DE INGENIERÍA
EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES
CON NÚMERO DE CONTROL 20370973.
P R E S E N T E.

Por este medio, me permito informar a usted que por aprobación de la comisión revisora asignada para valorar el trabajo que mediante la opción **TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL** y cuyo tema es: **SISTEMA DE TUTORÍA INTELIGENTE CONVERSACIONAL PARA MEJORAR LA COMPETENCIA EN LA LENGUA NÁHUATL DE HABLANTES NO NATIVOS A TRAVÉS DEL ANÁLISIS DE TRANSCRIPCIONES (FASE 1)**, conforme a lo establecido en el procedimiento para la obtención del Título Profesional en el Instituto Tecnológico, la División de Estudios Profesionales a mi cargo le otorga la:

AUTORIZACIÓN DE IMPRESIÓN

Debiendo entregar **2 USB** del mismo, apropiadamente empastados, a la brevedad, para presentar su Acto de Recepción Profesional.

Sin otro particular por el momento me despido.

ATENTAMENTE
Excelencia en Educación Tecnológica®
Pensar para Servir, Servir para Triunfar®

CECILIA YARELLY ARREGUÍN LARA
JEFA DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES

ccp. Archivo

CYAL/rrc



2025
Año de
La Mujer
Indígena

Av. Instituto Tecnológico No. 418, San Andrés Ahuashuatepec,
Municipio de Tzompantepec, Tlaxcala, Mex. C.P. 90491 Tel. (241) 4172010
Ext. 304, e-mail: dep_titulacion@apizaco.tecnm.mx | apizaco.tecnm.mx



Agradecimientos

Quiero expresar agradecimientos a quienes son los pilares fundamentales en mi vida: mis padres y mis hermanos. Papá y mamá, su amor incondicional, sus sacrificios y el constante apoyo ha sido lo que me ha impulsado a alcanzar mis sueños. Cada lección, cada regaño y cada cosa que me han enseñado han forjado la persona que soy. A mis hermanos, por ser mis primeros compañeros de aventuras, por las risas compartidas y por estar siempre ahí, ofreciendo una mano amiga o una palabra que me ayude. A mi pareja, gracias por ser mi confidente, mi apoyo incondicional y que ha hecho que cada desafío parezca más sencillo de superar y afrontar. Contigo he aprendido el verdadero significado de la complicidad y el crecimiento mutuo. Gracias por creer en mí, por celebrar mis victorias y estar en mis derrotas. No puedo dejar de mencionar a ese grupo de personas extraordinarias que llamo amigos. Gracias por las risas incontrolables, por las aventuras inolvidables, su lealtad, su alegría y su capacidad de escuchar han sido de gran ayuda al superar lo que es la etapa de la universidad. Han estado presentes en muchas etapas importantes, celebrando los éxitos y apoyándome en los fracasos. A los profesionales del Instituto Tecnológico de Apizaco, cuyo conocimiento y disposición para compartir sus experiencias ayudaron a realizar este trabajo. Su paciencia y guía fueron fundamentales para el desarrollo de mis habilidades técnicas; por brindarme las herramientas académicas y el entorno necesario para mi crecimiento. A los docentes, compañeros y personal administrativo, cuya colaboración y aliento hicieron posible este logro. Mención especial al Dr. José Crispín Hernández Hernández, mi asesor, por su invaluable orientación, paciencia y confianza en mi capacidad. Sus consejos fueron clave para superar obstáculos y alcanzar los objetivos planteados.

A todos, mi eterno agradecimiento por ser parte de esta trayectoria.

Contenido

Agradecimientos	4
Resumen	9
Introducción.	10
Generalidades de la institución	11
Ubicación	11
Logotipo de la empresa	11
Tecnológico Nacional de México	11
Misión	11
Visión	11
Valores	11
Datos generales del proyecto	12
Nombre del proyecto	12
Objetivos	12
Objetivo principal	12
Delimitación	12
Justificación	12
Marco teórico	13
Estado del arte	13
Investigación sobre PNL	14

Procesamiento del lenguaje natural (NLP)	14
Clasificación de textos.	15
Conjunto de datos para la clasificación de textos	15
Clasificación de documentos de texto mediante Naive Bayes	18
Clasificación de texto mediante regresión logística	23
RNN para clasificaciones de texto en PNL	29
Clasificación de textos mediante CNN	36
Extracción de información	39
Extracción de información en PNL	40
Reconocimiento de entidades nombradas	42
Extracción de relaciones en PNL	47
Traducción automática	53
Traducción automática con Trasnformer	54
Resumen de texto	62
Resumen utilizando modelo Hugging Face Model	62
Resumen utilizando Sumy	66
Generación de texto	72
Fnet	72
Generación de texto usando Recurrent Long Short Term Memory Network	74
Herramientas Tecnológicas	79
Visual Studio Code	79
Lenguaje Python	80
Pandas	80
NumPy	81

TensorFlow	81
Keras	81
Tkinter	81
Sistema	82
BLEU	82
Partes del sistema	86
Normalización	86
Segmentador	90
N-Gramas	94
Patrones Gramaticales	97
Detección de errores	103
Detección de errores PNL	111
Evaluación BLEU	116
Pequeño traductor utilizando RNN (Red Neuronal Recurrente)	119
Ejecutable	123
Conclusiones	126
Recomendaciones	127
Experiencia Obtenida	127
Competencias Desarrolladas	128
Referencias	129
Anexo A	132
A1 BLEU	132
A2 Normalización	138
A3 Segmentador	143

A4 N-Gramas	149
A5 Patrones gramaticales	153
A6 Detección de errores	163
A7 Detección de errores PNL	179
A8 Evaluación BLEU	186
A9 Pequeño traductor utilizando RNN (Red Neuronal Recurrente)	191
A10 Ejecutable	198

Resumen

El presente trabajo nos muestra una investigación realizada sobre lo que es el Procesamiento de Lenguaje Natural (PNL) dentro de la investigación se abordaron temas que nos serán de mucha utilidad a la hora de empezar con la realización del proyecto tenemos temas como lo son la clasificación de textos, por ejemplo, permite organizar y categorizar documentos automáticamente, facilitando la búsqueda y recuperación de información. La extracción de información se encarga de identificar y extraer datos estructurados de textos no estructurados, lo que es crucial para la creación de bases de conocimiento y el análisis de grandes volúmenes de texto.

La traducción automática ha revolucionado la comunicación global, rompiendo barreras idiomáticas y permitiendo la interacción entre personas de diferentes lenguas. El PLN también abarca la generación y el resumen de textos. El resumen de textos condensa documentos largos en versiones más cortas y coherentes, ahorrando tiempo y destacando la información esencial. La generación de textos, un campo en constante evolución capacita a las máquinas para producir contenido original y relevante, desde artículos de noticias hasta respuestas conversacionales, abriendo un sinfín de posibilidades para la automatización y la creatividad.

Finalmente el proyecto realizado tomamos las bases de los primeros pasos para la generación de dicho proyecto como lo son la segmentación de oraciones con el cual se desarrolla código, una tarea crucial que permite desglosar textos extensos en unidades más manejables para un análisis detallado, también vemos lo que son la identificación de patrones lingüísticos del cual también generamos código que nos ayude a entender y visualizar esto, de igual manera realizamos la identificación de errores y lo mostramos en graficas, finalmente mostramos evaluación de oraciones utilizando método BLEU y un pequeño traductor que realizamos tomando el modelo de una RNN y su función.

Todo con el fin de superar las barreras lingüísticas entre el español y el náhuatl, sentando las bases para una innovadora herramienta educativa.

Introducción.

El náhuatl, conocido por sus hablantes como Nawatlahtolli, es una lengua que se inscribe en la vasta familia lingüística yuto-nahua. Su presencia en México es profundamente histórica y se extiende a lo largo de cuatro principales regiones etnolingüísticas: el Bajío Pacífico, el Centro Huasteca Golfo, el Pacífico Sur y la región Maya o Sureste. Además de estas áreas ancestrales, la lengua ha encontrado nuevos asentamientos en el norte del país en tiempos contemporáneos, demostrando su capacidad de adaptación y pervivencia.

De acuerdo con el "Catálogo de las Lenguas Indígenas Nacionales: variantes lingüísticas de México con sus autodenominaciones y referencias geoestadísticas", la huella histórica del Nawatlahtolli se extiende por 16 estados de la República Mexicana.

Hemos explorado la rica historia y la amplia distribución geográfica del náhuatl (Nawatlahtolli), una lengua milenaria que se extiende por gran parte del territorio mexicano. Ahora, adentrémonos en cómo esta lengua ancestral se encuentra con la vanguardia tecnológica de nuestro tiempo: los sistemas conversacionales.

Los sistemas conversacionales han tomado una importancia mayor y estos ahora se encuentran cada vez más presentes en nuestro día a día, transformando la forma en que interactuamos con la tecnología. Estos sistemas, en su primera fase de desarrollo, enfrentan desafíos significativos, especialmente cuando se trata de la comprensión y generación de lenguaje natural en idiomas con estructuras y particularidades únicas. Aquí es donde el Procesamiento del Lenguaje Natural (PNL) juega un papel crucial, sirviendo como la columna vertebral tecnológica que permite a estas interfaces interpretar y responder de manera inteligente.

Uno de los componentes más vitales en la construcción de un sistema conversacional es el análisis de transcripciones. Este proceso implica la meticulosa examinación de conversaciones grabadas o textuales para identificar patrones, extraer información clave y comprender las intenciones del usuario. Cuando nos adentramos en el ámbito de las lenguas indígenas, como el náhuatl, la tarea se vuelve aún más compleja y fascinante. El análisis de transcripciones del español al náhuatl no solo requiere un conocimiento profundo de ambas lenguas, sino también una sensibilidad a las variaciones dialectales y las particularidades culturales inherentes al náhuatl. Esta fase inicial es fundamental empezar con las bases simples del PNL como lo son la segmentación de oraciones, identificación de patrones lingüísticos, detección de errores, evaluación de traducción utilizando el modelo de BLEU; todo con el fin de construir un modelo lingüístico sólido que permita al sistema conversacional comprender y comunicarse eficazmente en esta lengua ancestral.

Generalidades de la institución

Ubicación

Av. Instituto Tecnológico No. 418, San Andrés Ahuashuatepec, Municipio de Tzompantepec, Tlaxcala, Méx. C.P. 90491.

Tel: 241 417 2010 Ext. 136

Correo: difusion@apizaco.tecnm.mx

Web: Apizaco.tecnm.mx

Logotipo de la empresa



Tecnológico Nacional de México

El Tecnológico Nacional de México (TecNM) es una institución de educación superior tecnológica que forma profesionales e investigadores con habilidades en el campo laboral, pensamiento crítico, sentido ético, colaborativos y competentes para contribuir al desarrollo nacional y regional. Fue creada en 2014. El TecNM ofrece educación de excelencia y es la institución de educación superior tecnológica con mayor movilidad social en México. La misión del TecNM es formar profesionales mediante un modelo educativo integral de calidad, que garantiza una formación técnica humanística, con capacidad para investigar y aplicar tecnología con impacto en el desarrollo de la sociedad.

Misión

Formar profesionistas que sean líderes visionarios, emprendedores, con capacidad de generar soluciones disruptivas, innovando en la ciencia y la tecnología, comprometidos con el desarrollo económico, responsabilidad social teniendo una visión sostenible y sustentable.

Visión

Ser una Institución de educación superior tecnológica de vanguardia, formadora de ciudadanos del mundo, con conocimientos intercultural, respetando la diversidad social.

Valores

- Honestidad
- Responsabilidad
- Lealtad

Datos generales del proyecto

Nombre del proyecto

“Sistema de Tutoría Inteligente Conversacional para Mejorar la Competencia en la lengua náhuatl de Hablantes No Nativos a Través del Análisis de Transcripciones (fase 1)”

Objetivos

- Análisis automático de transcripciones.
- Identificación de errores comunes en gramática, vocabulario y pronunciación.
- Generación de retroalimentación personalizada con ejemplos prácticos y ejercicios.

Objetivo principal

El objetivo principal de este proyecto es diseñar e implementar un CITS que permita a las personas que no son hablantes de la lengua indígena, mejorar su dominio del náhuatl.

Delimitación

El proyecto se implementará exclusivamente en el TecNM/ Instituto Tecnológico de Apizaco Se proporcionará capacitación específica para garantizar que el personal pueda utilizar la plataforma de manera efectiva, para mejorar la competencia en la lengua náhuatl de personas que no son hablantes de la lengua materna (los docentes y asistentes educativos) a través del análisis de transcripciones que permitan la apropiación de la escritura y gramática básica de la lengua náhuatl.

Justificación

Dada la necesidad de incorporar el uso de las lenguas indígenas en la escuela, es importante tener en cuenta que los docentes carecen de formación o experiencia en la enseñanza de las lenguas originarias como lenguas maternas, así como en la enseñanza de estos idiomas como segunda lengua. Por lo tanto, es esencial enfocarse en dos aspectos: la apropiación de los conceptos y metodologías básicas por parte de los docentes y asistentes educativos, y la organización de los procesos de formación de manera que los docentes de lenguas construyan colectivamente su experiencia y aprendan a partir de su participación en los procesos cotidianos de enseñanza.

Tomando en cuenta el primer aspecto, se propone un Sistema de tutoría Inteligente conversacional (CITS) para mejorar la competencia en la lengua náhuatl de personas que no son hablantes de la lengua materna (los docentes y asistentes educativos) a través del análisis de transcripciones que permitan la apropiación de la escritura y gramática básica de la lengua náhuatl.

Marco teórico

El Procesamiento del Lenguaje Natural (PNL) es el pilar central que permite a los sistemas conversacionales comprender, interpretar y generar lenguaje humano. Desde una perspectiva lingüística computacional, el PNL aborda la morfología (estructura de las palabras), la sintaxis (estructura de las oraciones), la semántica (significado) y la pragmática (uso del lenguaje en contexto). Para lograr esto, el PNL se apoya en modelos estadísticos y de aprendizaje automático. Además, el náhuatl es una lengua de bajos recursos, lo que significa que hay una escasez de corpus de texto, diccionarios y herramientas computacionales en comparación con idiomas como el español. Las variantes dialectales del náhuatl añaden otra capa de complejidad. Para superar estos obstáculos, estrategias como la Traducción Automática Neuronal (NMT), el transfer learning y el fine-tuning son cruciales, permitiendo adaptar modelos pre-entrenados en lenguas con más recursos. Los sistemas conversacionales, como los chatbots, buscan simular interacciones humanas. Su base teórica se asienta en la ciencia de la comunicación y la lingüística computacional, buscando replicar un diálogo efectivo. La Interacción Humano-Máquina (HCI) es crucial para asegurar que estas interacciones sean intuitivas y naturales.

Estado del arte

Se muestra un progreso notable, aunque con limitaciones inherentes a las lenguas de bajos recursos:

- Avances en Traducción Automática Neuronal (NMT) con limitación de datos: La noticia de Toolify AI destaca que ChatGPT, un modelo de lenguaje de última generación es capaz de hablar náhuatl. Esto es un indicio del poder de los modelos de Transformer y el transfer learning. Aunque ChatGPT no fue diseñado específicamente para la traducción bidireccional náhuatl-español, su capacidad para generar y entender náhuatl sugiere que los modelos grandes de lenguaje (LLMs) pueden ser adaptados para estas lenguas. Sin embargo, Toolify menciona que los resultados "son variados y a menudo requieren correcciones", lo que subraya la necesidad de datos más específicos y de mayor calidad para un rendimiento óptimo.
- Herramientas de Traducción en Línea con Enfoques Variados: Mr-dialect.com ofrece un "Traductor de dialectos mexicanos", incluyendo el náhuatl. Esto indica que existen herramientas accesibles para el público general, lo que es un paso adelante en la democratización del acceso a estas traducciones. Aunque no se detalla la tecnología subyacente, es probable que combine enfoques basados en reglas, diccionarios y, posiblemente, modelos estadísticos o neuronales con conjuntos de datos limitados.
- Investigación y Desarrollo Académico Específico: La UNAM ha desarrollado un "sistema traductor de lengua náhuatl". Esto es crucial porque la investigación académica suele abordar los desafíos lingüísticos de manera más profunda y metódica. Aunque la noticia no especifica si es traducción automática estadística (SMT) o neuronal (NMT), dentro de la nota se menciona también que este sistema de traducción tiene una base de datos en la cual se basa para cuando se busca una palabra, en vez de un resultado como tal de traducción, el traductor mostrara referencias de textos y fragmentos de aquellos donde el sistema allá encontrado coincidencias de la palabra buscada.

Investigación sobre PNL

Procesamiento del lenguaje natural (NLP)

El procesamiento de lenguaje natural o PLN es un subcampo de la informática y la inteligencia artificial que emplea el machine learning para permitir que las computadoras comprendan y se comuniquen con el lenguaje humano.

La investigación de PLN ha ayudado a habilitar la era de la IA generativa, desde las habilidades de comunicación de los modelos de lenguaje de gran tamaño (LLM) hasta la capacidad de los modelos de generación de imágenes para comprender las solicitudes. El PLN ya forma parte de la vida cotidiana de muchos, accionando los motores de búsqueda, impulsando los chatbots de atención al cliente con instrucciones habladas, los sistemas GPS operados por voz y los asistentes digitales de preguntas frecuentes en teléfonos inteligentes como Alexa de Amazon, Siri de Apple y Cortana de Microsoft.



Clasificación de textos.

¿Qué es?

La clasificación de texto es una tarea fundamental en el procesamiento del lenguaje natural (PLN) que implica categorizar documentos de texto en clases o categorías predefinidas según su contenido. Los conjuntos de datos para la clasificación de texto sirven como base para el entrenamiento, la validación y la prueba de modelos y algoritmos de aprendizaje automático que automatizan el proceso de clasificación.

Conjunto de datos para la clasificación de textos

¿Por qué es importante la clasificación de texto?

Los conjuntos de datos de clasificación de texto desempeñan un papel crucial en el avance de la investigación y el desarrollo en PLN y campos relacionados. Permiten a investigadores, científicos de datos y profesionales:

- **Desarrollar y evaluar modelos:** los conjuntos de datos proporcionan ejemplos etiquetados de documentos de texto que pertenecen a diferentes clases, lo que permite a los investigadores entrenar y evaluar el desempeño de los modelos de clasificación de texto en datos del mundo real.
- **Rendimiento de referencia:** Los conjuntos de datos estandarizados sirven como referencia para comparar el rendimiento de diferentes algoritmos y técnicas. Facilitan comparaciones justas y ayudan a identificar enfoques de vanguardia en la clasificación de texto.
- **Aplicaciones específicas del dominio:** los conjuntos de datos adaptados a dominios o industrias específicos (por ejemplo, finanzas, atención médica, redes sociales) permiten el desarrollo de modelos de clasificación de texto optimizados para tareas y aplicaciones especializadas.

Existen diferentes listas de conjuntos de datos para clasificación de texto, cada una de estas está enfocada a diversos usos, a continuación, mostraremos algunas de las cuales son más conocidas y los usos que se les da a las diversas listas de conjuntos.

- **Reseñas de películas de IMDB:** El conjunto de datos de reseñas de películas de IMDB contiene reseñas de películas del sitio web de IMDB, clasificadas según su opinión positiva o negativa. Se utiliza habitualmente para análisis de opiniones y tareas de clasificación de texto binario. El conjunto de datos proporciona una amplia colección de muestras de texto con sus correspondientes etiquetas de opinión, lo que lo hace ideal para el entrenamiento y la evaluación de modelos de análisis de opiniones.
- **Noticias AG:** El conjunto de datos AG News consta de artículos periodísticos categorizados en cuatro categorías: Mundial, Deportes, Negocios y Ciencia/Tecnología. Se utiliza habitualmente para tareas de clasificación de temas y categorización de textos. El conjunto de datos proporciona una colección diversa de artículos periodísticos de diferentes dominios, lo que permite a los investigadores entrenar modelos para la clasificación de temas y la categorización de noticias.

- **20 grupos de noticias:** El conjunto de datos "20 Grupos de Noticias" contiene publicaciones de 20 grupos de noticias diferentes que abarcan diversos temas, como política, religión, deportes y tecnología. Se utiliza habitualmente para la categorización de temas, la clasificación de textos y la investigación de agrupamiento de documentos. Este conjunto de datos proporciona un punto de referencia para evaluar algoritmos y técnicas de clasificación de textos y modelado de temas.
- **Reuters-21578:** El conjunto de datos Reuters-21578 consta de artículos de noticias de la agencia de noticias Reuters, etiquetados por temas y categorías. Se utiliza ampliamente para la categorización de documentos, la clasificación de textos y la recuperación de información. Abarca una amplia gama de temas y proporciona un punto de referencia estándar para la evaluación de algoritmos y técnicas de clasificación de textos.
- **Conjuntos de datos de detección de correo electrónico no deseado:** Los conjuntos de datos de detección de correo no deseado contienen mensajes etiquetados como spam o no spam (ham). Estos conjuntos de datos se utilizan para tareas de filtrado de correo electrónico, detección de spam y clasificación de texto. Suelen incluir características extraídas del contenido y los metadatos del correo electrónico, como la información del remitente, el asunto y el cuerpo del mensaje.
- **Conjuntos de datos de análisis de sentimiento de Twitter:** Los conjuntos de datos de análisis de sentimiento de Twitter consisten en tuits etiquetados con etiquetas de sentimiento, como positivo, negativo o neutral. Estos conjuntos de datos se utilizan para análisis de sentimiento, minería de opiniones y análisis de redes sociales. Ofrecen una visión general de la opinión pública y el sentimiento expresado en Twitter.
- **Reseñas de Yelp:** El conjunto de datos de reseñas de Yelp contiene reseñas y valoraciones de usuarios de la plataforma Yelp, que abarcan diversos negocios y establecimientos. Se utiliza habitualmente para análisis de sentimientos, minería de opiniones e investigación de sistemas de recomendación. El conjunto de datos incluye reseñas de texto con sus correspondientes valoraciones, lo que lo hace ideal para tareas de clasificación de texto.
- **Reseñas de Amazon:** El conjunto de datos de reseñas de Amazon contiene reseñas y valoraciones de usuarios sobre los productos vendidos en la plataforma de Amazon. Se utiliza para análisis de sentimientos, recomendaciones de productos y tareas de clasificación de texto. El conjunto de datos proporciona una amplia colección de reseñas de texto de diferentes categorías de productos, lo que permite a los investigadores entrenar modelos para diversas tareas de análisis de texto.
- **Preguntas de Stack Overflow:** El conjunto de datos de preguntas de Stack Overflow contiene preguntas publicadas en la plataforma Stack Overflow, un popular sitio web de preguntas y respuestas sobre programación. Se utiliza para la clasificación de texto, el modelado de temas y la categorización de preguntas. El conjunto de datos ofrece una colección diversa de preguntas sobre lenguajes de programación, frameworks y tecnologías.
- **Conjunto de datos de clasificación de noticias de la BBC:** El conjunto de datos de clasificación de noticias de la BBC consta de artículos de noticias del sitio web de la BBC, clasificados en categorías como negocios, entretenimiento, política, deportes y tecnología. Se utiliza habitualmente para tareas de clasificación de texto y categorización de noticias. Este

conjunto de datos proporciona un punto de referencia para evaluar modelos de clasificación de texto en el ámbito informativo.

Pero estos conjuntos de listas no son los únicos que existen, ya que existen muchos más conjuntos de listas, cada uno con un respetivo uso al que nos estemos enfocando.

🔔 [Share your dataset with the ML community!](#)

168 dataset results for **Text Classification** ✕

Search for datasets 🔍

Best match ▾

Filter by Modality

- Texts 124
- Graphs 11
- Images 7
- Tabular 3
- Audio 1

Filter by Task (clear)

- Text Classification** ✕
- Question Answering 413
- Semantic Segmentation 344
- Object Detection 331
- Image Classification 285

Filter by Language

MNIST

The MNIST database (Modified National Institute of Standards and Technology database) is a large collection of handwritten digits. It has a training set of 60,000 examples, and a test set of...

7,605 PAPERS • 52 BENCHMARKS

GLUE (General Language Understanding Evaluation benchmark)

General Language Understanding Evaluation (GLUE) benchmark is a collection of nine natural language understanding tasks, including single-sentence tasks CoLA and SST-2, similarity and...

3,174 PAPERS • 29 BENCHMARKS

SST (Stanford Sentiment Treebank)

The Stanford Sentiment Treebank is a corpus with fully labeled parse trees that allows for a complete analysis of the compositional effects of sentiment in language. The corpus is based...

2,334 PAPERS • 11 BENCHMARKS

SST-2

The Stanford Sentiment Treebank is a corpus with fully labeled parse trees that allows for a complete analysis of the compositional effects of sentiment in language. The corpus is based...

1,790 PAPERS • 5 BENCHMARKS

IMDb Movie Reviews

The IMDb Movie Reviews dataset is a binary sentiment analysis dataset consisting of 50,000 reviews from the Internet Movie Database (IMDb) labeled as positive or negative. The dataset...

1,775 PAPERS • 11 BENCHMARKS

AG News (AG's News Corpus)

AG News (AG's News Corpus) is a subdataset of AG's corpus of news articles constructed by assembling titles and description fields of articles from the 4 largest classes ("World", "Sports", "...)

956 PAPERS • 10 BENCHMARKS

Dentro de la pagina <https://paperswithcode.com/datasets?task=text-classification&page=1>

Podemos encontrar muchos conjuntos que nos pueden ayudar dependiendo del enfoque o la tarea en la que nos estemos enfocando.

Clasificación de documentos de texto mediante Naive Bayes

¿Qué es Naive Bayes?

Naive Bayes es un algoritmo de clasificación que utiliza la probabilidad para predecir la categoría a la que pertenece un punto de datos, asumiendo que todas las características no están relacionadas.

Indagando un poco más, este algoritmo se basa principalmente en lo que es el teorema de Bayes la cual es una fórmula matemática que ayuda a determinar la probabilidad condicional de un evento basándose en conocimiento previo y nueva evidencia.



Ajusta las probabilidades cuando llega nueva información y ayuda a tomar mejores decisiones en situaciones inciertas.

El Teorema de Bayes permite averiguar la probabilidad de que ocurra una serie de eventos dado que han sucedido otros anteriormente. Y, además, se puede lograr un mayor grado de precisión en el cálculo de probabilidades a medida que introducimos nueva información. ¿Qué tiene esto que ver con el aprendizaje automático?; Algunas de las aplicaciones de Machine Learning se basan en modelos de Naive Bayes, un tipo de algoritmo de clasificación basado en la técnica del Teorema de Bayes.

Gracias a los algoritmos basados en el Teorema de Bayes, se pueden llevar a cabo predicciones y análisis de probabilidades precisos que ayuden en la toma de decisiones o la automatización de procesos.

El Data Science permite que se puedan procesar miles y miles de datos, haciendo los cálculos de probabilidades en menos tiempo y de una forma precisa, combinando una gran cantidad de variables combinadas.

Ventajas del clasificador Naive Bayes

- Fácil de implementar y computacionalmente eficiente.
- Eficaz en casos con gran número de funciones.
- Funciona bien incluso con datos de entrenamiento limitados.
- Funciona bien en presencia de características categóricas.

- Para las características numéricas se supone que los datos provienen de distribuciones normales.

Desventajas del clasificador Bayes ingenuo

- Supone que las características son independientes, lo que no siempre es así en los datos del mundo real.
- Puede verse influenciado por atributos irrelevantes.
- Puede asignar probabilidad cero a eventos no vistos, lo que conduce a una generalización deficiente.

Aplicaciones del clasificador Naive Bayes

- Filtrado de correo electrónico no deseado: clasifica los correos electrónicos como spam o no spam según sus características.
- Clasificación de texto: se utiliza en el análisis de sentimientos, la categorización de documentos y la clasificación de temas.
- Diagnóstico médico: ayuda a predecir la probabilidad de una enfermedad según los síntomas.
- Puntuación crediticia: evalúa la solvencia de las personas para la aprobación de un préstamo.
- Predicción del tiempo: clasifica las condiciones climáticas en función de varios factores.

En el procesamiento del lenguaje natural y el aprendizaje automático, Naive Bayes es un método popular para clasificar documentos de texto. Permite clasificar documentos en tipos predefinidos según la probabilidad de que aparezca una palabra mediante el teorema de Bayes.

Multinomial Naive Bayes es una variante del Naive Bayes; Este algoritmo de clasificación, basado en el teorema de Bayes, es ideal para datos discretos y se utiliza habitualmente en problemas de clasificación de texto. Modela la frecuencia de palabras como recuentos y asume que cada característica o palabra tiene una distribución multinomial. El MNB se utiliza ampliamente para tareas como la clasificación de documentos según la frecuencia de palabras, como en la detección de correo no deseado.

Un ejemplo muy claro es el de mensajes de SPAM del correo electrónico donde el algoritmo busca ciertas palabras dentro de los mensajes y determinar si es o no es SPAM.

Ejemplo Naive Bayes

Ahora veremos cómo es que funciona, pero con un ejemplo más práctico en el lenguaje de Python extraído de la página de <https://www.geeksforgeeks.org/classification-of-text-documents-using-the-approach-of-naive-bayes/>.

Para lo cual utilizaremos las siguientes librerías: **pandas**, **numpy**, **scikit-learn**.

```
numpy > Ejemplo.py > ...
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import seaborn as sns
5  from sklearn.model_selection import train_test_split
6  from sklearn.feature_extraction.text import CountVectorizer
7  from sklearn.naive_bayes import MultinomialNB
8  from sklearn.metrics import accuracy_score, confusion_matrix
9  from sklearn.feature_extraction.text import CountVectorizer
10
11
12  data = pd.read_csv('numpys\synthetic_text_data.csv')
13  X = data['text']
14  y = data['label']
15
16  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
17  vectorizer = CountVectorizer()
18  X_train_vectorized = vectorizer.fit_transform(X_train)
19  X_test_vectorized = vectorizer.transform(X_test)
20
21  model = MultinomialNB()
22  model.fit(X_train_vectorized, y_train)
23  y_pred = model.predict(X_test_vectorized)
24  accuracy = accuracy_score(y_test, y_pred)
25  conf_matrix = confusion_matrix(y_test, y_pred)
26
27  print(f'Accuracy: {accuracy * 100}%')
28
29  class_labels = np.unique(y_test)
30  """
31  plt.figure(figsize=(8, 6))
32  sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
33  plt.title('Confusion Matrix Heatmap')
34  plt.xlabel('Predicted Label')
35  plt.ylabel('True Label')
36  plt.show()
37  """
38  user_input = ("I love artificial intelligence and machine learning")
39
40  user_input_vectorized = vectorizer.transform([user_input])
41  predicted_label = model.predict(user_input_vectorized)
42  print(f"The input text belongs to the '{predicted_label[0]}' category.")
```

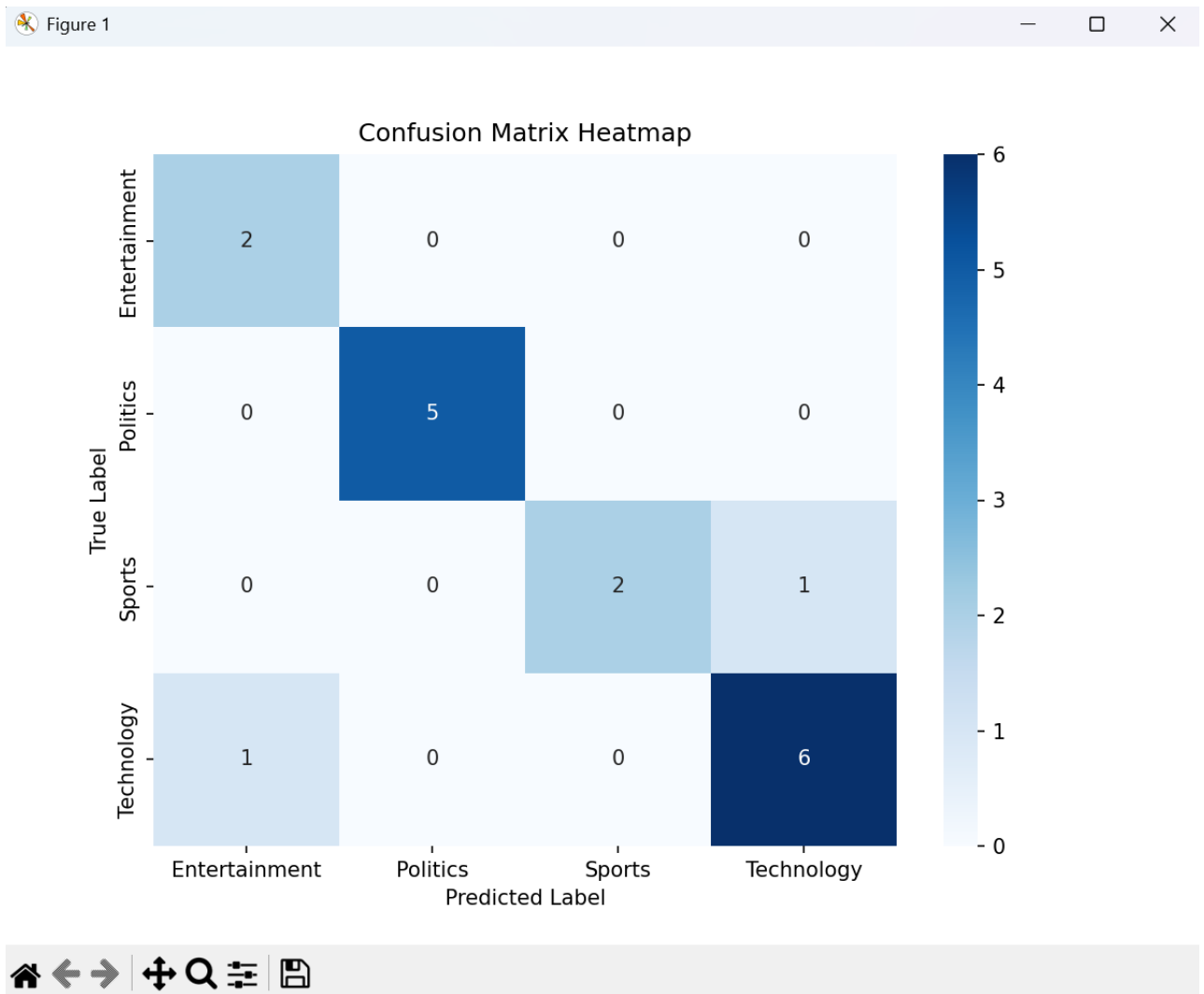
Con el siguiente código podemos realizar una pequeña prueba de cómo es que funciona Naive Bayes, primero necesitamos importar nuestras librerías que nos ayudaran a realizar este ejercicio, también necesitamos de un archivo del cual sacaremos nuestros elementos (text, label) en este caso es una hoja de cálculo (un Excel), los datos que tenemos dentro son los siguientes.

text	label
Artificial intelligence is advancing in healthcare with numerous applications emerging.	Technology
Football fans are excited about the upcoming World Cup, with teams preparing intensively.	Sports
New policies regarding climate change have sparked a heated debate in the political arena.	Politics
The latest blockbuster movie has shattered box office records and is trending worldwide.	Entertainment
Quantum computing promises to revolutionize industries, from finance to pharmaceuticals.	Technology
Political debates are becoming more heated as elections approach in several countries.	Politics
Tech companies are investing heavily in artificial intelligence to improve their products.	Technology
The summer Olympics are fast approaching, and athletes are training harder than ever.	Sports
Music festivals are back this year with global headliners performing in major cities.	Entertainment
The rise of self-driving cars is changing the transportation landscape globally.	Technology
Cybersecurity threats are increasing as hackers become more sophisticated.	Technology
Political leaders are scrambling to address the growing issue of income inequality.	Politics
Video game developers are excited about the release of new gaming consoles this fall.	Entertainment
AI-powered tools are making significant strides in areas like healthcare and manufacturing.	Technology
The latest FIFA tournament was an unforgettable event for soccer fans around the world.	Sports
Discussions about universal healthcare have intensified across many nations.	Politics
New research in AI and robotics shows promising advances in automation.	Technology
The entertainment industry is grappling with the rise of streaming services.	Entertainment
Baseball teams are preparing for the playoffs with high hopes of reaching the finals.	Sports
Politicians are debating whether to raise taxes on the wealthy to fund social programs.	Politics
Electric vehicles are becoming increasingly popular as consumers look for more sustainable options.	Technology
Cinema-goers are eagerly awaiting the release of the latest action-packed movie.	Entertainment
The debate on free speech continues to be a hot topic in political circles.	Politics
Gaming companies are exploring new ways to enhance virtual reality experiences.	Technology
The national football team just announced its squad for the upcoming tournament.	Sports
Political candidates are making promises about job creation and economic growth.	Politics
Cutting-edge technologies are reshaping the landscape of medical research.	Technology
Social media platforms are transforming the way people consume news and entertainment.	Entertainment
Political campaigns are turning to digital platforms to reach voters during the pandemic.	Politics
International conferences on technology are being held virtually due to travel restrictions.	Technology
The basketball season is well underway, with teams competing for playoff spots.	Sports
Artificial intelligence could soon have a huge impact on the job market.	Technology
The latest political scandal has shaken the political landscape across the country.	Politics

Podemos ver que tenemos 2 columnas las cuales están formadas por nuestros:

- **Text:** pequeñas oraciones que tienen que ver con un tema en específico.
- **Label:** etiquetas las cuales hacen referencia al tema con el cual tienen que ver.

Lo siguiente es que el programa va a dividir en 2 secciones las cuales son la práctica (80%) y el test (20%), las cuales entraran dentro de una matriz de recuentos de palabras para que posteriormente el sistema pueda identificarlas y aprenderlas; después se realiza un test para ver el éxito que este tiene y lo mostrara dentro de un gráfico.



Con un índice de precisión del: **88.23529411764706%**

Una vez realizado la prueba podemos introducir una palabra y podemos ver el resultado que arroja (la categoría en el que entraría dicha oración).

```
user_input = ("I love artificial intelligence and machine learning")
```

Mostrándonos el resultado

The input text belongs to the 'Technology' category.

Que el texto pertenece a la categoría de "Tecnología".

Como podemos ver este programa de Python nos enseña un poco más como es que podemos utilizar el algoritmo de Naive Bayes, al igual que vemos como es que este algoritmo aprende las palabras para poder procesar y decidir en qué rama queda, aunque no es 100% efectivo como podemos observar.

Clasificación de texto mediante regresión logística

La clasificación de texto es una tarea fundamental en el Procesamiento del Lenguaje Natural (PLN) que implica la asignación de categorías o etiquetas predefinidas a datos textuales. Tiene una amplia gama de aplicaciones, como la detección de spam, el análisis de sentimientos, la categorización de temas y la identificación de idiomas.

La regresión logística es un método estadístico utilizado para problemas de clasificación binaria y también puede extenderse para la clasificación multiclase. Al aplicarla a la clasificación de texto, el objetivo es predecir la categoría o clase de un documento de texto determinado en función de sus características.

Dentro de los puntos 2.3.1 y 2.3.2 explicaremos más lo que son la regresión logística y la clasificación binaria para tener un mejor entendimiento de cada una.

Regresión logística

La regresión logística es un algoritmo de aprendizaje automático supervisado que se utiliza para problemas de clasificación. A diferencia de la regresión lineal, que predice valores continuos, predice la probabilidad de que una entrada pertenezca a una clase específica.

Tipos de regresión logística

La regresión logística se puede clasificar en tres tipos principales según la naturaleza de la variable dependiente:

1. **Regresión logística binomial:** Este tipo se utiliza cuando la variable dependiente solo tiene dos categorías posibles. Algunos ejemplos son Sí/No, Aprobado/Reprobado o 0/1. Es la forma más común de regresión logística y se utiliza para problemas de clasificación binaria.
2. **Regresión logística multinomial:** Se utiliza cuando la variable dependiente tiene tres o más categorías posibles sin orden. Por ejemplo, clasificar animales en categorías como "gato", "perro" u "oveja". Extiende la regresión logística binaria para manejar múltiples clases.
3. **Regresión logística ordinal:** Este tipo se aplica cuando la variable dependiente tiene tres o más categorías con un orden o clasificación natural. Algunos ejemplos incluyen clasificaciones como "baja", "media" y "alta". El orden de las categorías se considera al modelar.

Supuestos de la regresión logística

Comprender los supuestos detrás de la regresión logística es importante para garantizar que el modelo se aplique correctamente. Los supuestos principales son:

- **Observaciones independientes:** se supone que cada punto de datos es independiente de los demás, lo que significa que no debería haber correlación ni dependencia entre las muestras de entrada.

- **Variables dependientes binarias:** Se asume que la variable dependiente debe ser binaria, lo que significa que solo puede tomar dos valores. Para más de dos categorías, se utilizan funciones SoftMax.
- **Relación de linealidad entre las variables independientes y las probabilidades logarítmicas:** el modelo supone una relación lineal entre las variables independientes y las probabilidades logarítmicas de la variable dependiente, lo que significa que los predictores afectan las probabilidades logarítmicas de forma lineal.
- **Sin valores atípicos:** el conjunto de datos no debe contener valores atípicos extremos, ya que pueden distorsionar la estimación de los coeficientes de regresión logística.
- **Tamaño de muestra grande:** se requiere un tamaño de muestra suficientemente grande para producir resultados confiables y estables.

Clasificación binaria

La clasificación binaria enseña a una máquina a clasificar elementos en categorías. Aprende analizando ejemplos etiquetados (como correos electrónicos marcados como "spam" o "no spam"). Tras aprender, puede decidir a qué categoría pertenecen los nuevos elementos, como identificar si un correo electrónico nuevo es spam o no. Por ejemplo, un modelo de clasificación podría entrenarse con un conjunto de datos de imágenes etiquetadas como perros o gatos y usarse para predecir la clasificación de imágenes nuevas y desconocidas como perros o gatos, basándose en sus características, como el color, la textura y la forma.

Tipos de clasificación

Cuando hablamos de clasificación en aprendizaje automático, nos referimos al proceso de ordenar datos en categorías según características específicas. Existen diferentes tipos de problemas de clasificación según la cantidad de categorías (o clases) con las que trabajemos y su organización. Existen dos tipos principales de clasificación en aprendizaje automático:

- **Clasificación binaria:** Este es el tipo de clasificación más simple. En la clasificación binaria, el objetivo es ordenar los datos en dos categorías distintas. Piénselo como una simple elección entre dos opciones. Imagine un sistema que clasifica los correos electrónicos como spam o no spam. Funciona analizando diferentes características del correo electrónico, como ciertas palabras clave o datos del remitente, y decide si es spam o no. Solo elige entre estas dos opciones.
- **Clasificación multiclase:** Aquí, en lugar de solo dos categorías, los datos deben clasificarse en más de dos. El modelo selecciona la que mejor se ajusta a la entrada. Imagine un sistema de reconocimiento de imágenes que clasifica imágenes de animales en categorías como gato, perro y pájaro. Básicamente, la máquina observa las características de la imagen (como la forma, el color o la textura) y elige qué animal es más probable que sea la imagen basándose en el entrenamiento que recibió.
- **Clasificación de múltiples etiquetas:** En la clasificación multietiqueta, un mismo dato puede pertenecer a varias categorías a la vez. A diferencia de la clasificación multiclase, donde cada punto de datos pertenece a una sola clase, la clasificación multietiqueta permite que los puntos

de datos pertenezcan a varias clases. Un sistema de recomendación de películas podría etiquetar una película como de acción y comedia. El sistema verifica diversas características (como la trama, los actores o las etiquetas de género) y asigna múltiples etiquetas a un solo dato, en lugar de solo una.

Modelado de clasificación en el aprendizaje automático

Ahora que comprendemos los fundamentos de la clasificación, es hora de explorar cómo podemos usar estos conceptos para construir modelos de clasificación. El modelado de clasificación se refiere al proceso de usar algoritmos de aprendizaje automático para categorizar datos en clases o etiquetas predefinidas. Estos modelos están diseñados para gestionar tareas de clasificación binaria y multiclase, según la naturaleza del problema. Veamos las características clave de los modelos de clasificación:

- **Separación de clases:** La clasificación se basa en distinguir entre clases distintas. El objetivo es desarrollar un modelo que pueda separar o categorizar los puntos de datos en clases predefinidas según sus características.
- **Límites de decisión:** El modelo dibuja límites de decisión en el espacio de características para diferenciar entre clases. Estos límites pueden ser lineales o no lineales.
- **Sensibilidad a la calidad de los datos:** Los modelos de clasificación son sensibles a la calidad y cantidad de los datos de entrenamiento. Unos datos bien etiquetados y representativos garantizan un mejor rendimiento, mientras que los datos ruidosos o sesgados pueden generar predicciones deficientes.
- **Manejo de datos desequilibrados:** Los problemas de clasificación pueden presentar dificultades cuando una clase está subrepresentada. Se utilizan técnicas especiales, como el remuestreo o la ponderación, para gestionar los desequilibrios de clase.
- **Interpretabilidad:** algunos algoritmos de clasificación, como los árboles de decisión, ofrecen una mayor interpretabilidad, lo que significa que es más fácil entender por qué un modelo hizo una predicción particular.

