

Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial
15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Matemáticas y Física
Maestría en Ciencia de Datos



**Modelo para la estructuración y clasificación de datos no estructurados
usando técnicas de Procesamiento de Lenguaje Natural: un caso de
estudio en el ámbito de la industria alimentaria.**

**TESIS para obtener el GRADO de
MAESTRA EN CIENCIA DE DATOS**

Presenta:
Patricia Muñoz López

Director:
Dr. César Lozano Díaz

Tlaquepaque, Jalisco, Enero, 2025

Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Matemáticas y Física Formato de Aprobación de la Maestría en Ciencia de Datos

Título de tesis: **Modelo para la estructuración y clasificación de datos no estructurados usando técnicas de Procesamiento de Lenguaje Natural: un caso de estudio en el ámbito de la industria alimentaria.**

Autora: **Patricia Muñoz López**

Thesis Approved to complete all degree requirements for the Master of Science Degree in Data Science.

Thesis Advisor, **Dr. César Lozano Díaz**

Thesis Reader, **Dr. Riemann Ruíz Cruz**

Thesis Reader, **Dra. María del Rosario Ruíz Hernández**

Tlaquepaque, Jalisco, Enero, 2025

Modelo para la estructuración y clasificación de datos no estructurados usando técnicas de Procesamiento de Lenguaje Natural: un caso de estudio en el ámbito de la industria alimentaria.

Patricia Muñoz López

Resumen

Este estudio aborda el análisis de *datos no estructurados* provenientes de etiquetas de productos alimenticios con el objetivo de clasificar alimentos bajo el contexto de diseño y desarrollo de nuevos productos. Dado que estos datos contienen información detallada sobre ingredientes y aditivos, su análisis resulta fundamental para identificar relaciones subyacentes y transformar el texto en valores cuantitativos susceptibles de evaluación sistemática.

Para alcanzar este propósito, se emplean técnicas de Procesamiento de Lenguaje Natural (PLN), como los *word embeddings* y el *reconocimiento de entidades nombradas* (Named Entity Recognition o NER). En particular, se utilizan modelos como GPT, Word2Vec y GLiNER, con los cuales se desarrolla una metodología orientada a la extracción de patrones relevantes, facilitando así la limpieza, estructuración y análisis de los datos en el marco del desarrollo de productos alimenticios.

Los resultados muestran que la metodología es efectiva para procesar *datos no estructurados*, mejorando la calidad mediante segmentación por idioma, adaptación de modelos preentrenados de NER y balance de clases. Estas optimizaciones permitieron obtener métricas destacadas (*accuracy*, *precision*, *recall*, *f1-score*) y confirmar la hipótesis inicial, evidenciando un impacto positivo en la clasificación de alimentos.

Finalmente, el procesamiento y estructuración de datos dentro de un flujo de trabajo automatizado tiene un alto valor, ya que permite integrar y sistematizar los datos de manera escalable y reproducible. Este enfoque ofrece un modelo escalable para gestionar datos, especialmente aquellos provenientes de bases de datos tipo *crowdsourcing*, mejorando así la calidad y consistencia de las bases de datos alimentarias y facilitando la actualización continua de información. La centralización de datos de alta calidad podría contribuir considerablemente al desarrollo y la innovación de productos alimenticios.

Palabras clave: Datos no estructurados, procesamiento de lenguaje natural, clasificación de alimentos, reconocimiento de entidades nombradas (NER), Word2Vec.

Contents

	Page
1 Introducción	27
2 Justificación.	29
2.1 De la procedencia de los datos	29
2.2 De los métodos de modelado	31
3 Problema.	33
3.1 Desarrollo del problema	33
3.1.1 Pregunta de investigación	36
3.2 Objetivos	36
3.2.1 Objetivo general	36
3.2.2 Objetivos específicos	36
3.2.3 Hipótesis	37
3.2.4 Hipótesis Nula	37
3.2.5 Novedad científica, tecnológica o aportación	37
3.2.6 Futuras líneas de trabajo	38
4 Estado del arte	39
4.1 Introducción	39
4.2 Aplicaciones de Inteligencia Artificial (IA) en el ámbito de alimentos	40
5 Marco teórico	49
5.1 Introducción	49
5.2 El lenguaje y la ambigüedad	51
5.3 Representaciones vectoriales del lenguaje	53
5.4 Fundamentos matemáticos en la representación del lenguaje	56
5.4.1 Probabilidad y espacio latente en el modelado de lenguaje	57
5.5 Modelos de lenguaje iterativos basados en redes neuronales	62
5.5.1 Contexto local y global	62
5.6 Arquitecturas de modelos de lenguaje neuronales	63
5.6.1 Word2Vec	65
5.6.2 Transición hacia la arquitectura de Transformers	71
5.6.3 Transformers	78
5.7 Tareas específicas	91