Ejemplo de regresión logística en Python

Ahora veremos cómo es que funciona, pero con un ejemplo más práctico en el lenguaje de Python extraído de la página de <https://www.geeksforgeeks.org/text-classification-using-logistic-regression/>.

```
numpys > Ejemplo-R-Logistica.py > ...
1  import pandas as pd
2  from sklearn.feature_extraction.text import CountVectorizer
3  from sklearn.model_selection import train_test_split
4  from sklearn.linear_model import LogisticRegression
5  from sklearn.metrics import accuracy_score, confusion_matrix
6
7  #Cargamos los datos
8  data = pd.read_csv('numpys\spam.csv', encoding='latin-1')
9  #Preparamos los datos
10 data.rename(columns={'v1': 'label', 'v2': 'text'}, inplace=True)
11 data['label'] = data['label'].map({'ham': 0, 'spam': 1})
12
13 #Vectorizamos los datos
14 vectorizer = CountVectorizer()
15 X = vectorizer.fit_transform(data['text'])
16 y = data['label']
17
18 #Dividimos los datos en entrenamiento y prueba
19 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
20
21 #Entrenamos el modelo
22 model = LogisticRegression(random_state=42)
23 model.fit(X_train, y_train)
24
25 #Evaluacion del modelo para tener su precision
26 y_pred = model.predict(X_test)
27 print("Accuracy_score", accuracy_score(y_test, y_pred))
28
29 cm = confusion_matrix(y_test, y_pred)
30 print("Confusion Matrix:")
31 print(f"[[{cm[0,0]} {cm[0,1]}]")
32 print(f" [{cm[1,0]} {cm[1,1]}]")
33
34 #Aplicamos el modelo
35 def classify_message(model, vectorizer, message):
36     message_vect = vectorizer.transform([message])
37     prediction = model.predict(message_vect)
38     return "spam" if prediction[0] == 0 else "ham"
39
40 message = "You've won a free ticket to Cancun!"
41 print(classify_message(model, vectorizer, message))
```

Algunos de los datos que ocupamos dentro del archivo de cálculo son los siguientes

v1	v2
ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
ham	Ok lar... Joking wif u oni...
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's
ham	U dun say so early hor... U c already then say...
ham	Nah I don't think he goes to usf, he lives around here though
spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, â€1.50 to rcv
ham	Even my brother is not like to speak with me. They treat me like aids patent.
ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurunugu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune
spam	WINNER!! As a valued network customer you have been selected to receivea â€900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.
spam	Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030
ham	I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today.
spam	SIX chances to win CASH! From 100 to 20,000 pounds txt> CSH11 and send to 87575. Cost 150p/day, 6days, 16+ TsandCs apply Reply HL 4 info
spam	URGENT! You have won a 1 week FREE membership in our â€100,000 Prize Jackpot! Txt the word: CLAIM to No: 81010 T&C www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18
ham	I've been searching for the right words to thank you for this breather. I promise i wont take your help for granted and will fulfil my promise. You have been wonderful and a blessing at all times.
ham	I HAVE A DATE ON SUNDAY WITH WILL!!
spam	XXXMobileMovieClub: To use your credit, click the WAP link in the next txt message or click here>> http://wap.xxxmobilemovieclub.com?n=QJKGIGHJGCBL
ham	Oh k...i'm watching here:)
ham	Eh u remember how 2 spell his name... Yes i did. He v naughty make until i v wet.
ham	Fine if thatâOs the way u feel. ThatâOs the way its gota b
spam	England v Macedonia - dont miss the goals/team news. Txt ur national team to 87077 eg ENGLAND to 87077 Try:WALES, SCOTLAND 4txt/!%1.20 POBOXox36504W45WQ 16+
ham	Is that seriously how you spell his name?
ham	I%U+m going to try for 2 months ha ha only joking
ham	So l _ pay first lar... Then when is da stock comin...
ham	Aft i finish my lunch then i go str down lor. Ard 3 smth lor. U finish ur lunch already?
ham	Fffffff. Alright no way i can meet up with you sooner?
ham	Just forced myself to eat a slice. I'm really not hungry tho. This sucks. Mark is getting worried. He knows I'm sick when I turn down pizza. Lol
ham	Lol your always so convincing.
ham	Did you catch the bus ? Are you frying an egg ? Did you make a tea? Are you eating your mom's left over dinner ? Do you feel my Love ?
ham	I'm back & we're packing the car now, I'll let you know if there's room
ham	Ahhh. Work. I vaguely remember that! What does it feel like? Lol
ham	Wait that's still not all that clear, were you not sure about me being sarcastic or that that's why x doesn't want to live with us
ham	Yeah he got in at 2 and was v apologetic. n had fallen out and she was actin like spoilt child and he got caught up in that. Till 2! But we won't go there! Not doing too badly cheers. You?
ham	K tell me anything about you.

Dentro del programa se realizan los siguientes pasos:

Representamos el texto

- Los datos de texto deben convertirse en características numéricas, conocidas como vectorización de texto. Las técnicas comunes para la vectorización de texto incluyen Bag of Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF) u otros métodos más avanzados.

Extracción de características

- Una vez que los datos se representan numéricamente, podemos utilizarlos como características para el modelo.
- Las características podrían ser los recuentos de palabras en BoW, los valores ponderados en TF-IDF o los vectores numéricos en las incrustaciones.

Evaluamos el modelo

- Para esta parte evaluamos el modelo para ver la eficacia con la que opera.

```
Accuracy_score 0.9748743718592965  
Confusion Matrix:  
[[1199 3]  
 [32 159]]
```

Modelo de regresión logística

- La regresión logística modela la relación entre las características y la probabilidad de pertenecer a una clase particular utilizando la función logística.
- La función logística (también llamada función sigmoidea) asigna cualquier número de valor real al rango $[0, 1]$, que es adecuado para representar probabilidades.
- El modelo de regresión logística calcula una suma ponderada de las características de entrada y aplica la función logística para obtener la probabilidad de pertenecer a la clase positiva.

Al realizar la prueba tenemos como resultado lo siguiente

```
data = pd.read_csv(  
El mensaje es spam
```

Esta es una manera diferente de cómo es que podemos identificar textos, también podemos observar cómo es que trata una manera diferente de aprendizaje este modelo para identificar las palabras.

RNN para clasificaciones de texto en PNL

¿Qué es una RNN (red neuronal recurrente)?

Una RNN o red neuronal recurrente es una red neuronal profunda entrenada en datos secuenciales o series temporales para crear un modelo de machine learning (ML) que pueda hacer predicciones secuenciales o conclusiones basadas en entradas secuenciales. Se podría usar una RNN para predecir los niveles diarios de inundación en función de las inundaciones diarias pasadas, las mareas y los datos meteorológicos.

Pero las RNN también se pueden utilizar para resolver problemas ordinales o temporales, como la traducción del lenguaje, **el procesamiento de lenguaje natural**, el análisis de sentimientos, el reconocimiento de voz y los subtítulos de imágenes.

Las redes neuronales recurrentes utilizan datos de entrenamiento para aprender. Se distinguen por su "memoria", ya que toman información de entradas anteriores para influir en la entrada y salida actual.

Las redes neuronales de aprendizaje profundo tradicionales funcionan bajo el supuesto de que sus entradas y salidas son independientes. Sin embargo, las redes neuronales recurrentes (RNN) son diferentes: sus resultados dependen de los elementos previos en una secuencia. Esto significa que, si bien el contexto futuro también sería útil para determinar una salida, las RNN unidireccionales no pueden usar esa información en sus predicciones.

Para entender mejor las RNN, tomemos la expresión "sentirse mal", que comúnmente usamos para referirnos a alguien que está enfermo. El significado de esta frase hecha solo se mantiene si las palabras están en ese orden específico. Por ello, las RNN son ideales para manejar este tipo de secuencias, ya que pueden considerar la posición de cada palabra para predecir la siguiente.

Cada palabra en "sentirse mal" forma parte de una secuencia donde el orden es crucial. Las RNN logran esto manteniendo un estado oculto en cada paso de tiempo, que actúa como una memoria. Este estado oculto se retroalimenta al siguiente paso, permitiendo a la RNN recordar información sobre las entradas anteriores. Así, en cada momento, la RNN procesa la entrada actual (como una palabra en una frase) junto con el estado oculto del paso anterior. Esta capacidad de "recordar" datos previos es lo que permite que las RNN influyan en la salida actual.

También dentro de este tema veremos lo que es el análisis de sentimientos y como participa dentro de este tipo de clasificación de textos.

¿Qué es el análisis de sentimientos?

El análisis de sentimientos es el proceso de clasificar un bloque de texto como positivo, negativo o neutral. El objetivo de la minería de sentimientos es analizar las opiniones de las personas para impulsar la expansión de las empresas. Se centra no solo en la polaridad (positiva, negativa y neutral), sino también en las emociones (feliz, triste, enojado, etc.). Utiliza diversos algoritmos de procesamiento del lenguaje natural, como el basado en reglas, el automático y el híbrido.

¿Por qué es importante el análisis de sentimientos?

El análisis de sentimientos es el significado contextual de las palabras que indica el sentimiento social de una marca y también ayuda a la empresa a determinar si el producto que están fabricando tendrá demanda en el mercado o no.

Según la encuesta, el 80% de los datos mundiales no están estructurados. Es necesario analizarlos y estructurarlos, ya sea en forma de correos electrónicos, textos, documentos, artículos, etc.

El análisis de sentimientos es necesario porque permite almacenar datos de forma eficiente y rentable. Resuelve problemas en tiempo real y puede ayudarle a resolver todos los escenarios en tiempo real.

Presentamos algunas razones clave por las que el análisis de sentimientos es importante para las empresas:

- **Análisis de comentarios de los clientes:** las empresas pueden analizar las reseñas, los comentarios y la retroalimentación de los clientes para comprender el sentimiento detrás de ellos, lo que ayuda a identificar áreas de mejora y abordar las inquietudes de los clientes, mejorando en última instancia la satisfacción del cliente.
- **Gestión de la Reputación de Marca:** El análisis de sentimientos permite a las empresas monitorizar la reputación de su marca en tiempo real.
- **Desarrollo e innovación de productos:** Comprender la opinión del cliente ayuda a identificar características y aspectos de sus productos o servicios que tienen buena aceptación o que necesitan mejoras. Esta información es fundamental para el desarrollo y la innovación de productos, ya que permite a las empresas adaptar sus ofertas a las preferencias de los clientes.
- **Análisis de la Competencia:** El análisis de sentimiento permite comparar la percepción de los productos o servicios de una empresa con la de la competencia.
- Efectividad de las campañas de marketing.
- Las empresas pueden evaluar el éxito de sus campañas de marketing analizando el sentimiento de las conversaciones en línea y las menciones en redes sociales.
- Un sentimiento positivo indica que la campaña está teniendo repercusión en el público objetivo, mientras que un sentimiento negativo puede indicar la necesidad de ajustes.

Ahora veremos un ejemplo de cómo es que se usa esto dentro de un programa en Python.

Ejemplo de RNN para clasificaciones de texto en PNL.

Ahora veremos cómo es que funciona, pero con un ejemplo más práctico en el lenguaje de Python extraído de la página de <https://www.geeksforgeeks.org/rnn-for-text-classifications-in-nlp/>. Se realizaron algunas conexiones para el correcto funcionamiento dentro del equipo de cómputo.

```
C:\Users\eliman\OneDrive\Escritorio\Ejercicios NumPy\numpys\Ejemplo-RNN.py
1  import tensorflow as tf
2  import tensorflow_datasets as tfds
3
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  # Obtain the imdb review dataset from tensorflow datasets
8  dataset = tfds.load('imdb_reviews', as_supervised=True)
9
10 # Separate test and train datasets
11 train_dataset, test_dataset = dataset['train'], dataset['test']
12
13 # Split the test and train data into batches of 32
14 # and shuffling the training set
15 batch_size = 32
16 train_dataset = train_dataset.shuffle(10000)
17 train_dataset = train_dataset.batch(batch_size)
18 test_dataset = test_dataset.batch(batch_size)
19
20 example, label = next(iter(train_dataset))
21 """
22 print('Text:\n', example.numpy()[0])
23 print('\nLabel: ', label.numpy()[0])
24
25 """
26
27 # Using the TextVectorization layer to normalize, split, and map strings
28 # to integers.
29 encoder = tf.keras.layers.TextVectorization(max_tokens=10000)
30 encoder.adapt(train_dataset.map(lambda text, _: text))
31
32 # Extracting the vocabulary from the TextVectorization layer.
33 vocabulary = np.array(encoder.get_vocabulary())
34
35 # Encoding a test example and decoding it back.
36 original_text = example.numpy()[0]
37 encoded_text = encoder(original_text).numpy()
38 decoded_text = ' '.join(vocabulary[encoded_text])
39
40 """
41 print('original: ', original_text)
42 print('encoded: ', encoded_text)
43 print('decoded: ', decoded_text)
44
45 """
46
47 # Creating the model
```

```

48  ∨ model = tf.keras.Sequential([
49      encoder,
50  ∨  tf.keras.layers.Embedding(
51      |    len(encoder.get_vocabulary()), 64, mask_zero=True),
52  ∨  tf.keras.layers.Bidirectional(
53      |    tf.keras.layers.LSTM(64, return_sequences=True)),
54      tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
55      tf.keras.layers.Dense(64, activation='relu'),
56      tf.keras.layers.Dense(1)
57  ])
58
59  # Call the model once with dummy data to build it and determine shapes
60  # This resolves the TypeError when calling model.summary()
61  model(example)
62
63  # Summary of the model
64  model.summary()
65
66  # Compile the model
67  ∨ model.compile(
68      |    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
69      |    optimizer=tf.keras.optimizers.Adam(),
70      |    metrics=['accuracy']
71  )
72
73  # Training the model and validating it on test set
74  ∨ history = model.fit(
75      |    train_dataset,
76      |    epochs=5,
77      |    validation_data=test_dataset,
78  )
79
80  # Plotting the accuracy and loss over time
81
82  # Training history
83  history_dict = history.history
84
85  # Seperating validation and training accuracy
86  acc = history_dict['accuracy']
87  val_acc = history_dict['val_accuracy']
88
89  # Seperating validation and training loss
90  loss = history_dict['loss']
91  val_loss = history_dict['val_loss']

```

```

# Seperating validation and training accuracy
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']

# Seperating validation and training loss
loss = history_dict['loss']
val_loss = history_dict['val_loss']
"""

# Plotting
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(acc)
plt.plot(val_acc)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Accuracy', 'Validation Accuracy'])

plt.subplot(1, 2, 2)
plt.plot(loss)
plt.plot(val_loss)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Loss', 'Validation Loss'])

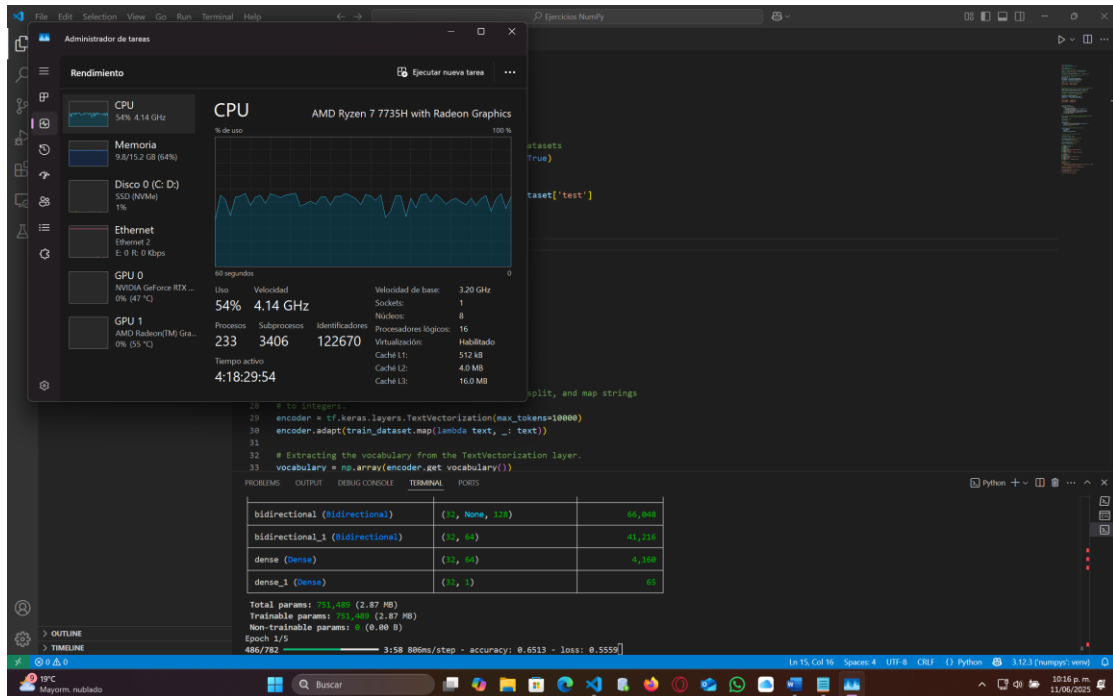
plt.show()
"""

# Making predictions
sample_text = (
    '''The anime solo leveling was so good and the animation are so dop
    I would recommend my friends to watch it.'''
)
predictions = model.predict(tf.constant([sample_text]))
print(*predictions[0])

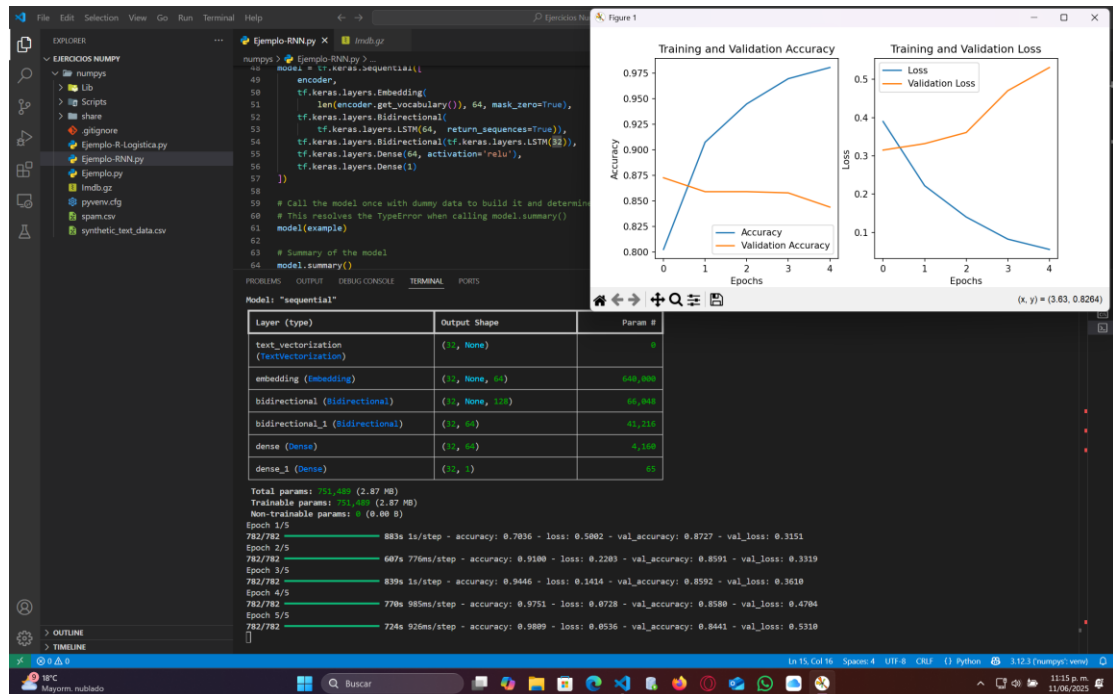
# Print the label based on the prediction
if predictions[0] > 0:
    print('La reseña es positiva ')
else:
    print('La reseña es negativa ') # Corregido el espacio

```

Aun así, este programa por tratarse de redes neurales consume bastantes recursos y tiempo por lo que se recomienda tener tiempo, de igual manera se puede ejecutar dentro del entorno de **Google colabs**, pero de igual manera se recomienda tener bastante tiempo ya que si tarda la ejecución.



Aproximadamente después de una hora tenemos nuestras graficas en las cuales tenemos las muestras del entrenamiento del modelo y los resultados que tuvo.



Como podemos ver en el primer grafico tenemos la vista de la validación y la exactitud y vemos como es que esta va subiendo, dentro del segundo grafico podemos ver cómo es que trabaja la validación de la perdida y como es que la perdida de iniciar en casi 4, se acerca lo más posible a 0, lo cual muestra cómo es que se fue entrenando el modelo, dentro de la terminal podemos observar la exactitud y la perdida.

También realizamos algunas correcciones al código donde se ingresa el texto y ya una vez corregido y compilado tenemos lo siguiente:

```
113 # Making predictions
114 sample_text = (
115     '''The anime solo leveling was so good and the animation are so dope.
116     I would recommend my friends to watch it.'''
117 )
118 predictions = model.predict(tf.constant([sample_text]))
119 print(*predictions[0])
120
121 # Print the label based on the prediction
122 if predictions[0] > 0:
123     print('La reseña es positiva ')
124 else:
125     print('La reseña es negativa ') # Corregido el espacio
126
```

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
			dense_1 (Dense)	(32, 1) 65

```
Total params: 751,489 (2.87 MB)
Trainable params: 751,489 (2.87 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/5
782/782 648s 823ms/step - accuracy: 0.7092 - loss: 0.5241 - val_accuracy: 0.8030 - val_loss: 0.4071
Epoch 2/5
782/782 482s 616ms/step - accuracy: 0.8609 - loss: 0.3273 - val_accuracy: 0.8563 - val_loss: 0.3293
Epoch 3/5
782/782 709s 907ms/step - accuracy: 0.9035 - loss: 0.2340 - val_accuracy: 0.8284 - val_loss: 0.3431
Epoch 4/5
782/782 851s 1s/step - accuracy: 0.9225 - loss: 0.2022 - val_accuracy: 0.7928 - val_loss: 0.4425
Epoch 5/5
782/782 913s 1s/step - accuracy: 0.9354 - loss: 0.1706 - val_accuracy: 0.8716 - val_loss: 0.3526
1/1 0s 449ms/step
5.244576
La reseña es positiva
```

Como podemos ver al final de la captura tenemos que la reseña es positiva, lo que demuestra que el modelo de redes neuronales recurrentes si funciona y tiene recordemos que entre más veces se entrene, poco a poco ira avanzando su identificación lo que lo hará más preciso, no podemos decir que no tenga ningún error, pero cada vez te dará resultados más precisos.

Como podemos ver en este nuevo modelo tenemos una nueva integración de nuevas bibliotecas y que tarda más en ejecutarse ya que se entrenan estos modelos para que aprendan.

Clasificación de textos mediante CNN

¿Qué es una red neuronal convolucional (CNN)?

Una red neuronal convolucional (CNN), también conocida como ConvNet, es un tipo especializado de algoritmo de aprendizaje profundo diseñado principalmente para tareas que requieren reconocimiento de objetos, como la clasificación, la detección y la segmentación de imágenes. Las CNN se emplean en diversos casos prácticos, como vehículos autónomos, sistemas de cámaras de seguridad y otros.

Una CNN consta de una capa de entrada, una capa de salida y varias capas ocultas entre ambas.

Estas capas realizan operaciones que modifican los datos, con el propósito de comprender sus características particulares. Las 3 capas más comunes son: convolución, activación o ReLU, y agrupación.

- **Convolución:** Aplica un conjunto de filtros convolucionales a las imágenes de entrada; cada filtro activa diferentes características de las imágenes.
- **Unidad lineal rectificadora (ReLU):** Mantiene los valores positivos y establece los valores negativos en cero, lo que permite un entrenamiento más rápido y eficaz. También se lo conoce como activación, ya que solo las características activadas prosiguen a la siguiente capa.
- **Agrupación:** Simplifica la salida mediante reducción no lineal de la tasa de muestreo, lo que disminuye el número de parámetros que la red debe aprender.

Estas operaciones se repiten en decenas o cientos de capas; cada capa aprende a identificar diferentes características.

Las redes neuronales convolucionales impulsan el reconocimiento de imágenes y las tareas de visión artificial. La visión artificial es un campo de la inteligencia artificial (IA) que permite a los equipos y sistemas obtener información significativa de imágenes digitales, videos y otras entradas visuales, en función de las cuales puede actuar. Esta capacidad de proporcionar recomendaciones la distingue de las tareas de reconocimiento de imágenes. Algunas aplicaciones comunes de la visión artificial se pueden apreciar hoy en día en:

- **Marketing:** las plataformas de redes sociales ofrecen sugerencias sobre quién podría estar en una fotografía que ha sido publicada en un perfil, lo que facilita etiquetar a amigos en álbumes de fotos.
- **Atención médica:** la visión artificial se ha incorporado a la tecnología radiológica, lo que permite a los médicos identificar mejor los tumores cancerosos en una anatomía saludable.
- **Venta minorista:** la búsqueda visual se ha incorporado a algunas plataformas de comercio electrónico, permitiendo a las marcas recomendar artículos que complementarían un guardarropa existente.
- **Automotriz:** si bien la era de los automóviles sin conductor aún no ha surgido, la tecnología subyacente ha comenzado a abrirse camino, mejorando la seguridad del conductor y los pasajeros a través de características como la detección de línea de carril.

Ejemplo de clasificación de textos mediante CNN

Ahora veremos cómo es que funciona, pero con un ejemplo más práctico en el lenguaje de Python extraído de la página de <https://www.geeksforgeeks.org/nlp/text-classification-using-cnn/>, realizamos comentarios al español dentro de este para que tengamos un mejor entendimiento del programa, nuevamente como podemos ver en este ejemplo hacemos uso de lo que es la base de datos de reseñas de películas de IMDB, nosotros hacemos uso mediante `keras.datasets`, en caso de que no queramos hacer uso de esa librería también podemos descargar el archivo desde su página principal <https://ai.stanford.edu/~amaas/data/sentiment/> y hacer uso de los datos, aunque estos datos ya vienen en otro formato para el uso directo.

```
1  # Importamos las herramientas necesarias para crear nuestro programa
2  import numpy as np
3  from keras.models import Sequential
4  from keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense
5  from keras.preprocessing.sequence import pad_sequences
6  from keras.datasets import imdb
7  from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
8
9  # Definimos los valores que controlará nuestro programa
10 maximum_features = 5000 # Cantidad máxima de palabras diferentes que consideraremos
11 maximum_length = 100 # Largo máximo que tendrán las reseñas (en número de palabras)
12 word_embedding_dims = 50 # Tamaño para representar cada palabra como números
13 no_of_filters = 250 # Cantidad de filtros que buscarán patrones en el texto
14 kernel_size = 3 # Tamaño de cada filtro (cuántas palabras analiza a la vez)
15 hidden_dims = 250 # Cantidad de neuronas en la capa intermedia de la red
16 batch_size = 32 # Cuántas reseñas procesamos juntas en cada paso
17 epochs = 2 # Cuántas veces repasaremos todos los datos para entrenar
18 threshold = 0.5 # Punto de corte para decidir si una reseña es positiva (0.5 = 50%)
19
20 # Cargamos la base de datos de reseñas de películas de IMDB
21 # Esta base ya viene preparada con reseñas clasificadas como positivas o negativas
22 (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=maximum_features)
23
24 # Hacemos que todas las reseñas tengan el mismo largo
25 # Si son muy cortas, las rellenamos; si son muy largas, las cortamos
26 x_train = pad_sequences(x_train, maxlen=maximum_length)
27 x_test = pad_sequences(x_test, maxlen=maximum_length)
28
29 # Construimos nuestro modelo de inteligencia artificial paso a paso
30 model = Sequential()
31
32 # Primera capa: convierte las palabras en números que la computadora entiende mejor
33 model.add(Embedding(maximum_features, word_embedding_dims,
34                     input_length=maximum_length))
35
36 # Segunda capa: busca patrones en grupos de palabras consecutivas
37 # Es como buscar frases características de reseñas positivas o negativas
38 model.add(Conv1D(no_of_filters, kernel_size, padding='valid',
39                 activation='relu', strides=1))
40
41 # Tercera capa: selecciona solo la información más importante encontrada
42 model.add(GlobalMaxPooling1D())
43
44 # Cuarta capa: procesa la información para tomar decisiones
45 model.add(Dense(hidden_dims, activation='relu'))
46
```

```

# Última capa: da el resultado final (¿es positiva o negativa la reseña?)
model.add(Dense(1, activation='sigmoid'))

# Configuramos cómo aprenderá nuestro modelo
# Le decimos qué tipo de errores debe evitar y cómo debe mejorar
model.compile(loss='binary_crossentropy',
              optimizer='adam', metrics=['accuracy'])

# Entrenamos el modelo con las reseñas que ya conocemos la respuesta
# El modelo aprende a reconocer patrones entre reseñas positivas y negativas
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, validation_data=(x_test, y_test))

# Pedimos al modelo que prediga si las reseñas de prueba son positivas o negativas
y_pred_prob = model.predict(x_test)

# Convertimos las probabilidades en respuestas definitivas (sí/no, positiva/negativa)
# Si la probabilidad es mayor al 50%, decimos que es positiva
y_pred = (y_pred_prob > threshold).astype(int)

# Calculamos qué tan bien funcionó nuestro modelo comparando sus respuestas
# con las respuestas correctas que ya conocíamos
accuracy = accuracy_score(y_test, y_pred) # Porcentaje de aciertos totales
precision = precision_score(y_test, y_pred) # De las que dijo positivas, cuántas eran realmente positivas
recall = recall_score(y_test, y_pred) # De todas las positivas reales, cuántas logró encontrar
f1 = f1_score(y_test, y_pred) # Promedio balanceado entre precisión y recuperación

# Mostramos los resultados de qué tan bien funcionó nuestro clasificador
print('Exactitud:', accuracy)
print('Precisión:', precision)
print('Recuperación:', recall)
print('Puntuación F1:', f1)

```

Este ejemplo al realizarlo nos da como resultado lo siguiente:

```

Epoch 1/2
782/782 ————— 6s 7ms/step - accuracy: 0.7009 - loss: 0.5274 - val_accuracy: 0.8516 - val_loss: 0.3319
Epoch 2/2
782/782 ————— 5s 7ms/step - accuracy: 0.8977 - loss: 0.2550 - val_accuracy: 0.8584 - val_loss: 0.3282
782/782 ————— 1s 1ms/step
Exactitud: 0.8584
Precisión: 0.862400906002265
Recuperación: 0.85288
Puntuación F1: 0.8576140294425227

```

Lo que son las epoch (épocas) nos sirven para decir cuantas veces queremos que los datos sean repasados para que el modelo estudie o más bien para entrenar al sistema, como vemos dentro de este ejemplo no hacemos uso de alguna oración externa, si no que toma aleatoriamente una de la base de datos y la analiza lo que genera esa exactitud y precisión.

Estos ejemplos y temas vistos dentro del tema 2, nos fueron de gran ayuda para posteriormente pasar a lo que es el tema 3, ahí veremos cómo es que se extrae la información, de igual manera veremos algunos ejemplos dentro del lenguaje de Python para la extracción de información y tener una mejor idea de cómo es que se hace.

Extracción de información

La extracción de información (EI) es el proceso de convertir texto en lenguaje natural, ya sea semiestructurado o no estructurado (como los PDF), en un formato organizado que las máquinas puedan leer y consultar. En esencia, transforma datos de texto en información estructurada. El procesamiento del lenguaje natural (PLN) a menudo se apoya en la extracción de información para identificar datos clave dentro de un texto.

Los algoritmos de extracción de información son capaces de reconocer y clasificar elementos como entidades (nombres, lugares), relaciones entre ellos, eventos e incluso sentimientos. Una vez identificados, estos datos se almacenan en una base de datos. La característica principal de esta información estructurada es su formato estandarizado, que generalmente se organiza en filas y columnas con atributos bien definidos. Esta organización estandarizada es lo que distingue fundamentalmente a los datos estructurados de los datos no estructurados.

Dentro de una misma base de datos, todos los valores de datos mantienen un formato estructurado consistente, compartiendo los mismos atributos definidos. Además, se establecen atributos relacionales para vincular diferentes bases de datos mediante atributos comunes.

¿Por qué es importante?

La extracción de datos es un paso inicial crucial en el análisis de datos. Consiste en recopilar información de distintas fuentes, como bases de datos, sitios web, documentos y servicios en la nube. Esta fase es vital para que las empresas y organizaciones puedan transformar datos brutos en información útil y procesable.

Técnicas de Extracción de Datos: Ventajas y Desventajas

Es importante evaluar las ventajas y desventajas de cada técnica de extracción de datos antes de elegir el método más adecuado para una tarea específica:

- **Web Scraping:** Es muy eficiente para obtener datos de sitios web públicos. Sin embargo, puede volverse inestable y fallar si el diseño del sitio web cambia.
- **Extracción mediante API:** Ofrece un acceso a los datos más estable y eficiente. No obstante, requiere tener acceso a las API relevantes y podría estar sujeto a ciertas limitaciones de uso.
- **Extracción directa de bases de datos:** Permite un acceso eficaz a los datos estructurados directamente desde su origen. La desventaja es que exige conocimientos técnicos en lenguajes de bases de datos como SQL (Structured Query Language).

Comprender los puntos fuertes y las limitaciones de cada método de extracción es fundamental para optimizar este proceso. Al elegir cuidadosamente la técnica más apropiada, las organizaciones pueden asegurar que la extracción de datos se realice de la manera más eficiente, precisa y confiable posible. Esta estrategia es clave para maximizar el valor de los datos tanto para el análisis como para las operaciones.

Según la naturaleza y el formato de los datos, se emplean diferentes métodos de extracción para obtener información valiosa de forma eficiente. Los principales tipos de extracción de datos incluyen la extracción de datos estructurados, no estructurados y semiestructurados.

Tipos de Extracción de Datos: Estructurados, No Estructurados y Semiestructurados

La extracción de datos es fundamental para obtener información valiosa, pero el método varía drásticamente según la organización de la fuente. Podemos clasificar la extracción en tres tipos principales: estructurada, no estructurada y semiestructurada.

- **Extracción de Datos Estructurados:** La extracción de datos estructurados se enfoca en recuperar información de fuentes altamente organizadas. Aquí, el formato y el esquema están predefinidos, como ocurre en las bases de datos, hojas de cálculo o tablas. Este proceso es muy preciso y eficiente porque la naturaleza ordenada de los datos simplifica la identificación y recopilación de elementos específicos. Las herramientas diseñadas para este fin son muy hábiles en estos entornos, permitiéndote especificar exactamente los datos que necesitas.
- **Extracción de Datos No Estructurados:** Por otro lado, la extracción de datos no estructurados aborda información que carece de un formato u organización predefinidos. Piensa en documentos de texto, correos electrónicos, vídeos o publicaciones en redes sociales. Extraer datos de estas fuentes es inherentemente más complejo y requiere técnicas avanzadas como el procesamiento del lenguaje natural (PLN) y el aprendizaje automático para interpretarlos y organizarlos. A pesar de estos desafíos, la extracción de datos no estructurados es invaluable para obtener información de una amplia gama de fuentes que, de otro modo, permanecerían sin explotar.
- **Extracción de Datos Semiestructurados:** La extracción de datos semiestructurados se encuentra en un punto intermedio entre los datos estructurados y los no estructurados. Fuentes como los archivos XML (Lenguaje de Marcado Extensible), los documentos JSON (Notación de Objetos JavaScript) y ciertas páginas web no tienen una estructura tan rígida como las bases de datos, pero sí contienen marcadores o etiquetas que proporcionan cierto nivel de organización. La extracción de estas fuentes generalmente implica analizar este formato semiestructurado para identificar y extraer la información relevante. Este tipo de extracción combina elementos de técnicas estructuradas y no estructuradas, aprovechando la organización inherente de los datos y aplicando un procesamiento más avanzado para manejar la variabilidad dentro de la estructura.

Extracción de información en PLN

La extracción mediante PLN es una tarea fundamental en PLN, diseñada para identificar y categorizar información significativa dentro de una oración o documento. Dado un texto de entrada, un modelo entrenado analiza la representación de las palabras y extrae datos estructurados con diferentes niveles de granularidad.

En esencia, la extracción de PLN mejora la forma en que se procesa la información textual, transformando datos no estructurados en representaciones estructuradas que alimentan sistemas inteligentes.

Comprender y extraer información clave del texto abre un amplio abanico de posibilidades para el análisis textual y la recuperación de información. Este proceso es esencial en múltiples ámbitos, desde la creación de grafos de conocimiento para asistentes basados en IA hasta la automatización de tareas de clasificación y predicción de documentos.

En el contexto empresarial, la extracción mediante PLN mejora la toma de decisiones al organizar documentos y revelar información relevante. Por ejemplo, las instituciones financieras la utilizan para extraer menciones de empresas, riesgos regulatorios y tendencias del mercado. En el sector sanitario, ayuda a extraer datos críticos de textos médicos, como afecciones, tratamientos y síntomas, lo que impulsa la investigación avanzada y los sistemas de apoyo a la toma de decisiones clínicas.

Más allá de las aplicaciones estructuradas, la extracción de PNL es crucial para entrenar conjuntos de datos, enriquecer modelos previamente entrenados y mejorar las arquitecturas basadas en la atención que impulsan los últimos avances en PNL.

Tipos de extracción de información

Las tareas de extracción de información se clasifican según el tipo de información que se identifica y etiqueta. Los sistemas IE pueden gestionar tareas que incluyen:

- Named entity recognition (NER)
- Extracción de relaciones
- Extracción de eventos
- Análisis de sentimiento

Named entity recognition (NER)

El reconocimiento de entidades nombradas (NER) sirve de puente entre el texto no estructurado y los datos estructurados, permitiendo a las máquinas cribar enormes cantidades de información textual y extraer pepitas de datos valiosos en formas categorizadas. Al localizar entidades concretas dentro de un mar de palabras, el NER transforma la forma en que procesamos y utilizamos los datos textuales.

Objetivo: el objetivo principal de NER es peinar el texto no estructurado e identificar trozos específicos como entidades con nombre, clasificándolos posteriormente en categorías predefinidas. Esta conversión de texto bruto en información estructurada hace que los datos sean más procesables, facilitando tareas como el análisis de datos, la recuperación de información y la construcción de grafos de conocimiento.

Cómo funciona:

- **Tokenización:** Antes de identificar las entidades, el texto se divide en tokens, que pueden ser palabras, frases o incluso oraciones. Por ejemplo, "Steve Jobs cofundó Apple" se dividiría en tokens como "Steve", "Jobs", "cofundó", "Apple".
- **Identificación de la entidad:** Utilizando diversas reglas lingüísticas o métodos estadísticos, se detectan las entidades con nombre potenciales. Esto implica reconocer patrones, como las mayúsculas en los nombres ("Steve Jobs") o formatos específicos (como fechas).
- **Clasificación de las entidades:** Una vez identificadas las entidades, se clasifican en clases predefinidas como "persona", "organización" o "ubicación". Esto se consigue a menudo

utilizando modelos de machine learning entrenados en conjuntos de datos etiquetados. Para nuestro ejemplo, "Steve Jobs" se clasificaría como "persona" y "Apple" como "organización".

- **Análisis contextual:** Los sistemas NER suelen tener en cuenta el contexto circundante para mejorar la precisión. Por ejemplo, en la frase "Apple lanzó un nuevo iPhone", el contexto ayuda al sistema a reconocer a "Apple" como una organización y no como una fruta.
- **Posprocesamiento:** Tras el reconocimiento y la clasificación iniciales, puede aplicarse un posprocesamiento para refinar los resultados. Esto podría implicar la resolución de ambigüedades, la fusión de entidades con varios tokens o el uso de bases de conocimiento para mejorar los datos de las entidades.

Navegar por el ámbito del reconocimiento de entidades nombradas (NER) presenta su propio conjunto de retos, aunque la técnica prometa conocimientos estructurados a partir de datos no estructurados. He aquí algunos de los principales obstáculos a los que se enfrenta este ámbito:

- **Ambigüedad.** Las palabras pueden ser engañosas. Un término como "Amazon" puede referirse al río Amazonas o a la empresa Amazon (en inglés), en función del contexto, lo que hace que el reconocimiento de la entidad sea una tarea complicada.
- **Dependencia del contexto.** Las palabras suelen derivar su significado del texto que las rodea. La palabra "Apple" en un artículo de tecnología probablemente se refiera a la corporación, mientras que, en una receta, probablemente sea una manzana (en inglés). Comprender estos matices es crucial para un reconocimiento preciso de las entidades.
- **Variaciones lingüísticas.** El colorido tapiz del lenguaje humano, con su jerga, dialectos y diferencias regionales, puede plantear retos. Lo que es lenguaje común en una región puede ser extraño en otra, lo que complica el proceso del NER.
- **Dispersión de datos.** Para los métodos de NER basados en machine learning, es crucial disponer de datos etiquetados completos. Sin embargo, obtener esos datos, sobre todo para las lenguas menos comunes o los dominios especializados, puede ser un reto.
- **Generalización del modelo.** Mientras que un modelo puede sobresalir en el reconocimiento de entidades en un dominio, puede flaquear en otro. Garantizar que los modelos de NER generalicen bien en varios dominios es un reto persistente.

Reconocimiento de entidades nombradas

Desde el inicio del NER, ha habido algunos avances metodológicos significativos, especialmente aquellos que dependen de técnicas basadas en el aprendizaje profundo. Las iteraciones más recientes incluyen:

- **Redes neuronales recurrentes (RNN) y memoria larga a corto plazo (LSTM).** Las RNN son un tipo de redes neuronales diseñadas para problemas de predicción de secuencias. Las LSTM, un tipo especial de RNN, pueden aprender a reconocer patrones a lo largo del tiempo y mantener la información en la "memoria" durante largas secuencias, lo que las hace particularmente útiles para comprender el contexto e identificar entidades.
- **Campos aleatorios condicionales (CRF).** Los CRF se utilizan a menudo en combinación con las LSTM para tareas de NER. Pueden modelar la probabilidad condicional de una

secuencia completa de etiquetas, en lugar de solo etiquetas individuales, lo que las hace útiles para tareas en las que la etiqueta de una palabra depende de las etiquetas de las palabras circundantes.

- **Transformadores y BERT.** Las redes de transformadores, en particular el modelo BERT (Representaciones de codificador bidireccional de transformadores), tuvieron un impacto significativo en el NER. Mediante un mecanismo de autoatención que sopesa la importancia de diferentes palabras, BERT da cuenta del contexto completo de una palabra al observar las palabras que vienen antes y después.

Usando SpaCy

SpaCy es una librería de Python que permite construir aplicaciones de procesamiento de lenguaje natural (NLP). SpaCy proporciona modelos preentrenados de diferentes lenguajes, lo cual junto a una sintaxis clara hace que sea ideal para principiantes en el campo de la NLP.

Además, permite crear modelos nuevos o reentrenar los modelos que proporciona con datos propios para entrenar los modelos en campos específicos. SpaCy está orientado al uso de producción, proporcionando un framework para construir aplicaciones completas que requieran de procesamiento de lenguaje natural, a diferencia de otras librerías como TensorFlow que permiten experimentar con diferentes arquitecturas de redes neuronales o implementar los últimos modelos desarrollados. Además, SpaCy es eficiente en CPU pese a utilizar modelos de redes neuronales. La velocidad y precisión de los modelos de SpaCy son de los mejores en el mercado.

En este documento gratuito en español, que puedes descargar directamente, tienes toda la información para conocer los puntos claves de SpaCy

Los modelos de SpaCy de procesamiento de lenguaje natural permiten analizar un texto y extraer información tanto del texto como de las predicciones del modelo sobre su significado por el contexto. Son útiles por ejemplo para el análisis de artículos, de tweets, de páginas web... SpaCy permite mucha personalización tanto en el entrenamiento de modelos como el añadir información al análisis del texto para crear aplicaciones a medida.

Instalamos SpaCy

```
(numpys) C:\Users\elman\OneDrive\Escritorio\Ejercicios Numpy>pip install spacy
```

SpaCy incluye varios modelos preentrenados para diferentes lenguajes y tareas, y vamos a descargar el modelo específico para el uso de este ejemplo.

```
(numpys) C:\Users\elman\OneDrive\Escritorio\Ejercicios Numpy>python -m spacy download en_core_web_sm
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-any.whl (12.8 MB)
    12.8/12.8 MB 2.7 MB/s eta 0:00:00
Installing collected packages: en-core-web-sm
Successfully installed en-core-web-sm-3.8.0
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

Usamos un ejemplo extraído de internet de la página de [Named Entity Recognition \(NER\) with Python – Wisecube AI – Research Intelligence Platform](#)

```
1 import spacy
2
3 nlp = spacy.load("en_core_web_sm")
4
5 # Define the named entity categories that we want to recognize
6 ner_categories = ["PERSON", "ORG", "GPE", "PRODUCT"]
7 # Tokenize the text
8 text = "Robert Downey Jr. is an American actor and producer. He is best known for his roles in films such as Iron Man, The Avengers, and Sherlock Holmes. " \
9 "Downey has won several awards for his acting, including two Screen Actors Guild Awards and a Golden Globe Award. " \
10 "He has also been nominated for an Academy Award."
11 doc = nlp(text)
12 # Identify and classify named entities
13 entities = []
14 for ent in doc.ents:
15     if ent.label_ in ner_categories:
16         entities.append((ent.text, ent.label_))
17
18 # Print the named entities and their categories
19 for entity, category in entities:
20     print(f"{entity}: {category}")
21
```

Como podemos ver usamos un pequeño texto en inglés, el cual habla sobre el actor Robert Downey Jr., y sobre algunos de sus papeles, también el modelo está preentrenado para buscar PERSON (nombres, apellidos), ORG (organizaciones, instituciones), GPE (países, estados, ciudades), PRODUCT (objetos, vehículos, comida), y al ejecutar este pequeño programa tenemos los siguientes resultados.