5.7.1	Summarization	91
5.7.2	Named Entity Recognition (NER)	91
5.7.3	Text Classification	92
5.8	ANOVA: Análisis de Varianza	92
5.8.1	Modelo estadístico	92
5.8.2	Formulación de la hipótesis	92
5.8.3	Fórmulas del ANOVA	93
5.8.4	Criterio de decisión	94
6	Desarrollo metodológico	95
6.1	Motivación	95
6.2	Conjunto de datos: <i>OpenFoodFacts</i>	96
6.2.1	Acceso a los datos	97
6.2.2	Preparación de los datos	98
6.3	Motivación para el uso de <i>GLiNER</i>	101
6.3.1	El Modelo <i>GLiNER</i>	102
6.4	Motivación para generación de conjunto de datos y uso de Web Scrapping	104
6.4.1	Extracción de datos mediante Web Scraping	104
6.4.2	Tarea de resumir el texto <i>summarization</i>	105
6.4.3	Proceso de tokenización y Reconocimiento de Entidades Nombradas (NER)	107
6.5	Ajuste (<i>Fine-tuning</i>) de <i>GLiNER</i>	109
6.6	Entrenamiento de Word2Vec para la clasificación de alimentos	112
6.7	Métricas de evaluación	117
6.7.1	Cálculo del tamaño de muestra para evaluación NER	123
6.8	Análisis estadístico de la hipótesis	124
6.8.1	Descripción de los escenarios	124
6.8.2	Variables de análisis	126
6.8.3	Metodología estadística	127
6.8.4	Definición de métricas de desempeño	128
7	Resultados y discusión.	131
7.1	Introducción	131
7.2	Identificación de idiomas y calidad de los datos	133
7.3	Desempeño del Modelo <i>GLiNER</i> ajustado	133
7.3.1	Evaluación de las métricas de rendimiento del modelo ajustado.	134
7.4	Resultados de Word2Vec en la clasificación de alimentos	134
7.4.1	Generación de representaciones vectoriales	140
7.4.2	Descripción del uso de las listas de ingredientes depuradas	143
7.5	Evaluación de los algoritmos de clasificación	145

7.6	Resultados de desempeño para la clasificación en los diferentes escenarios	153
7.6.1	Resultados del escenario de línea base	153
7.6.2	Resultados del escenario parcial	153
7.6.3	Resultados del escenario completo	154
7.6.4	Comparación de métricas entre escenarios	155
7.6.5	Validación de supuestos	156
7.6.6	Resultados de ANOVA de una vía, tamaño del efecto y Post Hoc	158
7.6.7	Resultados de Kruskal-Wallis y Post Hoc	159
7.7	Discusión	160
8	Conclusiones	164
8.1	Conclusiones finales	164
8.1.1	Principales hallazgos	164
8.1.2	Aportaciones del estudio	165
8.1.3	Limitaciones y futuras líneas de trabajo	165
8.2	Trabajo a futuro	165
8.2.1	Conclusión general	166
8.3	Anexos	167
8.3.1	Cálculo de la muestra para evaluación de idioma en las listas de ingredientes	167
8.3.2	Repositorio con código replicable	167
8.3.3	Términos de uso de Wikipedia	168
8.3.4	Listado de URLs	168
	Bibliography	169