```
Robert Downey Jr.: PERSON
Iron Man: PERSON
Holmes: PERSON
Downey: PERSON
Screen Actors Guild Awards: ORG
an Academy Award: ORG
```

Podemos ver cómo es que clasifico e identifico las palabras del texto.

Usando NLTK

El Natural Language Toolkit (NLTK) es un conjunto de librerías y programas para Python que nos permiten llevar a cabo muchas tareas relacionadas con el Procesamiento del Lenguaje Natural. Muchas de las tareas que necesitaremos hacer ya estarán programadas de manera eficiente en NLTK y las podremos usar directamente en nuestros programas. Además de los programas, se distribuyen también corpus y otros datos lingüísticos. Es una plataforma muy útil tanto para la enseñanza como para el desarrollo y la investigación.

NLTK es una plataforma líder para desarrollar programas Python que trabajan con datos de lenguaje humano. Ofrece interfaces fáciles de usar para más de 50 corpus y recursos léxicos como WordNet, además de un conjunto de bibliotecas de procesamiento de texto para clasificación, tokenización, lematización, etiquetado, análisis sintáctico y razonamiento semántico, contenedores para bibliotecas de PLN de alto rendimiento y un foro de discusión activo.

Gracias a una guía práctica que presenta los fundamentos de la programación junto con temas de lingüística computacional, además de una completa documentación de la API, NLTK es ideal tanto para lingüistas, ingenieros, estudiantes, educadores, investigadores como para usuarios de la industria. NLTK está disponible para Windows, Mac OS X y Linux. Y lo mejor de todo, NLTK es un proyecto gratuito, de código abierto y comunitario.

Se ha dicho que NLTK es “una herramienta maravillosa para enseñar y trabajar en lingüística computacional usando Python” y “una biblioteca increíble para jugar con el lenguaje natural”.

Procesamiento del Lenguaje Natural con Python ofrece una introducción práctica a la programación para el procesamiento del lenguaje. Escrito por los creadores de NLTK, guía al lector a través de los fundamentos de la escritura de programas en Python, el trabajo con corpus, la categorización de texto, el análisis de la estructura lingüística y más.

EJEMPLO EN PYTHON

El código que utilizamos a continuación es de la página de <https://medium.com/@danielwume/getting-started-with-nltk-10-essential-examples-for-natural-language-processing-in-python-54451eae1366>, dentro de esta página podemos encontrar diversos ejemplos que nos enseñan el uso de NLTK, por el momento haremos uso del ejemplo de NER.

Instalamos primero lo que es NLTK dentro de nuestro entorno

```
(numpy) C:\Users\elman\OneDrive\Escritorio\Ejercicios Numpy>pip install nltk
Collecting nltk
  Downloading nltk-3.9.1-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: click in c:\users\elman\onedrive\escritorio\ejercicios numpy\numpys\lib\site-packages (from nltk) (8.2.1)
Requirement already satisfied: joblib in c:\users\elman\onedrive\escritorio\ejercicios numpy\numpys\lib\site-packages (from nltk) (1.5.1)
Collecting regex>=2021.8.3 (from nltk)
  Downloading regex-2024.11.6-cp312-cp312-win_amd64.whl.metadata (41 kB)
Requirement already satisfied: tqdm in c:\users\elman\onedrive\escritorio\ejercicios numpy\numpys\lib\site-packages (from nltk) (4.67.1)
Requirement already satisfied: colorama in c:\users\elman\onedrive\escritorio\ejercicios numpy\numpys\lib\site-packages (from click->nltk) (0.4.6)
Downloading nltk-3.9.1-py3-none-any.whl (1.5 MB)
   1.5/1.5 MB 864.6 kB/s eta 0:00:00
Downloading regex-2024.11.6-cp312-cp312-win_amd64.whl (273 kB)
Installing collected packages: regex, nltk
Successfully installed nltk-3.9.1 regex-2024.11.6
```

Ahora lo que necesitamos es poner el código para ver cómo es que funciona el NER con el uso de NLTK, pero le vamos a realizar diversas modificaciones para que funcione, puesto que el programa está desactualizado y en mi caso no funciona por lo que realice ciertas correcciones para que funcione correctamente.

```

1  import nltk
2  # Descargar recursos necesarios para NER
3  nltk.download('maxent_ne_chunker_tab')
4  nltk.download('words')
5  nltk.download('punkt_tab') # Formato de tokenizador
6  nltk.download('averaged_perceptron_tagger_eng') # Necesario para POS tagging
7
8  from nltk.tokenize import word_tokenize
9  from nltk import pos_tag, ne_chunk
10
11 # Texto corregido con ortografía correcta
12 texto = "Eiichiro Oda was born in Japan, and he is the author of masterpiece called One Piece."
13
14 tokens = word_tokenize(texto)
15 tagged_tokens = pos_tag(tokens)
16 ne_chunked_tokens = ne_chunk(tagged_tokens)
17
18 print("Tokens:")
19 print(tokens)
20 print("\nPOS Tags:")
21 print(tagged_tokens)
22 print("\nNamed Entities:")
23 print(ne_chunked_tokens)
24
25 # Opción para mostrar solo las entidades nombradas identificadas
26 print("\nEntidades nombradas encontradas:")
27 for chunk in ne_chunked_tokens:
28     if hasattr(chunk, 'label'):
29         entity = ' '.join([token for token, pos in chunk.leaves()])
30         print(f"{entity}: {chunk.label()}")

```

Como vemos lo que nos muestra de salida es:

Named Entities: (S (PERSON Eiichiro/NNP) (PERSON Oda/NNP) was/VBD born/VBN in/IN (GPE Japan/NNP) ,/, and/CC he/PRP is/VBZ	the/DT author/NN of/IN masterpiece/NN called/VBN One/CD Piece/NNP ./.) Entidades nombradas encontradas: Eiichiro: PERSON Oda: PERSON Japan: GPE
---	---

Como vemos los datos que toma y como los organiza para que nos lo muestra, de igual manera como podemos ver las entidades que encontró fueron, PERSON, GPE. Lo que nos demuestra cómo es que se realiza la implementación de NLTK para el NER.

Extracción de relaciones en PNL

La extracción de relaciones es una subtask de la tarea de Extracción de Información, cuyo objetivo es identificar relaciones entre entidades y asignarles algún tipo de etiqueta o clase.

Es un proceso importante en el Procesamiento del Lenguaje Natural que identifica y categoriza automáticamente las conexiones entre entidades dentro de un texto en lenguaje natural. Estas entidades pueden incluir individuos, organizaciones, ubicaciones, fechas o cualquier otro sustantivo o concepto mencionado en el texto.

Tipos de extracción de relaciones en PNL

Podemos clasificar la extracción relacional en varios tipos, que se enumeran a continuación:

Extracción de relaciones binarias

- Este proceso de extracción se centra en identificar y categorizar las relaciones entre pares de entidades, lo cual es una forma fundamental de extracción de relaciones y se utiliza a menudo cuando las relaciones son simples y pueden expresarse como pares.
- El proceso de extracción de relaciones binarias implica los siguientes pasos:
 - El reconocimiento de entidades nombradas se realiza para identificar y clasificar las entidades en la prueba. Las entidades pueden ser personas, lugares o cualquier otro sustantivo relevante en el contexto.
 - Para analizar la estructura gramatical del texto y comprender la relación entre las palabras se utiliza el análisis de dependencia.
 - El modelo determina la relación entre dos entidades identificadas examinando el contexto y la coocurrencia de las entidades en el texto.
 - Si se identifica una relación, suele clasificarse en un tipo o categoría específica. Por ejemplo, si las entidades son una persona y una organización, la relación podría clasificarse como "trabaja para".
 - El paso final extrae las relaciones binarias del texto y las almacena en un formato estructurado.

Extracción de relaciones ternarias

- La extracción de relaciones ternarias es una tarea avanzada de procesamiento del lenguaje natural (PLN) que amplía el concepto de extracción de relaciones binarias mediante la identificación y extracción de relaciones que involucren tres entidades a partir de datos de texto

no estructurados. Esta tarea es esencial para construir grafos de conocimiento más complejos y extraer información con matices del texto.

Extracción de relaciones anidadas

- Este proceso de extracción se utiliza cuando las relaciones dentro de un texto son jerárquicas o están integradas unas dentro de otras.
- El proceso es bastante similar a la extracción de relaciones ternarias y binarias.
 - El proceso comienza con el reconocimiento de la entidad del nombre.
 - El análisis de dependencia se utiliza para analizar la estructura gramatical del texto.
 - El objetivo principal de la extracción de relaciones anidadas es identificar y extraer relaciones jerárquicas o anidadas entre entidades. Esto implica determinar cómo se estructuran jerárquicamente las entidades y cómo se relacionan entre sí. Por ejemplo, podría querer extraer relaciones como "forma parte de", "contiene", "es miembro de", etc.
 - Una vez identificadas estas relaciones anidadas, suelen clasificarse en tipos o categorías específicas. Esta categorización ayuda a comprender la naturaleza de las relaciones jerárquicas entre las entidades. Por ejemplo, si se extraen relaciones entre partes de una máquina, se podrían clasificar como "subcomponente", "componente" o "ensamblaje".
 - El paso final es extraer las relaciones anidadas del texto y representarlas en un formato estructurado.

Extracción de relaciones temporales

- Esta extracción se centra en identificar relaciones con una dimensión temporal que incluye determinar cuándo ocurrió un evento o cuándo una relación fue válida.
- Las relaciones temporales son esenciales para comprender el orden cronológico, la duración y las dependencias temporales entre eventos.
- El proceso de extracción de relación temporal implica los siguientes pasos:
 - La extracción de relaciones temporales implica reconocer e identificar eventos, acciones o entidades mencionadas en el texto.
 - Paralelamente, el sistema necesita identificar y clasificar las expresiones temporales dentro del texto. Estas pueden ser fechas, horas y duraciones específicas, o frases más complejas como "hace dos semanas" o "en el futuro".

- El análisis de dependencia se utiliza a menudo para analizar la estructura gramatical del texto, lo que permite comprender cómo se conectan los diferentes elementos del texto, incluido cómo las expresiones temporales se relacionan con eventos o entidades.
- La tarea principal de la extracción de relaciones temporales implica identificar las relaciones temporales entre eventos o entidades y las expresiones temporales asociadas.
- Una vez identificadas las relaciones temporales, suelen clasificarse en tipos o clases específicas. Estas clases pueden incluir "antes", "después", "durante", "simultáneo", "inicio", "fin" y más, según los matices de las relaciones temporales.
- El paso final es extraer y representar estas relaciones temporales en un formato estructurado, como una línea de tiempo o un gráfico de conocimiento.

Extracción de relaciones causales

- Esta extracción identifica relaciones que expresan causa y efecto, lo cual es crucial en aplicaciones como identificar las causas de enfermedades o comprender las razones detrás de ciertos eventos.
- El proceso de extracción de relaciones causales consiste en determinar si existe una relación causal entre los eventos o entidades identificados.
 - Esto requiere examinar el contexto y las claves lingüísticas del texto para identificar si un evento o entidad está causando otro, o si existe una correlación entre ellos.
 - Una vez que se identifica una relación causal, se puede clasificar en tipos o categorías específicos.
 - Por ejemplo, las relaciones causales pueden clasificarse como causalidad directa, causalidad indirecta, correlación, causalidad inversa o causalidad temporal.
 - El paso final es extraer y representar las relaciones causales en un formato estructurado, como un gráfico causal o una base de conocimiento.

Extracción de relaciones entre oraciones

- Cuando las relaciones abarcan varias oraciones, se utiliza la extracción de relaciones entre oraciones para identificar y extraer relaciones que existen a través de los límites de las oraciones, lo que a menudo requiere resolución de coreferencia y análisis contextual.
- El proceso implica:
 - La extracción de relaciones entre oraciones consiste en identificar y extraer relaciones entre entidades o eventos mencionados en diferentes oraciones. Esto puede implicar reconocer cómo las acciones o entidades mencionadas en una oración se relacionan con las de otra, como la causalidad, la dependencia o la asociación.

- Una vez identificadas las relaciones, se pueden categorizar en tipos o clases específicas. La clasificación ayuda a comprender la naturaleza de las relaciones que abarcan varias oraciones, como las relaciones causales, las dependencias temporales o la coocurrencia.

Extracción de información abierta (OpenIE)

- En este proceso de extracción especial, las relaciones entre entidades se extraen en función de las dependencias gramaticales y los patrones del texto sin depender de esquemas o categorías predefinidos, lo que permite capturar una amplia gama de relaciones.
- Las características principales de la extracción de información abierta incluyen:
 - Los sistemas OpenIE están diseñados para descubrir relaciones entre las entidades mencionadas en el texto. Estas relaciones pueden ser explícitas, como "X funciona para Y", o implícitas, como "X es una instancia de Y". Los sistemas OpenIE buscan descubrir una amplia variedad de relaciones, sin limitarse a un conjunto predefinido.
 - Para extraer relaciones, los sistemas OpenIE también necesitan identificar entidades (por ejemplo, personas, organizaciones, ubicaciones) dentro del texto.
 - OpenIE no requiere conocimiento previo de las relaciones específicas que se van a extraer. Puede descubrir relaciones nuevas que no se habían previsto.
 - Los sistemas OpenIE están diseñados para escalar a grandes corpus y pueden extraer información de amplios conjuntos de datos textuales.

Ejemplo de extracción de relaciones en Python

Para el siguiente ejemplo usamos el programa dentro de la página de <https://www.geeksforgeeks.org/nlp/relationship-extraction-in-nlp/> , tuvimos que realizar diversas modificaciones al programa ya que no funcionaba.

```
1  import spacy
2  from transformers import pipeline
3
4  # Load spaCy's transformer-based model
5  nlp = spacy.load("en_core_web_sm")
6
7  # Define a sample text for relationship extraction
8  text = "GeekforGeeks was founded by Sandeep Jain. Sandeep Jain was a former teacher. Geeksforschool was founded by Sandeep Jain."
9
10 # Process the text with spaCy
11 doc = nlp(text)
12
13 # Create a list to store extracted relationships
14 relationships = []
15
16 # Use Hugging Face's transformers NER pipeline for named entity recognition
17 nlp_ner = pipeline("ner", model="dbmdz/bert-large-cased-finetuned-conll03-english")
18
19 # Extract named entities and their labels
20 named_entities = nlp_ner(text)
21
22 print("Named entities detected by Transformers:")
23 for entity in named_entities:
24     print(f"Entity: {entity['word']}, Label: {entity['entity']}, Score: {entity['score']:.4f}")
25
26 print("\nSpaCy entities:")
27 for ent in doc.ents:
28     print(f"Entity: {ent.text}, Label: {ent.label_}")
29
30 # Iterate through the sentences in the document
31 for sent in doc.sents:
32     print(f"\nProcessing sentence: {sent.text}")
33
```

```
34 # Iterate through the named entities (people, organizations etc.) in the sentence
35 for ent in sent.ents:
36     # Check if the entity has a known label and is a person or organization
37     if ent.label_ in ["PERSON", "ORG"]:
38         print(f"Found entity: {ent.text} ({ent.label_})")
39
40     # Extract the relationship - look for verbs and their objects/subjects
41     for token in sent:
42         # Look for verbs that might indicate relationships
43         if token.pos_ == "VERB" and token.dep_ in ["ROOT"]:
44             # Find related tokens (subjects, objects, attributes)
45             for child in token.children:
46                 if child.dep_ in ["attr", "nsubj", "nsubjpass", "dobj", "pobj"] and child != ent:
47                     relationships.append((ent.text, token.lemma_, child.text))
48                     print(f"Relationship: {ent.text} --{token.lemma_}--> {child.text}")
49
50 print("\nExtracted relationships:")
51 for subject, relation, obj in relationships:
52     print(f"{subject} --{relation}--> {obj}")
53
54 # Alternative approach: Simple pattern-based extraction
55 print("\nAlternative pattern-based relationships:")
56 simple_relationships = []
57
58 for sent in doc.sents:
59     for token in sent:
60         # Look for "founded by" pattern
61         if token.lemma_ == "found" and token.head.dep_ == "ROOT":
62             # Find subject (what was founded)
63             subject = None
64             founded_by = None

```

```

65
66         for child in token.head.children:
67             if child.dep_ == "nsubjpass":
68                 subject = child.text
69
70         # Look for "by" preposition
71         for child in token.children:
72             if child.dep_ == "agent" and child.head == token:
73                 for grandchild in child.children:
74                     if grandchild.dep_ == "pobj":
75                         founded_by = grandchild.text
76
77         if subject and founded_by:
78             simple_relationships.append((subject, "founded_by", founded_by))
79
80 for subject, relation, obj in simple_relationships:
81     print(f"{subject} --{relation}--> {obj}")

```

Como podemos ver el programa puede detectar las entidades encontradas, además del score con las cuales estaba asociadas estas palabras.

```

Some weights of the model checkpoint at dbmdz/bert-large-cased-fine-tuned-v2.0
- This IS expected if you are initializing BertForTokenClassification from
- This IS NOT expected if you are initializing BertForTokenClassification from
tokenizer_config.json: 100% ██████████ 60.0/60.0
vocab.txt: 100% ██████████ 213k/213k [00:00<]
Device set to use cpu
Named entities detected by Transformers:
Entity: Gee, Label: I-ORG, Score: 0.9952
Entity: ##k, Label: I-ORG, Score: 0.9548
Entity: ##fo, Label: I-ORG, Score: 0.9875
Entity: ##r, Label: I-ORG, Score: 0.9836

```

```

Entity: ##G, Label: I-ORG, Score: 0.9928
Entity: ##eek, Label: I-ORG, Score: 0.9268
Entity: ##s, Label: I-ORG, Score: 0.9153
Entity: Sand, Label: I-PER, Score: 0.9996
Entity: ##eep, Label: I-PER, Score: 0.9997

```

Una vez que realiza esta evaluación podemos ver como en la siguiente imagen tenemos las entidades que encontraron y la etiqueta con las cuales esta etiquetadas las entidades, después podemos ver cómo es que procesa las oraciones y como es que la relaciona.

```

SpaCy entities:
Entity: Sandeep Jain, Label: ORG
Entity: GeeksforGeeks, Label: ORG

Processing sentence: GeekforGeeks was founded by Sandeep Jain.
Found entity: Sandeep Jain (ORG)

Processing sentence: Sandeep Jain was a former teacher.

Processing sentence: GeeksforGeeks is a computer science portal and offers various courses and articles.
Found entity: GeeksforGeeks (ORG)

Extracted relationships:

Alternative pattern-based relationships:
GeekforGeeks --founded_by--> Jain

```

Traducción automática

La traducción automática estadística (TMA) es un tipo de traducción automática (TA) que utiliza modelos estadísticos para traducir texto de un idioma a otro. A diferencia de los sistemas tradicionales basados en reglas, la TMA se basa en grandes corpus de textos bilingües para construir modelos probabilísticos que determinan la probabilidad de que una oración en el idioma de destino se corresponda con una oración en el idioma de origen. Este enfoque marcó un cambio significativo en el procesamiento del lenguaje natural (PLN) y abrió la puerta a tecnologías de traducción automática más avanzadas.

La Traducción Automática Estadística (TME) funciona analizando grandes corpus bilingües, como textos paralelos o pares de traducciones alineadas con oraciones, para identificar patrones y relaciones entre palabras y frases en diferentes idiomas. Estos patrones se utilizan para construir modelos probabilísticos que pueden generar traducciones para nuevas oraciones o documentos.

Dada la complejidad de la traducción, no sorprende que los sistemas de traducción automática más eficaces se desarrollen entrenando un modelo probabilístico con estadísticas derivadas de un vasto corpus textual. Este método no requiere una ontología compleja de conceptos interlingüísticos, gramáticas elaboradas a mano para los idiomas de origen y destino, ni un árbol de traducción etiquetado manualmente. En cambio, simplemente requiere datos en forma de traducciones de ejemplo a partir de los cuales se pueda aprender un modelo de traducción.

Tipos de traducción automática estadística

Los tipos de SMT son los siguientes:

Traducción basada en palabras

En este caso, la unidad básica de la traducción es una palabra en el idioma de origen. Es decir, se trata de un modelo que traduce palabra por palabra. Aunque debido a los modismos, a la morfología y a las palabras compuestas, la cantidad de palabras del texto traducido puede ser distinta a la del texto de origen.

La fertilidad es la cantidad de palabras en la lengua de destino producidas por una palabra en la lengua de origen. Por ejemplo, la voz inglesa “nail” se puede traducir al español como “uña” o “clavo”. Con una alta fertilidad, es posible asignar una palabra en la lengua de origen a varias palabras en la lengua de destino, pero no se pueden agrupar dos palabras de la lengua de destino que signifiquen una sola en la lengua de origen.

Traducción basada en frases

Este tipo de tecnología de traducción automática traduce secuencias completas de palabras, buscando disminuir las restricciones de la SMT en palabras. Estas secuencias se denominan frases o bloques. Se trata de frases que no atienden a estructuras lingüísticas, sino a métodos estadísticos, con el fin de no reducir la calidad de traducción.

Traducción basada en sintaxis

En este tipo de tecnología de traducción automática, el modelo de SMT deja a un lado las palabras individuales y los bloques para traducir unidades sintácticas. Esto significa que traduce mediante el análisis de oraciones o expresiones.

Traducción basada en modelos de lenguaje

Los modelos de lenguaje ayudan a que la traducción resulte más fluida y natural. Se trata de una función que, en base a una oración traducida, selecciona la que es más probable que sea usada por un hablante nativo. También facilita elegir la palabra más adecuada ante la posibilidad de múltiples traducciones.

Ventajas de la traducción automática.

Los traductores humanos utilizan los servicios de traducción automática para traducir con mayor rapidez y eficacia. A continuación, presentamos algunas ventajas de la traducción automática:

- **Ayuda de traducción automatizada:** La traducción automática es un buen punto de partida para los traductores humanos profesionales. Numerosos sistemas de administración de traducciones integran uno o varios modelos de traducción automática en el flujo de trabajo. Disponen de configuraciones para ejecutar las traducciones automática y posteriormente enviarlas a los traductores humanos para su edición posterior.
- **Velocidad y volumen:** La traducción automática funciona muy rápido, pues traduce millones de palabras de forma casi instantánea. Es capaz de traducir grandes cantidades de datos, como un chat en tiempo real o casos jurídicos a gran escala. Además, tiene la capacidad de procesar documentos en un idioma extranjero, buscar términos relevantes y recordarlos para usos futuros.
- **Amplia selección de idiomas:** La mayoría de los principales proveedores de traducción automática admiten entre 50 y 100 idiomas. Las traducciones también se realizan simultáneamente para varios idiomas, lo que resulta útil para el lanzamiento global de productos y la actualización de la documentación.
- **Traducción rentable:** La traducción automática aumenta la productividad y la capacidad de entregar las traducciones más rápidamente, lo que reduce el tiempo de comercialización. Hay menos participación humana en el proceso, ya que la traducción automática proporciona traducciones básicas pero valiosas, lo que reduce tanto el costo como el plazo de entrega. Por ejemplo, en proyectos de gran volumen, se puede integrar la traducción automática con los sistemas de administración de contenidos para etiquetar y organizar automáticamente el contenido antes de traducirlo a los distintos idiomas.

Traducción automática con Trasnformer

La traducción automática consiste en convertir texto de un idioma a otro. Herramientas como Google Translate utilizan esta tecnología. Muchos sistemas de traducción utilizan modelos de transformación que son eficaces para comprender el significado de las oraciones.

Transformer es un modelo de aprendizaje profundo utilizado en el procesamiento del lenguaje natural (PLN) porque puede comprender cómo se relacionan las palabras de una oración, incluso si están muy separadas. Consta de dos partes principales:

- Codificador: lee y comprende la oración de entrada (inglés en nuestro ejemplo).
- Descodificador: crea la oración traducida en el idioma de destino (aquí hindi).

La característica más importante de Transformers es la autoatención, que ayuda al modelo a concentrarse en las palabras correctas durante la traducción. A diferencia de los modelos anteriores que procesan las palabras una por una, este analiza la oración completa simultáneamente, lo que los hace más rápidos y eficientes.

Ventajas e inconvenientes de los Transformers en PLN

Los grandes modelos del lenguaje han revolucionado el campo del procesamiento del lenguaje natural (PLN) al proporcionar una capacidad sin precedentes para comprender y generar texto de manera coherente. Entre estos modelos, destacan los llamados Transformers, que han demostrado ser altamente efectivos en diversas tareas de PLN. En este artículo, exploraremos qué son los Transformers en PLN, cómo funcionan, así como sus ventajas e inconvenientes.

Ventajas de un transformador PLN

Los Transformers en PLN presentan varias ventajas significativas en comparación con otros enfoques de modelado del lenguaje. En primer lugar, son altamente paralelizables, lo que significa que pueden procesar múltiples partes de una secuencia al mismo tiempo, lo que acelera significativamente el entrenamiento y la inferencia.

Además, los Transformers son capaces de capturar dependencias a largo plazo en el texto, lo que les permite comprender mejor el contexto global y generar texto más coherente. También son más flexibles y escalables, lo que facilita su adaptación a diferentes tareas y dominios.

Inconvenientes de un Transformer en PLN

Aunque los Transformers tienen muchas ventajas, también tienen algunas limitaciones. Uno de los principales inconvenientes es su alta demanda computacional. Debido a su tamaño y complejidad, los modelos basados en Transformers requieren grandes cantidades de recursos computacionales y tiempo de entrenamiento.

Además, los Transformers son muy sensibles a la calidad y cantidad de los datos de entrenamiento. Si los datos de entrenamiento son limitados o están sesgados, el rendimiento del modelo puede verse afectado negativamente. Esto puede ser un desafío en situaciones donde los datos son escasos o difíciles de obtener.

Ejemplo de Python con Transformer

Este ejemplo fue extraído de la pagina de <https://www.geeksforgeeks.org/nlp/machine-translation-with-transformer-in-python/> , realizamos algunas modificaciones para la estética y pusimos lo que son comentarios para tener mejor estructurado el código del programa.

```
1  # =====
2  # IMPORTACIÓN DE LIBRERÍAS
3  # =====
4
5  # Librerías para manejo de datos y datasets
6  from datasets import load_dataset
7  import numpy as np
8
9  # Librerías de transformers para modelos de traducción
10 from transformers import (
11     AutoTokenizer,
12     AutoModelForSeq2SeqLM,
13     DataCollatorForSeq2Seq,
14     Seq2SeqTrainingArguments,
15     Seq2SeqTrainer
16 )
17
18 # Librerías para evaluación y computación
19 import evaluate
20 import torch
21
22 # Librería para crear la interfaz web
23 import gradio as gr
24
25 # =====
26 # CONFIGURACIÓN INICIAL
27 # =====
28
29 # Definimos la longitud máxima que pueden tener nuestras oraciones
30 max_length = 256
31
32 # Configuramos si usar GPU (más rápido) o CPU
33 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
34
```

```

35 # =====
36 # CARGA DE DATOS Y MODELO
37 # =====
38
39 # Cargamos un conjunto de datos que contiene traducciones entre inglés e hindi
40 dataset = load_dataset("cfilt/iitb-english-hindi")
41
42 # Cargamos un tokenizer (convierte texto en números que entiende la computadora)
43 tokenizer = AutoTokenizer.from_pretrained("Helsinki-NLP/opus-mt-en-hi")
44
45 # Cargamos un modelo pre-entrenado que ya sabe traducir de inglés a hindi
46 model = AutoModelForSeq2SeqLM.from_pretrained("Helsinki-NLP/opus-mt-en-hi")
47
48 # Movemos el modelo a la GPU si está disponible
49 model.to(device)
50
51 # =====
52 # PREPARACIÓN DE DATOS
53 # =====
54
55 def preprocess_function(examples):
56     """
57     Función que prepara los datos para entrenar el modelo
58     Convierte el texto a números que puede entender la computadora
59     """
60     # Extraemos las oraciones en inglés
61     inputs = [ex["en"] for ex in examples["translation"]]
62     # Extraemos las oraciones en hindi (las traducciones correctas)
63     targets = [ex["hi"] for ex in examples["translation"]]
64
65     # Convertimos el texto en inglés a números que entiende el modelo
66     model_inputs = tokenizer(inputs, max_length=max_length, truncation=True)
67     # Convertimos el texto en hindi a números
68     labels = tokenizer(targets, max_length=max_length, truncation=True)
69     # Guardamos las traducciones correctas como "etiquetas" para que el modelo aprenda
70     model_inputs["labels"] = labels["input_ids"]
71
72     return model_inputs
73
74 # Preparamos los datos de validación (para probar qué tan bien funciona el modelo)
75 tokenized_datasets_validation = dataset['validation'].map(
76     preprocess_function,
77     batched=True,
78     remove_columns=dataset["validation"].column_names,
79     batch_size=2
80 )
81

```

```

82 # Preparamos los datos de prueba
83 tokenized_datasets_test = dataset['test'].map(
84     preprocess_function,
85     batched=True,
86     remove_columns=dataset["test"].column_names,
87     batch_size=2
88 )
89
90 # Creamos una herramienta que organiza los datos en grupos para entrenar
91 data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
92
93 # =====
94 # CONFIGURACIÓN DEL MODELO
95 # =====
96
97 # Primero permitimos que todas las partes del modelo puedan aprender
98 for parameter in model.parameters():
99     parameter.requires_grad = True
100
101 # Congelamos algunas capas del encoder (parte que lee el inglés)
102 # Esto significa que estas capas no cambiarán durante el entrenamiento
103 num_layers_to_freeze = 10
104 for layer_index, layer in enumerate(model.model.encoder.layers):
105     if layer_index < len(model.model.encoder.layers) - num_layers_to_freeze:
106         for parameter in layer.parameters():
107             parameter.requires_grad = False
108
109 # Congelamos algunas capas del decoder (parte que genera el hindi)
110 num_layers_to_freeze = 10
111 for layer_index, layer in enumerate(model.model.decoder.layers):
112     if layer_index < len(model.model.encoder.layers) - num_layers_to_freeze:
113         for parameter in layer.parameters():
114             parameter.requires_grad = False
115
116 # =====
117 # MÉTRICAS DE EVALUACIÓN
118 # =====
119
120 # BLEU es una métrica que compara traducciones automáticas con traducciones humanas
121 metric = evaluate.load("sacrebleu")
122
123 def compute_metrics(eval_preds):
124     """
125     Función que calcula qué tan bien está traduciendo nuestro modelo
126     Devuelve una puntuación BLEU (entre 0 y 100, más alto es mejor)
127     """

```

```

128     preds, labels = eval_preds
129     if isinstance(preds, tuple):
130         preds = preds[0]
131
132     # Convertimos los números de vuelta a texto legible
133     decoded_preds = tokenizer.batch_decode(preds, skip_special_tokens=True)
134
135     # Preparamos las traducciones correctas para compararlas
136     labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
137     decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)
138
139     # Limpiamos los textos eliminando espacios extra
140     decoded_preds = [pred.strip() for pred in decoded_preds]
141     decoded_labels = [[label.strip()] for label in decoded_labels]
142
143     # Calculamos la puntuación BLEU
144     result = metric.compute(predictions=decoded_preds, references=decoded_labels)
145     return {"bleu": result["score"]}
146
147 # =====
148 # CONFIGURACIÓN DE ENTRENAMIENTO
149 # =====
150
151 # Configuramos cómo queremos entrenar el modelo
152 training_args = Seq2SeqTrainingArguments(
153     output_dir="finetuned-nlp-en-hi",          # Nombre del modelo entrenado
154     gradient_checkpointing=True,                # Ahorra memoria
155     per_device_train_batch_size=32,             # Cuántos ejemplos procesar a la vez
156     learning_rate=1e-5,                        # Qué tan rápido aprende el modelo
157     warmup_steps=2,                            # Pasos de calentamiento
158     max_steps=2000,                            # Cuántos pasos de entrenamiento hacer
159     fp16=True,                                 # Usa menos memoria
160     optim='adafactor',                         # Método de optimización
161     per_device_eval_batch_size=16,             # Tamaño del lote para evaluación
162     metric_for_best_model="eval_bleu",         # Métrica para elegir el mejor modelo
163     predict_with_generate=True,                # Genera predicciones durante evaluación
164     push_to_hub=False,                         # No subir a internet
165 )
166
167 # Creamos el entrenador con todos nuestros datos y configuraciones
168 trainer = Seq2SeqTrainer(
169     model=model,
170     args=training_args,
171     train_dataset=tokenized_datasets_test,     # Datos para entrenar

```

```

171     train_dataset=tokenized_datasets_test,      # Datos para entrenar
172     eval_dataset=tokenized_datasets_validation, # Datos para evaluar
173     data_collator=data_collator,               # Organizador de datos
174     tokenizer=tokenizer,                       # Convertidor de texto
175     compute_metrics=compute_metrics,          # Calculador de métricas
176 )
177
178 # =====
179 # ENTRENAMIENTO DEL MODELO
180 # =====
181
182 print(";Iniciando el entrenamiento del modelo!")
183 trainer.train()
184 print(";Entrenamiento completado!")
185 # =====
186 # FUNCIÓN DE TRADUCCIÓN
187 # =====
188 def translate(text):
189     """
190     Función que traduce texto nuevo de inglés a hindi
191     """
192     # Convertimos el texto a números
193     inputs = tokenizer(text, return_tensors="pt").to(device)
194     # El modelo genera la traducción
195     translated_tokens = model.generate(**inputs, max_length=256)
196     # Convertimos los números de vuelta a texto
197     results = tokenizer.batch_decode(translated_tokens, skip_special_tokens=True)[0]
198     return results
199
200 # =====
201 # INTERFAZ WEB
202 # =====
203
204 # Creamos una interfaz web simple donde la gente puede escribir texto y obtener traducciones
205 interface = gr.Interface(
206     fn=translate,
207     inputs=gr.Textbox(
208         lines=2,
209         placeholder='Escribe aquí el texto en inglés que quieres traducir...'
210     ),
211     outputs='text',
212     title="Traductor Inglés → Hindi",
213     description="Introduce texto en inglés y obtén su traducción al hindi"
214 )
215 # Lanzamos la interfaz web
216 print("Lanzando la interfaz web...")
217 interface.launch()

```

Una vez ejecutado, lo cual tarda aproximadamente 9 horas o más, como podemos ver tenemos esto.

```

{"loss": 3.2834, "grad_norm": 2.361265182495117, "learning_rate": 7.507507507507507e-06, "epoch": 6.33}
25% | 500/2000 [1:51:03<5:01:34, 12.06s/it]
/Users/elman/OneDrive/Escritorio/Ejercicios NumPy/numpys\Lib\site-packages\torch\utils\data\loader.py:665: UserWarning: Moving the following attributes in the config to the generation config: ('max_length': 512, 'num_beams': 4, 'bad_words_ids': [[61949]]). You are seeing this warning because you've set generation parameters in the model config, as opposed to the generation config.
warnings.warn(
28% | 553/2000 [2:02:31<4:11:27, 10.43s/it]
/Users/elman/OneDrive/Escritorio/Ejercicios NumPy/numpys\Lib\site-packages\torch\utils\data\loader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinned memory won't be used.
warnings.warn(warn_msg)
{"loss": 2.5376, "grad_norm": 1.731118883000183, "learning_rate": 5.005005005005006e-06, "epoch": 12.66}
51% | 1027/2000 [3:43:53<2:37:51, 9.73s/it]
/Users/elman/OneDrive/Escritorio/Ejercicios NumPy/numpys\Lib\site-packages\torch\utils\data\loader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinned memory won't be used.
warnings.warn(warn_msg)
{"loss": 2.3501, "grad_norm": 1.414618968963623, "learning_rate": 2.502502502502503e-06, "epoch": 18.99}
75% | 1501/2000 [5:41:04<1:33:40, 11.26s/it]
/Users/elman/OneDrive/Escritorio/Ejercicios NumPy/numpys\Lib\site-packages\torch\utils\data\loader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinned memory won't be used.
warnings.warn(warn_msg)
{"loss": 2.2866, "grad_norm": 1.400938868522644, "learning_rate": 0.0, "epoch": 25.32}
{"train_runtime": 27981.3055, "train_samples_per_second": 2.287, "train_steps_per_second": 0.071, "train_loss": 2.594427498234375, "epoch": 25.32}
100% | 2000/2000 [7:46:21<00:00, 13.99s/it]
Entrenamiento completado!
Lanzando la interfaz web...

```

Traductor Inglés → Hindi

Introduce texto en inglés y obtén su traducción al hindi

text

Hello

Clear

output

सलाम

Submit

Flag

Como podemos ver ya funciona y ahora veremos si el texto traducido es correcto.

Hindi

↔

Inglés

Prueba en Maratí

×

सलाम

salam

🔊

🎤

📄

📷

↺

Hello

🔄

Tono

🔊

📄

Otras formas de decir esto

Como podemos ver si funciona el traductor de texto de la aplicación, para ser sincero, necesita ser entrenado más y con más datos para un funcionamiento con un estándar de precisión por lo menos del 90% pero el problema es que por los datos y que es un modelo que puedas correr en la computadora, sabiendo la cantidad de recursos que consume, pues nuestra computadora no es tan potente como para poder correr uno.

Resumen de texto

El resumen de texto permite generar un resumen conciso a partir de un texto extenso. Esto resulta especialmente útil al trabajar con una gran cantidad de datos y extraer las ideas principales de forma concisa y legible. Los modelos de IA son algoritmos diseñados para realizar tareas específicas, en particular en el procesamiento del lenguaje natural. Estos modelos analizan datos de texto para extraer información, generar respuestas o clasificar la información basándose en patrones aprendidos.

Se entrenan con grandes conjuntos de datos, lo que les permite generalizar y funcionar correctamente con datos desconocidos



Resumen utilizando modelo Hugging Face Model

Hugging Face ofrece modelos de aprendizaje (BART, GPT-2 o RoBERTa), estos son encargados simplificar el procesamiento del lenguaje natural. Ofrece una plataforma donde los usuarios pueden acceder y competir modelos de IA diseñados para diversas tareas de PNL.

El resumen consiste en producir una versión más corta de un documento, conservando su información importante. Algunos modelos pueden extraer texto de la entrada original, mientras que otros pueden generar texto completamente nuevo.

- Modelo abstractivo: Genera un resumen estableciendo nuevas expresiones que formulan las ideas más importantes del texto original.
- Basado en arquitectura Transformers: Es una biblioteca de modelos preentrenados de procesamiento de lenguaje natural, visión artificial, audio y multimodales para inferencia y entrenamiento. Use Transformers para entrenar modelos con sus datos, crear aplicaciones

de inferencia y generar texto con modelos de lenguaje extensos.



- Adecuando textos complejos y lenguaje natural: basados en Transformers para resumir documentos largos, complejos, con múltiples temas o lenguaje natural.

Ejemplo en Python de Hugging Face Model

Vamos a utilizar el ejemplo de la página de <https://www.geeksforgeeks.org/nlp/text-summarizations-using-huggingface-model/> como ejemplo y modificaremos algunas cosas, también agregaremos más para que veamos el ejemplo, por lo cual mostramos el código que se hizo, también con la ayuda de IA para solucionar errores de código.

```
1  # Importamos las librerías necesarias
2  import gradio as gr # Para crear la interfaz web
3  from transformers import T5Tokenizer, T5ForConditionalGeneration # Para usar el modelo de IA
4  import torch # Librería base para inteligencia artificial
5  import evaluate # Para evaluar qué tan bueno es nuestro resumen
6
7  # Configuramos el modelo de inteligencia artificial que vamos a usar
8  # T5-small es un modelo pequeño y rápido que puede resumir textos
9  model_name = 't5-small'
10
11 # Cargamos las herramientas del modelo:
12 # - El tokenizer convierte las palabras en números que la IA puede entender
13 # - El model es la "mente" que hace el resumen
14 print("Cargando el modelo de IA... esto puede tomar un momento")
15 tokenizer = T5Tokenizer.from_pretrained(model_name)
16 model = T5ForConditionalGeneration.from_pretrained(model_name)
17
18 # Cargamos una herramienta para medir qué tan buenos son nuestros resúmenes
19 # ROUGE es como un "calificador" que compara resúmenes
20 rouge_metric = evaluate.load('rouge')
21
```

```

22 # Función principal que hace el resumen
23 def summarize(text):
24     """
25     Esta función toma un texto largo y lo convierte en un resumen corto
26     """
27     # Preparamos el texto para que la IA lo entienda
28     # Le decimos "summarize:" para que sepa que queremos un resumen
29     inputs = tokenizer.encode(
30         "summarize: " + text,
31         return_tensors="pt", # Formato que entiende la IA
32         max_length=1024,     # Máximo 1024 "palabras-número" de entrada
33         truncation=True      # Si es muy largo, lo cortamos
34     )
35
36     # Aquí la IA hace su "magia" y genera el resumen
37     summary_ids = model.generate(
38         inputs,
39         max_length=150,      # El resumen no puede ser más largo que 150 "palabras-número"
40         min_length=40,       # El resumen debe tener al menos 40 "palabras-número"
41         length_penalty=2.0,  # Preferimos resúmenes más cortos
42         num_beams=4,         # Probamos 4 opciones diferentes y elegimos la mejor
43         early_stopping=True  # Paramos cuando encontramos una buena respuesta
44     )
45
46     # Convertimos los números de vuelta a palabras que podemos leer
47     summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)

```

```

47     summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
48     return summary
49
50 # Función para evaluar qué tan bueno es nuestro resumen
51 def evaluate_summary(original_text, summary, reference_summary=None):
52     """
53     Esta función califica qué tan bueno es el resumen que hicimos
54     """
55     if reference_summary:
56         # Si tenemos un resumen "perfecto" para comparar, lo usamos
57         scores = rouge_metric.compute(
58             predictions=[summary],
59             references=[reference_summary]
60         )
61         return scores
62     else:
63         # Si no tenemos referencia, solo mostramos información básica
64         return {
65             "texto_original_palabras": len(original_text.split()),
66             "resumen_palabras": len(summary.split()),
67             "reduccion_porcentaje": round((1 - len(summary.split())/len(original_text.split())) * 100, 2)
68         }
69
70 # Texto de ejemplo para probar nuestro programa
71 text = """ La librería Hugging Face ha revolucionado el campo del procesamiento de lenguaje natural con su
72 Esta librería proporciona modelos de última generación para varias tareas de procesamiento de lenguaje natu
73 Con APIs fáciles de usar y modelos pre-entrenados, los desarrolladores pueden integrar rápidamente capacida
74 El enfoque impulsado por la comunidad asegura una mejora continua e innovación en la librería, convirtiénd
75
76 # Probamos nuestro programa con el texto de ejemplo
77 print("Texto original:")
78 print(text)
79 print("\n" + "="*50 + "\n")

```

```

76 # Probamos nuestro programa con el texto de ejemplo
77 print("Texto original:")
78 print(text)
79 print("\n" + "="*50 + "\n")
80
81 summary = summarize(text)
82 print("Resumen generado por la IA:")
83 print(summary)
84 print("\n" + "="*50 + "\n")
85
86 # Evaluamos qué tan bueno es el resumen
87 evaluation = evaluate_summary(text, summary)
88 print("Evaluación del resumen:")
89 for key, value in evaluation.items():
90     print(f"- {key}: {value}")
91
92 # Creamos una interfaz web para que cualquier persona pueda usar nuestro programa
93 # Es como crear una página web simple donde puedes escribir texto y obtener resúmenes
94 iface = gr.Interface(
95     fn=summarize, # La función que hace el trabajo
96     inputs="text", # Entrada: una caja de texto donde escribir
97     outputs="text", # Salida: otra caja de texto con el resumen
98     title="Resumidor de Textos con Inteligencia Artificial",
99     description="Escribe o pega cualquier texto largo y obtendrás un resumen automático. ¡Es como tener un asistente que lee por tí!",
100     examples=[
101         ["Escribe aquí tu texto largo y presiona 'Submit' para obtener un resumen automático."],
102         [text] # Nuestro ejemplo de prueba
103     ]
104 )
105
106 # Lanzamos la interfaz web
107 # Esto abrirá una página web en tu navegador donde puedes probar el programa
108 print("\nLanzando la interfaz web...")
109 print("Se abrirá una página en tu navegador donde puedes probar el resumidor")
110 iface.launch()

```

Este código es el que utilizaremos, ahora al ejecutarlo podemos ver que tenemos lo siguiente.

```

Texto original:
La librería Hugging Face ha revolucionado el campo del procesamiento de lenguaje natural con su librería transformers.
Esta librería proporciona modelos de última generación para varias tareas de procesamiento de lenguaje natural, incluyendo resumen de textos, clasificación de textos, respuesta a preguntas y más.
Con APIs fáciles de usar y modelos pre-entrenados, los desarrolladores pueden integrar rápidamente capacidades avanzadas de procesamiento de lenguaje natural en sus aplicaciones.
El enfoque impulsado por la comunidad asegura una mejora continua e innovación en la librería, convirtiéndola en un recurso valioso tanto para investigadores como para profesionales.

=====

Resumen generado por la IA:
la librería Hugging Face ha revolucionado el campo de procesamiento de lenguaje natural con su librería transformers. los desarrolladores pueden integrar rápidamente capacidades avanzadas de procesamiento de lenguaje natural.