List of Figures

	Page
4.1 Modelo para el diseño de productos alimenticios híbrido. El objetivo es determinar la mejor combinación de ingredientes y condiciones de procesamiento para producir un alimento. Figura retomada de X. Zhang, T. Zhou, L. Zhang, K. Y. Fung, and K. M. Ng, "Food Product Design: A Hybrid Machine Learning and Mechanistic Modeling Approach".	41
4.2 Diagrama de flujo del algoritmo de búsqueda de recetas basado en ingredientes. Figura retomada de S. Chaudhari, A. R., V. G. Tekkur, P. G. L., and S. R. Karki, "Ingredient/Recipe Algorithm Using Web Mining and Web Scraping for Smart Chef," in Proceedings of the IEEE CONECCT 2020: 6th International Conference on Electronics, Computing and Communication Technologies, 2020.	42
4.3 Marco del modelo generativo con múltiples ramas: El modelo consta de tres generadores (G_0 , G_1 , G_2) que crean imágenes en diferentes resoluciones (64^2 , 128^2 , 256^2). A cada generador le corresponde un discriminador (D_0 , D_1 , D_2) que evalúa la autenticidad de las imágenes, clasificándolas como reales o falsas. El proceso está regulado por una pérdida de consistencia cíclica y una pérdida de similitud coseno, garantizando coherencia entre las imágenes generadas y las reales. Figura retomada de F. Han, R. Guerrero, and V. Pavlovic, "Cookgan: Meal image synthesis from ingredients". . . .	43

- 4.4 Modelo de incrustación neuronal conjunta con regularización semántica: Este modelo aprende un espacio de incrustación compartido entre imágenes de alimentos y recetas de cocina. El encoder de ingredientes (arriba) procesa una secuencia de ingredientes, mientras que el encoder de instrucciones (abajo) procesa las instrucciones de la receta. Ambos encoders generan representaciones que se combinan mediante capas completamente conectadas. Figura retomada de J. Marin, A. Biswas, F. Ofli, N. Hynes, A. Salvador, Y. Aytar, I. Weber, and A. Torralba, "Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images" 44
- 4.5 Modelo de codificación de recetas: Este diagrama muestra dos marcos de trabajo. (a) Codificación de ingredientes, que utiliza reconocimiento de entidades, vectores numéricos y un modelo LSTM para representar los ingredientes de una receta. (b) Codificación de instrucciones, que usa LSTM para procesar secuencias de instrucciones y sus relaciones, prediciendo el siguiente paso en función del anterior. Figura retomada de M. Al-Sarayreh, M. Gomes Reis, A. Carr, and M. M. dos Reis, "Inverse design and AI/Deep generative networks in food design: A comprehensive review". 45
- 4.6 Adaptación de recetas para restricciones dietéticas: Este diagrama muestra el proceso para ajustar recetas según restricciones, como el veganismo. Los ingredientes se analizan y los que no cumplen con la restricción son reemplazados por alternativas adecuadas, como cambiar queso por margarina en un sándwich. Figura retomada de A. Morales-Garzón, J. Gómez-Romero, and M. J. Martin-Bautista, "A word embedding-based method for unsupervised adaptation of cooking recipes" 46

4.7	Incrustación de grafos con metapath2vec+CSP en FlavorGraph: Este modelo combina grafos de ingredientes y compuestos químicos, representando sus interacciones mediante metapaths y predicciones de estructuras químicas. Se entrenan capas ocultas con vectores de ingredientes y compuestos, optimizando la predicción de las relaciones mediante pérdidas de CSP y Skip-gram, para mejorar la comprensión de las conexiones en el FlavorGraph. Figura retomada de D. Park, K. Kim, S. Kim, M. Spranger, and J. Kang, “FlavorGraph: a large-scale food-chemical graph for generating food representations and recommending food pairings”	47
5.1	Relación entre disciplinas que estudian en lenguaje. La Figura original fue adaptada al idioma español de J. Tsujii, “Natural Language Processing and Computational Linguistics,” <i>Computational Linguistics</i> , vol. 47, pp. 707–727, 12 2021	50
5.2	Representación de la taxonomía de WordNet para dos sentidos diferentes de una palabra y cómo estos sentidos se organizan jeráquicamente dentro de categorías más amplias. La Figura original fue adaptada al idioma español de A. Finch, E. Black, Y.-S. Hwang, and E. Sumita, “Using lexical dependency and ontological knowledge to improve a detailed syntactic and semantic tagger of English,” in <i>Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions</i> , (Sydney, Australia), pp. 215–222, Association for Computational Linguistics, July 2006	52
5.3	Diferencia entre la representación localista y distribuida. Figura retomada de J. Chen, “Intro to nlp part ii: Word embedding,” 2021.	54
5.4	Se muestra una matriz de co-ocurrencia X generada a partir de un corpus de texto. Las filas y las columnas representan palabras del vocabulario y cada celda contiene un valor que indica la frecuencia con la que cada par de palabras aparecen juntas dentro de una ventana de contexto definida.	60
5.5	Taxonomía de las incrustaciones de texto (<i>word embeddings</i>). La Figura original fue adaptada al idioma español de Wang, 2020 y Chiusano, 2021.	60
5.6	Word embedding independiente de contexto.	61
5.7	Word embedding dependiente de contexto.	62

- 5.8 Ejemplificación del contexto local y global. Un fragmento de texto del conjunto de datos “20 Newsgroup”. La parte transparente representa el contexto local de la palabra “harmful”. La parte semitransparente denota el resto del documento. Figura retomada de H. J. W. G. W. Z. Z. C. Meng, Yu and J. Han, “Unsupervised word embedding learning by incorporating local and global contexts,” *Frontiers in Big Data*, vol. 3, p. 9, 2020. 63
- 5.9 Arquitectura de la primera red neuronal profunda para PLN propuesta por Y. Bengio, R. Ducharme, and P. Vincent, “A neural probabilistic language model,” in *Advances in Neural Information Processing Systems* (T. Leen, T. Dietterich, and V. Tresp, eds.), vol. 13, MIT Press, 2000. 64
- 5.10 Se ilustra el concepto de una ventana de contexto utilizada en modelos PLN. La palabra central (en color rojo) se encuentra en la posición t , mientras que las palabras de su contexto inmediato dentro de una ventana de tamaño 2 (en color azul), incluyen w_{t-2} , w_{t-1} , w_{t+1} , w_{t+2} . Las fechas representan las relaciones de probabilidad condicional $P(w|w_t)$ utilizadas para modelar la dependencia entre la palabra central y las palabras de su contexto. La imagen original fue adaptada al idioma español de C. Manning, J. Hewitt, F. Chaubard, M. Fang, G. Genthial, R. Mundra, and R. Socher, “Note 1: Introduction and word2vec.” johnhew@cs.stanford.edu, 2023. CS 224n: Natural Language Processing with Deep Learning, Winter 2023. 66
- 5.11 Arquitectura de un modelo secuencia a secuencia (*sequence-to-sequence*, *Seq2Seq*), compuesto por un codificador (*encoder*) y un decodificador (*decoder*). El el codificador, se procesan las entradas $x_1, x_2, x_3, \dots, x_n$, a través de una capa de *embeddings* seguida por capas ocultas (*hidden layers*), produciendo una representación codificada de la secuencia. Figura retomada de K. Chilamkurthy, “Nlp theory and code: Encoder-decoder models (part 11/30),” *Medium*, March 2021. Published in Nerd For Tech 76
- 5.12 Línea de tiempo con algunos de los modelos o arquitecturas de modelos de lenguaje representativos desde el 2018 a 2023. Figura retomada de S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, “Unifying large language models and knowledge graphs: A roadmap,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, p. 3580–3599, July 2024. 79

- 5.13 Arquitectura del modelo Tranformer. Figura tomada de A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. 79
- 5.14 Diagrama de la arquitectura del modelo Transformer, que ilustra la estructura de capas de codificación (Encoder Layer Class) y decodificación (Decoder Layer Class). A la izquierda, el bloque de codificación consta de seis capas con módulos de atención múltiple (Multi-Head Attention) y una red de avance (Feed Forward) con normalización. A la derecha, el bloque de decodificación también incluye seis capas, donde cada capa tiene atención enmascarada (Masked Multi-Head Attention) para procesar secuencias de salida de manera autoregresiva. Entre los bloques de codificación y decodificación, se produce la atención cruzada, permitiendo que el decodificador se alinee con la salida del codificador. Este diseño permite la traducción de una oración en inglés ("I am a student") a francés ("je suis étudiant"). Figura retomada de R. M. Go, *Deep Learning Bible - Part K: Transformer from scratch*, ch. K1.2. How it works, step by step. Mintrobot, 2024. 80
- 5.15 Esquema de la atención escalada por producto punto (izquierda) y atención de múltiples cabezas (derecha). Figura retomada de A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. 81
- 5.16 Esquema del mecanismo de *Atención Multi-Cabeza* en el modelo Transformer. La secuencia de entrada se proyecta en matrices de *queries* (Q), *keys* (K) y *values* (V), que luego se procesan de manera paralela en varias cabezas de atención. Cada cabeza de atención aplica el mecanismo de atención escalada por producto punto (*Scaled Dot-Product Attention*), enfocándose en diferentes partes de la secuencia. Las salidas de todas las cabezas se concatenan y proyectan a través de una capa lineal, generando una representación final que se utiliza en las siguientes capas del modelo. Figura retomada de R. M. Go, *Deep Learning Bible - Part K: Transformer from scratch*, ch. K1.2. How it works, step by step. Mintrobot, 2024. 84