=====

Evaluación del resumen:
- texto_original palabras: 93
- resumen_palabras: 29
- reduccion_porcentaje: 68.82

Lanzando la interfaz web...
Se abrirá una página en tu navegador donde puedes probar el resumidor
* Running on local URL: http://127.0.0.1:7860
* To create a public link, set 'share=True' in 'launch()'.

```

Ahora dentro de la interfaz también podemos meter texto para resumirlo.

Resumidor de Textos con Inteligencia Artificial

Escribe o pega cualquier texto largo y obtendrás un resumen automático. ¡Es como tener un asistente que lee por tí!

text

Un texto es un conjunto de frases ordenadas que tienen sentido y presentan coherencia y cohesión, lo que permite su correcta interpretación por parte del lector o oyente. En lingüística, se le llama texto a la configuración de lengua o habla organizada según reglas específicas. Además, los textos pueden clasificarse en diferentes tipos, como textos expositivos, instructivos y literarios, cada uno con características y propósitos distintos.

output

un texto es un conjunto de frases ordenadas que tienen sentido y presentan coherencia y cohesión, lo que permite su correcta interpretación por parte del lector o oyente.

Flag

Clear

Submit

Examples

Escribe aquí tu texto largo y presiona 'Submit' para obtener...

La librería Hugging Face ha revolucionado el campo del proc...

Dentro del código también podemos cambiar parámetros como lo son el número máximo o mínimo de las palabras queremos que se resume, además de cuantos diferentes resúmenes generen y del cual elegirá el mejor.

Página 65 | 201

Resumen utilizando Sumy

Sumy es una biblioteca de Python para tareas de procesamiento de lenguaje natural, se utiliza principalmente para el resumen de párrafos mediante diferentes algoritmos. **Esta biblioteca es ideal para resumir documentos extensos.**

Podemos utilizar diferentes algoritmos para el resumen de texto, incluidos:

- Lsa (Análisis Semántico Latente)
- Resumen de Lsa
- KISum (Kullback- Leibler)
- Resumen de KI
- LexRank
- Resumen LexRank
- Luhn
- Resumen de Luhn
- Edmundson
- Resumen de Edmundson

Dentro de los ejemplos para ver cómo funciona Sumy ocuparemos lo que son 3 de los mencionados en la lista por lo cual les daremos una definición de cada uno.

Resumen Luhn

El algoritmo de Luhn, también conocido como fórmula de Luhn, "módulo 10" o algoritmo "mod 10", es una fórmula de suma de comprobación simple que se utiliza para validar números de identificación. El algoritmo, desarrollado por el ingeniero de IBM Hans Peter Luhn, se puede utilizar para validar números de tarjetas de crédito, números de identidad internacional de equipos móviles (IMEI) y otros datos numéricos. Las empresas suelen usar este algoritmo para aumentar la precisión y fiabilidad de sus procesos de entrada de datos, especialmente cuando manejan información confidencial como los datos de pago.

El algoritmo de Luhn ha marcado un punto de inflexión en la validación de datos y la prevención del fraude en múltiples sectores, especialmente en los sectores de las finanzas, los pagos y los servicios gubernamentales.

Estas son algunas de las razones por las que es tan importante:

- **Precisión mejorada de los datos:** el algoritmo de Luhn es un método sencillo para detectar tipos comunes de errores de entrada de datos, como errores de un solo dígito o errores de transposición que invierten dos dígitos, algo que es especialmente importante en sectores como las finanzas y la banca.
- **Implementación sencilla:** el algoritmo es fácil de entender e implementar en varios lenguajes de programación sin necesidad de utilizar recursos de cálculo importantes, por lo que es posible utilizarlo en una amplia variedad de aplicaciones.

- **Funcionalidades en tiempo real:** el algoritmo de Luhn se puede aplicar en tiempo real para verificar la exactitud de los datos numéricos a medida que se introducen y se ha convertido en una verificación importante para los sistemas que requieren una validación instantánea de los datos.
- **Prevención de fraude:** si bien no es infalible contra intentos de fraude, el algoritmo de Luhn actúa como primera línea de defensa. Ayuda a descartar números no válidos antes de que se lleve a cabo una transacción u operación, lo que puede disuadir los intentos de fraude y reducir la incidencia de errores que podrían dar lugar a fraudes o vulneraciones de la seguridad.
- **Eficiencia en función del costo:** al detectar errores al principio del proceso, el algoritmo ayuda a evitar los costos asociados con el procesamiento incorrecto de datos, como pérdidas financieras, insatisfacción del cliente y costos administrativos derivados de la corrección de errores.
- **Influencia global:** debido a su eficacia y simplicidad, el mundo ha adoptado ampliamente el algoritmo de Luhn para la validación de números de identificación por parte de las principales compañías de tarjetas de crédito, agencias gubernamentales e instituciones financieras.

Resumen Edmundson

Método heurístico con investigación estadística previa: la mejora del método de Luhn mencionado anteriormente. Edmundson agregó tres heurísticas más al método para medir la importancia de las oraciones. Encuentra las llamadas palabras pragmáticas, las palabras que están en los encabezados y la posición de los términos extraídos. Por lo tanto, este método tiene cuatro submétodos y la combinación adecuada de ellos da como resultado el método de Edmundson. La parte importante es que este método es el más dependiente del idioma porque necesita la lista de palabras adicionales y palabras estigma, excepto las palabras vacías (llamadas aquí palabras nulas). Si le toma en serio el resumen, debería leer cómo Edmundson extrajo estas palabras del corpus en el artículo original. De lo contrario, aquí hay algunas pistas que Edmundson notó y describió allí.

- **Palabras adicionales:** positivamente relevantes. Edmundson encontró aquí la mayoría de los comparativos, superlativos, adverbios de conclusión, términos de valor, interrogativos relativos y términos de causalidad. Por ejemplo: significativo, grande, famoso, glorioso, interesante, famoso, óptimo, estupendo.
- **Palabras estigmatizadas:** de relevancia negativa. Edmundson encontró aquí la mayoría de las expresiones anafóricas, de menosprecio, de detalles insignificantes y de evasivas. Por ejemplo: difícilmente, imposible.
- **Palabras nulas:** palabras irrelevantes. Edmundson encontró aquí la mayoría de los ordinales, cardinales, el verbo "ser", preposiciones, pronombres, adjetivos, auxiliares verbales, artículos y conjunciones coordinantes. Por ejemplo: con, ser, de, o, y, cada, por qué, aquí.

Resumen LSA

El Análisis Semántico Latente (LSA, por sus siglas en inglés) es una técnica utilizada en procesamiento de lenguaje natural para identificar patrones y relaciones entre palabras dentro de un conjunto de textos.

Su objetivo es comprender el significado de los términos en función del contexto en el que aparecen, en lugar de analizar palabras de manera aislada.

Este método se basa en la idea de que las palabras que aparecen en contextos similares tienden a estar relacionadas semánticamente. Para lograrlo, emplea modelos matemáticos que descomponen grandes volúmenes de texto en conceptos latentes, lo que permite encontrar conexiones entre términos que no son evidentes a simple vista.

¿Cómo funciona el Análisis Semántico Latente?

El LSA utiliza una técnica matemática llamada Descomposición en Valores Singulares (SVD), que reduce la dimensionalidad de los datos y extrae patrones ocultos en el texto. Este proceso permite agrupar términos relacionados y asignarles un significado basado en su contexto.

El procedimiento se puede dividir en tres pasos principales:

- **Construcción de la matriz de términos y documentos:** Se crea una matriz donde las filas representan palabras y las columnas representan documentos o fragmentos de texto. Cada celda contiene la frecuencia con la que un término aparece en un documento.
- **Aplicación de la Descomposición en Valores Singulares (SVD):** Se descompone la matriz en tres componentes principales, lo que permite identificar relaciones entre términos y reducir el ruido de datos irrelevantes.
- **Identificación de conceptos latentes:** Se extraen patrones semánticos que permiten agrupar términos relacionados y mejorar la comprensión del contenido.

Este proceso ayuda a los motores de búsqueda a interpretar el significado de las palabras más allá de su forma literal, lo que mejora la precisión en la recuperación de información.

A continuación, mostraremos lo que son ejemplos extraídos de la página de:

<https://www.geeksforgeeks.org/mastering-text-summarization-with-sumy-a-python-library-overview/> cada uno de estos ejemplos ejecuta una pequeña fracción de código que nos ayuda para mostrar el texto resumido.

Ejemplo de Luhn en Python

```
1 from sumy.parsers.plaintext import PlaintextParser
2 from sumy.nlp.tokenizers import Tokenizer
3 from sumy.summarizers.luhn import LuhnSummarizer
4 from sumy.nlp.stemmers import Stemmer
5 from sumy.utils import get_stop_words
6 import nltk
7 nltk.download('punkt')
8
9 def summarize_paragraph(paragraph, sentences_count=2):
10     parser = PlaintextParser.from_string(paragraph, Tokenizer("english"))
11
12     summarizer = LuhnSummarizer(Stemmer("english"))
13     summarizer.stop_words = get_stop_words("english")
14
15     summary = summarizer(parser.document, sentences_count)
16     return summary
17
18 if __name__ == "__main__":
19     paragraph = """Artificial intelligence (AI) is intelligence demonstrated by machines, in contrast
20 to the natural intelligence displayed by humans and animals. Leading AI textbooks define
21 the field as the study of "intelligent agents": any device that perceives its environment
22 and takes actions that maximize its chance of successfully achieving its goals. Colloquially,
23 the term "artificial intelligence" is often used to describe machines (or computers) that mimic
24 "cognitive" functions that humans associate with the human mind, such as "learning" and "problem solving"."""
25
26     sentences_count = 2
27     summary = summarize_paragraph(paragraph, sentences_count)
28
29     for sentence in summary:
30         print(sentence)
```

Teniendo como resultado lo siguiente

```
Artificial intelligence (AI) is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and animals.
Colloquially, the term "artificial intelligence" is often used to describe machines (or computers) that mimic "cognitive" functions that humans associate with the human mind, such as
"learning" and "problem solving".
```

Este programa implementa un sistema de resumen automático de texto utilizando la biblioteca Sumy de Python y el algoritmo de Luhn para condensar párrafos largos manteniendo la información más relevante. El código configura un analizador de texto plano que procesa el contenido en inglés, utilizando un tokenizador para dividir el texto en unidades menores, un stemmer para reducir las palabras a su forma raíz, y un filtro que elimina palabras vacías como "the", "and", "is" que no aportan significado relevante al análisis.

En el ejemplo de ejecución, el programa toma un párrafo sobre inteligencia artificial y aplica el algoritmo de Luhn para identificar las dos oraciones más representativas basándose en la frecuencia y distribución de palabras significativas. El resultado es un resumen automático que extrae las oraciones que mejor capturan los conceptos clave del texto original, proporcionando una versión condensada útil para procesar grandes volúmenes de documentos y obtener resúmenes rápidos de cualquier contenido textual extenso.

Ejemplo de Edmundson en Python

```
1 from sumy.parsers.plaintext import PlaintextParser
2 from sumy.nlp.tokenizers import Tokenizer
3 from sumy.summarizers.edmundson import EdmundsonSummarizer
4 from sumy.nlp.stemmers import Stemmer
5 from sumy.utils import get_stop_words
6 import nltk
7 nltk.download('punkt')
8
9 def summarize_paragraph(paragraph, sentences_count=2, bonus_words=None, stigma_words=None, null_words=None):
10     parser = PlaintextParser.from_string(paragraph, Tokenizer("english"))
11
12     summarizer = EdmundsonSummarizer(Stemmer("english"))
13     summarizer.stop_words = get_stop_words("english")
14
15     if bonus_words:
16         summarizer.bonus_words = bonus_words
17     if stigma_words:
18         summarizer.stigma_words = stigma_words
19     if null_words:
20         summarizer.null_words = null_words
21
22     summary = summarizer(parser.document, sentences_count)
23     return summary
24
25 if __name__ == "__main__":
26     paragraph = """Artificial intelligence (AI) is intelligence demonstrated by machines, in contrast
27 to the natural intelligence displayed by humans and animals. Leading AI textbooks define
28 the field as the study of "intelligent agents": any device that perceives its environment
29 and takes actions that maximize its chance of successfully achieving its goals. Colloquially,
30 the term "artificial intelligence" is often used to describe machines (or computers) that mimic
31 "cognitive" functions that humans associate with the human mind, such as "learning" and "problem solving"."""
32
33     sentences_count = 2
34     bonus_words = ["intelligence", "AI"]
35     stigma_words = ["contrast"]
36     null_words = ["the", "of", "and", "to", "in"]
37
38     summary = summarize_paragraph(paragraph, sentences_count, bonus_words, stigma_words, null_words)
39
40     for sentence in summary:
41         print(sentence)
```

Teniendo como resultado lo siguiente

```
Artificial intelligence (AI) is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and animals.
Leading AI textbooks define the field as the study of "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of successfully achiev
ing its goals.
```

Este programa crea resúmenes automáticos de textos largos utilizando una técnica más sofisticada que permite controlar qué tipo de información se incluye o se excluye del resumen final. A diferencia de métodos básicos, este sistema utiliza el algoritmo de Edmundson que permite definir tres tipos de palabras especiales: palabras importantes que queremos destacar (como "inteligencia" e "IA"), palabras que queremos evitar (como "contraste"), y palabras neutras que no aportan significado relevante. El programa analiza un párrafo sobre inteligencia artificial y busca las oraciones que contengan más palabras importantes mientras evita aquellas con palabras no deseadas.

El funcionamiento es como tener un asistente inteligente que lee todo el texto y selecciona automáticamente las dos oraciones más relevantes según nuestros criterios. En el ejemplo, el programa prioriza oraciones que hablen de "inteligencia" e "IA", evita aquellas que mencionen "contraste", e ignora palabras comunes como "el", "de", "y" que aparecen frecuentemente pero no aportan información clave. El resultado es un resumen personalizado que se enfoca en los aspectos que consideramos más importantes del texto original, siendo útil para procesar documentos largos y extraer rápidamente la información que más nos interesa.

Ejemplo de LSA en Python

```
1 from sumy.parsers.plaintext import PlaintextParser
2 from sumy.nlp.tokenizers import Tokenizer
3 from sumy.summarizers.lsa import LsaSummarizer
4 from sumy.nlp.stemmers import Stemmer
5 from sumy.utils import get_stop_words
6 import nltk
7 nltk.download('punkt')
8
9 def summarize_paragraph(paragraph, sentences_count=2):
10     parser = PlaintextParser.from_string(paragraph, Tokenizer("english"))
11
12     summarizer = LsaSummarizer(Stemmer("english"))
13     summarizer.stop_words = get_stop_words("english")
14
15     summary = summarizer(parser.document, sentences_count)
16     return summary
17
18 if __name__ == "__main__":
19     paragraph = """Artificial intelligence (AI) is intelligence demonstrated by machines, in contrast
20 to the natural intelligence displayed by humans and animals. Leading AI textbooks define
21 the field as the study of "intelligent agents": any device that perceives its environment
22 and takes actions that maximize its chance of successfully achieving its goals. Colloquially,
23 the term "artificial intelligence" is often used to describe machines (or computers) that mimic
24 "cognitive" functions that humans associate with the human mind, such as "learning" and "problem solving"."""
25
26     sentences_count = 2
27     summary = summarize_paragraph(paragraph, sentences_count)
28
29     for sentence in summary:
30         print(sentence)
```

Teniendo como resultado lo siguiente

```
Leading AI textbooks define the field as the study of "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals.
Colloquially, the term "artificial intelligence" is often used to describe machines (or computers) that mimic "cognitive" functions that humans associate with the human mind, such as "learning" and "problem solving".
```

Este programa crea resúmenes automáticos de textos utilizando una técnica matemática avanzada llamada LSA (Análisis Semántico Latente) que funciona como un detector inteligente de temas y conceptos relacionados. A diferencia de otros métodos que solo cuentan palabras repetidas, este sistema analiza el significado oculto y las conexiones entre diferentes conceptos del texto para identificar las oraciones más representativas. Es como tener un lector experto que no solo ve las palabras individuales, sino que comprende cómo se relacionan entre sí para formar ideas principales.

El programa toma un párrafo sobre inteligencia artificial y utiliza esta técnica matemática para descubrir automáticamente cuáles son los temas centrales y seleccionar las dos oraciones que mejor los representen. Funciona creando un "mapa conceptual" invisible del texto donde agrupa ideas similares y encuentra patrones de significado, luego elige las oraciones que mejor capturen estos conceptos principales. El resultado es un resumen que no solo incluye las oraciones con palabras más frecuentes, sino aquellas que realmente contienen la esencia y los temas más importantes del texto original, proporcionando una comprensión más profunda y precisa del contenido.

Generación de texto

La generación de texto es el proceso de producir automáticamente texto coherente y significativo, que puede formar oraciones, párrafos o incluso documentos completos. Implica varias técnicas que se pueden encontrar en el campo, como procesamiento de lenguaje natural (PNL), el machine learning y los algoritmos de aprendizaje profundo para analizar los datos de entrada y generar texto similar al humano.

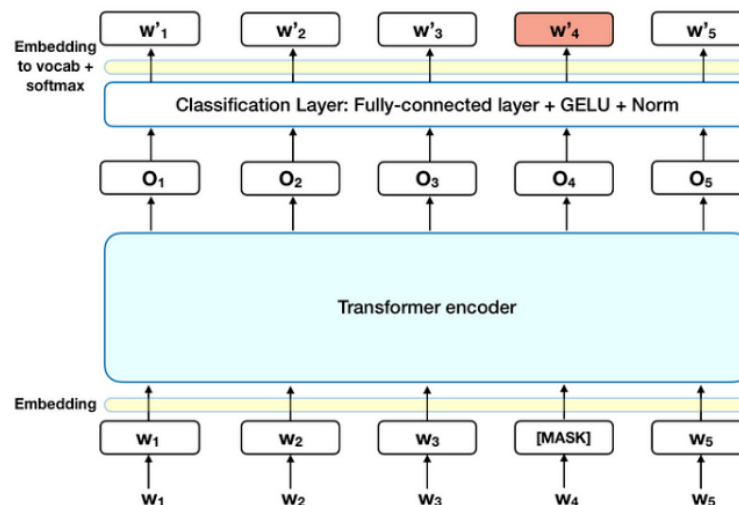
Natural language understanding (NLU) es la capacidad de una máquina para comprender, interpretar y extraer información significativa del lenguaje humano. La NLU ayuda a las máquinas a comprender el contexto, la intención y el significado semántico de las entradas del lenguaje humano.

La generación de lenguaje natural (NLG) es la capacidad de una máquina para producir texto o voz similar a la humana que sea clara, concisa y atractiva. Implica resúmenes de textos, narración de historias, sistemas de diálogo y síntesis de voz. NLG ayuda a las máquinas a generar respuestas significativas y coherentes de una manera que los humanos entiendan fácilmente.



Fnet

El modelo reemplaza la capa de autoatención es un modelo BERT con una transformación de Fourier de devuelve solo las partes reales de la transformada. El modelo es significativo más rápido que es modelo BERT porque tiene menos parámetros y es más eficiente en el uso de memoria.



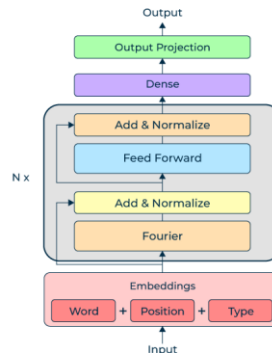
A diferencia de las arquitecturas de transformadores convencionales, como el popular modelo de transformador, FNet utiliza una transformada de Fourier parametrizada en lugar del mecanismo de autoatención. La transformada de Fourier, una operación matemática, se aplica con frecuencia al análisis de imágenes y al procesamiento de señales.

FNet ofrece un método diferente para capturar dependencias entre tokens en una secuencia, aplicando la Transformada de Fourier a las secuencias de entrada. Los autores sostienen que este enfoque puede ser más eficaz y escalable que la autoatención, especialmente para secuencias más largas.

Arquitectura de FNet

FNet es una arquitectura de transformador sin atención como se describe a continuación:

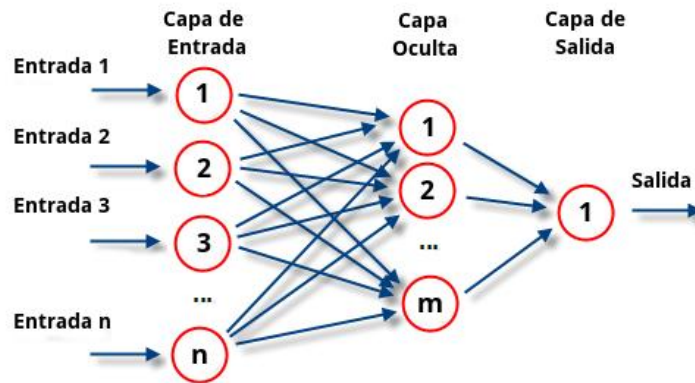
- **Entrada:** El autor ha mantenido la misma entrada que la del transformador. La entrada se integra con incrustaciones de palabras y se combina con incrustaciones de posición absoluta de los tokens y incrustaciones de tipo de la oración.
- **Bloque codificador:** Cada bloque consta de una subcapa de mezcla de Fourier seguida de una subcapa de prealimentación. En esencia, la subcapa de autoatención de la capa codificadora del transformador se reemplaza por una subcapa de Fourier.
- **Capa densa:** La salida del bloque codificador puede utilizarse para diversas tareas, de forma similar a como BERT la utiliza. Por ejemplo, para la clasificación de oraciones, la salida del codificador puede pasar por una capa densa y luego por una capa softmax para la proyección final.



- **Interpretación de los términos clave de F-Net**
- **Transformada de Fourier como mecanismo de mezcla:** La Transformada de Fourier puede describirse como un mecanismo eficaz para mezclar tokens, donde cada operación permite que cada elemento de los datos de entrada influya significativamente en todos los demás. Esto proporciona a las subcapas de prealimentación acceso a todos los tokens.
- **Multiplicación y convolución:** Se dice que multiplicar por los coeficientes de la subcapa de avance en el dominio de la frecuencia es equivalente a convolucionar en el dominio del tiempo. FNet puede considerarse como una alternancia entre multiplicaciones y convoluciones.

Generación de texto usando Recurrent Long Short Term Memory Network

La memoria a largo plazo (LSTM) es una arquitectura de red neuronal recurrente (RNN) diseñada para abordar los problemas de gradiente de desaparición y explosión de las RNN convencionales. Se han utilizado con éxito en tareas de etiquetado y predicción de secuencias, como el reconocimiento de escritura a mano, el modelado del lenguaje y el etiquetado fonético de fotogramas acústicos.



Ejemplo de Generación de texto usando Recurrent Long Short Term Memory Network en Python

```
1  import tensorflow as tf
2  import numpy as np
3  import pandas as pd
4  import random
5  import sys
6
7  # CARGA Y PREPARACIÓN DE DATOS
8  # Carga el archivo CSV con datos de entrenamiento
9  df = pd.read_csv('numpys/train.csv')
10 # Une todos los textos en una sola cadena, elimina valores nulos y convierte a minúsculas
11 text = " ".join(df['text'].dropna().astype(str)).lower()
12
13 # CREACIÓN DEL VOCABULARIO
14 # Crea un vocabulario único ordenado con todos los caracteres del texto
15 vocab = sorted(set(text))
16 print(f'Vocabulary size: {len(vocab)}')
17
18 # MAPEO DE CARACTERES A NÚMEROS
19 # Crea diccionarios para convertir caracteres a números y viceversa
20 char2idx = {c: i for i, c in enumerate(vocab)} # carácter -> índice numérico
21 idx2char = np.array(vocab)                   # índice numérico -> carácter
22
23 # Convierte todo el texto a una secuencia de números
24 text_as_int = np.array([char2idx[c] for c in text])
25
26 # PREPARACIÓN DE SECUENCIAS DE ENTRENAMIENTO
27 seq_length = 100 # Longitud de cada secuencia de entrada
28
29 # Crea un dataset de TensorFlow con los números del texto
30 char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
31
32 # Divide el texto en secuencias de longitud seq_length + 1
33 sequences = char_dataset.batch(seq_length + 1, drop_remainder=True)
34
35 # FUNCIÓN PARA CREAR PARES ENTRADA-OBJETIVO
36 def split_input_target(chunk):
37     """
38     Divide cada secuencia en entrada (X) y objetivo (y)
39     Por ejemplo: si chunk = [1,2,3,4,5]
40     entrada = [1,2,3,4] y objetivo = [2,3,4,5]
41     """
42     return chunk[:-1], chunk[1:]
43
44 # Aplica la función de división a todas las secuencias
45 dataset = sequences.map(split_input_target)
46
```

```

47 # CONFIGURACIÓN DEL DATASET PARA ENTRENAMIENTO
48 BATCH_SIZE = 64 # Número de secuencias por lote
49 BUFFER_SIZE = 10000 # Tamaño del buffer para mezclar datos
50
51 # Mezcla los datos y los agrupa en lotes
52 dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)
53
54 # DEFINICIÓN DE LA ARQUITECTURA DEL MODELO
55 vocab_size = len(vocab) # Tamaño del vocabulario
56 embedding_dim = 64 # Dimensión de los embeddings
57 rnn_units = 128 # Número de unidades en la capa LSTM
58
59 # Crea el modelo secuencial con 3 capas
60 model = tf.keras.Sequential([
61     # Capa 1: Embedding - convierte índices a vectores densos
62     tf.keras.layers.Embedding(vocab_size, embedding_dim, input_shape=(None,)),
63     # Capa 2: LSTM - procesa secuencias y mantiene memoria a largo plazo
64     tf.keras.layers.LSTM(rnn_units, return_sequences=True, recurrent_initializer='glorot_uniform'),
65     # Capa 3: Dense - genera probabilidades para cada carácter del vocabulario
66     tf.keras.layers.Dense(vocab_size)
67 ])
68
69 # CONFIGURACIÓN DE LA FUNCIÓN DE PÉRDIDA
70 def loss(labels, logits):
71     """
72     Función de pérdida para clasificación multi-clase
73     Compara las predicciones con los valores reales
74     """
75     return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)
76
77 # Compila el modelo con optimizador Adam y la función de pérdida definida
78 model.compile(optimizer='adam', loss=loss)
79
80 # Muestra el resumen de la arquitectura del modelo
81 model.summary()
82
83 # ENTRENAMIENTO DEL MODELO
84 EPOCHS = 5 # Número de épocas de entrenamiento
85 # Entrena el modelo con los datos preparados
86 history = model.fit(dataset, epochs=EPOCHS)
87
88 # FUNCIÓN PARA GENERAR TEXTO NUEVO
89 def generate_text(model, start_string, num_generate=100, temperature=1.0):
90     """
91     Genera texto nuevo usando el modelo entrenado
92 
```

```

93     Parámetros:
94     - model: modelo entrenado
95     - start_string: texto inicial para comenzar la generación
96     - num_generate: número de caracteres a generar
97     - temperature: controla la aleatoriedad (menor = más conservador)
98     """
99     # Convierte el texto inicial a números
100    input_eval = [char2idx.get(s, 0) for s in start_string.lower()]
101    input_eval = tf.expand_dims(input_eval, 0) # Añade dimensión de lote
102
103    text_generated = [] # Lista para almacenar caracteres generados
104
105    # Reinicia los estados internos de la capa LSTM
106    model.layers[1].reset_states()
107
108    # Genera caracteres uno por uno
109    for _ in range(num_generate):
110        # Obtiene predicciones del modelo
111        predictions = model(input_eval)
112        # Aplica temperatura para controlar aleatoriedad
113        predictions = tf.squeeze(predictions, 0) / temperature
114        # Selecciona el siguiente carácter usando muestreo categórico
115        predicted_id = tf.random.categorical(predictions, num_samples=1)[-1, 0].numpy()
116
117        # Usa el carácter predicho como entrada para la siguiente predicción
118        input_eval = tf.expand_dims([predicted_id], 0)
119        # Convierte el índice predicho de vuelta a carácter y lo guarda
120        text_generated.append(idx2char[predicted_id])
121
122    # Retorna el texto inicial más el texto generado
123    return start_string + ''.join(text_generated)
124
125    # PRUEBA DE GENERACIÓN DE TEXTO
126    # Genera 200 caracteres comenzando con "The " usando temperatura 0.8
127    print(generate_text(model, start_string="The ", num_generate=200, temperature=0.8))

```

Este programa informático establece un sistema de creación de texto automatizado mediante una red neuronal LSTM (Long Short-Term Memory). Su propósito primordial es aprender a redactar contenido mimetizando el modo de los datos empleados en su formación. En esencia, se adiestra a una máquina para que escriba de manera similar a un ser humano, escudriñando regularidades en el texto. El procedimiento involucra la preparación de información, donde se lee un fichero de texto, se homogeneiza a minúsculas y se construye un léxico de caracteres únicos. Posteriormente, se transforman las letras en cifras para su procesamiento. Se generan secuencias de instrucción, dividiendo el material en segmentos y creando binomios de "entrada" y "salida deseada". La arquitectura del modelo de inteligencia artificial incluye una capa de incrustación que convierte números en vectores, una capa LSTM que funge como la "memoria" para identificar patrones secuenciales, y una capa densa que determina el carácter subsiguiente. Durante el periodo de aprendizaje, el modelo practica la predicción del siguiente carácter en un sinnúmero de ejemplos, asimilando patrones de co-ocurrencia de letras y formación de vocablos a lo

largo de varias rondas. Finalmente, en la fase de producción de texto, al proporcionarle una expresión inicial, el modelo anticipa el siguiente carácter más factible, y así sucesivamente, con una "temperatura" que regula la inventiva o la fidelidad del texto resultante. En pocas palabras, es como instruir a alguien a redactar exhibiéndole múltiples muestras, y luego solicitarle que prosiga una oración. El modelo LSTM sobresale en esta labor por su capacidad de retener el marco de referencia de palabras precedentes, generando un texto armónico que emula el estilo del material original.

Al ejecutar con solo 5 épocas de entrenamiento tenemos lo siguiente:

```
Epoch 1/5
5522/5522 ██████████ 311s 56ms/step - loss: 2.0871
Epoch 2/5
5522/5522 ██████████ 297s 54ms/step - loss: 1.4791
Epoch 3/5
5522/5522 ██████████ 279s 50ms/step - loss: 1.3934
Epoch 4/5
5522/5522 ██████████ 284s 51ms/step - loss: 1.3577
Epoch 5/5
5522/5522 ██████████ 285s 52ms/step - loss: 1.3370
The cat he t t l f iays tinthisa ins tireind f hord core de m irongint an tin s. ersan formeatonfine t that ce tes then allens tomfren at'souess. iomyereng thedelen thithet ca atior
ed surumere lacashiou
PS C:\Users\elman\OneDrive\Escritorio\Ejercicios NumPy> █
```

Al ejecutar con 10 épocas de entrenamiento tenemos lo siguiente:

```
5522/5522 ██████████ 261s 47ms/step - loss: 1.3667
Epoch 5/10
5522/5522 ██████████ 264s 48ms/step - loss: 1.3445
Epoch 6/10
5522/5522 ██████████ 265s 48ms/step - loss: 1.3295
Epoch 7/10
5522/5522 ██████████ 265s 48ms/step - loss: 1.3183
Epoch 8/10
5522/5522 ██████████ 261s 47ms/step - loss: 1.3096
Epoch 9/10
5522/5522 ██████████ 261s 47ms/step - loss: 1.3027
Epoch 10/10
5522/5522 ██████████ 261s 47ms/step - loss: 1.2973
The seseqatalanivirithethawedinthen tond. or t aving re blins wn ry wang f t the id, ot aratste ay te cong in sth in cke tite. he pe ot theres. h acopio f ssthatry: s ind amep icouter
-her t ctig oran tind
PS C:\Users\elman\OneDrive\Escritorio\Ejercicios NumPy> █
```

Entre más entrenemos o más aumentemos las épocas, formara mejor las oraciones y los textos serán mejor comprensibles y con coherencia.

Herramientas Tecnológicas

Para el desarrollo del sistema haremos uso de la herramienta de Visual Studio Code, que es el entorno donde estaremos desarrollando el código, de igual manera dentro del entorno de programación utilizaremos el lenguaje de Python.

Visual Studio Code



Visual Studio Code (VS Code) es un editor de código fuente de código abierto y multiplataforma desarrollado por Microsoft, que se ha consolidado como una herramienta esencial para desarrolladores de todos los niveles. Su diseño ligero, pero potente, lo distingue de los entornos de desarrollo integrados (IDE) más tradicionales, ofreciendo una experiencia de codificación eficiente y altamente personalizable. Aunque se integra de forma nativa con tecnologías de Microsoft como TypeScript y JavaScript, su versatilidad se extiende a una vasta gama de lenguajes de programación, convirtiéndolo en un centro de trabajo unificado para proyectos de diversa índole, desde el desarrollo web y móvil hasta la creación de scripts y la automatización de tareas.

En cuanto a su funcionalidad, VS Code no es solo un editor de texto; es un entorno de desarrollo adaptable que puede transformarse según las necesidades del usuario. Permite la edición simultánea de múltiples archivos, la navegación rápida entre ellos y la gestión de proyectos complejos. Su arquitectura basada en extensiones lo convierte en una plataforma modular, donde los usuarios pueden seleccionar y añadir solo las herramientas que necesitan, evitando la sobrecarga de un IDE completo. Esta capacidad de adaptación lo hace ideal tanto para el desarrollo rápido de prototipos como para la gestión de proyectos de gran escala, proporcionando un equilibrio entre simplicidad y sofisticación.

Si bien Visual Studio Code es fundamentalmente una herramienta para la programación, su relación con el PNL se establece a través de las extensiones y el tipo de proyectos que facilita. Los desarrolladores que trabajan en proyectos de PNL, como la creación de modelos de lenguaje, el análisis de sentimientos o la traducción automática, a menudo utilizan lenguajes como Python, R o Java. VS Code ofrece un soporte excepcional para estos lenguajes a través de extensiones que proporcionan entornos de desarrollo dedicados, con características como la visualización de datos, la gestión de entornos virtuales y la integración con librerías específicas de PNL (como NLTK, spaCy o TensorFlow). De esta manera, VS Code se convierte en el entorno de desarrollo preferido para

construir, depurar y optimizar las complejas aplicaciones y modelos que constituyen el corazón del PLN.

Lenguaje Python

Python se ha consolidado como el lenguaje preferido para el Procesamiento del Lenguaje Natural (PLN) gracias a su simplicidad y, crucialmente, a un ecosistema vasto de librerías y frameworks. Este lenguaje facilita el preprocesamiento de texto, una fase fundamental donde el lenguaje humano se prepara para ser entendido por las máquinas. Esto incluye la tokenización (dividir el texto en unidades), la eliminación de palabras vacías, y la radicalización (stemming) o lematización para reducir las palabras a su forma base. Estas operaciones básicas son esenciales antes de cualquier análisis más complejo.

Una vez preprocesado, Python permite transformar el texto en representaciones numéricas, un paso vital para que los algoritmos de aprendizaje automático puedan trabajar con él. Técnicas como Bag of Words y TF-IDF cuantifican la frecuencia e importancia de las palabras. Sin embargo, un avance significativo son las Word Embeddings (como Word2Vec), que representan palabras como vectores capturando sus relaciones semánticas. Modelos de lenguaje avanzados como BERT o GPT van un paso más allá, entendiendo el contexto y las relaciones complejas entre palabras, lo que permite representaciones mucho más ricas y versátiles para tareas sofisticadas.

Con estas representaciones, Python se convierte en una herramienta potente para abordar tareas avanzadas de PLN. Esto abarca desde la clasificación de texto (como identificar spam o sentimiento), el análisis de sentimiento para determinar emociones, y el reconocimiento de entidades nombradas (NER) para extraer información clave (personas, lugares). También facilita el modelado de temas, la traducción automática, la generación de texto e incluso la respuesta a preguntas, permitiendo a las máquinas interactuar y comprender el lenguaje humano de formas cada vez más complejas.

El poder de Python en PLN reside en sus numerosas librerías. NLTK es ideal para tareas de preprocesamiento y aprendizaje, mientras que spaCy destaca por su eficiencia y modelos preentrenados, perfectos para producción. Gensim es clave para el modelado de temas y Word Embeddings. Para el PLN moderno y profundo, Hugging Face Transformers es indispensable, ofreciendo acceso a modelos de lenguaje de última generación. Finalmente, scikit-learn, TensorFlow y PyTorch proporcionan los algoritmos de aprendizaje automático y los frameworks de Deep Learning necesarios para construir y entrenar modelos sofisticados que impulsan las capacidades actuales del PLN.

Pandas

Pandas es una herramienta que nos ayuda a organizar toda esa información en tablas muy ordenadas y fáciles de leer, casi como si fueran hojas de cálculo inteligentes. Nos permite agrupar datos, encontrar los valores más altos o bajos, o incluso combinar información de diferentes tablas. Es como tener un asistente super eficiente que nos ayuda a poner los datos en orden para que podamos entenderlos mejor y trabajar con ellos sin problemas.

NumPy

NumPy es como una calculadora extremadamente potente, pero en lugar de solo sumar dos números, puede hacer cálculos rapidísimos con listas enteras de números, incluso si son millones. Es perfecto para cuando necesitamos realizar operaciones matemáticas complejas o repetitivas sobre grandes cantidades de datos. Es una herramienta fundamental cuando trabajamos con números y necesitamos procesarlos muy rápido.

TensorFlow

TensorFlow es una plataforma de código abierto integral desarrollada por Google, diseñada para el aprendizaje automático y, en particular, para el aprendizaje profundo. Permite a los desarrolladores y científicos de datos construir y entrenar modelos complejos de redes neuronales, que son capaces de realizar tareas avanzadas como el reconocimiento de voz, la visión por computadora y el procesamiento del lenguaje natural. Su arquitectura flexible permite la implementación en diversas plataformas, desde servidores en la nube hasta dispositivos móviles, ofreciendo las herramientas necesarias para la investigación y el despliegue de soluciones de inteligencia artificial a gran escala.

Keras

Keras es una API de alto nivel para redes neuronales, escrita en Python y capaz de ejecutarse sobre TensorFlow (entre otros backends como Theano o CNTK). Su principal objetivo es permitir una prototipación rápida de modelos de aprendizaje profundo, enfocándose en la facilidad de uso y la modularidad. Al abstraer la complejidad subyacente de TensorFlow, Keras facilita la construcción, entrenamiento y evaluación de redes neuronales profundas, lo que la convierte en una opción ideal tanto para investigadores que buscan experimentar con nuevas arquitecturas como para desarrolladores que necesitan implementar soluciones de IA de manera eficiente.

Tkinter

Tkinter es la biblioteca estándar de Python para crear interfaces gráficas de usuario (GUI). Forma parte de la distribución estándar de Python, lo que significa que no requiere instalación adicional. Permite a los desarrolladores construir aplicaciones de escritorio interactivas, ofreciendo un conjunto de widgets básicos como botones, etiquetas, campos de texto y menús. Aunque no es la única opción para el desarrollo de GUI en Python, su simplicidad y disponibilidad la hacen una excelente elección para proyectos que requieren una interfaz de usuario directa y funcional, siendo útil para herramientas internas, prototipos o aplicaciones con requisitos de interfaz menos complejos.

Sistema

Se busca realizar en Python un programa con el cual podamos realizar diversas acciones como el procesamiento de transcripciones, para posteriormente segmentemos las oraciones de las transcripciones, identifiquemos algunos de los patrones lingüísticos y la detección de errores que corresponde a la primera fase del proyecto, de igual manera también realizaremos una RNN (Red Neuronal Recurrente) para tratar de realizar un pequeño traductor de español a náhuatl. De igual manera para la evaluación con BLEU realizamos un ejemplo para que vea como es que funciona BLEU sin llamarlo con un import.

BLEU

BLEU, que significa "Bilingual Evaluation Understudy", es un método para la evaluación automática de la traducción automática. Es una métrica que compara la salida de un sistema de traducción automática con una o más traducciones de referencia (humanas) y asigna una puntuación que indica la calidad de la traducción. Cuanto más alta sea la puntuación BLEU, más cerca estará la traducción automática de las traducciones de referencia, lo que sugiere una traducción de mejor calidad.

Para el uso del proyecto veremos un ejemplo de BLEU en Python, este ejemplo es realizado utilizando solo librerías de Python, veremos cómo es que funciona.

```
1 import math
2 from collections import Counter
3
4 def preprocesar_texto(texto):
5     """
6     Preprocesamiento básico del texto: tokenización por palabras y conversión a minúsculas.
7     """
8     return texto.lower().split()
9
10 def calcular_precision_n_gramas(candidato, referencias, n):
11     """
12     Calcula la precisión modificada para n-gramas.
13
14     Args:
15         candidato: Lista de tokens de la traducción candidata
16         referencias: Lista de listas de tokens de las traducciones de referencia
17         n: Tamaño de los n-gramas (1 para unigramas, 2 para bigramas, etc.)
18
19     Returns:
20         La cuenta de n-gramas del candidato que aparecen en las referencias (modificado)
21         y el total de n-gramas en el candidato
22     """
23     # Generar n-gramas para el candidato
24     n_gramas_candidato = [tuple(candidato[i:i+n]) for i in range(len(candidato)-n+1)]
25
26     if not n_gramas_candidato:
27         return 0, 0
28
29     # Contar n-gramas en el candidato
30     conteo_candidato = Counter(n_gramas_candidato)
31
32     # Para cada n-grama, encontrar el máximo conteo en todas las referencias
33     conteo_max_referencias = Counter()
34     for referencia in referencias:
35         n_gramas_referencia = [tuple(referencia[i:i+n]) for i in range(len(referencia)-n+1)]
36         conteo_referencia = Counter(n_gramas_referencia)
37
38         for n_grama in conteo_candidato:
39             conteo_max_referencias[n_grama] = max(conteo_max_referencias[n_grama], conteo_referencia[n_grama])
```

BLEU SIN IMPORT 1

```

41     # Calcular conteos recortados (clipped counts)
42     conteo_recortado = 0
43     for n_grama in conteo_candidato:
44         conteo_recortado += min(conteo_candidato[n_grama], conteo_max_referencias[n_grama])
45
46     return conteo_recortado, len(n_gramas_candidato)
47
48 def calcular_brevedad_penalizacion(candidato, referencias):
49     """
50     Calcula la penalización por brevedad (BP).
51
52     Args:
53         candidato: Lista de tokens de la traducción candidata
54         referencias: Lista de listas de tokens de las traducciones de referencia
55
56     Returns:
57         Factor de penalización por brevedad
58     """
59     longitud_candidato = len(candidato)
60
61     # Encontrar la longitud de referencia más cercana
62     longitudes_referencia = [len(ref) for ref in referencias]
63     longitud_referencia_efectiva = min(longitudes_referencia, key=lambda ref_len: abs(ref_len - longitud_candidato))
64
65     if longitud_candidato > longitud_referencia_efectiva:
66         return 1.0
67     else:
68         return math.exp(1 - (longitud_referencia_efectiva / longitud_candidato))
69
70 def calcular_puntaje_bleu(candidato, referencias, pesos=None, verbose=False):
71     """
72     Calcula el puntaje BLEU para una traducción candidata dadas una o más referencias.
73
74     Args:
75         candidato: String con la traducción candidata
76         referencias: Lista de strings con las traducciones de referencia
77         pesos: Lista de pesos para cada n-grama (por defecto [0.25, 0.25, 0.25, 0.25])
78         verbose: Si True, muestra información detallada del cálculo
79
80     Returns:
81         Puntaje BLEU entre 0 y 1
82     """
83     if pesos is None:
84         pesos = [0.25, 0.25, 0.25, 0.25] # Pesos por defecto para 1-4 gramas
85

```

BLEU SIN IMPORT 2

```

86     # Preprocesar textos
87     candidato_tokens = preprocesar_texto(candidato)
88     referencias_tokens = [preprocesar_texto(ref) for ref in referencias]
89
90     if verbose:
91         print(f"Candidato: {candidato_tokens}")
92         print(f"Referencias: {referencias_tokens}")
93
94     # Calcular precisiones para n-gramas de 1 a 4
95     precisiones = []
96     for n in range(1, len(pesos)+1):
97         recortado, total = calcular_precision_n_gramas(candidato_tokens, referencias_tokens, n)
98         if verbose:
99             print(f"Precisión {n}-gramas: {recortado}/{total}")
100
101         if total == 0:
102             precision = 0
103         else:
104             precision = recortado / total
105
106         precisiones.append(precision)
107

```

BLEU SIN IMPORT 3

```

108     # Calcular media geométrica ponderada de las precisiones
109     suma_ponderada = 0
110     for w, p in zip(pesos, precisiones):
111         if p == 0:
112             suma_ponderada = -float('inf')
113             break
114         suma_ponderada += w * math.log(p)
115
116     if suma_ponderada == -float('inf'):
117         media_geo = 0
118     else:
119         media_geo = math.exp(suma_ponderada)
120
121     # Calcular penalización por brevedad
122     bp = calcular_brevedad_penalizacion(candidato_tokens, referencias_tokens)
123
124     # Calcular puntaje BLEU final
125     bleu = bp * media_geo
126
127     if verbose:
128         print(f"Precisiones: {precisiones}")
129         print(f"Media geométrica: {media_geo}")
130         print(f"Penalización por brevedad: {bp}")
131         print(f"Puntaje BLEU: {bleu}")
132
133     return bleu
134
135 # Ejemplo de uso
136 if __name__ == "__main__":
137     candidato = "el gato está debajo de la mesa"
138     referencia1 = "el gato está bajo la mesa"
139     referencia2 = "debajo de la mesa está el gato"
140
141     print("-- Ejemplo de cálculo BLEU --")
142     puntaje = calcular_puntaje_bleu(candidato, [referencia1, referencia2], verbose=True)
143     print(f"\nPuntaje BLEU final: {puntaje:.4f}")

```

BLEU SIN IMPORT 4

Al ejecutar el programa tenemos lo siguiente:

```

-- Ejemplo de cálculo BLEU --
Candidato: ['el', 'gato', 'está', 'debajo', 'de', 'la', 'mesa']
Referencias: [['el', 'gato', 'está', 'bajo', 'la', 'mesa'], ['debajo', 'de', 'la', 'mesa', 'está', 'el', 'gato']]
Precisión 1-gramas: 7/7
Precisión 2-gramas: 5/6
Precisión 3-gramas: 3/5
Precisión 4-gramas: 1/4
Precisión 3-gramas: 3/5
Precisión 4-gramas: 1/4
Precisión 3-gramas: 3/5
Precisión 4-gramas: 1/4
Precisión 3-gramas: 3/5
Precisión 4-gramas: 1/4
Precisión 3-gramas: 3/5
Precisión 4-gramas: 1/4
Precisión 3-gramas: 3/5
Precisión 4-gramas: 1/4
Precisiones: [1.0, 0.8333333333333334, 0.6, 0.25]
Media geométrica: 0.5946035575013605
Penalización por brevedad: 1.0
Puntaje BLEU: 0.5946035575013605

Puntaje BLEU final: 0.5946

```

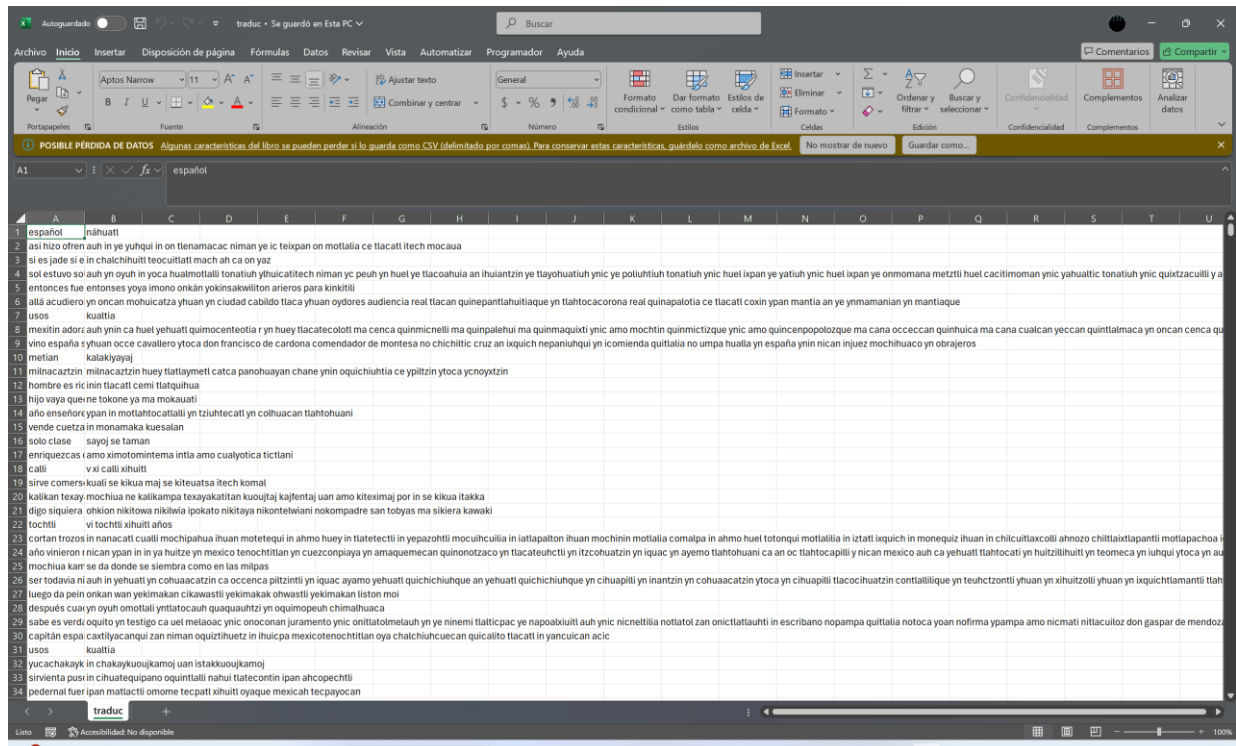
BLEU SIN IMPORT 5

Este programa sirve para evaluar qué tan buena es una traducción hecha por una máquina comparándola con una o varias traducciones hechas por personas. Para lograr esto, analiza pequeños fragmentos de palabras (como grupos de una, dos, tres o cuatro palabras seguidas) y verifica cuántos de esos fragmentos aparecen tanto en la traducción de la máquina como en las traducciones humanas. Mientras más fragmentos coincidan, mejor se considera que está la traducción. Además de contar esas coincidencias, el programa también revisa si la traducción de la máquina es demasiado corta comparada con las humanas. Si es mucho más corta, le pone una penalización, porque en general se espera que una buena traducción tenga una longitud parecida a la original. Esto evita que una traducción recorte demasiado y aun así obtenga una puntuación alta por coincidencias cortas. Finalmente, el programa combina todas esas comparaciones para calcular un número entre 0 y 1, donde 1 significa una traducción perfecta (idéntica a la humana) y 0 significa que no se parece en nada. Ese número final se llama puntaje BLEU y es muy usado en el mundo de la inteligencia artificial para medir el rendimiento de traductores automáticos.

Partes del sistema

Para el desarrollo del sistema se crearon diferentes archivos de Python encargados de diferentes tareas para el correcto funcionamiento del programa general, tenemos como programas los diferentes archivos, para el uso inicial del ocupamos un archivo tipo .csv dentro del cual se encuentran el español y el náhuatl, este es el archivo de donde originaremos los demás resultados y archivos que se van a generar.

Además, dentro de cada programa metimos una interfaz para que podamos visualizar los resultados que salen en la terminal y sea un poco mejor o más estético ver estos resultados.



ARCHIVO .CSV 1

Normalización

Dentro del primer programa de normalización primero creamos un programa el cual carga un archivo tipo .csv que es donde tenemos el español, y la columna de náhuatl como lo vimos en la imagen llamada Archivo.csv 1, la normalización de esto se debe a que debemos primero quitar caracteres especiales y dejar exclusivamente texto, lo cual realizamos dentro del siguiente programa.