5.17 Esquema del proceso completo en el encoder de un Transformer. (1) Los tokens de entrada se convierten en *embeddings* a través de un tokenizer. (2) Se suman *positional embeddings* sinusoidales para incorporar la información de posición en la secuencia. (3) Se aplica una máscara de padding (*Self Padding Mask*) para ignorar tokens vacíos durante la autoatención. (4) La secuencia de tokens pasa a través de N capas del encoder, cada una compuesta de bloques de autoatención de múltiples cabezas (*MHA*), redes *feed-forward*, y conexiones residuales con normalización (*Add & Norm*). (5) La salida del encoder es una representación codificada de la secuencia de entrada, que será utilizada por el decodificador. Imagen tomada de R. M. Go, *Deep Learning Bible - Part K: Transformer from scratch*, ch. K1.2. How it works, step by step. Mintrobot, 2024. 88

5.18 Esquema del proceso en el decodificador con N capas en un Transformer. El decodificador genera la salida en los siguientes pasos: (1) Se incorpora el embedding del token de entrada del decodificador. (2) Se añade la codificación posicional sinusoidal para incorporar la información de posición en la secuencia de entrada del decodificador. (3) Se crea una máscara de padding a partir de las entradas del codificador para ignorar los tokens de relleno. (4) Se genera una máscara de anticipación (*Look Ahead Mask*) a partir de las entradas del decodificador, evitando que el modelo vea información futura durante la generación. (5) Las N capas del decodificador están apiladas, donde cada capa realiza operaciones de autoatención multi-cabeza y atención entre el encoder y el decodificador. (6) Finalmente, la salida de la última capa del decodificador se utiliza para predecir el próximo token de la secuencia de salida. Imagen tomada de R. M. Go, *Deep Learning Bible - Part K: Transformer from scratch*, ch. K1.2. How it works, step by step. Mintrobot, 2024. 89

6.1 Distribución de los cinco idiomas principales en los datos. 98

6.2 Frecuencia de las categorías y del conjunto de datos final. 99

6.3 **Flujo de la Etapa I.** El procesamiento de datos comenzó con (1) la extracción de ≈ 3.13 millones de registros desde la base de datos de Open Food Facts. (2) Posteriormente, se realizó un filtrado de columnas, reduciendo el número de columnas de 206 a 3 en el DataFrame. (3) A continuación, se eliminaron los registros que contenían datos faltantes, lo que resultó en un total de 938,719 registros. (4) Se procedió a identificar el idioma, seleccionando únicamente los registros en inglés, lo que redujo el conjunto de datos a 446,730 registros. (5) Finalmente, se obtuvo un conjunto de datos preparado para clasificación con 42,027 registros. Este proceso fue diseñado para garantizar datos utilizables para los siguientes pasos del análisis. Elaboración propia. 101

6.4 Arquitectura del modelo GLiNER. Figura retomada de U. Zaratiana, N. Tomeh, P. Holat, and T. Charnois, "Gliner: Generalist model for named entity recognition using bidirectional transformer," 2023. 102

6.5 **Flujo de la Etapa 2.** El procesamiento de datos comienza con (1) la generación del listado de aditivos alimentarios basado en el *Codex Alimentarius*, el cual se traduce en un conjunto de URLs para cada aditivo; (2) posteriormente, se realizó un proceso de *Web Scraping* sobre este listado, obteniendo como resultado un archivo en formato JSON; (3) con los datos obtenidos, se generó un resumen para cada aditivo, centrado en su relevancia dentro de la industria alimentaria; (4) finalmente, se estructuraron los datos mediante diversos microprocesos que incluyeron la identificación y extracción de entidades clave, la tokenización y la determinación de las posiciones de dichas entidades en el texto. El resultado final fue un conjunto de datos en formato JSON, el cual fue compartido con la comunidad de *HuggingFace* para su uso. Elaboración propia. 112