```

1  #Librerias para el funcionamiento del programa
2  import csv
3  import re
4  import unicodedata
5  #Librerias para el funcionamiento de la interfaz
6  import sys
7  import tkinter as tk
8  from tkinter import scrolledtext
9
10 #Codigo del programa
11 def normalizar_texto(texto):
12     """
13     Normaliza el texto realizando las siguientes operaciones:
14     1. Convierte a minúsculas
15     2. Elimina acentos y caracteres diacríticos
16     3. Elimina signos de puntuación
17     4. Elimina espacios extra
18     5. Elimina caracteres especiales
19     """
20     if not isinstance(texto, str):
21         return ""
22
23     # Paso 1: Convertir a minúsculas
24     texto = texto.lower()
25
26     # Paso 2: Eliminar acentos y caracteres diacríticos
27     texto = ''.join(c for c in unicodedata.normalize('NFD', texto)
28                     if unicodedata.category(c) != 'Mn')
29
30     # Paso 3: Eliminar signos de puntuación (conserva letras, números y espacios)
31     texto = re.sub(r'^\w\s', '', texto)
32
33     # Paso 4: Eliminar espacios extra
34     texto = ' '.join(texto.split())
35
36     return texto
37

```

NORMALIZACION 1

```

38 def procesar_csv(archivo_entrada, archivo_salida):
39     """
40     Procesa el archivo CSV de entrada y genera un nuevo archivo CSV con el texto normalizado
41     """
42     try:
43         with open(archivo_entrada, mode='r', encoding='latin-1',) as infile, \
44             open(archivo_salida, mode='w', encoding='utf-8', newline='') as outfile:
45

```

NORMALIZACION 2

```

46 # Leer el archivo CSV
47 lector = csv.reader(infile)
48 escritor = csv.writer(outfile)
49
50 # Escribir encabezados
51 encabezados = next(lector, None)
52 if encabezados:
53     escritor.writerow([f"{encabezados[0]}_normalizado", f"{encabezados[1]}_normalizado"])
54
55 # Procesar cada fila
56 for fila in lector:
57     if len(fila) >= 2:
58         espanol_normalizado = normalizar_texto(fila[0])
59         nahuatl_normalizado = normalizar_texto(fila[1])
60         escritor.writerow([espanol_normalizado, nahuatl_normalizado])
61     else:
62         print(f"Advertencia: Fila con formato incorrecto: {fila}")
63
64 print(f"Procesamiento completado. \nResultados guardados en: {archivo_salida}")
65
66 except FileNotFoundError:
67     print(f"Error: No se pudo encontrar el archivo {archivo_entrada}")
68 except Exception as e:
69     print(f"Error inesperado: {str(e)}")

```

NORMALIZACIÓN 3

Para la interfaz realizamos un pequeño código que se pudiera reutilizar y utilizarlo en los siguientes programas sin realizar muchos cambios.

```

71 #Codigo de la interfaz
72 class TerminalOutputGUI:
73     def __init__(self):
74         self.root = tk.Tk()
75         self.root.title("Salida de Terminal")
76         self.root.geometry("800x600")
77
78         self.output_area = scrolledtext.ScrolledText(self.root, wrap=tk.WORD)
79         self.output_area.pack(fill=tk.BOTH, expand=True)
80         self.output_area.configure(bg='black', fg='white', font=('Consolas', 10))
81
82         self.original_stdout = sys.stdout
83         sys.stdout = self
84
85     def write(self, text):
86         self.output_area.insert(tk.END, text)
87         self.output_area.see(tk.END)
88
89     def flush(self):
90         pass
91
92     def start(self):
93         self.root.mainloop()
94         sys.stdout = self.original_stdout
95

```

NORMALIZACIÓN 4

```

96 def ejecutar_con_interfaz(func):
97     """Decorador para ejecutar una función con la interfaz gráfica"""
98     def wrapper(*args, **kwargs):
99         gui = TerminalOutputGUI()
100         try:
101             func(*args, **kwargs)
102         except Exception as e:
103             print(f"Error: {str(e)}")
104         gui.start()
105     return wrapper
106
107 @ejecutar_con_interfaz
108 def mi_programa():
109     print("Iniciando programa...")
110     #
111     procesar_csv(archivo_entrada, archivo_salida)
112     print("Puede consultar en archivo generado dentro del proyecto")
113

```

NORMALIZACIÓN 5

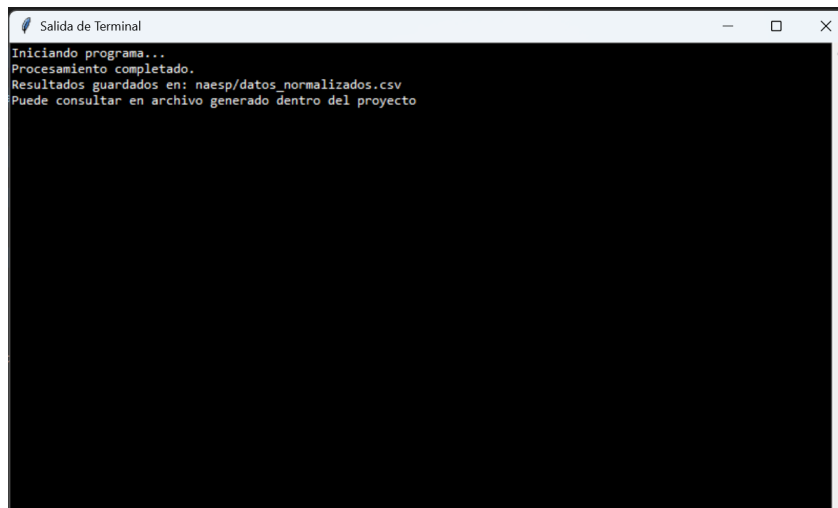
```

114 # Ejemplo de uso
115 if __name__ == "__main__":
116     archivo_entrada = "naesp/traduc.csv"
117     archivo_salida = "naesp/datos_normalizados.csv"
118
119     mi_programa()

```

NORMALIZACIÓN 6

Al ejecutarlo obtenemos como resultado lo siguiente:



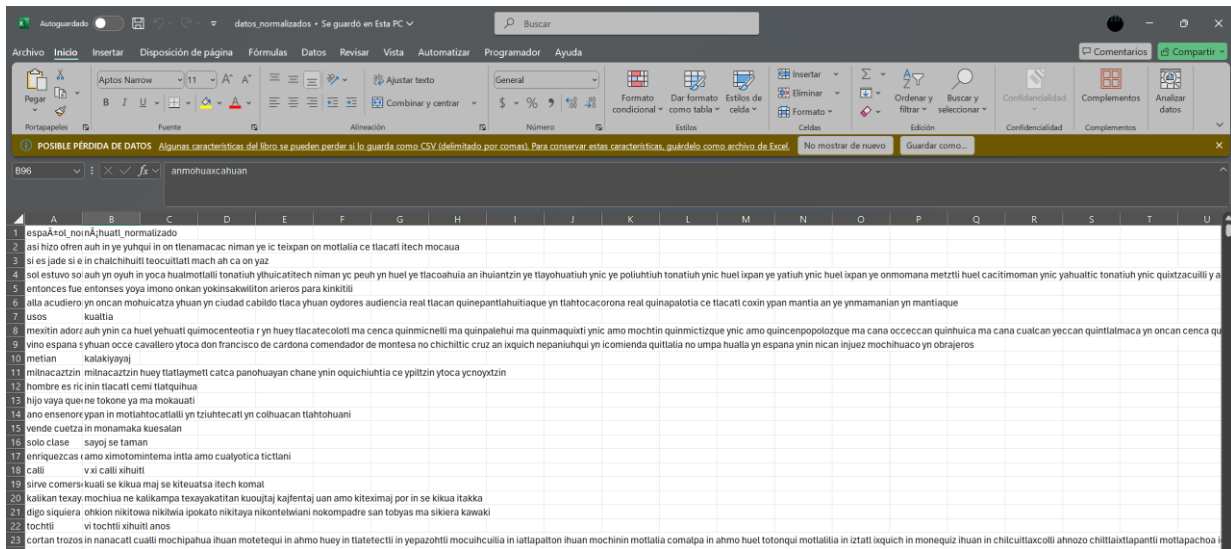
```

Salida de Terminal
Iniciando programa...
Procesamiento completado.
Resultados guardados en: naesp/datos_normalizados.csv
Puede consultar en archivo generado dentro del proyecto

```

NORMALIZACIÓN 7

Lo cual genera otro archivo .csv que contiene los datos normalizados como lo podemos ver dentro de la imagen de Archivo.csv 2.



ARCHIVO .CSV 2

Segmentador

Para el programa de segmentador realizamos un programa el cual toma nuestro Archivo .csv generado con anterioridad, una vez que este ya está normalizado ya podemos segmentar en oraciones para tener un mejor proceso del texto, finalmente genera un archivo .csv de igual manera posee una interfaz que nos mostrara lo que hace.

```
1 #Librerías para el funcionamiento del programa
2 import csv
3 #Librerías para el funcionamiento de la interfaz
4 import sys
5 import tkinter as tk
6 from tkinter import scrolledtext
7
8 #Codigo funcionamiento del programa
9 def segmentar_oraciones(texto):
10     """Segmenta un texto en oraciones basándose en signos de puntuación."""
11     if not texto or not isinstance(texto, str):
12         return []
13
14     oraciones = []
15     inicio = 0
16
17     # Caracteres que pueden terminar una oración
18     terminadores = ('.', '?', '!', ';')
19
```

SEGMENTADOR 1

```

20     for i, caracter in enumerate(texto):
21         if caracter in terminadores:
22             # Verificar que no sea parte de una abreviación (como "EE.UU.")
23             if i + 1 < len(texto) and texto[i+1].isspace():
24                 oracion = texto[inicio:i+1].strip()
25                 if oracion: # Evitar oraciones vacías
26                     oraciones.append(oracion)
27                 inicio = i + 1
28
29     # Añadir el último fragmento si queda texto restante
30     ultima_oracion = texto[inicio:].strip()
31     if ultima_oracion:
32         oraciones.append(ultima_oracion)
33
34     return oraciones
35
36 def procesar_csv(archivo_entrada, archivo_salida):
37     """Procesa el archivo CSV de entrada y genera uno nuevo con oraciones segmentadas."""
38     with open(archivo_entrada, mode='r', encoding='utf-8') as infile:
39         lector = csv.DictReader(infile)
40
41         # Mostrar las columnas disponibles para depuración
42         columnas = lector.fieldnames
43         print(f"Columnas encontradas en el CSV: {columnas}")
44
45         # Limpiar nombres de columnas (quitar espacios)
46         columnas_limpias = [col.strip() for col in columnas]
47
48         # Intentar encontrar las columnas correctas
49         col_espanol = None
50         col_nahuatl = None
51
52         for col in columnas_limpias:
53             if 'español' in col.lower() or 'espanol' in col.lower():
54                 col_espanol = col
55             elif 'náhuatl' in col.lower() or 'nahuatl' in col.lower():
56                 col_nahuatl = col
57
58         if not col_espanol or not col_nahuatl:
59             print("Error: No se pudieron encontrar las columnas de español y náhuatl")
60             print("Columnas disponibles:")
61             for i, col in enumerate(columnas):
62                 print(f" {i}: '{col}'")
63         return
64

```

SEGMENTADOR 2

```

65     print(f"Usando columna español: '{col_espanol}'")
66     print(f"Usando columna náhuatl: '{col_nahuatl}'")
67
68     # Procesar el archivo
69     with open(archivo_entrada, mode='r', encoding='utf-8') as infile, \
70         open(archivo_salida, mode='w', encoding='utf-8', newline='') as outfile:
71
72         lector = csv.DictReader(infile)
73         escritor = csv.writer(outfile)
74
75         # Escribir encabezados
76         escritor.writerow(['Idioma', 'Texto', 'Oracion_ID'])
77
78         oracion_id = 1
79

```

SEGMENTADOR 3

```

80         for fila in lector:
81             try:
82                 # Procesar español
83                 texto_espanol = fila.get(col_espanol, '')
84                 oraciones_es = segmentar_oraciones(texto_espanol)
85                 for oracion in oraciones_es:
86                     escritor.writerow(['español', oracion, f'oracion_{oracion_id}'])
87                     oracion_id += 1
88
89                 # Procesar náhuatl
90                 texto_nahuatl = fila.get(col_nahuatl, '')
91                 oraciones_na = segmentar_oraciones(texto_nahuatl)
92                 for oracion in oraciones_na:
93                     escritor.writerow(['náhuatl', oracion, f'oracion_{oracion_id}'])
94                     oracion_id += 1
95
96             except Exception as e:
97                 print(f"Error procesando fila: {e}")
98                 continue
99
100 #Codigo funcionamiento de la interfaz
101 class TerminalOutputGUI:
102     def __init__(self):
103         self.root = tk.Tk()
104         self.root.title("Salida de Terminal")
105         self.root.geometry("800x600")
106
107         self.output_area = scrolledtext.ScrolledText(self.root, wrap=tk.WORD)
108         self.output_area.pack(fill=tk.BOTH, expand=True)
109         self.output_area.configure(bg='black', fg='white', font=('Consolas', 10))
110
111         self.original_stdout = sys.stdout
112         sys.stdout = self
113
114     def write(self, text):
115         self.output_area.insert(tk.END, text)
116         self.output_area.see(tk.END)
117
118     def flush(self):
119         pass
120
121     def start(self):
122         self.root.mainloop()
123         sys.stdout = self.original_stdout
124

```

SEGMENTADOR 4

```

125 def ejecutar_con_interfaz(func):
126     """Decorador para ejecutar una función con la interfaz gráfica"""
127     def wrapper(*args, **kwargs):
128         gui = TerminalOutputGUI()
129         try:
130             func(*args, **kwargs)
131         except Exception as e:
132             print(f"Error: {str(e)}")
133         gui.start()
134     return wrapper

```

SEGMENTADOR 5

```

136 @ejecutar_con_interfaz
137 def mi_programa():
138     print("Iniciando programa...")
139     #
140     try:
141         procesar_csv(archivo_entrada, archivo_salida)
142         print(f"Procesamiento completado. Resultados guardados en {archivo_salida}")
143     except FileNotFoundError:
144         print("Error: No se encontró el archivo de entrada.")
145     except Exception as e:
146         print(f'Ocurrió un error: {str(e)}')
147
148 if __name__ == "__main__":
149     archivo_entrada = "naesp/datos_normalizados.csv"
150     archivo_salida = "naesp/oraci.csv"
151
152     mi_programa()

```

SEGMENTADOR 6

```

Salida de Terminal
Iniciando programa...
Columnas encontradas en el CSV: ['español_normalizado', 'náhuatl_normalizado']
Usando columna español: 'español_normalizado'
Usando columna náhuatl: 'náhuatl_normalizado'
Procesamiento completado. Resultados guardados en naesp/oraci.csv

```

SEGMENTADOR 7

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Idioma	Texto	Oracion_ID												
2	español	asi hizo ofren	oracion_1												
3	náhuatl	auh in ye yuh	oracion_2												
4	español	si es jade si e	oracion_3												
5	náhuatl	in chatchihuitl	oracion_4												
6	español	sot estuvo so	oracion_5												
7	náhuatl	auh yn oyuh i	oracion_6												
8	español	entonces fue	oracion_7												
9	náhuatl	entonces yoy	oracion_8												
10	español	alla acudiero	oracion_9												
11	náhuatl	yn oncan mol	oracion_10												
12	español	usos	oracion_11												
13	náhuatl	kuattia	oracion_12												
14	español	mexitin ador	oracion_13												
15	náhuatl	auh ynin ca h	oracion_14												
16	español	vino espana s	oracion_15												
17	náhuatl	yhuan occe c	oracion_16												
18	español	metian	oracion_17												

ARCHIVO .CSV 3

N-Gramas

Dentro del siguiente programa lo que vamos a hacer es la extracción de N-Gramas de cada una de las oraciones que teníamos en el Archivo .csv 3, para genera un nuevo archivo con los N-gramas de las oraciones, recordemos que los N-Gramas es una secuencia contigua de n elementos, generalmente palabras, de un texto dado. Un n-grama permite analizar patrones y relaciones dentro de un texto al considerar grupos de palabras consecutivas. Los modelos de n-gramas son útiles para diversas tareas como análisis de sentimiento, clasificación de texto y generación de texto.

```
1  #Librerías funcionamiento del programa
2  import csv
3  #Librerías para el funcionamiento de la interfaz
4  import sys
5  import tkinter as tk
6  from tkinter import scrolledtext
7
8  #Codigo funcionamiento del programa
9  def generar_ngramas(texto, n):
10     """Genera n-gramas de un texto dado."""
11     palabras = texto.split()
12     ngramas = []
13     for i in range(len(palabras) - n + 1):
14         ngrama = ' '.join(palabras[i:i+n])
15         ngramas.append(ngrama)
16     return ngramas
17
18  def determinar_n(texto):
19     """Determina el tamaño del n-grama basado en la longitud del texto."""
20     palabras = texto.split()
21     if len(palabras) <= 1:
22         return 1
23     elif len(palabras) <= 3:
24         return 2
25     elif len(palabras) <= 5:
26         return 3
27     else:
28         return 4 # Puedes ajustar estos valores según tus necesidades
29
30  def procesar_csv(archivo_entrada, archivo_salida):
31     """Procesa el archivo CSV y genera el archivo de salida con n-gramas."""
32     with open(archivo_entrada, mode='r', encoding='utf-8') as infile, \
33         open(archivo_salida, mode='w', encoding='utf-8', newline='') as outfile:
34
35         reader = csv.DictReader(infile)
36         fieldnames = ['Idioma', 'Oracion_ID', 'Texto_Original', 'N_Grama', 'Ngrama_Texto']
37         writer = csv.DictWriter(outfile, fieldnames=fieldnames)
38         writer.writeheader()
```

N-GRAMA 1

```

40         for fila in reader:
41             idioma = fila['Idioma']
42             texto = fila['Texto']
43             oracion_id = fila['Oracion_ID']
44
45             n = determinar_n(texto)
46             ngramas = generar_ngramas(texto, n)
47
48             for ngrama in ngramas:
49                 writer.writerow({
50                     'Idioma': idioma,
51                     'Oracion_ID': oracion_id,
52                     'Texto_Original': texto,
53                     'N_Grama': n,
54                     'Ngrama_Texto': ngrama
55                 })
56
57 #Codigo funcionamiento de la interfaz
58 class TerminalOutputGUI:
59     def __init__(self):
60         self.root = tk.Tk()
61         self.root.title("Salida de Terminal")
62         self.root.geometry("800x600")
63
64         self.output_area = scrolledtext.ScrolledText(self.root, wrap=tk.WORD)
65         self.output_area.pack(fill=tk.BOTH, expand=True)
66         self.output_area.configure(bg='black', fg='white', font=('Consolas', 10))
67
68         self.original_stdout = sys.stdout
69         sys.stdout = self
70
71     def write(self, text):
72         self.output_area.insert(tk.END, text)
73         self.output_area.see(tk.END)
74
75     def flush(self):
76         pass
77
78     def start(self):
79         self.root.mainloop()
80         sys.stdout = self.original_stdout
81

```

N-GRAMA 2

```

82 def ejecutar_con_interfaz(func):
83     """Decorador para ejecutar una función con la interfaz gráfica"""
84     def wrapper(*args, **kwargs):
85         gui = TerminalOutputGUI()
86         try:
87             func(*args, **kwargs)
88         except Exception as e:
89             print(f"Error: {str(e)}")
90         gui.start()
91     return wrapper

```

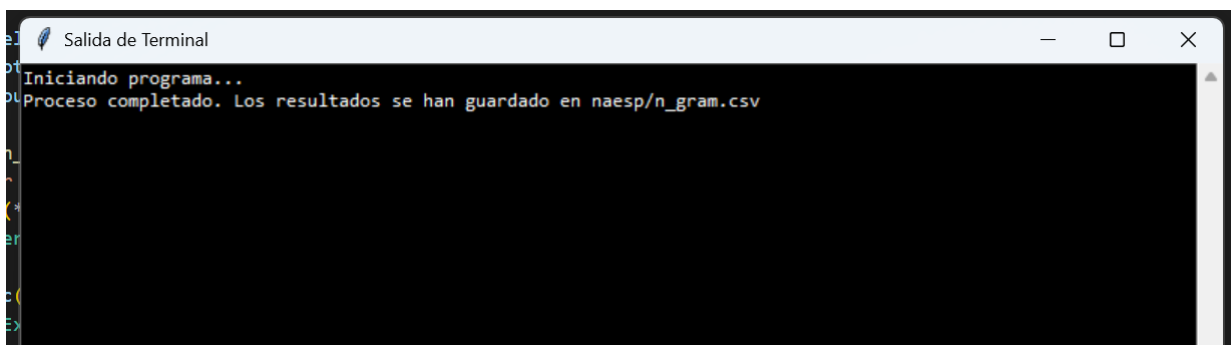
N-GRAMA 3

```

93 @ejecutar_con_interfaz
94 def mi_programa():
95     print("Iniciando programa...")
96     #
97     procesar_csv(archivo_entrada, archivo_salida)
98     print(f"Proceso completado. Los resultados se han guardado en {archivo_salida}")
99
100 if __name__ == "__main__":
101     archivo_entrada = "naesp/oraci.csv"
102     archivo_salida = "naesp/n_gram.csv"
103
104     mi_programa()

```

N-GRAMA 4



N-GRAMA 5

Autoguardado: n_gr... Se guardó en Est...

Archivos: Inicio Insertar Disposición de página Fórmulas Datos Revisar Vista Automatizar Programador Ayuda

Comentarios Compartir

POSIBLE PÉRDIDA DE DATOS: Algunas características del libro se pueden perder si lo guarda como CSV (delimitado por comas). Para conservar estas características, guárdelo como archivo de Excel.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Idioma	Oracion_ID	Texto_Origen	N_Grama	Ngrama_Texto										
español	oracion_1	asi hizo ofren	4	asi hizo ofrenda fuego										
español	oracion_1	asi hizo ofren	4	hizo ofrenda fuego sienta										
español	oracion_1	asi hizo ofren	4	ofrenda fuego sienta delante										
español	oracion_1	asi hizo ofren	4	fuego sienta delante demas										
español	oracion_1	asi hizo ofren	4	sienta delante demas persona										
español	oracion_1	asi hizo ofren	4	delante demas persona queda										
español	oracion_1	asi hizo ofren	4	demas persona queda junto										
náhuatl	oracion_2	auh in ye yuh	4	auh in ye yuhqui										
náhuatl	oracion_2	auh in ye yuh	4	in ye yuhqui in										
náhuatl	oracion_2	auh in ye yuh	4	ye yuhqui in on										
náhuatl	oracion_2	auh in ye yuh	4	yuhqui in on tlenamacac										
náhuatl	oracion_2	auh in ye yuh	4	in on tlenamacac niman										
náhuatl	oracion_2	auh in ye yuh	4	on tlenamacac niman ye										
náhuatl	oracion_2	auh in ye yuh	4	tlenamacac niman ye ic										
náhuatl	oracion_2	auh in ye yuh	4	niman ye ic teixpan										
náhuatl	oracion_2	auh in ye yuh	4	ye ic teixpan on										
náhuatl	oracion_2	auh in ye yuh	4	ic teixpan on mottalia										
náhuatl	oracion_2	auh in ye yuh	4	teixpan on mottalia ce										

n_gram

ARCHIVO .CSV 4

Patrones Gramaticales

Lo siguiente es encontrar los patrones gramaticales los cuales nos ayudaran a examinar nuestras oraciones, recordemos que los patrones gramaticales son las estructuras y secuencias de palabras que siguen las oraciones en un idioma. Estas estructuras son fundamentales para la construcción de oraciones significativas y coherentes. Los patrones gramaticales abarcan el orden de las palabras, las relaciones entre las palabras y las diferentes combinaciones de frases y cláusulas dentro de una oración; una vez ejecutado el programa analizara los patrones gramaticales que se encuentran dentro de las oraciones y nos mostrara 3 graficas sobre estos, para este programa como tal no utilizamos una interfaz y solo mostramos lo que son las gráficas.

```
1  import csv
2  import re
3  import matplotlib.pyplot as plt
4  from collections import defaultdict
5
6  class AnalizadorGramatical:
7      """
8      Clase para analizar patrones gramaticales en oraciones en español.
9      Clasifica palabras en categorías gramaticales y identifica patrones frecuentes.
10     """
11
12     def __init__(self):
13         """
14         Inicializa el analizador con diccionarios para almacenar patrones
15         y listas de palabras categorizadas por tipo gramatical.
16         """
17         # Diccionario para contar la frecuencia de cada patrón gramatical
18         self.patrones = defaultdict(int)
19
20         # Diccionario anidado: longitud -> patrón -> frecuencia
21         # Permite analizar patrones según el número de palabras
22         self.patrones_por_longitud = defaultdict(lambda: defaultdict(int))
23
24         # Listas de palabras categorizadas por tipo gramatical
25         self.articulos = ['el', 'la', 'los', 'las', 'un', 'una', 'unos', 'unas']
26
27         self.preposiciones = ['a', 'al', 'ante', 'bajo', 'cabe', 'con', 'contra', 'de', 'del',
28                               'desde', 'durante', 'en', 'entre', 'hacia', 'hasta', 'mediante',
29                               'para', 'por', 'según', 'sin', 'so', 'sobre', 'tras']
30
31         self.conjunciones = ['y', 'e', 'ni', 'que', 'o', 'u', 'pero', 'aunque', 'mas', 'sino']
32
33         self.pronombres = ['yo', 'tú', 'él', 'ella', 'usted', 'nosotros', 'vosotros',
34                             'ellos', 'ellas', 'ustedes', 'me', 'te', 'se', 'nos', 'os']
35
```

PATRONES GRAMATICALES 1

```

36 def determinar_categoria(self, palabra):
37     """
38     Determina la categoría gramatical de una palabra usando:
39     1. Listas predefinidas para palabras funcionales
40     2. Expresiones regulares para patrones morfológicos
41
42     Args:
43     |     palabra (str): La palabra a categorizar
44
45     Returns:
46     |     str: Código de la categoría gramatical
47     """
48     palabra = palabra.lower()
49
50     # Verificar primero las listas de palabras funcionales (más precisas)
51     if palabra in self.articulos:
52         return 'Articulo' # Artículos: el, la, un, una, etc.
53     elif palabra in self.preposiciones:
54         return 'Preposi' # Preposiciones: a, de, en, por, etc.
55     elif palabra in self.conjunciones:
56         return 'Conjci' # Conjunciones: y, o, pero, aunque, etc.
57     elif palabra in self.pronombres:
58         return 'Pronombr' # Pronombres: yo, tú, él, me, te, se, etc.
59
60     # Usar expresiones regulares para identificar formas verbales y otras categorías
61     elif re.match(r'^(*ando|*iendo|*endo)$', palabra):
62         return 'Gerun' # Gerundios: cantando, corriendo, viviendo
63     elif re.match(r'^(*es|*as|*os|*amos|*emos|*imos|*asteis|*isteis|*aron|*i',
64     |     return 'V_Preter' # Verbos en pretérito: cantaste, corrimos, vivieron
65     elif re.match(r'^(*é|*í|*aste|*iste|*ó|*ió|*imos|*isteis|*eron)$', palabra):
66     |     return 'V_Preteri' # Más formas de pretérito: canté, corrí, vivió
67     elif re.match(r'^(*aba|*abas|*ábamos|*abais|*aban|*ía|*ías|*íamos|*íais|',
68     |     return 'V_Imper' # Verbos en imperfecto: cantaba, corría, vivía
69     elif re.match(r'^(*aré|*eré|*iré|*arás|*erás|*irás|*aremos|*eremos|*iremo',
70     |     return 'V_Futur' # Verbos en futuro: cantaré, correré, viviré
71     elif re.match(r'^(*aría|*ería|*iría|*arías|*erías|*irías|*aríamos|*eríamos',
72     |     return 'V_Condic' # Verbos en condicional: cantaría, correría, viviría
73     elif re.match(r'^(*o|*as|*a|*amos|*áis|*an|*es|*e|*emos|*éis|*en)$', palabra):
74     |     return 'V_Present' # Verbos en presente: canto, corres, vive
75     elif re.match(r'^(*ad[oa]s?|*ed[oa]s?|*id[oa]s?)$', palabra):
76     |     return 'Adjet' # Adjetivos: cansado, verde, rápido

```

PATRONES GRAMATICALES 2

```

77     elif re.match(r'^(*[aeo]s?|*es|*[aeo]s)$', palabra):
78     |     return 'Sustant' # Sustantivos: casa, perro, ciudad
79     elif re.match(r'^[0-9]+$', palabra):
80     |     return 'Num' # Números: 1, 23, 456
81     else:
82     |     return 'Otro' # Cualquier palabra que no entre en las categorías anteriores
83

```

PATRONES GRAMATICALES 3

```

84     def analizar_oracion(self, oracion):
85         """
86         Analiza una oración completa y extrae su patrón gramatical.
87
88         Args:
89         |     oracion (str): La oración a analizar
90
91         Returns:
92         |     str: El patrón gramatical como string (ej: "Articulo Sust V_Pres")
93         """
94         # Extraer solo las palabras (letras y números) ignorando puntuación
95         palabras = re.findall(r'\b\w+\b', oracion.lower())
96
97         # Lista para almacenar las categorías de cada palabra
98         patron = []
99
100        # Clasificar cada palabra y construir el patrón
101        for palabra in palabras:
102            categoria = self.determinar_categoria(palabra)
103            patron.append(categoria)
104
105        # Convertir la lista de categorías en un string
106        patron_str = ' '.join(patron)
107
108        # Incrementar el contador para este patrón
109        self.patrones[patron_str] += 1
110
111        # También contar el patrón según su longitud
112        longitud = len(patron)
113        self.patrones_por_longitud[longitud][patron_str] += 1
114
115        return patron_str
116
117    def cargar_csv(self, archivo):
118        """
119        Carga un archivo CSV con oraciones y analiza solo las oraciones en español.
120
121        Args:
122        |     archivo (str): Ruta del archivo CSV
123        """
124        with open(archivo, mode='r', encoding='utf-8') as file:
125            lector = csv.DictReader(file)
126
127            # Procesar cada fila del CSV
128            for fila in lector:
129                # Solo analizar oraciones en español

```

PATRONES GRAMATICALES 4

```

129                # Solo analizar oraciones en español
130                if fila['Idioma'].lower() == 'español' or fila['Idioma'].lower() == 'spanish':
131                    self.analizar_oracion(fila['Texto'])
132

```

PATRONES GRAMATICALES 5

```

133 def graficar_patrones_comunes(self, top_n=10):
134     """
135     Genera un gráfico de barras horizontales con los patrones más comunes.
136
137     Args:
138         top_n (int): Número de patrones más frecuentes a mostrar
139     """
140     # Ordenar patrones por frecuencia (mayor a menor) y tomar los top_n
141     patrones_ordenados = sorted(self.patrones.items(), key=lambda x: x[1], reverse=True)[:top_n]
142
143     # Separar patrones y sus frecuencias
144     patrones = [p[0] for p in patrones_ordenados]
145     frecuencias = [p[1] for p in patrones_ordenados]
146
147     # Crear el gráfico
148     plt.figure(figsize=(12, 6))
149     plt.barh(patrones, frecuencias) # Barras horizontales
150     plt.xlabel('Frecuencia')
151     plt.ylabel('Patrón gramatical')
152     plt.title('Patrones gramaticales más comunes')
153     plt.gca().invert_yaxis() # Invertir el eje Y para mostrar el más frecuente arriba
154     plt.tight_layout() # Ajustar el layout para que no se corten las etiquetas
155     plt.show()
156
157 def graficar_distribucion_longitudes(self):
158     """
159     Genera un gráfico de barras que muestra la distribución de oraciones
160     según la cantidad de palabras que contienen.
161     """
162     # Obtener las longitudes ordenadas
163     longitudes = sorted(self.patrones_por_longitud.keys())
164
165     # Contar cuántas oraciones hay de cada longitud
166     conteos = [sum(self.patrones_por_longitud[l].values()) for l in longitudes]
167
168     # Crear el gráfico
169     plt.figure(figsize=(10, 5))
170     plt.bar(longitudes, conteos)
171     plt.xlabel('Longitud del patrón (en palabras)')
172     plt.ylabel('Número de oraciones')
173     plt.title('Distribución de longitudes de patrones gramaticales')
174     plt.show()
175

```

PATRONES GRAMATICALES 6

```

176 def graficar_patrones_por_longitud(self, longitud):
177     """
178     Genera un gráfico de los patrones más comunes para una longitud específica.
179
180     Args:
181         longitud (int): Número de palabras de los patrones a analizar
182     """
183     # Verificar que existan patrones de esa longitud
184     if longitud not in self.patrones_por_longitud:
185         print(f"No hay oraciones con patrones de longitud {longitud}")
186         return
187
188     # Obtener los patrones de la longitud especificada
189     patrones = self.patrones_por_longitud[longitud]
190
191     # Ordenar por frecuencia y tomar los 10 más comunes
192     patrones_ordenados = sorted(patrones.items(), key=lambda x: x[1], reverse=True)[:10]
193
194     # Verificar que hay patrones para mostrar
195     if not patrones_ordenados:
196         return
197
198     # Separar etiquetas y valores
199     etiquetas = [p[0] for p in patrones_ordenados]
200     valores = [p[1] for p in patrones_ordenados]
201
202     # Crear el gráfico
203     plt.figure(figsize=(12, 6))
204     plt.barh(etiquetas, valores)
205     plt.xlabel('Frecuencia')
206     plt.ylabel('Patrón gramatical')
207     plt.title(f'Patrones gramaticales más comunes para longitud {longitud}')
208     plt.gca().invert_yaxis()
209     plt.tight_layout()
210     plt.show()

```

PATRONES GRAMATICALES 7

```

213 if __name__ == "__main__":
214     """
215     Sección principal que demuestra cómo usar el analizador:
216     1. Crear una instancia del analizador
217     2. Cargar datos desde un archivo CSV
218     3. Generar diferentes tipos de gráficos
219     """
220
221     # Crear el analizador
222     analizador = AnalizadorGramatical()
223
224     # Cargar el archivo CSV
225     archivo_csv = 'naesp/oraci.csv'
226     analizador.cargar_csv(archivo_csv)
227
228     # Generar gráficos
229     print("Generando gráficos...")
230
231     # Gráfico 1: Los 10 patrones más comunes en general
232     analizador.graficar_patrones_comunes()
233
234     # Gráfico 2: Distribución de longitudes (cuántas palabras tienen las oraciones)
235     analizador.graficar_distribucion_longitudes()
236
237     # Gráfico 3: Patrones más comunes para oraciones de 5 palabras
238     analizador.graficar_patrones_por_longitud(5)

```

PATRONES GRAMATICALES 8

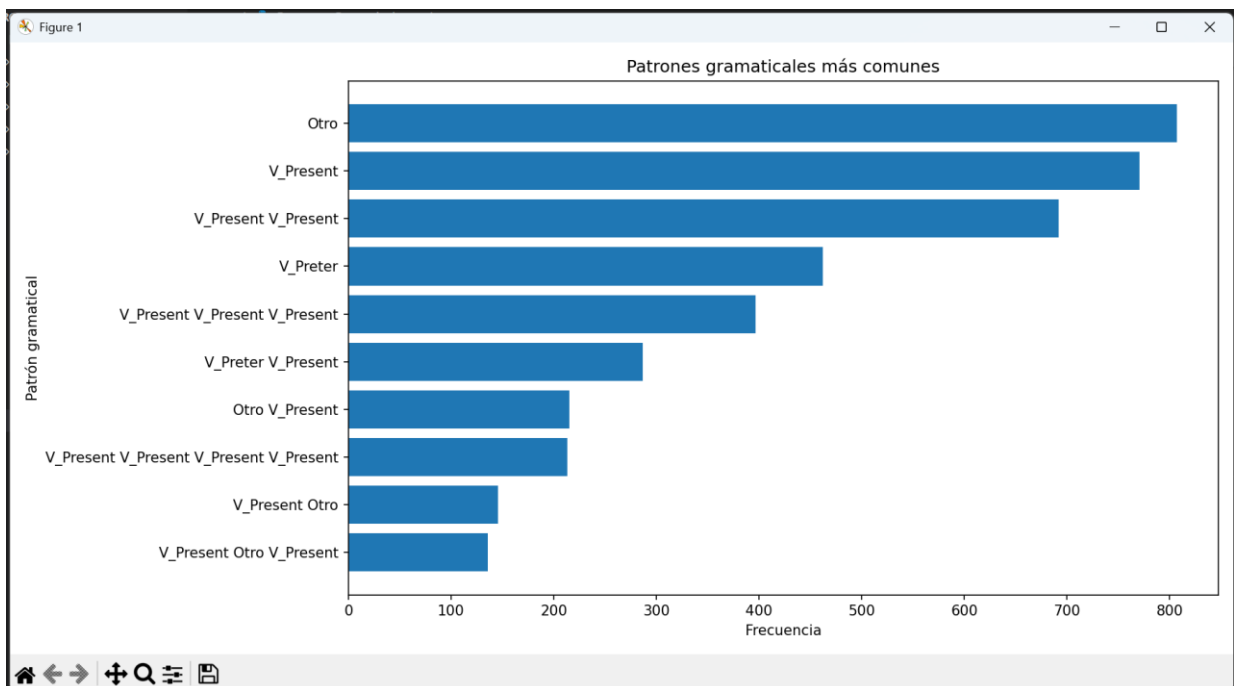


GRÁFICO 1

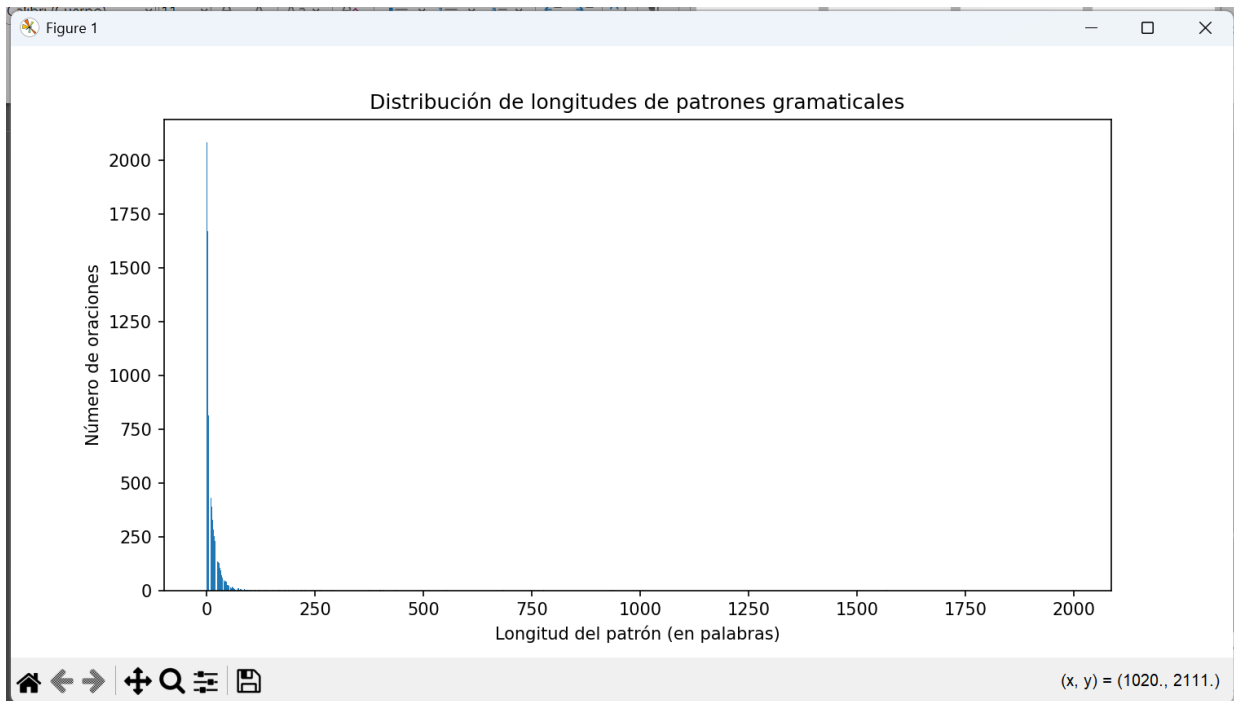


GRÁFICO 2

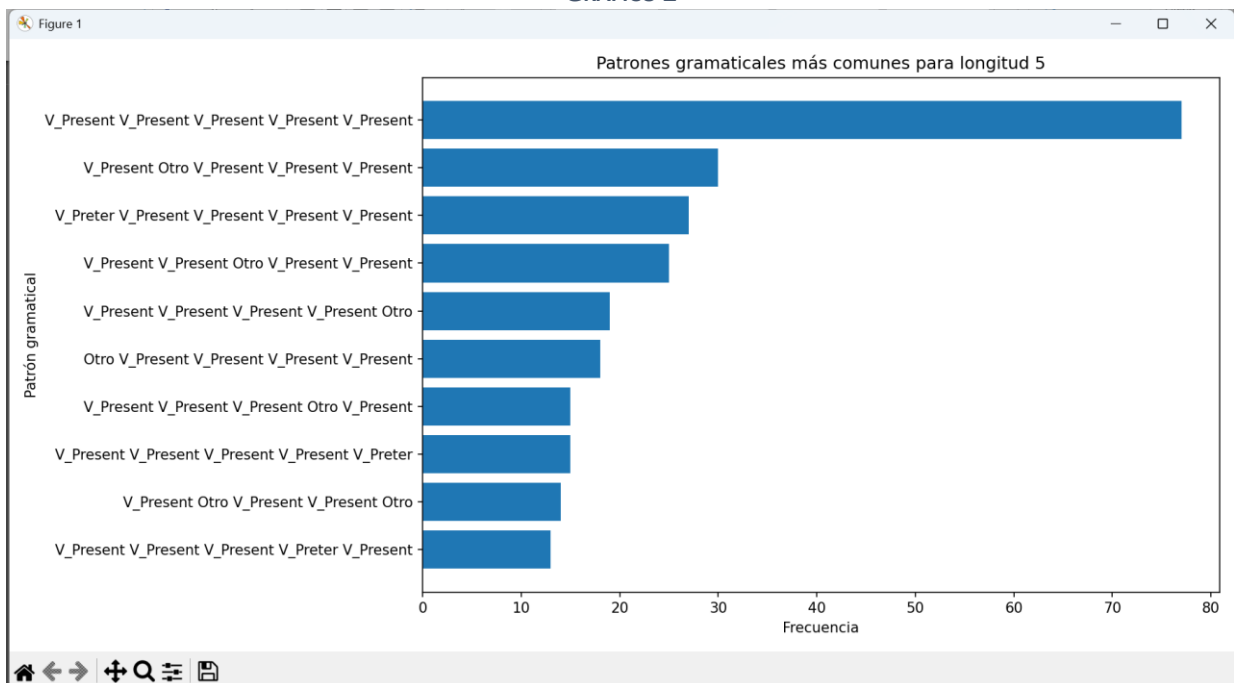


GRÁFICO 3

Estas 3 graficas lo que nos muestran es lo encontrado en las oraciones analizadas del documento .csv y cada uno nos dice que tipo de palabras encontró, para este programa creamos nuestras listas, ya que no utilizamos algún otro tipo de biblioteca ya existente.

Detección de errores

Para el programa de detección de errores esta vez hicimos el uso SpaCy para poder analizar las oraciones, pero utilizaremos un diccionario propio creado para el análisis de esto, de igual manera tenemos 2 programas uno que utilizamos y creamos, digamos es un poco mas casero y realizado, el otro ocupa librerías que harán de manera más robusta la evaluación de los errores, de igual manera al final genera un gráfico.

SpaCy es una biblioteca de código abierto escrita en Python y Cython para procesamiento avanzado del lenguaje natural (PLN). Está diseñada para la producción y ayuda a los usuarios a comprender grandes volúmenes de texto de manera eficiente. Ofrece herramientas para tareas como tokenización, análisis de dependencias, reconocimiento de entidades nombradas y más, siendo popular en aplicaciones de PLN.

```
1  #Librerías de funcionamiento del programa
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import spacy
5  import re
6  from collections import Counter
7  #Librerías para el funcionamiento de la interfaz
8  import sys
9  import tkinter as tk
10 from tkinter import scrolledtext
11
12 #Codigo funcionamiento del programa
13 def verificar_dependencias():
14     """Verifica que las dependencias básicas estén instaladas"""
15     try:
16         import spacy
17         import pandas as pd
18         import matplotlib.pyplot as plt
19         print("✓ Dependencias básicas instaladas")
20         return True
21     except ImportError as e:
22         print(f"X Error de importación: {e}")
23         print("\nPara instalar las dependencias básicas, ejecuta:")
24         print("pip install pandas matplotlib spacy")
25         print("python -m spacy download es_core_news_sm")
26         return False
27
28 def cargar_modelo_spacy():
29     """Carga el modelo de spaCy con manejo de errores"""
30     try:
31         nlp = spacy.load("es_core_news_sm")
32         print("✓ Modelo de spaCy cargado correctamente")
33         return nlp
34     except OSError:
35         print("X Error: No se encontró el modelo de spaCy en español")
36         print("Para instalarlo, ejecuta:")
37         print("python -m spacy download es_core_news_sm")
38         return None
39
40 def cargar_datos(archivo_csv):
41     """Carga el archivo CSV y devuelve un DataFrame"""
42     try:
43         datos = pd.read_csv(archivo_csv)
```

```

40 def cargar_datos(archivo_csv):
44     print(f"✓ Archivo {archivo_csv} cargado correctamente")
45     print(f" - Filas: {len(datos)}")
46     print(f" - Columnas: {list(datos.columns)}")
47     return datos
48 except FileNotFoundError:
49     print(f"X Error: No se encontró el archivo {archivo_csv}")
50     print("Asegúrate de que el archivo existe y la ruta es correcta")
51     return None
52 except Exception as e:
53     print(f"X Error al cargar el archivo: {e}")
54     return None
55
56 def filtrar_espanol(datos):
57     """Filtra las oraciones en español"""
58     if datos is None:
59         return None
60
61     # Verificar que exista la columna 'Idioma'
62     if 'Idioma' not in datos.columns:
63         print("X Error: No se encontró la columna 'Idioma' en el CSV")
64         print(f"Columnas disponibles: {list(datos.columns)}")
65         return None
66
67     # Filtrar oraciones en español
68     datos_espanol = datos[datos['Idioma'].str.lower() == 'español']
69     print(f"✓ Filtradas {len(datos_espanol)} oraciones en español")
70
71     return datos_espanol
72
73 def crear_diccionario_basico():
74     """Crea un diccionario básico de palabras comunes en español"""
75     # Palabras comunes en español (se puede expandir)
76     palabras_comunes = {
77         'el', 'la', 'de', 'que', 'y', 'a', 'en', 'un', 'es', 'se', 'no', 'te', 'lo', 'le',
78         'da', 'su', 'por', 'son', 'con', 'para', 'al', 'del', 'los', 'las', 'una', 'como',
79         'pero', 'sus', 'muy', 'sin', 'sobre', 'también', 'me', 'hasta', 'hay', 'donde',
80         'han', 'quien', 'están', 'estado', 'desde', 'todo', 'nos', 'durante', 'estados',
81         'todos', 'uno', 'les', 'ni', 'contra', 'otros', 'fueron', 'ese', 'eso', 'había',
82         'antes', 'sí', 'mismo', 'como', 'año', 'años', 'después', 'vida', 'tiempo',
83         'día', 'días', 'casa', 'nombre', 'hombre', 'mujer', 'mundo', 'país', 'trabajo',
84         'parte', 'forma', 'lugar', 'caso', 'grupo', 'problema', 'hecho', 'momento',
85         'manera', 'sistema', 'programa', 'ciudad', 'gobierno', 'partido', 'empresa',
86         'agua', 'mes', 'meses', 'noche', 'historia', 'hora', 'horas', 'gente', 'niño',
87         'niña', 'niños', 'niñas', 'estado', 'desarrollo', 'proceso', 'uso', 'nivel',
88         'servicio', 'información', 'ejemplo', 'situación', 'resultado', 'precio',
89         'producto', 'mercado', 'centro', 'proyecto', 'modelo', 'condición', 'persona',

```

DETECCION_ERRORES_V1 2

```

115     'entran', 'trabajar', 'escribir', 'perder', 'producir', 'ocurrir', 'entender',
116     'pedir', 'recibir', 'recordar', 'terminar', 'permitir', 'aparecer', 'conseguir',
117     'comenzar', 'servir', 'sacar', 'necesitar', 'mantener', 'resultar', 'leer',
118     'caer', 'cambiar', 'presentar', 'crear', 'abrir', 'considerar', 'oír',
119     'puede', 'pueden', 'debe', 'deben', 'tiene', 'tienen', 'hace', 'hacen',
120     'dice', 'dicen', 'va', 'van', 'viene', 'vienen', 'sale', 'salen',
121     'entra', 'entran', 'vive', 'viven', 'trabaja', 'trabajan', 'estudia',
122     'estudian', 'juega', 'juegan', 'come', 'comen', 'bebe', 'beben',
123     'duerme', 'duermen', 'corre', 'corren', 'camina', 'caminan', 'habla',
124     'hablan', 'escucha', 'escuchan', 'mira', 'miran', 'lee', 'leen',
125     'escribe', 'escriben', 'compra', 'compran', 'vende', 'venden'
126 }
127
128 return palabras_comunes

```

DETECCION_ERRORES_V1 3

```

130 def detectar_errores_ortograficos_basico(oracion, diccionario):
131     """Detecta errores ortográficos básicos usando un diccionario simple"""
132     if not oracion or pd.isna(oracion):
133         return 0
134
135     # Limpiar la oración y extraer palabras
136     oracion_limpia = re.sub(r'^\w\s', '', oracion.lower())
137     palabras = oracion_limpia.split()
138
139     # Contar palabras no encontradas en el diccionario
140     errores = 0
141     for palabra in palabras:
142         if palabra and len(palabra) > 2: # Ignorar palabras muy cortas
143             if palabra not in diccionario:
144                 errores += 1
145
146     return errores
147
148 def detectar_errores_gramaticales(oracion, nlp):
149     """Detecta errores gramaticales básicos usando spaCy"""
150     if not oracion or pd.isna(oracion) or nlp is None:
151         return 0
152
153     try:
154         doc = nlp(oracion)
155         errores = 0
156
157         # Verificar concordancia básica entre sustantivos y adjetivos
158         for token in doc:
159             if token.pos_ == 'ADJ' and token.head.pos_ == 'NOUN':
160                 # Verificar concordancia de género y número
161                 token_gender = token.morph.get('Gender')
162                 token_number = token.morph.get('Number')
163                 head_gender = token.head.morph.get('Gender')
164                 head_number = token.head.morph.get('Number')
165
166                 if (token_gender and head_gender and token_gender != head_gender) or \
167                     (token_number and head_number and token_number != head_number):
168                     errores += 1
169
170         # Verificar estructura básica de la oración
171         # Buscar verbos principales
172         verbos_principales = [token for token in doc if token.pos_ == 'VERB' and token.dep_ == 'ROOT']
173         if not verbos_principales and len(doc) > 3: # Oraciones largas sin verbo principal
174             errores += 1
175
176     return errores

```

DETECCION_ERRORES_V1 4

```

178     except Exception as e:
179         print(f"Error en análisis gramatical: {e}")
180         return 0
181
182 def detectar_errores(oracion, nlp, diccionario):
183     """Detecta errores gramaticales y ortográficos en una oración"""
184     errores = {
185         'ortograficos': 0,
186         'gramaticales': 0,
187         'total': 0
188     }
189
190     # Detectar errores ortográficos
191     errores['ortograficos'] = detectar_errores_ortograficos_basico(oracion, diccionario)
192
193     # Detectar errores gramaticales
194     errores['gramaticales'] = detectar_errores_gramaticales(oracion, nlp)
195
196     errores['total'] = errores['ortograficos'] + errores['gramaticales']
197
198     return errores

```

DETECCION_ERRORES_V1 5

```

200 def analizar_oraciones(datos_espanol, nlp):
201     """Analiza todas las oraciones en español y devuelve resultados"""
202     if datos_espanol is None:
203         return None
204
205     # Verificar que existan las columnas necesarias
206     columnas_requeridas = ['Texto', 'Oracion_ID']
207     for col in columnas_requeridas:
208         if col not in datos_espanol.columns:
209             print(f"X Error: No se encontró la columna '{col}' en el CSV")
210             return None
211
212     # Crear diccionario básico
213     diccionario = crear_diccionario_basico()
214     print(f"✓ Diccionario creado con {len(diccionario)} palabras")
215
216     resultados = []
217     total_oraciones = len(datos_espanol)
218
219     print(f"Analizando {total_oraciones} oraciones...")
220
221     for i, (_, fila) in enumerate(datos_espanol.iterrows()):
222         if i % 100 == 0: # Mostrar progreso cada 100 oraciones
223             print(f"Procesando oración {i+1}/{total_oraciones}")

```

DETECCION_ERRORES_V1 6

```

225     oracion = fila['Texto']
226     oracion_id = fila['Oracion_ID']
227     errores = detectar_errores(oracion, nlp, diccionario)
228
229     resultados.append({
230         'Oracion_ID': oracion_id,
231         'Texto': oracion,
232         'Errores_Ortograficos': errores['ortograficos'],
233         'Errores_Gramaticales': errores['gramaticales'],
234         'Total_Errores': errores['total']
235     })
236
237     print("✓ Análisis completado")
238     return pd.DataFrame(resultados)

```

DETECCION_ERRORES_V1 7

```

240 def generar_grafico(resultados):
241     """Genera un gráfico de barras con los resultados"""
242     if resultados is None or len(resultados) == 0:
243         print("X No hay resultados para graficar")
244         return
245
246     try:
247         # Preparar datos para el gráfico
248         datos_grafico = resultados[['Errores_Ortograficos', 'Errores_Gramaticales']].sum()
249
250         # Crear gráfico
251         plt.figure(figsize=(12, 7))
252         colores = ['#FF6B6B', '#4ECDC4']
253         bars = plt.bar(datos_grafico.index, datos_grafico.values, color=colores,
254                        edgecolor='black', linewidth=1.2, alpha=0.8)
255
256         plt.title('Distribución de Errores en Oraciones en Español',
257                  fontsize=16, fontweight='bold', pad=20)
258         plt.xlabel('Tipo de Error', fontsize=12, fontweight='bold')
259         plt.ylabel('Cantidad de Errores', fontsize=12, fontweight='bold')
260         plt.xticks(rotation=0)
261         plt.grid(axis='y', linestyle='--', alpha=0.7)

```

DETECCION_ERRORES_V1 8

```

263         # Añadir valores en las barras
264         for bar, value in zip(bars, datos_grafico.values):
265             plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.5,
266                     str(value), ha='center', va='bottom', fontweight='bold', fontsize=11)
267
268         plt.tight_layout()
269         plt.show()
270
271         print("✓ Gráfico generado exitosamente")
272
273     except Exception as e:
274         print(f"X Error al generar el gráfico: {e}")
275
276 def generar_reporte_estadisticas(resultados):
277     """Genera un reporte con estadísticas descriptivas"""
278     if resultados is None or len(resultados) == 0:
279         return
280
281     print("\n" + "="*60)
282     print("REPORTE DE ESTADÍSTICAS - ANÁLISIS DE ERRORES")
283     print("="*60)
284
285     total_oraciones = len(resultados)
286     oraciones_con_errores = len(resultados[resultados['Total_Errores'] > 0])

```

DETECCION_ERRORES_V1 9

```

286     oraciones_con_errores = len(resultados[resultados['Total_Errores'] > 0])
287     oraciones_sin_errores = total_oraciones - oraciones_con_errores
288
289     print(f"📄 RESUMEN GENERAL:")
290     print(f"    Total de oraciones analizadas: {total_oraciones}")
291     print(f"    Oraciones con errores: {oraciones_con_errores} ({oraciones_con_errores/total_oraciones*100:.1f}%)")
292     print(f"    Oraciones sin errores: {oraciones_sin_errores} ({oraciones_sin_errores/total_oraciones*100:.1f}%)")
293
294     print(f"\n📄 ERRORES ORTOGRÁFICOS:")
295     print(f"    Total: {resultados['Errores_Ortograficos'].sum()}")
296     print(f"    Promedio por oración: {resultados['Errores_Ortograficos'].mean():.2f}")
297     print(f"    Máximo en una oración: {resultados['Errores_Ortograficos'].max()}")
298
299     print(f"\n📄 ERRORES GRAMATICALES:")
300     print(f"    Total: {resultados['Errores_Gramaticales'].sum()}")
301     print(f"    Promedio por oración: {resultados['Errores_Gramaticales'].mean():.2f}")
302     print(f"    Máximo en una oración: {resultados['Errores_Gramaticales'].max()}")
303
304     # Mostrar las 5 oraciones con más errores
305     print(f"\n📄 TOP 5 ORACIONES CON MÁS ERRORES:")
306     top_errores = resultados.nlargest(5, 'Total_Errores')
307     for i, (_, fila) in enumerate(top_errores.iterrows(), 1):
308         print(f"    {i}. [{fila['Total_Errores']} errores] {fila['Texto'][:80]}...")
309

```

DETECCION_ERRORES_V1 10

```

310 def principal():
311     """Función principal"""
312     print("DETECTOR DE ERRORES GRAMATICALES Y ORTOGRÁFICOS")
313     print("="*60)
314
315     # Verificar dependencias
316     if not verificar_dependencias():
317         return
318
319     # Cargar modelo de spaCy
320     nlp = cargar_modelo_spacy()
321
322     # Nombre del archivo CSV
323     archivo_csv = 'naesp/oraci.csv'
324
325     # Procesamiento de datos
326     datos = cargar_datos(archivo_csv)
327     datos_espanol = filtrar_espanol(datos)
328     resultados = analizar_oraciones(datos_espanol, nlp)
329
330     if resultados is not None and len(resultados) > 0:
331         # Mostrar primeras filas de resultados
332         print("\n 📄 MUESTRA DE RESULTADOS (primeras 5 oraciones):")
333         print(resultados.head().to_string(index=False))
334
335         # Generar estadísticas
336         generar_reporte_estadisticas(resultados)
337
338         # Generar gráfico
339         generar_grafico(resultados)
340
341         # Guardar resultados en un nuevo CSV
342         try:
343             resultados.to_csv('naesp/resultados_errores.csv', index=False, encoding='utf-8')
344             print(f"\n ✅ Resultados guardados en 'naesp/resultados_errores.csv'")
345         except Exception as e:
346             print(f"❌ Error al guardar resultados: {e}")
347     else:
348         print("❌ No se encontraron oraciones en español para analizar.")
349
350     #Codigo funcionamiento de la interfaz
351     class TerminalOutputGUI:
352         def __init__(self):
353             self.root = tk.Tk()
354             self.root.title("Salida de Terminal")
355             self.root.geometry("1000x800")
356

```

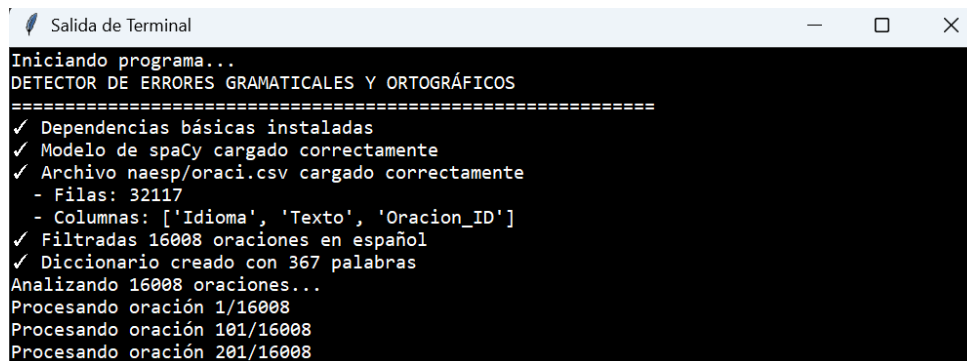
```

357         self.output_area = scrolledtext.ScrolledText(self.root, wrap=tk.WORD)
358         self.output_area.pack(fill=tk.BOTH, expand=True)
359         self.output_area.configure(bg='black', fg='white', font=('Consolas', 15))
360
361         self.original_stdout = sys.stdout
362         sys.stdout = self
363
364         def write(self, text):
365             self.output_area.insert(tk.END, text)
366             self.output_area.see(tk.END)
367
368         def flush(self):
369             pass
370
371         def start(self):
372             self.root.mainloop()
373             sys.stdout = self.original_stdout
374
375     def ejecutar_con_interfaz(func):
376         """Decorador para ejecutar una función con la interfaz gráfica"""
377         def wrapper(*args, **kwargs):
378             gui = TerminalOutputGUI()
379             try:
380                 func(*args, **kwargs)
381             except Exception as e:
382                 print(f"Error: {str(e)}")
383             gui.start()
384         return wrapper
385
386     @ejecutar_con_interfaz
387     def mi_programa():
388         print("Iniciando programa...")
389         #
390         principal()
391
392 if __name__ == "__main__":
393     mi_programa()

```

DETECCION_ERRORES_V1 12

Al ejecutar tenemos como resultado lo siguiente y vemos como es que ha examinado las oraciones y el grafico nos ayuda a saber de una mejor manera los errores encontrados:



```

Salida de Terminal
Iniciando programa...
DETECTOR DE ERRORES GRAMATICALES Y ORTOGRÁFICOS
=====
✓ Dependencias básicas instaladas
✓ Modelo de spaCy cargado correctamente
✓ Archivo naesp/oraci.csv cargado correctamente
  - Filas: 32117
  - Columnas: ['Idioma', 'Texto', 'Oracion_ID']
✓ Filtradas 16008 oraciones en español
✓ Diccionario creado con 367 palabras
Analizando 16008 oraciones...
Procesando oración 1/16008
Procesando oración 101/16008
Procesando oración 201/16008

```

DETECCION_ERRORES_V1 13

```

Procesando oración 16001/16008
✓ Análisis completado

MUESTRA DE RESULTADOS (primeras 5 oraciones):
Oración_ID      Errores_Ortográficos  Errores_Gramaticales  Total_Errores  Texto
oracion_1      0                    9                    9              así hizo ofrenda fuego sienta delante demas persona queda junto
oracion_3      0                    5                    5              si es jade si es oro acaso tendra ir alla
oracion_5      1                    24                   25              sol estuvo solo cielo enseguida comenzo amarillear fue oscureciendo sol desaparecio frente fue colocarse luna alcanzando cubrir disco sol asi lentamente desaparecio sol
oracion_7      1                    9                    10             entonces fue mono luego fue atajar arrieros quitarles
oracion_9      0                    16                   16             alla acudieron asimismo senores cabildo ciudad olores real audiencia pusieron enmedio hombre llevaba cojin corona real iban avanzando mismo tiempo

=====
REPORTE DE ESTADÍSTICAS - ANÁLISIS DE ERRORES
=====
RESUMEN GENERAL:
Total de oraciones analizadas: 16008
Oraciones con errores: 15923 (99.5%)
Oraciones sin errores: 85 (0.5%)

ERRORES ORTOGRÁFICOS:
Total: 183696
Promedio por oración: 11.48
Máximo en una oración: 1722

```

DETECCION_ERRORES_V1 14

```

ERRORES GRAMATICALES:
Total: 11644
Promedio por oración: 0.73
Máximo en una oración: 165

TOP 5 ORACIONES CON MÁS ERRORES:
1. [1887 errores] artículo toda persona tiene derecho trabajo digno socialmente util efecto promov...
2. [1739 errores] artículo propiedad tierras aguas comprendidas dentro límites territorio nacional...
3. [1604 errores] artículo definida artículo ordenamiento naturaleza jurídica distrito federal gob...
4. [1480 errores] artículo pueblo ejerce soberanía medio poderes union casos competencia estos est...
5. [1123 errores] artículo congreso tiene facultad i admitir nuevos estados union federal ii derog...

```

DETECCION_ERRORES_V1 15

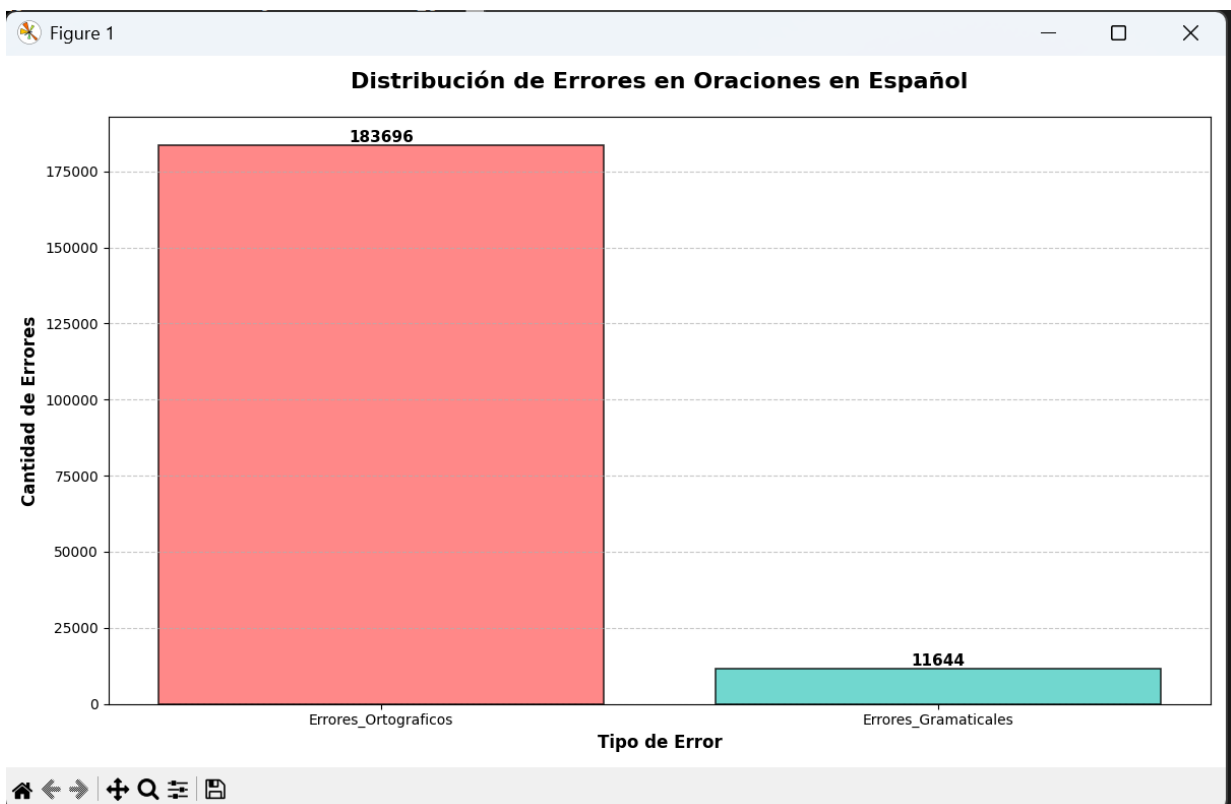


GRÁFICO 4

Detección de errores PNL

Para el siguiente programa realizamos lo mismo, pero esta vez utilizando bibliotecas más especializadas y cómo podemos ver el programa es más chico que el anterior utilizado, de igual manera utilizamos lo que es spellchecker, este último es una biblioteca que detecta errores ortográficos en un texto y sugiere correcciones. Funciona comparando las palabras escritas con un diccionario interno o una lista de términos aceptados. Si encuentra una palabra que no coincide, la marca como un error y, a menudo, ofrece opciones para reemplazarla con la forma correcta.

```
1  #Librerías funcionamiento del programa
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import spacy
5  from spellchecker import SpellChecker
6  #Librerías para el funcionamiento de la interfaz
7  import sys
8  import tkinter as tk
9  from tkinter import scrolledtext
10
11 # Cargar modelos de lenguaje
12 nlp = spacy.load("es_core_news_sm")
13 spell = SpellChecker(language='es')
14 #Codigo funcionamiento del programa
15 def cargar_datos(archivo_csv):
16     """Carga el archivo CSV y devuelve un DataFrame"""
17     try:
18         datos = pd.read_csv(archivo_csv)
19         return datos
20     except FileNotFoundError:
21         print(f"Error: No se encontró el archivo {archivo_csv}")
22         return None
23
24 def filtrar_espanol(datos):
25     """Filtra las oraciones en español"""
26     if datos is None:
27         return None
28     return datos[datos['Idioma'].str.lower() == 'español']
29
30 def detectar_errores(oracion):
31     """Detecta errores gramaticales y ortográficos en una oración"""
32     errores = {
33         'ortograficos': 0,
34         'gramaticales': 0,
35         'total': 0
36     }
37
38     # Verificación ortográfica
39     palabras = oracion.split()
40     mal_escritas = spell.unknown(palabras)
41     errores['ortograficos'] = len(mal_escritas)
42
```

```

43     # Análisis gramatical con spaCy
44     doc = nlp(oracion)
45
46     # Contar errores gramaticales básicos (esto es un enfoque simplificado)
47     gramaticales = 0
48     for token in doc:
49         if token.is_punct or token.is_space:
50             continue
51         # Verificar concordancia básica entre sustantivos y adjetivos
52         if token.pos_ == 'ADJ' and token.head.pos_ == 'NOUN':
53             # Verificar concordancia de género y número
54             if (token.morph.get('Gender') != token.head.morph.get('Gender') or
55                 token.morph.get('Number') != token.head.morph.get('Number')):
56                 gramaticales += 1
57
58     errores['gramaticales'] = gramaticales
59     errores['total'] = errores['ortograficos'] + errores['gramaticales']
60
61     return errores
62
63 def analizar_oraciones(datos_espanol):
64     """Analiza todas las oraciones en español y devuelve resultados"""
65     if datos_espanol is None:
66         return None
67
68     resultados = []
69     for _, fila in datos_espanol.iterrows():
70         oracion = fila['Texto']
71         oracion_id = fila['Oracion_ID']
72         errores = detectar_errores(oracion)
73         resultados.append({
74             'Oracion_ID': oracion_id,
75             'Texto': oracion,
76             'Errores_Ortograficos': errores['ortograficos'],
77             'Errores_Gramaticales': errores['gramaticales'],
78             'Total_Errores': errores['total']
79         })
80
81     return pd.DataFrame(resultados)
82
83 def generar_grafico(resultados):
84     """Genera un gráfico de barras con los resultados"""
85     if resultados is None:
86         return
87

```

```

88     # Preparar datos para el gráfico
89     datos_grafico = resultados[['Errores_Ortograficos', 'Errores_Gramaticales']].sum()
90
91     # Crear gráfico
92     plt.figure(figsize=(10, 6))
93     datos_grafico.plot(kind='bar', color=['red', 'blue'])
94     plt.title('Distribución de Errores en Oraciones en Español')
95     plt.xlabel('Tipo de Error')
96     plt.ylabel('Cantidad de Errores')
97     plt.xticks(rotation=0)
98     plt.grid(axis='y', linestyle='--', alpha=0.7)
99
100    # Mostrar gráfico
101    plt.tight_layout()
102    plt.show()
103
104    def principal():
105        # Nombre del archivo CSV
106        archivo_csv = 'naesp/oraci.csv'
107
108        # Procesamiento de datos
109        datos = cargar_datos(archivo_csv)
110        datos_espanol = filtrar_espanol(datos)
111        resultados = analizar_oraciones(datos_espanol)
112
113        if resultados is not None:
114            # Mostrar resultados
115            print("\nResultados del análisis:")
116            print(resultados.head())
117
118            # Generar gráfico
119            generar_grafico(resultados)
120
121            # Guardar resultados en un nuevo CSV
122            resultados.to_csv('naesp/resultados_errores_pnl.csv', index=False)
123            print("\nResultados guardados en 'naesp/resultados_errores_pnl.csv'")
124        else:
125            print("No se encontraron oraciones en español para analizar.")
126
127    #Codigo funcionamiento de la interfaz
128    class TerminalOutputGUI:
129        def __init__(self):
130            self.root = tk.Tk()
131            self.root.title("Salida de Terminal")
132            self.root.geometry("1000x800")
133

```

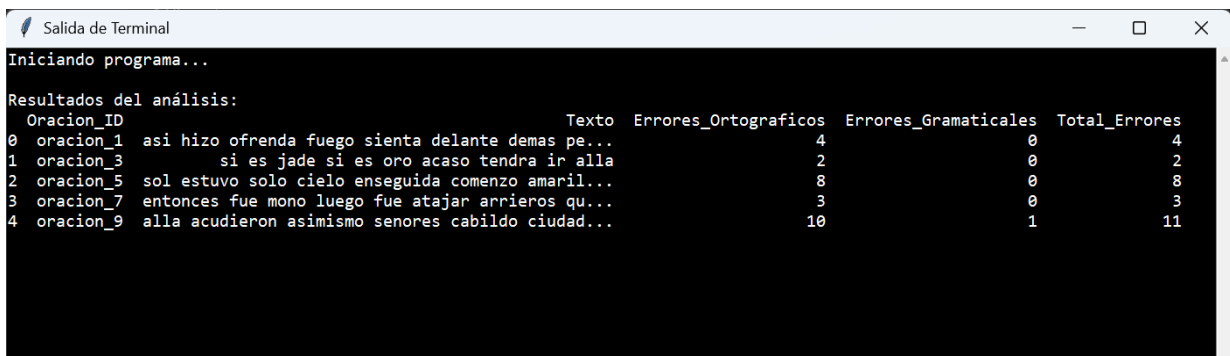
```

134         self.output_area = scrolledtext.ScrolledText(self.root, wrap=tk.WORD)
135         self.output_area.pack(fill=tk.BOTH, expand=True)
136         self.output_area.configure(bg='black', fg='white', font=('Consolas', 15))
137
138         self.original_stdout = sys.stdout
139         sys.stdout = self
140
141         def write(self, text):
142             self.output_area.insert(tk.END, text)
143             self.output_area.see(tk.END)
144
145         def flush(self):
146             pass
147
148         def start(self):
149             self.root.mainloop()
150             sys.stdout = self.original_stdout
151
152     def ejecutar_con_interfaz(func):
153         """Decorador para ejecutar una función con la interfaz gráfica"""
154         def wrapper(*args, **kwargs):
155             gui = TerminalOutputGUI()
156             try:
157                 func(*args, **kwargs)
158             except Exception as e:
159                 print(f"Error: {str(e)}")
160             gui.start()
161         return wrapper
162
163     @ejecutar_con_interfaz
164     def mi_programa():
165         print("Iniciando programa...")
166         #
167         principal()
168
169     if __name__ == "__main__":
170         mi_programa()

```

DETECCIÓN_ERRORES_v2 4

Al ejecutar tenemos los siguientes resultados, como podemos ver:



```

Salida de Terminal
Iniciando programa...

Resultados del análisis:

```

Oracion_ID	Texto	Errores_Ortograficos	Errores_Gramaticales	Total_Errores
0 oracion_1	asi hizo ofrenda fuego sienta delante demas pe...	4	0	4
1 oracion_3	si es jade si es oro acaso tendra ir alla	2	0	2
2 oracion_5	sol estuvo solo cielo enseguida comenzo amaril...	8	0	8
3 oracion_7	entonces fue mono luego fue atajar arrieros qu...	3	0	3
4 oracion_9	alla acudieron asimismo senores cabildo ciudad...	10	1	11

DETECCIÓN_ERRORES_v2 5

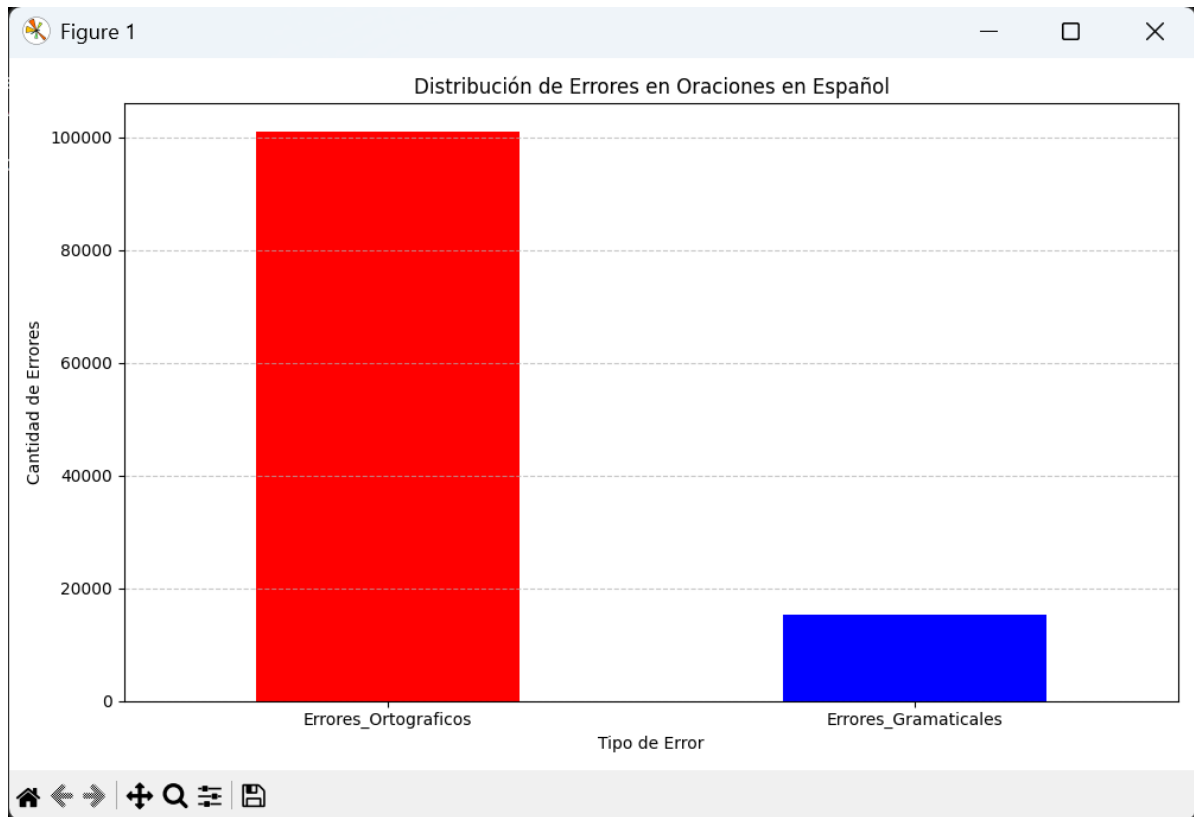


GRÁFICO 5

Autoguardado resultados_errores... • Se guardó en Esta PC

Buscar

Archivo Inicio Insertar Disposición de página Fórmulas Datos Revisar Vista Automatizar Programador Ayuda

Comentarios Compartir

Portapapeles Fuente Alineación Número Estilos Celdas Edición Confidencialidad Complementos Analizar datos

POSIBLE PÉRDIDA DE DATOS Algunas características del libro se pueden perder si lo guarda como CSV (delimitado por comas). Para conservar estas características, guárdelo como archivo de Excel. No mostrar de nuevo Guardar como...

A1 Oracion_ID

Oracion_ID	Texto	Errores_Orto	Errores_Gran	Total_Errores
oracion_1	asi hizo ofren	4	0	4
oracion_3	si es jade si e	2	0	2
oracion_5	sol estuvo so	8	0	8
oracion_7	entonces fue	3	0	3
oracion_9	alta acudiero	10	1	11
oracion_11	usos	1	0	1
oracion_13	mexitin adora	13	2	15
oracion_15	vino espana s	5	1	6
oracion_17	metian	1	0	1
oracion_19	mlinacaztzin	3	0	3
oracion_21	hombre es ric	0	0	0
oracion_23	hijo vaya que	0	0	0
oracion_25	ano ensenore	4	1	5
oracion_27	vende cuetza	2	0	2
oracion_29	solo clase	0	1	1
oracion_31	enriquezcas	2	1	3
oracion_33	catli	1	0	1
oracion_35	sirve comers	3	0	3

resultados_errores_pnl

100%

ARCHIVO .CSV 5

Evaluación BLEU

Para el siguiente programa realizamos la evaluación de una de las traducciones del español y ver que tal, como ya lo habíamos mencionado íbamos a ocupar el método de evaluación que nos daba BLEU y vimos cómo es que funciona este sin mandarlo a llamar, ahora para este programa realizaremos lo que la evaluación BLEU, utilizando lo que es la biblioteca de nltk lo cual resume todo lo ya realizado anteriormente y genera una evaluación de una oración al azar del archivo de N-Gramas, lo que corresponde a la imagen **Archivo .csv 4**, dentro de este archivo ya tenemos los N-gramas realizados de todas las oraciones del primer archivo que se nos dio.

```
1  #Librerías funcionamiento del programa
2  import pandas as pd
3  import random
4  from nltk.translate.bleu_score import sentence_bleu
5  import nltk
6  #Librerías para el funcionamiento de la interfaz
7  import sys
8  import tkinter as tk
9  from tkinter import scrolledtext
10
11 # Descargar recursos necesarios de NLTK (solo primera vez)
12 nltk.download('punkt')
13 #Codigo funcionamiento del programa
14 def evaluar_bleu(texto_original, ngrama_texto, n):
15     """
16     Evalúa el puntaje BLEU entre el texto original y los n-gramas.
17
18     Args:
19         texto_original (str): Texto de referencia.
20         ngrama_texto (str): N-gramas a evaluar.
21         n (int): Número de gramas (1, 2, 3, etc.).
22
23     Returns:
24         float: Puntaje BLEU.
25     """
26     # Tokenizar el texto original como referencia
27     referencia = [nltk.word_tokenize(texto_original.lower())]
28
29     # Tokenizar los n-gramas como candidato
30     candidato = nltk.word_tokenize(ngrama_texto.lower())
31
32     # Calcular BLEU con los pesos según el n-grama
33     # Por ejemplo, para 2-gramas: (0.5, 0.5)
34     pesos = tuple([1.0/n] * n)
35
36     return sentence_bleu(referencia, candidato, weights=pesos)
37
38 def principal():
39     # Cargar el archivo CSV
40     try:
41         df = pd.read_csv('naesp/n_gram.csv')
42     except FileNotFoundError:
43         print("Error: No se encontró el archivo CSV.")
44         return
45
```

```

46 # Filtrar solo las oraciones en español
47 df_espanol = df[df['Idioma'].str.lower() == 'español']
48
49 if df_espanol.empty:
50     print("No se encontraron oraciones en español en el archivo.")
51     return
52
53 # Seleccionar un Oracion_ID aleatorio
54 oracion_id = random.choice(df_espanol['Oracion_ID'].unique())
55
56 # Obtener todos los registros para ese Oracion_ID
57 registros = df_espanol[df_espanol['Oracion_ID'] == oracion_id]
58
59 # Obtener el texto original (debería ser el mismo para todos los registros del mismo Oracion_ID)
60 texto_original = registros.iloc[0]['Texto_Original']
61
62 print(f"\nEvaluando Oracion_ID: {oracion_id}")
63 print(f"Texto original: {texto_original}\n")
64
65 # Evaluar cada n-grama para esta oración
66 for _, row in registros.iterrows():
67     n = row['N_Grama']
68     ngrama_texto = row['Ngrama_Texto']
69
70     # Calcular puntaje BLEU
71     puntaje_bleu = evaluar_bleu(texto_original, ngrama_texto, n)
72
73     print(f"{n}-grama: {ngrama_texto}")
74     print(f"Puntaje BLEU: {puntaje_bleu:.4f}\n")
75
76 #Codigo funcionamiento de la interfaz
77 class TerminalOutputGUI:
78     def __init__(self):
79         self.root = tk.Tk()
80         self.root.title("Salida de Terminal")
81         self.root.geometry("1000x800")
82
83         self.output_area = scrolledtext.ScrolledText(self.root, wrap=tk.WORD)
84         self.output_area.pack(fill=tk.BOTH, expand=True)
85         self.output_area.configure(bg='black', fg='white', font=('Consolas', 15))
86
87         self.original_stdout = sys.stdout
88         sys.stdout = self
89

```

EVALUACIÓN BLEU 2

```

90     def write(self, text):
91         self.output_area.insert(tk.END, text)
92         self.output_area.see(tk.END)
93
94     def flush(self):
95         pass
96
97     def start(self):
98         self.root.mainloop()
99         sys.stdout = self.original_stdout
100

```

EVALUACIÓN BLEU 3

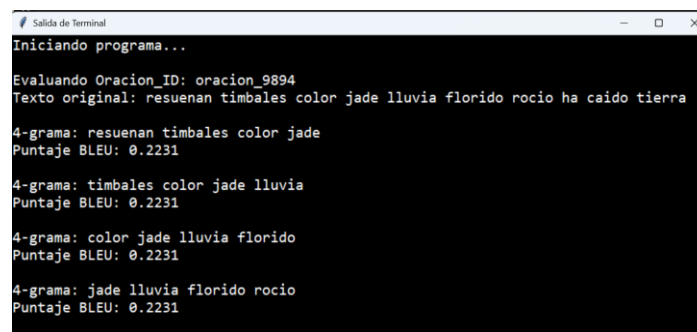
```

101 def ejecutar_con_interfaz(func):
102     """Decorador para ejecutar una función con la interfaz gráfica"""
103     def wrapper(*args, **kwargs):
104         gui = TerminalOutputGUI()
105         try:
106             func(*args, **kwargs)
107         except Exception as e:
108             print(f"Error: {str(e)}")
109         gui.start()
110     return wrapper
111
112 @ejecutar_con_interfaz
113 def mi_programa():
114     print("Iniciando programa...")
115     #
116     principal()
117
118 if __name__ == "__main__":
119     mi_programa()

```

EVALUACIÓN BLEU 4

Al ejecutar la primera vez como se ve en la imagen **Evaluación BLEU 5** vemos la puntuación que tiene asignada la oración, al ejecutar una segunda vez podemos observar en la imagen **Evaluación BLEU 6** que esta tiene una oración más entendible y que tiene una calificación de 1 lo que significa que la oración cumple con el estándar:

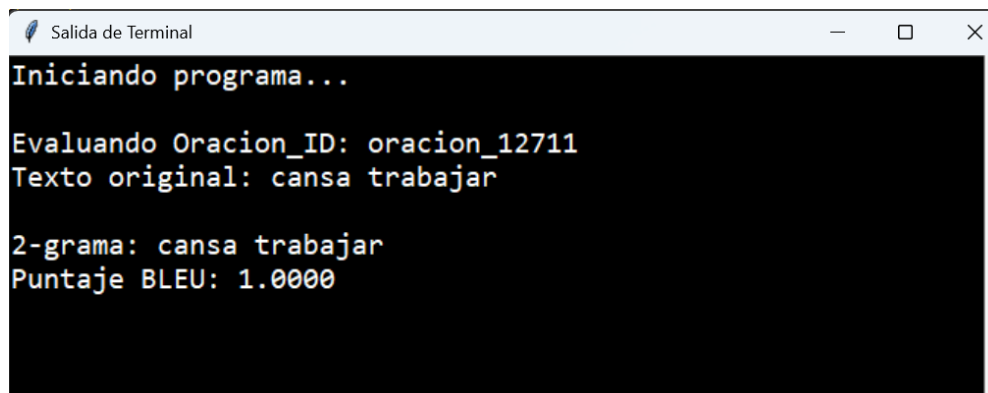


```

Salida de Terminal
Iniciando programa...
Evaluando Oracion_ID: oracion_9894
Texto original: resuenan timbales color jade lluvia florido rocío ha caído tierra
4-grama: resuenan timbales color jade
Puntaje BLEU: 0.2231
4-grama: timbales color jade lluvia
Puntaje BLEU: 0.2231
4-grama: color jade lluvia florido
Puntaje BLEU: 0.2231
4-grama: jade lluvia florido rocío
Puntaje BLEU: 0.2231

```

EVALUACIÓN BLEU 5



```

Salida de Terminal
Iniciando programa...
Evaluando Oracion_ID: oracion_12711
Texto original: cansa trabajar
2-grama: cansa trabajar
Puntaje BLEU: 1.0000

```

EVALUACIÓN BLEU 6

Pequeño traductor utilizando RNN (Red Neuronal Recurrente)

Para el siguiente y último programa que tenemos realizado es el uso de una red neuronal recurrente, esta red ocupamos más bibliotecas que tenemos a la mano y que fueron vistas dentro de la investigación de PNL realizada y presente dentro del documento, ocupamos esta RNN para crear un modelo que aprenda de las traducciones proporcionadas y generar un traductor de las palabras encontradas, también este programa al final genera un archivo .csv dentro del cual escribiremos la puntuación de cada una de las traducciones que este haya generado. Ocuparemos más bibliotecas las cuales ya fueron explicadas al inicio del documento.