6.6 Diagrama del flujo de trabajo para la clasificación

y procesamiento de datos de ingredientes y aditivos alimentarios. **Etapas I (A):** Extracción y limpieza de datos provenientes de la base colaborativa OpenFoodFacts, donde se filtran registros incompletos y se identifican los idiomas relevantes. El resultado es un conjunto final de 42,027 registros para clasificación. **Etapas II (B):** Extracción de datos mediante técnicas de web scraping en Wikipedia y con el apoyo del listado de aditivos del Codex Alimentarius. (C) El resultado de lo anterior es un nuevo conjunto de datos necesario para el *fine-tuning* del modelo GLiNER. Posteriormente este conjunto de datos se separa en *train* y *test*, se pasa por el modelo preentrenado y (D) genera un nuevo modelo. **Etapas III (E):** Aplicar el nuevo modelo a los datos de la etapa I, seguido de la generación de embeddings con Word2Vec para representar semánticamente los alimentos. Finalmente, (F) los vectores generados son utilizados para la clasificación de productos alimenticios. 116

- 6.7 Ejemplo de una matriz de confusión multiclase *NN*. Cada celda de la matriz representa la clasificación de las instancias según las clases reales (en filas) y las clases predichas (en columnas). Las celdas en azul corresponden a los verdaderos positivos (TP), donde la clase predicha coincide con la clase real. Las celdas verdes representan los falsos positivos (FP), donde una clase fue incorrectamente predicha como la clase actual. Las celdas amarillas indican los falsos negativos (FN), donde las instancias de la clase actual fueron clasificadas incorrectamente en otras clases. Las celdas restantes corresponden a los verdaderos negativos (TN), que son todas las instancias correctamente clasificadas como no pertenecientes a la clase actual. Esta estructura se repite para cada clase en problemas de clasificación multiclase. Figura retomada de W. B. Bustamante, "Calculando la precisión en un modelo de clasificación multiclase," 2021. Consultado: octubre, 2024. 129

7.1	Visualización de la palabra "apple" (índice 146) en el <i>TensorFlow Embedding Projector</i> , utilizando una proyección tridimensional por PCA . En el gráfico, se ha resaltado "apple" y sus 10 palabras más cercanas en el espacio vectorial original, calculando la distancia mediante la similitud coseno. Esta visualización permite identificar las palabras que comparten mayor similitud semántica con "apple", lo que se refleja en la proximidad de los puntos en el gráfico y que coinciden con lo mostrado en la Tabla 7.5 para la misma palabra.	141
7.2	Proyección de vectores mediante t-SNE con perplejidad 10.142	
7.3	Proyección de vectores mediante t-SNE con perplejidad 40.143	
7.4	Descripción de escenarios posibles para la representación vectorial de listas de ingredientes.	145
7.5	Distribución de frecuencias de productos alimenticios por categoría.	146
7.6	Matriz de confusión resultado de la regresión logística con los vectores promediados.	149
7.7	Comportamiento de la función de pérdida y precisión a lo largo de 150 épocas para el modelo de red neuronal. La gráfica de la izquierda ilustra la disminución en la función de pérdida, mientras que la gráfica de la derecha muestra el aumento en la precisión del modelo.	151
7.8	Diferencias entre métricas por escenario.	155
7.9	Gráficos Q-Q por métricas y escenario para observar la normalidad de los residuos.	157
7.10	Histogramas de los residuos para las métricas <i>Accuracy</i> , <i>Precision</i> , <i>Recall</i> y <i>F1score</i> , con curvas de densidad ajustadas. Las líneas rojas indican el valor promedio de los residuos. Estos gráficos evidencian una desviación respecto a la distribución normal, mostrando patrones de asimetría y concentraciones de los residuos hacia valores específicos.	157

List of Tables

	Page
5.1 Elaboración propia. Comparación entre características de modelos Seq2Seq y Transformers	78
6.1 Descripción de variables del conjunto de datos.	98
6.2 Modelos disponibles de <i>GLiNER</i> y sus características. . .	103
6.3 Ejemplo de coincidencia perfecta entre predicción y estándar dorado.	118
6.4 Ejemplo donde el sistema predice una entidad que no existe en el estándar dorado.	118
6.5 Ejemplo donde el sistema no predice una entidad que sí está en el estándar dorado.	119
6.6 Ejemplo donde el sistema asigna el tipo de entidad incorrecto.	119
6.7 Ejemplo donde el sistema delimita incorrectamente los límites del string de la entidad.	120
6.8 Ejemplo donde el sistema se equivoca tanto en la delimitación como en el tipo de entidad.	120
6.9 Interacción entre los tipos de errores y los esquemas de evaluación.	122
6.10 Resultados de precisión, recall y F1-score para diferentes esquemas de evaluación.	123
7.1 Resultados generales para las diferentes configuraciones de evaluación del modelo NER ajustado.	133
7.2 Resultados para la etiqueta INGREDIENTE	134
7.3 Resultados para la etiqueta ADITIVO	134
7.4 Ejemplos de palabras entrenadas en el modelo Word2Vec con errores.	135
7.5 Similitud de ingredientes con <i>apple</i> y <i>sugar</i> según el modelo Word2Vec.	136
7.6 No similitud de ingredientes con <i>apple</i> y <i>sugar</i> según el modelo Word2Vec.	137

7.7	Ejemplificación de agrupación de ingredientes similares según el modelo Word2Vec para ingredientes con el prefijo "organic".	139
7.8	Resultados del análisis de las palabras que no encajan en cada lista de ingredientes.	139
7.9	Arquitectura del Modelo de Red Neuronal	147
7.10	Desempeño de los modelos de clasificación en términos de métricas clave y tiempo de ejecución. [†] Representa el promedio de vectores y [‡] la suma de vectores.	148
7.11	Resultados obtenidos por clase para las métricas de desempeño de la regresión logística con multinomial con la técnica de promedio de vectores	148
7.12	Resultados obtenidos por clase para las métricas de desempeño de la regresión logística con multinomial y con las técnica de suma de vectores	149
7.13	Comparación de las métricas de desempeño obtenidas por clase para SVM con kernel lineal y kernel RBF utilizando la técnica de promedio de vectores	150
7.14	Comparación de las métricas de desempeño obtenidas por clase para SVM con kernel lineal y kernel RBF utilizando la técnica de suma de vectores	150
7.15	Resultados obtenidos del modelo de red neuronal tras 150 épocas. Las métricas incluyen precisión, <i>recall</i> , exactitud (<i>accuracy</i>) y pérdida (<i>loss</i>) tanto para los datos de entrenamiento como para los de validación para los vectores promediados . El tiempo total de entrenamiento fue de 7936.5440 segundos.	150
7.16	Resultados obtenidos del modelo de red neuronal tras 150 épocas. Las métricas incluyen precisión, <i>recall</i> , exactitud (<i>accuracy</i>) y pérdida (<i>loss</i>) tanto para los datos de entrenamiento como para los de validación para los vectores sumados . El tiempo total de entrenamiento fue de 7728.0316 segundos.	151
7.17	Desempeño de los modelos de clasificación en términos de métricas clave y tiempo de ejecución con técnica del promedio de vectores y balanceo por submuestras de clases.	152
7.18	Desempeño de los modelos de clasificación en términos de métricas clave y tiempo de ejecución con técnica del promedio de vectores y balanceo por SMOTE.	152
7.19	Desempeño de los modelos de clasificación del escenario de línea base.	153
7.20	Tiempos de ejecución promedio de los modelos de clasificación en el escenario de línea base.	153