```
1  import pandas as pd
2  import numpy as np
3  import tensorflow as tf
4  from keras.preprocessing.text import Tokenizer
5  from keras.preprocessing.sequence import pad_sequences
6  from keras.models import Sequential
7  from keras.layers import Embedding, SimpleRNN, Dense
8  import tkinter as tk
9  from tkinter import ttk, messagebox
10 import csv
11 import os
12 from datetime import datetime
13
14 # Configuración
15 CSV_TRAIN_FILE = 'naesp/traduc.csv'
16 CSV_FEEDBACK_FILE = 'naesp/retro_traducciones.csv'
17 MAX_VOCAB_SIZE = 10000
18 EMBEDDING_DIM = 256
19 RNN_UNITS = 512
20 MAX_LENGTH = 20
21 BATCH_SIZE = 32
22 EPOCHS = 10
23
24 # Preprocesamiento de datos
25 def load_and_preprocess_data(csv_file):
26     df = pd.read_csv(csv_file, encoding='latin-1')
27     español = df['español'].astype(str).tolist()
28     náhuatl = df['náhuatl'].astype(str).tolist()
29
30     # Tokenización español
31     español_tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE, filters='')
32     español_tokenizer.fit_on_texts(español)
33     español_sequences = español_tokenizer.texts_to_sequences(español)
34     español_data = pad_sequences(español_sequences, maxlen=MAX_LENGTH, padding='post')
35
36     # Tokenización náhuatl
37     náhuatl_tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE, filters='')
38     náhuatl_tokenizer.fit_on_texts(náhuatl)
39     náhuatl_sequences = náhuatl_tokenizer.texts_to_sequences(náhuatl)
40     náhuatl_data = pad_sequences(náhuatl_sequences, maxlen=MAX_LENGTH, padding='post')
41
42     return (español_data, español_tokenizer, náhuatl_data, náhuatl_tokenizer)
```

```

44 # Creación del modelo
45 def create_model(input_vocab_size, target_vocab_size):
46     model = Sequential([
47         Embedding(input_vocab_size, EMBEDDING_DIM, input_length=MAX_LENGTH),
48         SimpleRNN(RNN_UNITS, return_sequences=True),
49         Dense(target_vocab_size, activation='softmax')
50     ])
51
52     model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
53     return model
54
55 # Entrenamiento del modelo
56 def train_model(model, X_train, y_train):
57     model.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCHS, validation_split=0.2)
58     return model
59
60 # Traducción
61 def translate(model, sequence, español_tokenizer, náhuatl_tokenizer):
62     prediction = model.predict(sequence)
63     predicted_sequence = np.argmax(prediction, axis=-1)
64     translated_text = náhuatl_tokenizer.sequences_to_texts(predicted_sequence)
65     return translated_text[0]
66
67 # Interfaz gráfica
68 class TranslationApp:
69     def __init__(self, root, model, español_tokenizer, náhuatl_tokenizer):
70         self.root = root
71         self.model = model
72         self.español_tokenizer = español_tokenizer
73         self.náhuatl_tokenizer = náhuatl_tokenizer
74
75         root.title("Traductor Español-Náhuatl")
76
77         # Frame principal
78         main_frame = ttk.Frame(root, padding="10")
79         main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
80
81         # Entrada español
82         ttk.Label(main_frame, text="Español:").grid(row=0, column=0, sticky=tk.W)
83         self.entry_español = tk.Text(main_frame, width=50, height=5)
84         self.entry_español.grid(row=1, column=0, padx=5, pady=5)

```

RNN 2

```

86 # Entrada náhuatl (resultado)
87 ttk.Label(main_frame, text="Náhuatl:").grid(row=2, column=0, sticky=tk.W)
88 self.entry_náhuatl = tk.Text(main_frame, width=50, height=5, state='disabled')
89 self.entry_náhuatl.grid(row=3, column=0, padx=5, pady=5)
90
91 # Botones
92 btn_frame = ttk.Frame(main_frame)
93 btn_frame.grid(row=4, column=0, pady=10)
94
95 ttk.Button(btn_frame, text="Traducir", command=self.translate_text).pack(side=tk.LEFT, padx=5)
96 ttk.Button(btn_frame, text="Buena traducción", command=lambda: self.rate_translation(True)).pack(side=tk.LEFT, padx=5)
97 ttk.Button(btn_frame, text="Mala traducción", command=lambda: self.rate_translation(False)).pack(side=tk.LEFT, padx=5)
98
99 # Inicializar archivo de feedback si no existe
100 if not os.path.exists(str(CSV_FEEDBACK_FILE)):
101     with open(str(CSV_FEEDBACK_FILE), 'w', newline='', encoding='latin-1') as f:
102         writer = csv.writer(f)
103         writer.writerow(['fecha', 'texto_original', 'traduccion', 'calificacion'])
104

```

RNN 3

```

105     def translate_text(self):
106         text = self.entry_español.get("1.0", tk.END).strip()
107         if not text:
108             messagebox.showwarning("Advertencia", "Por favor ingrese un texto para traducir")
109             return
110
111         sequence = self.español_tokenizer.texts_to_sequences([text])
112         padded_sequence = pad_sequences(sequence, maxlen=MAX_LENGTH, padding='post')
113
114         translated_text = translate(self.model, padded_sequence, self.español_tokenizer, self.náhuatl_tokenizer)
115
116         self.entry_náhuatl.config(state='normal')
117         self.entry_náhuatl.delete("1.0", tk.END)
118         self.entry_náhuatl.insert("1.0", translated_text)
119         self.entry_náhuatl.config(state='disabled')
120
121         # Guardar la última traducción para calificación
122         self.last_translation = {
123             'original': text,
124             'translation': translated_text
125         }
126

```

RNN 4

```

127     def rate_translation(self, is_good):
128         if not hasattr(self, 'last_translation'):
129             messagebox.showwarning("Advertencia", "Primero realice una traducción para calificar")
130             return
131
132         rating = "buena" if is_good else "mala"
133
134         # Guardar en CSV de feedback
135         with open(str(CSV_FEEDBACK_FILE), 'a', newline='', encoding='latin-1') as f:
136             writer = csv.writer(f)
137             writer.writerow([
138                 datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
139                 self.last_translation['original'],
140                 self.last_translation['translation'],
141                 rating
142             ])
143
144         messagebox.showinfo("Gracias", f"¡Gracias por calificar esta traducción como {rating}!")
145
146     # Función principal
147     def main():
148         # Cargar y preprocesar datos
149         try:
150             español_data, español_tokenizer, náhuatl_data, náhuatl_tokenizer = load_and_preprocess_data(CSV_TRAIN_FILE)
151         except FileNotFoundError:
152             print(f"Error: No se encontró el archivo {CSV_TRAIN_FILE}")
153             return
154         except Exception as e:
155             print(f"Error al cargar los datos: {str(e)}")
156             return
157
158         # Crear y entrenar modelo
159         input_vocab_size = len(español_tokenizer.word_index) + 1
160         target_vocab_size = len(náhuatl_tokenizer.word_index) + 1
161
162         model = create_model(input_vocab_size, target_vocab_size)
163         model = train_model(model, español_data, náhuatl_data)
164
165         # Crear interfaz gráfica
166         root = tk.Tk()
167         app = TranslationApp(root, model, español_tokenizer, náhuatl_tokenizer)
168         root.mainloop()
169
170     if __name__ == "__main__":
171         main()

```

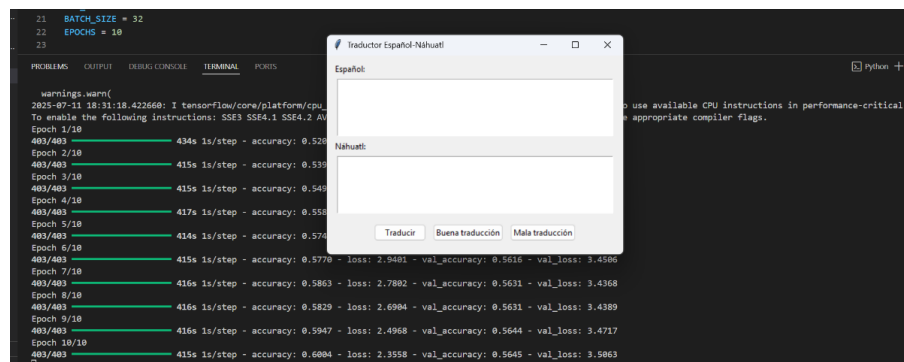
RNN 5

Al ejecutar tenemos notamos que tarda bastante más que todos los programas ya que este primero debe de entrenar al modelo para su uso como vemos en la imagen debe de entrenar 10 veces nosotros podemos decidir tanto el total de la muestra como el total de veces que entrene pero el proceso es muy tardado, pero al final nos genera un archivo .csv el cual nos muestra la traducción y el puntaje dado, es un traductor muy básico y para que funcione mejor necesitamos más recursos tanto de náhuatl a español y viceversa.

```
2025-07-11 18:31:10.251587: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due
ferent computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-07-11 18:31:13.052998: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due
ferent computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
C:\Users\elman\OneDrive\Escritorio\Proy-Naesp\naesp\Lib\site-packages\keras/src\layers\core\embedding.py:97: UserWarning: Argument 'input_length' is
warnings.warn(
2025-07-11 18:31:18.422660: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instruction
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/10
14/403 ----- 6:59 1s/step - accuracy: 0.4181 - loss: 9.5011
```

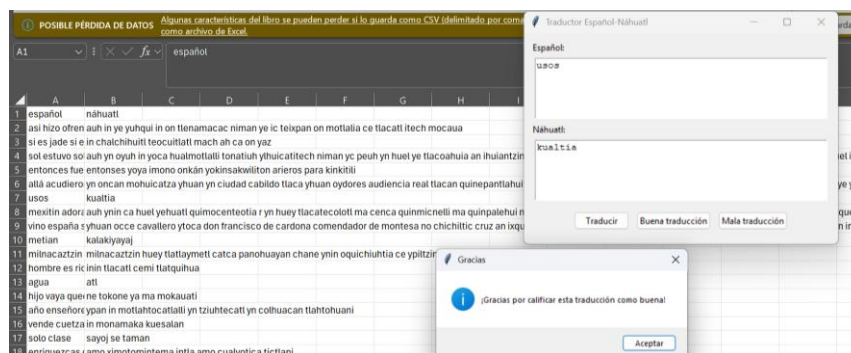
RNN 6

Una vez ejecutado vemos la siguiente interfaz, de igual manera vemos como es que han terminado de entrenarse las épocas, que son las veces que este ha entrenado.



RNN 7

Podemos escribir una palabra para ver si el traductor la encuentra o no, como en el siguiente ejemplo, también calificamos lo que fue la traducción de acuerdo con él .csv que tenemos y posteriormente dentro del archivo nuevo que genera tenemos lo siguiente.



ARCHIVO .CSV 6

	A	B	C	D	E	F
1	fecha	texto_origina	traduccion	calificacion		
2	#####	sol	in	mala		
3	#####	el dia	in	mala		
4	#####	nube	auh	mala		
5	#####	montaña	mahtactiome	mala		
6	#####	lugar	auh	mala		
7	#####	usos	kualtia	buena		
8	#####	hijo	yn	mala		
9						

ARCHIVO .CSV 7

Como se puede apreciar realice más traducciones pero no fueron o no cumplieron con lo solicitado, por lo cual podemos calificarlo, para una mejor precisión de las traducciones deberíamos de revisar todo el archivo .csv que se nos fue proporcionado y dar las palabras correctas ya que en un inicio cuando se empezó a trabajar con este, hubo perdida de información y cambio de ciertas palabras, por lo cual deberíamos de realizar un mayor trabajo de campo para unas mejores traducciones y en fases posteriores a esta poder generar un programa más robusto.

Ejecutable

Para la creación de este proyecto, solamente es una pequeña interfaz que contiene todos los demás programas realizados para poder ejecutarlos y no buscarlos dentro del proyecto.

```

1  import tkinter as tk
2  from tkinter import messagebox
3  import subprocess
4  import sys
5  import os
6
7  class ProgramLauncher:
8      def __init__(self, root):
9          self.root = root
10         self.root.title("Procesamiento de Lenguaje Natural")
11         self.root.geometry("1200x800")
12         self.root.resizable(False, False)
13
14         # Configuración de colores
15         bg_color = "#f0f0f0"
16         button_color = "#e1e1e1"
17         active_color = "#d1d1d1"
18
19         self.root.config(bg=bg_color)
20
21         # Título
22         title_label = tk.Label(
23             root,
24             text="Procesamiento de las transcripciones utilizando técnicas de PLN (Procesamiento de Lenguaje Natural)",
25             font=("Arial", 14, "bold"),
26             bg=bg_color
27         )
28         title_label.pack(pady=10)
29
30         # Frame para los botones
31         button_frame = tk.Frame(root, bg=bg_color)
32         button_frame.pack(expand=True)
33
34         # Botones
35         self.buttons = []
36         self.programs = [
37             ("Normalización", "naesp/Normalizacion.py"),
38             ("Oraciones", "naesp/Segmentador.py"),
39             ("N-gramas", "naesp/N_Gramas.py"),
40             ("Patrones gramaticales", "naesp/Patrones_Gramaticales.py"),
41             ("Deteccion errores", "naesp/Deteccion_errores.py"),
42             ("Deteccion errores PNL", "naesp/Deteccion_er_pnl.py"),
43             ("Evaluacion BLEU", "naesp/Evaluacion_BLEU.py"),
44             ("Traductor", "naesp/RNN_trad.py")
45         ]

```

EJECUTABLE 1

```

47 # Crear 8 botones en una cuadrícula de 2x4
48 for i, (name, file) in enumerate(self.programs):
49     row = i // 4
50     col = i % 4
51
52     btn = tk.Button(
53         button_frame,
54         text=name,
55         width=25,
56         height=3,
57         bg=button_color,
58         activebackground=active_color,
59         command=lambda f=file: self.run_program(f),
60         font=("Arial", 10)
61     )
62     btn.grid(row=row, column=col, padx=10, pady=10)
63     self.buttons.append(btn)
64
65 # Botón de salir
66 exit_button = tk.Button(
67     root,
68     text="Salir",
69     command=root.quit,
70     bg=█ "#ff9999",
71     activebackground=█ "#ff7777",
72     width=10,
73     font=("Arial", 10, "bold")
74 )
75 exit_button.pack(pady=10)
76
77 def run_program(self, program_file):
78     """Ejecuta el programa Python especificado"""
79     try:
80         # Verificar si el archivo existe
81         if not os.path.exists(program_file):
82             messagebox.showerror("Error", f"El archivo {program_file} no existe.")
83             return
84
85         # Ejecutar el programa usando el intérprete de Python actual
86         subprocess.Popen([sys.executable, program_file])
87
88     except Exception as e:
89         messagebox.showerror("Error", f"No se pudo ejecutar {program_file}:\n{str(e)}")
90

```

EJECUTABLE 2

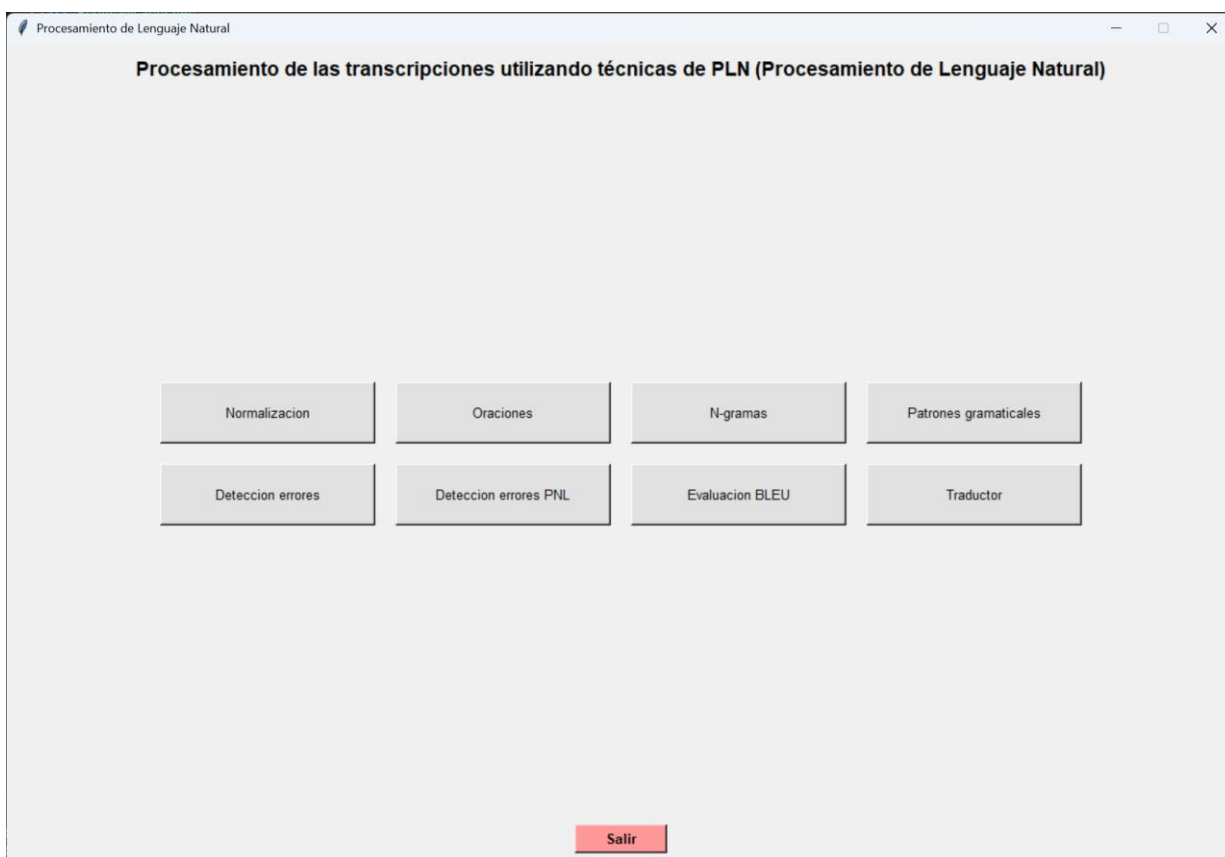
```

91
92 if __name__ == "__main__":
93     root = tk.Tk()
94     app = ProgramLauncher(root)
95     root.mainloop()

```

EJECUTABLE 3

Como resultado al ejecutar nos muestra la siguiente interfaz.



Como podemos ver tenemos un botón para cada uno de estos programas y un botón salir, como lo mencione, es una interfaz simple para mostrar los diversos programas realizados dentro de la fase 1 del proyecto.

Conclusiones

Esta primera fase del proyecto me ayudo a establecer una base en la investigación de Procesamiento de Lenguaje Natural (PLN), enfocándose en tareas fundamentales: la segmentación de oraciones, la identificación de patrones lingüísticos y la detección de errores básicos. Logrando implementar y validar los primeros prototipos para cada uno de estos temas.

Dentro de la segmentación de oraciones, desarrollamos algoritmos iniciales que permiten dividir textos en unidades oracionales discretas. Las pruebas que realizamos demostraron la efectividad de los métodos para manejar estructuras de oraciones comunes, sentando las bases para un análisis más detallado en fases posteriores.

Dentro de la identificación de patrones lingüísticos, durante esta fase nos centramos en el reconocimiento de las estructuras gramaticales y sintácticas que son más frecuentes. Logrando identificar patrones básicos que nos proporcionan una comprensión inicial de la composición del texto, lo que será esencial para tareas más complejas como el análisis semántico.

Finalmente, durante la detección de errores en esta etapa nos concentramos en la identificación de fallos ortográficos y algunos errores gramaticales simples. Los resultados obtenidos son prometedores y demuestran la viabilidad de incorporar sistemas de corrección en futuras iteraciones del proyecto.

En resumen, la Fase 1 ha sido crucial para sentar los cimientos de nuestro sistema de PLN. Hemos validado las metodologías iniciales y confirmado el potencial de las herramientas. Los logros de esta fase nos preparan para avanzar con confianza hacia desarrollos más sofisticados en las siguientes etapas del proyecto.

Recomendaciones

Las recomendaciones que doy y al ser la Fase 1 del proyecto son las siguientes:

- Enfoque en el archivo .csv inicial ya que este puede ser considerado el núcleo del proyecto
- Enfoque en las traducciones dadas
- Mejor retroalimentación a las traducciones
- Una investigación más amplia sobre lo que es el PNL
- Mas pruebas en caso de los diferentes formas y modelos que tiene PNL
- Búsqueda de recurso de náhuatl a español

Seguir con la construcción del modelo y utilizar equipos con más recursos, porque los modelos, aunque sean pequeños al tratarse del procesamiento del lenguaje natural consumen bastantes recursos del equipo.

Experiencia Obtenida

Me voy con la satisfacción de haber obtenido una mayor cantidad de conocimiento, ya que al tratarse de un tema que me interesaba por el uso de aprendizaje de la computadora; tanto la investigación realizada como el trabajo donde se realizaron las primeras pruebas, me deja con nuevas herramientas que pueden ser ocupadas con un mayor provecho en posteriores proyectos, de igual manera enfrentamos algunas dificultades ya que el náhuatl es una lengua en México que aunque tiene hablantes de esta, no se encuentran muchos de sus textos o materiales digitalizados por lo cual es fundamental realizar un trabajo de campo más extenso que nos lleve a ampliar nuestras bases que posteriormente ayudaran al proyecto a llegar a niveles superiores. De este proyecto me llevo muchas experiencias principalmente trabajando con el lenguaje de Python porque, aunque ya sabía que este lenguaje es uno de los más utilizados y que tiene un gran potencial, es sorprendente como es que se puede utilizar como lo es trabajando con PNL.

Competencias Desarrolladas

El desarrollo del presente proyecto nos ayudó a hacer un énfasis en los siguientes temas

- Fundamentos de PLN: Comprensión básica de cómo el lenguaje natural puede ser procesado computacionalmente, incluyendo delimitación de oraciones y la identificación de las unidades más pequeñas del lenguaje.
- Programación Aplicada a Datos Textuales: Reforzamos habilidades de programación en Python, aprendiendo a manipular y procesar cadenas de texto.
- Análisis y Preprocesamiento de Datos Básicos: Desarrolle la capacidad de tomar texto crudo y prepararlo para el análisis. Esto implica tareas como la limpieza de datos, la eliminación de caracteres no deseados y la normalización inicial del texto.
- Implementación de Algoritmos Simples: Nos familiarizamos con la implementación de algoritmos directos para la segmentación de oraciones (basados en puntuación, por ejemplo) y la detección de patrones (como secuencias de palabras o caracteres específicos) y errores ortográficos básicos (usando diccionarios simples).
- Uso de Librerías de PLN Básicas: Te introduces al uso de bibliotecas fundamentales de PLN (como NLTK en Python) para realizar tareas como la tokenización y la segmentación, lo que te familiariza con las herramientas estándar de la industria.

Referencias

Astera. (s. f.). *Extracción de información con PNL: Tipos, casos de uso y herramientas*. Recuperado de <https://www.astera.com/es/type/blog/nlp-information-extraction/>

BotPenguin. (s. f. a). *Hugging Face*. En *Glossary*. Recuperado de <https://botpenguin.com/glossary/hugging-face>

BotPenguin. (s. f. b). *Text Summarization*. En *Glossary*. Recuperado de <https://huggingface.co/tasks/summarization>

DataCamp. (s. f. a). *Introducción a las Redes Neuronales Convolucionales (CNNs)*. Recuperado de <https://www.datacamp.com/es/tutorial/introduction-to-convolutional-neural-networks-cnns>

DataCamp. (s. f. b). *¿Qué es el reconocimiento de entidades nombradas (NER)?*. Recuperado de <https://www.datacamp.com/es/blog/what-is-named-entity-recognition-ner>

DataCamp. (s. f. c). *Tutorial para Redes Neuronales Recurrentes*. Recuperado de <https://www.datacamp.com/es/tutorial/tutorial-for-recurrent-neural-network>

GeeksforGeeks. (s. f.). *Natural Language Processing (NLP) Tutorial*. Recuperado de <https://www.geeksforgeeks.org/natural-language-processing-nlp-tutorial/>

Hugging Face. (s. f. a). *facebook/bart-large-cnn*. Recuperado de <https://huggingface.co/facebook/bart-large-cnn>

Hugging Face. (s. f. b). *Transformers*. Recuperado de <https://huggingface.co/docs/transformers/index>

Hugging Face. (s. f. c). *FNet*. Recuperado de https://huggingface-co.translate.google/docs/transformers/v4.42.0/model_doc/fnet?x_tr_sl=en&x_tr_tl=es&x_tr_hl=es&x_tr_pto=tc

IBM. (s. f. a). *Redes neuronales convolucionales*. Recuperado de <https://www.ibm.com/mx-es/think/topics/convolutional-neural-networks>

IBM. (s. f. b). *Extracción de información*. Recuperado de <https://www.ibm.com/mx-es/think/topics/information-extraction>

IBM. (s. f. c). *Redes neuronales recurrentes*. Recuperado de <https://www.ibm.com/mx-es/think/topics/recurrent-neural-networks>

IBM. (s. f. d). *Generación de texto*. Recuperado de <https://www.ibm.com/mx-es/think/topics/text-generation>

Instituto Nacional de Lenguas Indígenas. (2022, 4 mayo). *Instituto Nacional de Lenguas Indígenas*. <https://www.inali.gob.mx/detalle/lengua-nawatlahtolli-nahuatl>

Kairntech. (s. f.). *Extracción de información de PNL: guía completa*. Recuperado de <https://kairntech.com/blog/articles/nlp-extraction/>

Kraz. (s. f.). *¿Qué es el Teorema de Bayes y qué importancia tiene en Machine Learning?*. Recuperado de <https://blog.kraz.ai/marketing/que-es-el-teorema-de-bayes-y-que-importancia-tiene-en-machine-learning/>

MathWorks. (s. f.). *Red neuronal convolucional*. Recuperado de <https://es.mathworks.com/discovery/convolutional-neural-network.html>

Pangeanic. (s. f.). *¿Qué son los Transformers en PLN?*. Recuperado de <https://blog.pangeanic.com/es/que-son-los-transformers-en-pln>

Reintech. (s. f.). *Cómo crear una herramienta de resumen de texto con Sumy: Tutorial para desarrolladores*. Recuperado de https://reintech-io.translate.google.com/blog/how-to-create-a-text-summarization-tool-with-sumy-tutorial-for-developers?x_tr_sl=en&x_tr_tl=es&x_tr_hl=es&x_tr_pto=sge

Siddique, A. (2021, julio 22). *NLTK: A Beginner's Hands-on Guide to Natural Language Processing*. Analytics Vidhya. Recuperado de <https://www.analyticsvidhya.com/blog/2021/07/nltk-a-beginners-hands-on-guide-to-natural-language-processing/>

Teradata. (s. f.). *Extracción de Datos*. Recuperado de <https://www.teradata.com/insights/data-platform/data-extraction>

Tirumala, R. (2024, julio 23). *Automated Text Summarization with Sumy Library*. Analytics Vidhya. Recuperado de https://www-analyticsvidhya-com.translate.google.com/blog/2024/07/automated-text-summarization-with-sumy-library/?x_tr_sl=en&x_tr_tl=es&x_tr_hl=es&x_tr_pto=sge#:~:text=Sumy%20es%20una%20de%20las,significativos%20según%20nuestras%20necesidades%20espec%C3%ADficas.

Vidal, F. (2023, julio 27). *NLTK Python: Procesamiento de Lenguaje Natural en Python*. Adictos al Trabajo. Recuperado de <https://adictosaltrabajo.com/2023/07/27/nltk-python/>

WiseCube. (s. f.). *Reconocimiento de Entidades Nombradas (NER) con Python*. Recuperado de <https://www.wisecube.ai/blog/named-entity-recognition-ner-with-python/>

Anexo A

A1 BLEU

```
import math

from collections import Counter


def preprocesar_texto(texto):
    """
    Preprocesamiento básico del texto: tokenización por palabras y
    conversión a minúsculas.
    """
    return texto.lower().split()


def calcular_precision_n_gramas(candidato, referencias, n):
    """
    Calcula la precisión modificada para n-gramas.

    Args:
        candidato: Lista de tokens de la traducción candidata
        referencias: Lista de listas de tokens de las traducciones de
        referencia
        n: Tamaño de los n-gramas (1 para unigramas, 2 para bigramas, etc.)

    Returns:
        La cuenta de n-gramas del candidato que aparecen en las referencias
        (modificado)
        y el total de n-gramas en el candidato
    """
    # Generar n-gramas para el candidato
    n_gramas_candidato = [tuple(candidato[i:i+n]) for i in
                           range(len(candidato)-n+1)]
```

```

if not n_gramas_candidato:
    return 0, 0

# Contar n-gramas en el candidato
conteo_candidato = Counter(n_gramas_candidato)

# Para cada n-grama, encontrar el máximo conteo en todas las referencias
conteo_max_referencias = Counter()
for referencia in referencias:
    n_gramas_referencia = [tuple(referencia[i:i+n]) for i in
range(len(referencia)-n+1)]
    conteo_referencia = Counter(n_gramas_referencia)

    for n_grama in conteo_candidato:
        conteo_max_referencias[n_grama] =
max(conteo_max_referencias[n_grama], conteo_referencia[n_grama])

# Calcular conteos recortados (clipped counts)
conteo_recortado = 0
for n_grama in conteo_candidato:
    conteo_recortado += min(conteo_candidato[n_grama],
conteo_max_referencias[n_grama])

return conteo_recortado, len(n_gramas_candidato)

def calcular_brevedad_penalizacion(candidato, referencias):
    """
    Calcula la penalización por brevedad (BP).

```

Args:

candidato: Lista de tokens de la traducción candidata

referencias: Lista de listas de tokens de las traducciones de referencia

Returns:

Factor de penalización por brevedad

"""

```
longitud_candidato = len(candidato)
```

```
# Encontrar la longitud de referencia más cercana
```

```
longitudes_referencia = [len(ref) for ref in referencias]
```

```
longitud_referencia_efectiva = min(longitudes_referencia, key=lambda  
ref_len: abs(ref_len - longitud_candidato))
```

```
if longitud_candidato > longitud_referencia_efectiva:
```

```
    return 1.0
```

```
else:
```

```
    return math.exp(1 - (longitud_referencia_efectiva /  
longitud_candidato))
```

```
def calcular_puntaje_bleu(candidato, referencias, pesos=None,  
verbose=False):
```

"""

Calcula el puntaje BLEU para una traducción candidata dadas una o más referencias.

Args:

candidato: String con la traducción candidata

referencias: Lista de strings con las traducciones de referencia

pesos: Lista de pesos para cada n-grama (por defecto [0.25, 0.25, 0.25, 0.25])

verbose: Si True, muestra información detallada del cálculo

Returns:

Puntaje BLEU entre 0 y 1

"""

if pesos is None:

pesos = [0.25, 0.25, 0.25, 0.25] # Pesos por defecto para 1-4
gramas

Preprocesar textos

candidato_tokens = preprocesar_texto(candidato)

referencias_tokens = [preprocesar_texto(ref) for ref in referencias]

if verbose:

print(f"Candidato: {candidato_tokens}")

print(f"Referencias: {referencias_tokens}")

Calcular precisiones para n-gramas de 1 a 4

precisiones = []

for n in range(1, len(pesos)+1):

recortado, total = calcular_precision_n_gramas(candidato_tokens,
referencias_tokens, n)

if verbose:

print(f"Precisión {n}-gramas: {recortado}/{total}")

if total == 0:

precision = 0

else:

```

        precision = recortado / total

    precisiones.append(precision)

# Calcular media geométrica ponderada de las precisiones
suma_ponderada = 0
for w, p in zip(pesos, precisiones):
    if p == 0:
        suma_ponderada = -float('inf')
        break
    suma_ponderada += w * math.log(p)

if suma_ponderada == -float('inf'):
    media_geo = 0
else:
    media_geo = math.exp(suma_ponderada)

# Calcular penalización por brevedad
bp = calcular_brevedad_penalizacion(candidato_tokens,
referencias_tokens)

# Calcular puntaje BLEU final
bleu = bp * media_geo

if verbose:
    print(f"Precisiones: {precisiones}")
    print(f"Media geométrica: {media_geo}")
    print(f"Penalización por brevedad: {bp}")
    print(f"Puntaje BLEU: {bleu}")

```

```
    return bleu

# Ejemplo de uso
if __name__ == "__main__":
    candidato = "el gato está debajo de la mesa"
    referencia1 = "el gato está bajo la mesa"
    referencia2 = "debajo de la mesa está el gato"

    print("-- Ejemplo de cálculo BLEU --")
    puntaje = calcular_puntaje_bleu(candidato, [referencia1, referencia2],
    verbose=True)
    print(f"\nPuntaje BLEU final: {puntaje:.4f}")
```

A2 Normalización

#Librerías para el funcionamiento del programa

```
import csv
```

```
import re
```

```
import unicodedata
```

#Librerías para el funcionamiento de la interfaz

```
import sys
```

```
import tkinter as tk
```

```
from tkinter import scrolledtext
```

#Codigo del programa

```
def normalizar_texto(texto):
```

```
    """
```

```
    Normaliza el texto realizando las siguientes operaciones:
```

```
    1. Convierte a minúsculas
```

```
    2. Elimina acentos y caracteres diacríticos
```

```
    3. Elimina signos de puntuación
```

```
    4. Elimina espacios extra
```

```
    5. Elimina caracteres especiales
```

```
    """
```

```
    if not isinstance(texto, str):
```

```
        return ""
```

```
    # Paso 1: Convertir a minúsculas
```

```
    texto = texto.lower()
```

```
    # Paso 2: Eliminar acentos y caracteres diacríticos
```

```
    texto = ''.join(c for c in unicodedata.normalize('NFD', texto)
```

```
        if unicodedata.category(c) != 'Mn')
```

```

    # Paso 3: Eliminar signos de puntuación (conserva letras, números y
    espacios)
    texto = re.sub(r'^\w\s]', '', texto)

    # Paso 4: Eliminar espacios extra
    texto = ' '.join(texto.split())

    return texto

def procesar_csv(archivo_entrada, archivo_salida):
    """
    Procesa el archivo CSV de entrada y genera un nuevo archivo CSV con el
    texto normalizado
    """
    try:
        with open(archivo_entrada, mode='r', encoding='latin-1',) as infile,
        \
            open(archivo_salida, mode='w', encoding='utf-8', newline='') as
        outfile:

            # Leer el archivo CSV
            lector = csv.reader(infile)
            escritor = csv.writer(outfile)

            # Escribir encabezados
            encabezados = next(lector, None)
            if encabezados:
                escritor.writerow([f"{encabezados[0]}_normalizado",
                f"{encabezados[1]}_normalizado"]])

```

```

        # Procesar cada fila
        for fila in lector:
            if len(fila) >= 2:
                espanol_normalizado = normalizar_texto(fila[0])
                nahuatl_normalizado = normalizar_texto(fila[1])
                escritor.writerow([espanol_normalizado,
nahuatl_normalizado])
            else:
                print(f"Advertencia: Fila con formato incorrecto:
{fila}")

        print(f"Procesamiento completado. \nResultados guardados en:
{archivo_salida}")

    except FileNotFoundError:
        print(f"Error: No se pudo encontrar el archivo {archivo_entrada}")
    except Exception as e:
        print(f"Error inesperado: {str(e)}")

#Codigo de la interfaz
class TerminalOutputGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Salida de Terminal")
        self.root.geometry("800x600")

        self.output_area = scrolledtext.ScrolledText(self.root,
wrap=tk.WORD)
        self.output_area.pack(fill=tk.BOTH, expand=True)

```

```

10))    self.output_area.configure(bg='black', fg='white', font=('Consolas',

self.output_area.insert(tk.END, text)
self.output_area.see(tk.END)

def flush(self):
    pass

def start(self):
    self.root.mainloop()
    sys.stdout = self.original_stdout

def ejecutar_con_interfaz(func):
    """Decorador para ejecutar una función con la interfaz gráfica"""
    def wrapper(*args, **kwargs):
        gui = TerminalOutputGUI()
        try:
            func(*args, **kwargs)
        except Exception as e:
            print(f"Error: {str(e)}")
        gui.start()
    return wrapper

@ejecutar_con_interfaz

```

```
def mi_programa():
    print("Iniciando programa...")
    #
    procesar_csv(archivo_entrada, archivo_salida)
    print("Puede consultar en archivo generado dentro del proyecto")

# Ejemplo de uso
if __name__ == "__main__":
    archivo_entrada = "naesp/traduc.csv"
    archivo_salida = "naesp/datos_normalizados.csv"

    mi_programa()
```

A3 Segmentador

```
#Librerias para el funcionamiento del programa
import csv

#Librerias para el funcionamiento de la interfaz
import sys
import tkinter as tk
from tkinter import scrolledtext

#Codigo funcionamiento del programa
def segmentar_oraciones(texto):
    """Segmenta un texto en oraciones basándose en signos de puntuación."""
    if not texto or not isinstance(texto, str):
        return []

    oraciones = []
    inicio = 0

    # Caracteres que pueden terminar una oración
    terminadores = ('.', '?', '!', ';')

    for i, caracter in enumerate(texto):
        if caracter in terminadores:
            # Verificar que no sea parte de una abreviación (como "EE.UU.")
            if i + 1 < len(texto) and texto[i+1].isspace():
                oracion = texto[inicio:i+1].strip()
                if oracion: # Evitar oraciones vacías
                    oraciones.append(oracion)
                inicio = i + 1
```

```

# Añadir el último fragmento si queda texto restante
ultima_oracion = texto[inicio:].strip()
if ultima_oracion:
    oraciones.append(ultima_oracion)

return oraciones

def procesar_csv(archivo_entrada, archivo_salida):
    """Procesa el archivo CSV de entrada y genera uno nuevo con oraciones
    segmentadas."""
    with open(archivo_entrada, mode='r', encoding='utf-8') as infile:
        lector = csv.DictReader(infile)

        # Mostrar las columnas disponibles para depuración
        columnas = lector.fieldnames
        print(f"Columnas encontradas en el CSV: {columnas}")

        # Limpiar nombres de columnas (quitar espacios)
        columnas_limpias = [col.strip() for col in columnas]

        # Intentar encontrar las columnas correctas
        col_espanol = None
        col_nahuatl = None

        for col in columnas_limpias:
            if 'español' in col.lower() or 'espanol' in col.lower():
                col_espanol = col
            elif 'náhuatl' in col.lower() or 'nahuatl' in col.lower():
                col_nahuatl = col

```

```

        if not col_espanol or not col_nahuatl:
            print("Error: No se pudieron encontrar las columnas de español y
náhuatl")

            print("Columnas disponibles:")
            for i, col in enumerate(columnas):
                print(f" {i}: '{col}'")
            return

        print(f"Usando columna español: '{col_espanol}'")
        print(f"Usando columna náhuatl: '{col_nahuatl}'")

    # Procesar el archivo
    with open(archivo_entrada, mode='r', encoding='utf-8') as infile, \
        open(archivo_salida, mode='w', encoding='utf-8', newline='') as
outfile:

        lector = csv.DictReader(infile)
        escritor = csv.writer(outfile)

        # Escribir encabezados
        escritor.writerow(['Idioma', 'Texto', 'Oracion_ID'])

        oracion_id = 1

        for fila in lector:
            try:
                # Procesar español
                texto_espanol = fila.get(col_espanol, '')

```

```

        oraciones_es = segmentar_oraciones(texto_espanol)
        for oracion in oraciones_es:
            escritor.writerow(['español', oracion,
f'oracion_{oracion_id}'])
            oracion_id += 1

    # Procesar náhuatl
    texto_nahuatl = fila.get(col_nahuatl, '')
    oraciones_na = segmentar_oraciones(texto_nahuatl)
    for oracion in oraciones_na:
        escritor.writerow(['náhuatl', oracion,
f'oracion_{oracion_id}'])
        oracion_id += 1

except Exception as e:
    print(f"Error procesando fila: {e}")
    continue

#Codigo funcionamiento de la interfaz
class TerminalOutputGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Salida de Terminal")
        self.root.geometry("800x600")

        self.output_area = scrolledtext.ScrolledText(self.root,
wrap=tk.WORD)
        self.output_area.pack(fill=tk.BOTH, expand=True)
        self.output_area.configure(bg='black', fg='white', font=('Consolas',
10))

```

```

        self.original_stdout = sys.stdout
        sys.stdout = self

    def write(self, text):
        self.output_area.insert(tk.END, text)
        self.output_area.see(tk.END)

    def flush(self):
        pass

    def start(self):
        self.root.mainloop()
        sys.stdout = self.original_stdout

def ejecutar_con_interfaz(func):
    """Decorador para ejecutar una función con la interfaz gráfica"""
    def wrapper(*args, **kwargs):
        gui = TerminalOutputGUI()
        try:
            func(*args, **kwargs)
        except Exception as e:
            print(f"Error: {str(e)}")
        gui.start()
    return wrapper

@ejecutar_con_interfaz
def mi_programa():
    print("Iniciando programa...")

```

```

#
try:
    procesar_csv(archivo_entrada, archivo_salida)
    print(f"Procesamiento completado. Resultados guardados en
{archivo_salida}")
except FileNotFoundError:
    print("Error: No se encontró el archivo de entrada.")
except Exception as e:
    print(f"Ocurrió un error: {str(e)}")

if __name__ == "__main__":
    archivo_entrada = "naesp/datos_normalizados.csv"
    archivo_salida = "naesp/oraci.csv"

    mi_programa()

```

A4 N-Gramas

#Librerías funcionamiento del programa

```
import csv
```

#Librerías para el funcionamiento de la interfaz

```
import sys
```

```
import tkinter as tk
```

```
from tkinter import scrolledtext
```

#Codigo funcionamiento del programa

```
def generar_ngramas(texto, n):
```

```
    """Genera n-gramas de un texto dado."""
```

```
    palabras = texto.split()
```

```
    ngramas = []
```

```
    for i in range(len(palabras) - n + 1):
```

```
        ngrama = ' '.join(palabras[i:i+n])
```

```
        ngramas.append(ngrama)
```

```
    return ngramas
```

```
def determinar_n(texto):
```

```
    """Determina el tamaño del n-grama basado en la longitud del texto."""
```

```
    palabras = texto.split()
```

```
    if len(palabras) <= 1:
```

```
        return 1
```

```
    elif len(palabras) <= 3:
```

```
        return 2
```

```
    elif len(palabras) <= 5:
```

```
        return 3
```

```
    else:
```

```
        return 4 # Puedes ajustar estos valores según tus necesidades
```

```

def procesar_csv(archivo_entrada, archivo_salida):
    """Procesa el archivo CSV y genera el archivo de salida con n-gramas."""
    with open(archivo_entrada, mode='r', encoding='utf-8') as infile, \
        open(archivo_salida, mode='w', encoding='utf-8', newline='') as
outfile:

        reader = csv.DictReader(infile)

        fieldnames = ['Idioma', 'Oracion_ID', 'Texto_Original', 'N_Grama',
'Ngrama_Texto']

        writer = csv.DictWriter(outfile, fieldnames=fieldnames)
        writer.writeheader()

        for fila in reader:
            idioma = fila['Idioma']
            texto = fila['Texto']
            oracion_id = fila['Oracion_ID']

            n = determinar_n(texto)
            ngramas = generar_ngramas(texto, n)

            for ngrama in ngramas:
                writer.writerow({
                    'Idioma': idioma,
                    'Oracion_ID': oracion_id,
                    'Texto_Original': texto,
                    'N_Grama': n,
                    'Ngrama_Texto': ngrama
                })

```

```

#Codigo funcionamiento de la interfaz
class TerminalOutputGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Salida de Terminal")
        self.root.geometry("800x600")

        self.output_area = scrolledtext.ScrolledText(self.root,
wrap=tk.WORD)

        self.output_area.pack(fill=tk.BOTH, expand=True)
        self.output_area.configure(bg='black', fg='white', font=('Consolas',
10))

        self.original_stdout = sys.stdout
        sys.stdout = self

    def write(self, text):
        self.output_area.insert(tk.END, text)
        self.output_area.see(tk.END)

    def flush(self):
        pass

    def start(self):
        self.root.mainloop()
        sys.stdout = self.original_stdout

def ejecutar_con_interfaz(func):

```

```

"""Decorador para ejecutar una función con la interfaz gráfica"""
def wrapper(*args, **kwargs):
    gui = TerminalOutputGUI()
    try:
        func(*args, **kwargs)
    except Exception as e:
        print(f"Error: {str(e)}")
    gui.start()
return wrapper

@ejecutar_con_interfaz
def mi_programa():
    print("Iniciando programa...")
    #
    procesar_csv(archivo_entrada, archivo_salida)
    print(f"Proceso completado. Los resultados se han guardado en
{archivo_salida}")

if __name__ == "__main__":
    archivo_entrada = "naesp/oraci.csv"
    archivo_salida = "naesp/n_gram.csv"

    mi_programa()

```

A5 Patrones gramaticales

```
import csv

import re

import matplotlib.pyplot as plt

from collections import defaultdict


class AnalizadorGramatical:
    """
    Clase para analizar patrones gramaticales en oraciones en español.
    Clasifica palabras en categorías gramaticales y identifica patrones
    frecuentes.
    """

    def __init__(self):
        """
        Inicializa el analizador con diccionarios para almacenar patrones
        y listas de palabras categorizadas por tipo gramatical.
        """
        # Diccionario para contar la frecuencia de cada patrón gramatical
        self.patrones = defaultdict(int)

        # Diccionario anidado: longitud -> patrón -> frecuencia
        # Permite analizar patrones según el número de palabras
        self.patrones_por_longitud = defaultdict(lambda: defaultdict(int))

        # Listas de palabras categorizadas por tipo gramatical
        self.articulos = ['el', 'la', 'los', 'las', 'un', 'una', 'unos',
                           'unas']
```

```
        self.preposiciones = ['a', 'al', 'ante', 'bajo', 'cabe', 'con',  
'contra', 'de', 'del',  
                               'desde', 'durante', 'en', 'entre', 'hacia',  
'hasta', 'mediante',  
                               'para', 'por', 'según', 'sin', 'so', 'sobre',  
'tras']
```

```
        self.conjunciones = ['y', 'e', 'ni', 'que', 'o', 'u', 'pero',  
'aunque', 'mas', 'sino']
```

```
        self.pronombres = ['yo', 'tú', 'él', 'ella', 'usted', 'nosotros',  
'vosotros',  
                            'ellos', 'ellas', 'ustedes', 'me', 'te', 'se',  
'nos', 'os']
```

```
def determinar_categoria(self, palabra):
```

```
    """
```

```
    Determina la categoría gramatical de una palabra usando:
```

1. Listas predefinidas para palabras funcionales
2. Expresiones regulares para patrones morfológicos

```
    Args:
```

```
        palabra (str): La palabra a categorizar
```

```
    Returns:
```

```
        str: Código de la categoría gramatical
```

```
    """
```

```
    palabra = palabra.lower()
```

```
    # Verificar primero las listas de palabras funcionales (más  
    precisas)
```

```

if palabra in self.articulos:
    return 'Articulo' # Artículos: el, la, un, una, etc.
elif palabra in self.preposiciones:
    return 'Preposi' # Preposiciones: a, de, en, por, etc.
elif palabra in self.conjunciones:
    return 'Conjci' # Conjunciones: y, o, pero, aunque, etc.
elif palabra in self.pronombres:
    return 'Pronombr' # Pronombres: yo, tú, él, me, te, se, etc.

# Usar expresiones regulares para identificar formas verbales y
otras categorías
elif re.match(r'^(*ando|*iando|*endo)$', palabra):
    return 'Gerun' # Gerundios: cantando, corriendo, viviendo
elif
re.match(r'^(*es|*as|*os|*amos|*emos|*imos|*asteis|*isteis|*aron|*
ieron)$', palabra):
    return 'V_Preter' # Verbos en pretérito: cantaste, corrimos,
vivieron
elif
re.match(r'^(*é|*í|*aste|*iste|*ó|*ió|*imos|*isteis|*eron)$',
palabra):
    return 'V_Preteri' # Más formas de pretérito: canté, corrí,
vivió
elif
re.match(r'^(*aba|*abas|*ábamos|*abais|*aban|*ía|*ías|*íamos|*íais|
*ían)$', palabra):
    return 'V_Imper' # Verbos en imperfecto: cantaba, corría,
vivía
elif
re.match(r'^(*aré|*eré|*iré|*arás|*erás|*irás|*aremos|*eremos|*irem
os|*aréis|*eréis|*iréis|*arán|*erán|*irán)$', palabra):
    return 'V_Futur' # Verbos en futuro: cantaré, correré,
viviré

```

```

        elif
re.match(r'^(*aría|*ería|*iría|*arías|*erías|*irías|*aríamos|*eríamos|*iríamos|*aríaais|*eríaais|*iríaais|*arían|*erían|*irían)$', palabra):

            return 'V_Condic'      # Verbos en condicional: cantarí,
correría, viviría

        elif
re.match(r'^(*o|*as|*a|*amos|*áis|*an|*es|*e|*emos|*éis|*en)$',
palabra):

            return 'V_Present'     # Verbos en presente: canto, corres, vive
elif re.match(r'^(*ad[oa]s?|*ed[oa]s?|*id[oa]s?)$', palabra):

            return 'Adjet'         # Adjetivos: cansado, verde, rápido
elif re.match(r'^(*[aeo]s?|*es|*[aeo]s)$', palabra):

            return 'Sustant'       # Sustantivos: casa, perro, ciudad
elif re.match(r'^[0-9]+$', palabra):

            return 'Num'           # Números: 1, 23, 456
else:

            return 'Otro'          # Cualquier palabra que no entre en las
categorías anteriores

def analizar_oracion(self, oracion):
    """
    Analiza una oración completa y extrae su patrón gramatical.

    Args:
        oracion (str): La oración a analizar

    Returns:
        str: El patrón gramatical como string (ej: "Articulo Sust
V_Pres")
    """
    # Extraer solo las palabras (letras y números) ignorando puntuación

```

```

palabras = re.findall(r'\b\w+\b', oracion.lower())

# Lista para almacenar las categorías de cada palabra
patron = []

# Clasificar cada palabra y construir el patrón
for palabra in palabras:
    categoria = self.determinar_categoria(palabra)
    patron.append(categoria)

# Convertir la lista de categorías en un string
patron_str = ' '.join(patron)

# Incrementar el contador para este patrón
self.patrones[patron_str] += 1

# También contar el patrón según su longitud
longitud = len(patron)
self.patrones_por_longitud[longitud][patron_str] += 1

return patron_str

def cargar_csv(self, archivo):
    """
    Carga un archivo CSV con oraciones y analiza solo las oraciones en
    español.

    Args:
        archivo (str): Ruta del archivo CSV

```

```

"""
with open(archivo, mode='r', encoding='utf-8') as file:
    lector = csv.DictReader(file)

    # Procesar cada fila del CSV
    for fila in lector:
        # Solo analizar oraciones en español
        if fila['Idioma'].lower() == 'español' or
fila['Idioma'].lower() == 'spanish':
            self.analizar_oracion(fila['Texto'])

def graficar_patrones_comunes(self, top_n=10):
    """
    Genera un gráfico de barras horizontales con los patrones más
    comunes.

    Args:
        top_n (int): Número de patrones más frecuentes a mostrar
    """
    # Ordenar patrones por frecuencia (mayor a menor) y tomar los top_n
    patrones_ordenados = sorted(self.patrones.items(), key=lambda x:
x[1], reverse=True)[:top_n]

    # Separar patrones y sus frecuencias
    patrones = [p[0] for p in patrones_ordenados]
    frecuencias = [p[1] for p in patrones_ordenados]

    # Crear el gráfico
    plt.figure(figsize=(12, 6))
    plt.barh(patrones, frecuencias) # Barras horizontales

```

```

plt.xlabel('Frecuencia')
plt.ylabel('Patrón gramatical')
plt.title('Patrones gramaticales más comunes')
plt.gca().invert_yaxis() # Invertir el eje Y para mostrar el más
frecuente arriba

plt.tight_layout() # Ajustar el layout para que no se corten las
etiquetas

plt.show()

def graficar_distribucion_longitudes(self):
    """
    Genera un gráfico de barras que muestra la distribución de oraciones
    según la cantidad de palabras que contienen.
    """
    # Obtener las longitudes ordenadas
    longitudes = sorted(self.patrones_por_longitud.keys())

    # Contar cuántas oraciones hay de cada longitud
    conteos = [sum(self.patrones_por_longitud[l].values()) for l in
longitudes]

    # Crear el gráfico
    plt.figure(figsize=(10, 5))
    plt.bar(longitudes, conteos)
    plt.xlabel('Longitud del patrón (en palabras)')
    plt.ylabel('Número de oraciones')
    plt.title('Distribución de longitudes de patrones gramaticales')
    plt.show()

def graficar_patrones_por_longitud(self, longitud):

```

```

"""
    Genera un gráfico de los patrones más comunes para una longitud
    específica.

    Args:
        longitud (int): Número de palabras de los patrones a analizar
"""
# Verificar que existan patrones de esa longitud
if longitud not in self.patrones_por_longitud:
    print(f"No hay oraciones con patrones de longitud {longitud}")
    return

# Obtener los patrones de la longitud especificada
patrones = self.patrones_por_longitud[longitud]

# Ordenar por frecuencia y tomar los 10 más comunes
patrones_ordenados = sorted(patrones.items(), key=lambda x: x[1],
reverse=True)[:10]

# Verificar que hay patrones para mostrar
if not patrones_ordenados:
    return

# Separar etiquetas y valores
etiquetas = [p[0] for p in patrones_ordenados]
valores = [p[1] for p in patrones_ordenados]

# Crear el gráfico
plt.figure(figsize=(12, 6))

```

```

plt.barh(etiquetas, valores)
plt.xlabel('Frecuencia')
plt.ylabel('Patrón gramatical')
plt.title(f'Patrones gramaticales más comunes para longitud
{longitud}')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()

```

```

if __name__ == "__main__":

```

```

    """

```

Sección principal que demuestra cómo usar el analizador:

1. Crear una instancia del analizador
2. Cargar datos desde un archivo CSV
3. Generar diferentes tipos de gráficos

```

    """

```

```

# Crear el analizador

```

```

analizador = AnalizadorGramatical()

```

```

# Cargar el archivo CSV

```

```

archivo_csv = 'naesp/oraci.csv'

```

```

analizador.cargar_csv(archivo_csv)

```

```

# Generar gráficos

```

```

print("Generando gráficos...")

```

```

# Gráfico 1: Los 10 patrones más comunes en general

```

```

analizador.graficar_patrones_comunes()

```

```
# Gráfico 2: Distribución de longitudes (cuántas palabras tienen las oraciones)
```

```
analizador.graficar_distribucion_longitudes()
```

```
# Gráfico 3: Patrones más comunes para oraciones de 5 palabras
```

```
analizador.graficar_patrones_por_longitud(5)
```

A6 Detección de errores

```
#Librerías de funcionamiento del programa

import pandas as pd
import matplotlib.pyplot as plt
import spacy
import re

from collections import Counter

#Librerías para el funcionamiento de la interfaz

import sys
import tkinter as tk
from tkinter import scrolledtext

#Codigo funcionamiento del programa
def verificar_dependencias():
    """Verifica que las dependencias básicas estén instaladas"""
    try:
        import spacy
        import pandas as pd
        import matplotlib.pyplot as plt
        print("✓ Dependencias básicas instaladas")
        return True
    except ImportError as e:
        print(f"X Error de importación: {e}")
        print("\nPara instalar las dependencias básicas, ejecuta:")
        print("pip install pandas matplotlib spacy")
        print("python -m spacy download es_core_news_sm")
        return False

def cargar_modelo_spacy():
```

```

"""Carga el modelo de spaCy con manejo de errores"""
try:
    nlp = spacy.load("es_core_news_sm")
    print("✓ Modelo de spaCy cargado correctamente")
    return nlp
except OSError:
    print("X Error: No se encontró el modelo de spaCy en español")
    print("Para instalarlo, ejecuta:")
    print("python -m spacy download es_core_news_sm")
    return None

def cargar_datos(archivo_csv):
    """Carga el archivo CSV y devuelve un DataFrame"""
    try:
        datos = pd.read_csv(archivo_csv)
        print(f"✓ Archivo {archivo_csv} cargado correctamente")
        print(f" - Filas: {len(datos)}")
        print(f" - Columnas: {list(datos.columns)}")
        return datos
    except FileNotFoundError:
        print(f"X Error: No se encontró el archivo {archivo_csv}")
        print("Asegúrate de que el archivo existe y la ruta es correcta")
        return None
    except Exception as e:
        print(f"X Error al cargar el archivo: {e}")
        return None

def filtrar_espanol(datos):

```

```

"""Filtra las oraciones en español"""

if datos is None:
    return None

# Verificar que exista la columna 'Idioma'
if 'Idioma' not in datos.columns:
    print("X Error: No se encontró la columna 'Idioma' en el CSV")
    print(f"Columnas disponibles: {list(datos.columns)}")
    return None

# Filtrar oraciones en español
datos_espanol = datos[datos['Idioma'].str.lower() == 'español']
print(f"✓ Filtradas {len(datos_espanol)} oraciones en español")

return datos_espanol

def crear_diccionario_basico():
    """Crea un diccionario básico de palabras comunes en español"""
    # Palabras comunes en español (se puede expandir)
    palabras_comunes = {
        'el', 'la', 'de', 'que', 'y', 'a', 'en', 'un', 'es', 'se', 'no',
        'te', 'lo', 'le',
        'da', 'su', 'por', 'son', 'con', 'para', 'al', 'del', 'los', 'las',
        'una', 'como',
        'pero', 'sus', 'muy', 'sin', 'sobre', 'también', 'me', 'hasta',
        'hay', 'donde',
        'han', 'quien', 'están', 'estado', 'desde', 'todo', 'nos',
        'durante', 'estados',
        'todos', 'uno', 'les', 'ni', 'contra', 'otros', 'fueron', 'ese',
        'eso', 'había',

```

'antes', 'sí', 'mismo', 'como', 'año', 'años', 'después', 'vida',
'tiempo',
'día', 'días', 'casa', 'nombre', 'hombre', 'mujer', 'mundo', 'país',
'trabajo',
'parte', 'forma', 'lugar', 'caso', 'grupo', 'problema', 'hecho',
'momento',
'manera', 'sistema', 'programa', 'ciudad', 'gobierno', 'partido',
'empresa',
'agua', 'mes', 'meses', 'noche', 'historia', 'hora', 'horas',
'gente', 'niño',
'niña', 'niños', 'niñas', 'estado', 'desarrollo', 'proceso', 'uso',
'nivel',
'servicio', 'información', 'ejemplo', 'situación', 'resultado',
'precio',
'producto', 'mercado', 'centro', 'proyecto', 'modelo', 'condición',
'persona',
'personas', 'razón', 'punto', 'puntos', 'tipo', 'tipos', 'director',
'dirección',
'número', 'números', 'control', 'objetivo', 'política', 'políticas',
'clase',
'clases', 'relación', 'relaciones', 'medio', 'medios', 'orden',
'final',
'principio', 'fin', 'económico', 'económica', 'social', 'sociales',
'nacional',
'nacional', 'internacional', 'general', 'generales', 'local',
'locales',
'público', 'pública', 'privado', 'privada', 'mejor', 'mejores',
'mayor',
'mayores', 'menor', 'menores', 'nuevo', 'nueva', 'nuevos', 'nuevas',
'primero', 'primera', 'último', 'última', 'últimos', 'últimas',
'grande',
'grandes', 'pequeño', 'pequeña', 'pequeños', 'pequeñas', 'bueno',
'buena',
'buenos', 'buenas', 'malo', 'mala', 'malos', 'malas', 'propio',
'propia',

'propios', 'propias', 'mismo', 'misma', 'mismos', 'mismas', 'otro',
'otra',
'otros', 'otras', 'cada', 'algún', 'alguna', 'algunos', 'algunas',
'ningún',
'ninguna', 'ningunos', 'ningunas', 'cierto', 'cierta', 'ciertos',
'ciertas',
'todo', 'toda', 'todos', 'todas', 'tanto', 'tanta', 'tantos',
'tantas',
'poco', 'poca', 'pocos', 'pocas', 'mucho', 'mucha', 'muchos',
'muchas',
'más', 'menos', 'tan', 'tanto', 'demasiado', 'demasiada',
'bastante',
'bastantes', 'suficiente', 'suficientes', 'necesario', 'necesaria',
'necesarios', 'necesarias', 'posible', 'posibles', 'imposible',
'imposibles',
'importante', 'importantes', 'interesante', 'interesantes',
'difícil',
'difíciles', 'fácil', 'fáciles', 'simple', 'simples', 'complejo',
'complejas',
'ser', 'estar', 'tener', 'hacer', 'poder', 'decir', 'ir', 'ver',
'dar',
'saber', 'querer', 'llegar', 'pasar', 'deber', 'poner', 'parecer',
'quedar',
'creer', 'hablar', 'llevar', 'dejar', 'seguir', 'encontrar',
'llamar',
'venir', 'pensar', 'salir', 'volver', 'tomar', 'conocer', 'vivir',
'sentir',
'tratar', 'mirar', 'contar', 'empezar', 'esperar', 'buscar',
'existir',
'entrar', 'trabajar', 'escribir', 'perder', 'producir', 'ocurrir',
'entender',
'pedir', 'recibir', 'recordar', 'terminar', 'permitir', 'aparecer',
'conseguir',
'comenzar', 'servir', 'sacar', 'necesitar', 'mantener', 'resultar',
'leer',

```

        'caer', 'cambiar', 'presentar', 'crear', 'abrir', 'considerar',
'oír',
        'puede', 'pueden', 'debe', 'deben', 'tiene', 'tienen', 'hace',
'hacen',
        'dice', 'dicen', 'va', 'van', 'viene', 'vienen', 'sale', 'salen',
        'entra', 'entran', 'vive', 'viven', 'trabaja', 'trabajan',
'estudia',
        'estudian', 'juega', 'juegan', 'come', 'comen', 'bebe', 'beben',
        'duerme', 'duermen', 'corre', 'corren', 'camina', 'caminan',
'habla',
        'hablan', 'escucha', 'escuchan', 'mira', 'miran', 'lee', 'leen',
        'escribe', 'escriben', 'compra', 'compran', 'vende', 'venden'
    }

```

```

return palabras_comunes

```

```

def detectar_errores_ortograficos_basico(oracion, diccionario):
    """Detecta errores ortográficos básicos usando un diccionario simple"""
    if not oracion or pd.isna(oracion):
        return 0

    # Limpiar la oración y extraer palabras
    oracion_limpia = re.sub(r'^\w\s', '', oracion.lower())
    palabras = oracion_limpia.split()

    # Contar palabras no encontradas en el diccionario
    errores = 0
    for palabra in palabras:
        if palabra and len(palabra) > 2: # Ignorar palabras muy cortas
            if palabra not in diccionario:

```

```

        errores += 1

    return errores

def detectar_errores_gramaticales(oracion, nlp):
    """Detecta errores gramaticales básicos usando spaCy"""
    if not oracion or pd.isna(oracion) or nlp is None:
        return 0

    try:
        doc = nlp(oracion)
        errores = 0

        # Verificar concordancia básica entre sustantivos y adjetivos
        for token in doc:
            if token.pos_ == 'ADJ' and token.head.pos_ == 'NOUN':
                # Verificar concordancia de género y número
                token_gender = token.morph.get('Gender')
                token_number = token.morph.get('Number')
                head_gender = token.head.morph.get('Gender')
                head_number = token.head.morph.get('Number')

                if (token_gender and head_gender and token_gender !=
head_gender) or \
                    (token_number and head_number and token_number !=
head_number):
                    errores += 1

        # Verificar estructura básica de la oración

```

```

        # Buscar verbos principales
        verbos_principales = [token for token in doc if token.pos_ == 'VERB'
and token.dep_ == 'ROOT']

        if not verbos_principales and len(doc) > 3: # Oraciones largas sin
verbo principal
            errores += 1

    return errores

except Exception as e:
    print(f"Error en análisis gramatical: {e}")
    return 0

def detectar_errores(oracion, nlp, diccionario):
    """Detecta errores gramaticales y ortográficos en una oración"""
    errores = {
        'ortograficos': 0,
        'gramaticales': 0,
        'total': 0
    }

    # Detectar errores ortográficos
    errores['ortograficos'] = detectar_errores_ortograficos_basico(oracion,
diccionario)

    # Detectar errores gramaticales
    errores['gramaticales'] = detectar_errores_gramaticales(oracion, nlp)

    errores['total'] = errores['ortograficos'] + errores['gramaticales']

```

```
return errores
```

```
def analizar_oraciones(datos_espanol, nlp):  
    """Analiza todas las oraciones en español y devuelve resultados"""  
    if datos_espanol is None:  
        return None  
  
    # Verificar que existan las columnas necesarias  
    columnas_requeridas = ['Texto', 'Oracion_ID']  
    for col in columnas_requeridas:  
        if col not in datos_espanol.columns:  
            print(f"X Error: No se encontró la columna '{col}' en el CSV")  
            return None  
  
    # Crear diccionario básico  
    diccionario = crear_diccionario_basico()  
    print(f"✓ Diccionario creado con {len(diccionario)} palabras")  
  
    resultados = []  
    total_oraciones = len(datos_espanol)  
  
    print(f"Analizando {total_oraciones} oraciones...")  
  
    for i, (_, fila) in enumerate(datos_espanol.iterrows()):  
        if i % 100 == 0: # Mostrar progreso cada 100 oraciones  
            print(f"Procesando oración {i+1}/{total_oraciones}")  
  
        oracion = fila['Texto']
```

```

    oracion_id = fila['Oracion_ID']
    errores = detectar_errores(oracion, nlp, diccionario)

    resultados.append({
        'Oracion_ID': oracion_id,
        'Texto': oracion,
        'Errores_Ortograficos': errores['ortograficos'],
        'Errores_Gramaticales': errores['gramaticales'],
        'Total_Errores': errores['total']
    })

print("✓ Análisis completado")
return pd.DataFrame(resultados)

def generar_grafico(resultados):
    """Genera un gráfico de barras con los resultados"""
    if resultados is None or len(resultados) == 0:
        print("X No hay resultados para graficar")
        return

    try:
        # Preparar datos para el gráfico
        datos_grafico = resultados[['Errores_Ortograficos',
        'Errores_Gramaticales']].sum()

        # Crear gráfico
        plt.figure(figsize=(12, 7))
        colores = ['#FF6B6B', '#4ECDC4']

```

```

        bars = plt.bar(datos_grafico.index, datos_grafico.values,
                        color=colores,

                                edgecolor='black', linewidth=1.2, alpha=0.8)

plt.title('Distribución de Errores en Oraciones en Español',
          fontsize=16, fontweight='bold', pad=20)
plt.xlabel('Tipo de Error', fontsize=12, fontweight='bold')
plt.ylabel('Cantidad de Errores', fontsize=12, fontweight='bold')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Añadir valores en las barras
for bar, value in zip(bars, datos_grafico.values):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() +
0.5,
                                str(value), ha='center', va='bottom', fontweight='bold',
fontsize=11)

plt.tight_layout()
plt.show()

print("✓ Gráfico generado exitosamente")

except Exception as e:
    print(f"X Error al generar el gráfico: {e}")

def generar_reporte_estadisticas(resultados):
    """Genera un reporte con estadísticas descriptivas"""
    if resultados is None or len(resultados) == 0:

```

```

        return

print("\n" + "="*60)
print("REPORTE DE ESTADÍSTICAS - ANÁLISIS DE ERRORES")
print("="*60)

total_oraciones = len(resultados)
oraciones_con_errores = len(resultados[resultados['Total_Errores'] > 0])
oraciones_sin_errores = total_oraciones - oraciones_con_errores

print(f"📊 RESUMEN GENERAL:")
print(f"    Total de oraciones analizadas: {total_oraciones}")
print(f"    Oraciones con errores: {oraciones_con_errores} ({oraciones_con_errores/total_oraciones*100:.1f}%)")
print(f"    Oraciones sin errores: {oraciones_sin_errores} ({oraciones_sin_errores/total_oraciones*100:.1f}%)")

print(f"\n📝 ERRORES ORTOGRÁFICOS:")
print(f"    Total: {resultados['Errores_Ortograficos'].sum()}")
print(f"    Promedio por oración: {resultados['Errores_Ortograficos'].mean():.2f}")
print(f"    Máximo en una oración: {resultados['Errores_Ortograficos'].max()}")

print(f"\n📖 ERRORES GRAMATICALES:")
print(f"    Total: {resultados['Errores_Gramaticales'].sum()}")
print(f"    Promedio por oración: {resultados['Errores_Gramaticales'].mean():.2f}")
print(f"    Máximo en una oración: {resultados['Errores_Gramaticales'].max()}")

```

```

# Mostrar las 5 oraciones con más errores

print(f"\n🔥 TOP 5 ORACIONES CON MÁS ERRORES:")

top_errores = resultados.nlargest(5, 'Total_Errores')

for i, (_, fila) in enumerate(top_errores.iterrows(), 1):
    print(f"    {i}. [{fila['Total_Errores']} errores]
{fila['Texto'][:80]}...")

def principal():
    """Función principal"""
    print("DETECTOR DE ERRORES GRAMATICALES Y ORTOGRÁFICOS")
    print("="*60)

    # Verificar dependencias
    if not verificar_dependencias():
        return

    # Cargar modelo de spaCy
    nlp = cargar_modelo_spacy()

    # Nombre del archivo CSV
    archivo_csv = 'naesp/oraci.csv'

    # Procesamiento de datos
    datos = cargar_datos(archivo_csv)
    datos_espanol = filtrar_espanol(datos)
    resultados = analizar_oraciones(datos_espanol, nlp)

    if resultados is not None and len(resultados) > 0:
        # Mostrar primeras filas de resultados

```

```

print("\n📋 MUESTRA DE RESULTADOS (primeras 5 oraciones):")
print(resultados.head().to_string(index=False))

# Generar estadísticas
generar_reporte_estadisticas(resultados)

# Generar gráfico
generar_grafico(resultados)

# Guardar resultados en un nuevo CSV
try:
    resultados.to_csv('naesp/resultados_errores.csv', index=False,
encoding='utf-8')
    print(f"\n✅ Resultados guardados en
'naesp/resultados_errores.csv'")
except Exception as e:
    print(f"X Error al guardar resultados: {e}")
else:
    print("X No se encontraron oraciones en español para analizar.")

#Codigo funcionamiento de la interfaz
class TerminalOutputGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Salida de Terminal")
        self.root.geometry("1000x800")

        self.output_area = scrolledtext.ScrolledText(self.root,
wrap=tk.WORD)

```

```

        self.output_area.pack(fill=tk.BOTH, expand=True)
        self.output_area.configure(bg='black', fg='white', font=('Consolas',
15))

        self.original_stdout = sys.stdout
        sys.stdout = self

    def write(self, text):
        self.output_area.insert(tk.END, text)
        self.output_area.see(tk.END)

    def flush(self):
        pass

    def start(self):
        self.root.mainloop()
        sys.stdout = self.original_stdout

def ejecutar_con_interfaz(func):
    """Decorador para ejecutar una función con la interfaz gráfica"""
    def wrapper(*args, **kwargs):
        gui = TerminalOutputGUI()
        try:
            func(*args, **kwargs)
        except Exception as e:
            print(f"Error: {str(e)}")
        gui.start()
    return wrapper

```

```
@ejecutar_con_interfaz
def mi_programa():
    print("Iniciando programa...")
    #
    principal()

if __name__ == "__main__":
    mi_programa()
```

A7 Detección de errores PNL

```
#Librerias funcionamiento del programa

import pandas as pd
import matplotlib.pyplot as plt
import spacy
from spellchecker import SpellChecker

#Librerias para el funcionamiento de la interfaz
import sys
import tkinter as tk
from tkinter import scrolledtext

# Cargar modelos de lenguaje
nlp = spacy.load("es_core_news_sm")
spell = SpellChecker(language='es')

#Codigo funcionamiento del programa
def cargar_datos(archivo_csv):
    """Carga el archivo CSV y devuelve un DataFrame"""
    try:
        datos = pd.read_csv(archivo_csv)
        return datos
    except FileNotFoundError:
        print(f"Error: No se encontró el archivo {archivo_csv}")
        return None

def filtrar_espanol(datos):
    """Filtra las oraciones en español"""
    if datos is None:
        return None
    return datos[datos['Idioma'].str.lower() == 'español']
```

```

def detectar_errores(oracion):
    """Detecta errores gramaticales y ortográficos en una oración"""
    errores = {
        'ortograficos': 0,
        'gramaticales': 0,
        'total': 0
    }

    # Verificación ortográfica
    palabras = oracion.split()
    mal_escritas = spell.unknown(palabras)
    errores['ortograficos'] = len(mal_escritas)

    # Análisis gramatical con spaCy
    doc = nlp(oracion)

    # Contar errores gramaticales básicos (esto es un enfoque simplificado)
    gramaticales = 0
    for token in doc:
        if token.is_punct or token.is_space:
            continue

        # Verificar concordancia básica entre sustantivos y adjetivos
        if token.pos_ == 'ADJ' and token.head.pos_ == 'NOUN':
            # Verificar concordancia de género y número
            if (token.morph.get('Gender') != token.head.morph.get('Gender'))
or
                token.morph.get('Number') !=
token.head.morph.get('Number')):

```

```

        gramaticales += 1

errores['gramaticales'] = gramaticales
errores['total'] = errores['ortograficos'] + errores['gramaticales']

return errores

def analizar_oraciones(datos_espanol):
    """Analiza todas las oraciones en español y devuelve resultados"""
    if datos_espanol is None:
        return None

    resultados = []
    for _, fila in datos_espanol.iterrows():
        oracion = fila['Texto']
        oracion_id = fila['Oracion_ID']
        errores = detectar_errores(oracion)
        resultados.append({
            'Oracion_ID': oracion_id,
            'Texto': oracion,
            'Errores_Ortograficos': errores['ortograficos'],
            'Errores_Gramaticales': errores['gramaticales'],
            'Total_Errores': errores['total']
        })

    return pd.DataFrame(resultados)

def generar_grafico(resultados):
    """Genera un gráfico de barras con los resultados"""

```

```

if resultados is None:
    return

# Preparar datos para el gráfico
datos_grafico = resultados[['Errores_Ortograficos',
'Errores_Gramaticales']].sum()

# Crear gráfico
plt.figure(figsize=(10, 6))
datos_grafico.plot(kind='bar', color=['red', 'blue'])
plt.title('Distribución de Errores en Oraciones en Español')
plt.xlabel('Tipo de Error')
plt.ylabel('Cantidad de Errores')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Mostrar gráfico
plt.tight_layout()
plt.show()

def principal():
    # Nombre del archivo CSV
    archivo_csv = 'naesp/oraci.csv'

    # Procesamiento de datos
    datos = cargar_datos(archivo_csv)
    datos_espanol = filtrar_espanol(datos)
    resultados = analizar_oraciones(datos_espanol)

```

```

if resultados is not None:
    # Mostrar resultados
    print("\nResultados del análisis:")
    print(resultados.head())

    # Generar gráfico
    generar_grafico(resultados)

    # Guardar resultados en un nuevo CSV
    resultados.to_csv('naesp/resultados_errores_pnl.csv', index=False)
    print("\nResultados guardados en
'naesp/resultados_errores_pnl.csv'")
else:
    print("No se encontraron oraciones en español para analizar.")

#Codigo funcionamiento de la interfaz
class TerminalOutputGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Salida de Terminal")
        self.root.geometry("1000x800")

        self.output_area = scrolledtext.ScrolledText(self.root,
wrap=tk.WORD)
        self.output_area.pack(fill=tk.BOTH, expand=True)
        self.output_area.configure(bg='black', fg='white', font=('Consolas',
15))

        self.original_stdout = sys.stdout
        sys.stdout = self

```

```

def write(self, text):
    self.output_area.insert(tk.END, text)
    self.output_area.see(tk.END)

def flush(self):
    pass

def start(self):
    self.root.mainloop()
    sys.stdout = self.original_stdout

def ejecutar_con_interfaz(func):
    """Decorador para ejecutar una función con la interfaz gráfica"""
    def wrapper(*args, **kwargs):
        gui = TerminalOutputGUI()
        try:
            func(*args, **kwargs)
        except Exception as e:
            print(f"Error: {str(e)}")
        gui.start()
    return wrapper

@ejecutar_con_interfaz
def mi_programa():
    print("Iniciando programa...")
    #
    principal()

```

```
if __name__ == "__main__":  
    mi_programa()
```

A8 Evaluación BLEU

#Librerías funcionamiento del programa

```
import pandas as pd
```

```
import random
```

```
from nltk.translate.bleu_score import sentence_bleu
```

```
import nltk
```

#Librerías para el funcionamiento de la interfaz

```
import sys
```

```
import tkinter as tk
```

```
from tkinter import scrolledtext
```

Descargar recursos necesarios de NLTK (solo primera vez)

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

#Codigo funcionamiento del programa

```
def evaluar_bleu(texto_original, ngrama_texto, n):
```

```
    """
```

Evalúa el puntaje BLEU entre el texto original y los n-gramas.

Args:

texto_original (str): Texto de referencia.

ngrama_texto (str): N-gramas a evaluar.

n (int): Número de gramas (1, 2, 3, etc.).

Returns:

float: Puntaje BLEU.

```
    """
```

Tokenizar el texto original como referencia

```
referencia = [nltk.word_tokenize(texto_original.lower())]
```

```

# Tokenizar los n-gramas como candidato
candidato = nltk.word_tokenize(ngrama_texto.lower())

# Calcular BLEU con los pesos según el n-grama
# Por ejemplo, para 2-gramas: (0.5, 0.5)
pesos = tuple([1.0/n] * n)

return sentence_bleu(referencia, candidato, weights=pesos)

def principal():
    # Cargar el archivo CSV
    try:
        df = pd.read_csv('naesp/n_gram.csv')
    except FileNotFoundError:
        print("Error: No se encontró el archivo CSV.")
        return

    # Filtrar solo las oraciones en español
    df_espanol = df[df['Idioma'].str.lower() == 'español']

    if df_espanol.empty:
        print("No se encontraron oraciones en español en el archivo.")
        return

    # Seleccionar un Oracion_ID aleatorio
    oracion_id = random.choice(df_espanol['Oracion_ID'].unique())

    # Obtener todos los registros para ese Oracion_ID
    registros = df_espanol[df_espanol['Oracion_ID'] == oracion_id]

```

```
# Obtener el texto original (debería ser el mismo para todos los
registros del mismo Oracion_ID)
```

```
texto_original = registros.iloc[0]['Texto_Original']
```

```
print(f"\nEvaluando Oracion_ID: {oracion_id}")
```

```
print(f"Texto original: {texto_original}\n")
```

```
# Evaluar cada n-grama para esta oración
```

```
for _, row in registros.iterrows():
```

```
    n = row['N_Grama']
```

```
    ngrama_texto = row['Ngrama_Texto']
```

```
# Calcular puntaje BLEU
```

```
puntaje_bleu = evaluar_bleu(texto_original, ngrama_texto, n)
```

```
print(f"{n}-grama: {ngrama_texto}")
```

```
print(f"Puntaje BLEU: {puntaje_bleu:.4f}\n")
```

```
#Codigo funcionamiento de la interfaz
```

```
class TerminalOutputGUI:
```

```
    def __init__(self):
```

```
        self.root = tk.Tk()
```

```
        self.root.title("Salida de Terminal")
```

```
        self.root.geometry("1000x800")
```

```
        self.output_area = scrolledtext.ScrolledText(self.root,
wrap=tk.WORD)
```

```
        self.output_area.pack(fill=tk.BOTH, expand=True)
```

```

15))    self.output_area.configure(bg='black', fg='white', font=('Consolas',

self.original_stdout = sys.stdout
sys.stdout = self

def write(self, text):
    self.output_area.insert(tk.END, text)
    self.output_area.see(tk.END)

def flush(self):
    pass

def start(self):
    self.root.mainloop()
    sys.stdout = self.original_stdout

def ejecutar_con_interfaz(func):
    """Decorador para ejecutar una función con la interfaz gráfica"""
    def wrapper(*args, **kwargs):
        gui = TerminalOutputGUI()
        try:
            func(*args, **kwargs)
        except Exception as e:
            print(f"Error: {str(e)}")
        gui.start()
    return wrapper

@ejecutar_con_interfaz

```

```
def mi_programa():  
    print("Iniciando programa...")  
    #  
    principal()  
  
if __name__ == "__main__":  
    mi_programa()
```

A9 Pequeño traductor utilizando RNN (Red Neuronal Recurrente)

```
import pandas as pd

import numpy as np

import tensorflow as tf

from keras_preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad_sequences

from keras.models import Sequential

from keras.layers import Embedding, SimpleRNN, Dense

import tkinter as tk

from tkinter import ttk, messagebox

import csv

import os

from datetime import datetime


# Configuración

CSV_TRAIN_FILE = 'naesp/traduc.csv'

CSV_FEEDBACK_FILE = 'naesp/retro_traduccion.csv'

MAX_VOCAB_SIZE = 10000

EMBEDDING_DIM = 256

RNN_UNITS = 512

MAX_LENGTH = 20

BATCH_SIZE = 32

EPOCHS = 10


# Preprocesamiento de datos

def load_and_preprocess_data(csv_file):

    df = pd.read_csv(csv_file, encoding='latin-1')

    español = df['español'].astype(str).tolist()

    náhuatl = df['náhuatl'].astype(str).tolist()
```

```

# Tokenización español
español_tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE, filters='')
español_tokenizer.fit_on_texts(español)
español_sequences = español_tokenizer.texts_to_sequences(español)
español_data = pad_sequences(español_sequences, maxlen=MAX_LENGTH,
padding='post')

# Tokenización náhuatl
náhuatl_tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE, filters='')
náhuatl_tokenizer.fit_on_texts(náhuatl)
náhuatl_sequences = náhuatl_tokenizer.texts_to_sequences(náhuatl)
náhuatl_data = pad_sequences(náhuatl_sequences, maxlen=MAX_LENGTH,
padding='post')

return (español_data, español_tokenizer, náhuatl_data,
náhuatl_tokenizer)

# Creación del modelo
def create_model(input_vocab_size, target_vocab_size):
    model = Sequential([
        Embedding(input_vocab_size, EMBEDDING_DIM, input_length=MAX_LENGTH),
        SimpleRNN(RNN_UNITS, return_sequences=True),
        Dense(target_vocab_size, activation='softmax')
    ])

    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

```

```

# Entrenamiento del modelo

def train_model(model, X_train, y_train):
    model.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCHS,
validation_split=0.2)
    return model

# Traducción

def translate(model, sequence, español_tokenizer, náhuatl_tokenizer):
    prediction = model.predict(sequence)
    predicted_sequence = np.argmax(prediction, axis=-1)
    translated_text =
náhuatl_tokenizer.sequences_to_texts(predicted_sequence)
    return translated_text[0]

# Interfaz gráfica

class TranslationApp:
    def __init__(self, root, model, español_tokenizer, náhuatl_tokenizer):
        self.root = root
        self.model = model
        self.español_tokenizer = español_tokenizer
        self.náhuatl_tokenizer = náhuatl_tokenizer

        root.title("Traductor Español-Náhuatl")

        # Frame principal
        main_frame = ttk.Frame(root, padding="10")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # Entrada español

```

```

        ttk.Label(main_frame, text="Español:").grid(row=0, column=0,
sticky=tk.W)

        self.entry_español = tk.Text(main_frame, width=50, height=5)
        self.entry_español.grid(row=1, column=0, padx=5, pady=5)

        # Entrada náhuatl (resultado)

        ttk.Label(main_frame, text="Náhuatl:").grid(row=2, column=0,
sticky=tk.W)

        self.entry_náhuatl = tk.Text(main_frame, width=50, height=5,
state='disabled')

        self.entry_náhuatl.grid(row=3, column=0, padx=5, pady=5)

        # Botones

        btn_frame = ttk.Frame(main_frame)
        btn_frame.grid(row=4, column=0, pady=10)

        ttk.Button(btn_frame, text="Traducir",
command=self.translate_text).pack(side=tk.LEFT, padx=5)

        ttk.Button(btn_frame, text="Buena traducción", command=lambda:
self.rate_translation(True)).pack(side=tk.LEFT, padx=5)

        ttk.Button(btn_frame, text="Mala traducción", command=lambda:
self.rate_translation(False)).pack(side=tk.LEFT, padx=5)

        # Inicializar archivo de feedback si no existe
        if not os.path.exists(str(CSV_FEEDBACK_FILE)):
            with open(str(CSV_FEEDBACK_FILE), 'w', newline='',
encoding='latin-1') as f:
                writer = csv.writer(f)
                writer.writerow(['fecha', 'texto_original', 'traduccion',
'calificacion'])

```

```

def translate_text(self):
    text = self.entry_español.get("1.0", tk.END).strip()
    if not text:
        messagebox.showwarning("Advertencia", "Por favor ingrese un
texto para traducir")
        return

    sequence = self.español_tokenizer.texts_to_sequences([text])
    padded_sequence = pad_sequences(sequence, maxlen=MAX_LENGTH,
padding='post')

    translated_text = translate(self.model, padded_sequence,
self.español_tokenizer, self.náhuatl_tokenizer)

    self.entry_náhuatl.config(state='normal')
    self.entry_náhuatl.delete("1.0", tk.END)
    self.entry_náhuatl.insert("1.0", translated_text)
    self.entry_náhuatl.config(state='disabled')

    # Guardar la última traducción para calificación
    self.last_translation = {
        'original': text,
        'translation': translated_text
    }

def rate_translation(self, is_good):
    if not hasattr(self, 'last_translation'):
        messagebox.showwarning("Advertencia", "Primero realice una
traducción para calificar")
        return

```

```

        rating = "buena" if is_good else "mala"

    # Guardar en CSV de feedback
    with open(str(CSV_FEEDBACK_FILE), 'a', newline='', encoding='latin-
1') as f:
        writer = csv.writer(f)
        writer.writerow([
            datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            self.last_translation['original'],
            self.last_translation['translation'],
            rating
        ])

    messagebox.showinfo("Gracias", f"¡Gracias por calificar esta
traducción como {rating}!")

# Función principal
def main():
    # Cargar y preprocesar datos
    try:
        español_data, español_tokenizer, náhuatl_data, náhuatl_tokenizer =
load_and_preprocess_data(CSV_TRAIN_FILE)
    except FileNotFoundError:
        print(f"Error: No se encontró el archivo {CSV_TRAIN_FILE}")
        return
    except Exception as e:
        print(f"Error al cargar los datos: {str(e)}")
        return

```

```
# Crear y entrenar modelo

input_vocab_size = len(español_tokenizer.word_index) + 1
target_vocab_size = len(náhuatl_tokenizer.word_index) + 1


model = create_model(input_vocab_size, target_vocab_size)
model = train_model(model, español_data, náhuatl_data)


# Crear interfaz gráfica
root = tk.Tk()
app = TranslationApp(root, model, español_tokenizer, náhuatl_tokenizer)
root.mainloop()


if __name__ == "__main__":
    main()
```

A10 Ejecutable

```
import tkinter as tk

from tkinter import messagebox

import subprocess

import sys

import os


class ProgramLauncher:

    def __init__(self, root):

        self.root = root

        self.root.title("Procesamiento de Lenguaje Natural")

        self.root.geometry("1200x800")

        self.root.resizable(False, False)


        # Configuración de colores

        bg_color = "#f0f0f0"

        button_color = "#e1e1e1"

        active_color = "#d1d1d1"


        self.root.config(bg=bg_color)


        # Título

        title_label = tk.Label(

            root,

            text="Procesamiento de las transcripciones utilizando técnicas

de PLN (Procesamiento de Lenguaje Natural)",

            font=("Arial", 14, "bold"),

            bg=bg_color

        )
```

```

title_label.pack(pady=10)

# Frame para los botones
button_frame = tk.Frame(root, bg=bg_color)
button_frame.pack(expand=True)

# Botones
self.buttons = []
self.programs = [
    ("Normalizacion", "naesp/Normalizacion.py"),
    ("Oraciones", "naesp/Segmentador.py"),
    ("N-gramas", "naesp/N_Gramas.py"),
    ("Patrones gramaticales", "naesp/Patrones_Gramaticales.py"),
    ("Deteccion errores", "naesp/Deteccion_errores.py"),
    ("Deteccion errores PNL", "naesp/Deteccion_er_pnl.py"),
    ("Evaluacion BLEU", "naesp/Evaluacion_BLEU.py"),
    ("Traductor", "naesp/RNN_trad.py")
]

# Crear 8 botones en una cuadrícula de 2x4
for i, (name, file) in enumerate(self.programs):
    row = i // 4
    col = i % 4

    btn = tk.Button(
        button_frame,
        text=name,
        width=25,
        height=3,

```

```

        bg=button_color,
        activebackground=active_color,
        command=lambda f=file: self.run_program(f),
        font=("Arial", 10)
    )
    btn.grid(row=row, column=col, padx=10, pady=10)
    self.buttons.append(btn)

# Botón de salir
exit_button = tk.Button(
    root,
    text="Salir",
    command=root.quit,
    bg="#ff9999",
    activebackground="#ff7777",
    width=10,
    font=("Arial", 10, "bold")
)
exit_button.pack(pady=10)

def run_program(self, program_file):
    """Ejecuta el programa Python especificado"""
    try:
        # Verificar si el archivo existe
        if not os.path.exists(program_file):
            messagebox.showerror("Error", f"El archivo {program_file} no
existe.")
        return

```

```
# Ejecutar el programa usando el intérprete de Python actual
subprocess.Popen([sys.executable, program_file])

except Exception as e:
    messagebox.showerror("Error", f"No se pudo ejecutar
{program_file}:\n{str(e)}")

if __name__ == "__main__":
    root = tk.Tk()
    app = ProgramLauncher(root)
    root.mainloop()
```