7.21	Desempeño de los modelos de clasificación. Estos valores se calcularon llevando a cabo los procesos propuestos en el modelo de este proyecto, excepto la separación a través del modelo ajustado NER. Los valores incluyen el promedio (μ), desviación estándar ($\pm\sigma$), error estándar (SE), e intervalo de confianza (IC) al 95%. La red neuronal fue ejecutada una sola vez, por lo que no es posible calcular SE ni IC	153
7.22	Tiempos de ejecución promedio de los modelos de clasificación. Los valores están en segundos; el tiempo de la Red Neuronal corresponde a una única ejecución. .	154
7.23	Desempeño de los modelos de clasificación en términos de métricas clave con técnica del promedio de vectores y balanceo por SMOTE. Los valores incluyen el promedio (μ), desviación estándar ($\pm\sigma$), error estándar (SE), e intervalo de confianza (IC) al 95%. La red neuronal fue ejecutada una sola vez, por lo que no es posible calcular SE ni IC	154
7.24	Tiempos de ejecución promedio de los modelos con técnica del promedio de vectores y balanceo por SMOTE. Los valores están en segundos; el tiempo de la Red Neuronal se refiere a una única ejecución.	154
7.25	Resultados de prueba de normalidad Shapiro-Wilk para las métricas.	156
7.26	Resultados de la prueba de Levene para homocedasticidad.	156
7.27	Asimetría y Curtosis de las métricas de evaluación . . .	158
7.28	Resultados del ANOVA de una vía para las métricas evaluadas	159
7.29	Resultados del tamaño del efecto para cada métrica . . .	159
7.30	Resultados post-hoc de Tukey HSD para todas las métricas	159
7.31	Resumen de la prueba de Kruskal-Wallis para todas las métricas	160
7.32	Resultados post-hoc (Conover) para las métricas evaluadas	160

Dedicada a mi pequeña familia.

1 Introducción

EN LA ACTUALIDAD, el desarrollo de productos alimenticios presenta la necesidad de adaptarse a un mercado en constante evolución, caracterizado por consumidores con exigencias en relación con la variedad, calidad y valor nutricional. La ingeniería de alimentos se encuentra, por tanto, ante el desafío de innovar mediante la creación de formulaciones que integren ingredientes y aditivos a nuevos productos alimenticios. En los últimos años, esta innovación ha sido impulsada por el creciente uso de modelos de Inteligencia Artificial (IA), los cuales ofrecen herramientas para analizar y optimizar formulaciones. Para llevar a cabo estos análisis, es fundamental contar con datos detallados y representativos de los productos disponibles en el mercado obtenidos, en parte, a través del análisis de las etiquetas de ingredientes. Este tipo de información brinda una visión única sobre los ingredientes y aditivos comúnmente empleados en la industria, así como sobre las preferencias de formulación que predominan en los productos comerciales actuales.

Sin embargo, en ocasiones, el uso de estos datos en investigación y desarrollo se ve limitado por el carácter no estructurado y heterogéneo de las fuentes disponibles. La información obtenida de etiquetas de productos y bases de datos colaborativas (*crowdsourcing*), aunque rica en detalles, no siempre se encuentra en formatos que faciliten su análisis y dificulta su integración en modelos para el diseño de alimentos. En este contexto, el uso de técnicas de Procesamiento de Lenguaje Natural (PLN) se presenta como una herramienta prometedora para transformar y estructurar estos datos textuales, facilitando su uso en sistemas de clasificación y modelado de ingredientes y aditivos.

El presente trabajo explora cómo los modelos de PLN, específicamente a través de técnicas de *word embeddings* y reconocimiento de entidades nombradas (*NER*, por sus siglas en inglés), pueden aprovechar los datos no estructurados de las etiquetas de productos alimenticios. Mediante el uso de modelos como *GPT*, *Word2vec* y *GLiNER*, se busca extraer patrones semánticos entre

ingredientes y aditivos, generando un flujo de trabajo para la estructuración de datos textuales en el marco del diseño de productos en la industria alimentaria. Este enfoque representa un avance con mira hacia la integración de datos colaborativos y etiquetas comerciales en un marco analítico, promoviendo la comprensión de las formulaciones comerciales y estableciendo un camino hacia una innovación alimentaria.

2 Justificación

Contents

2.1	De la procedencia de los datos	29
2.2	De los métodos de modelado	31

2.1 De la procedencia de los datos

DADA LA DEMANDA en la ingeniería de alimentos por explorar nuevas combinaciones de ingredientes y aditivos para el desarrollo de productos innovadores, resulta útil incorporar datos provenientes de productos alimenticios del mercado actual. Estos datos pueden obtenerse a partir de la información proveniente del etiquetado de ingredientes de productos comercializados, lo cual ofrece un panorama representativo de las prácticas y preferencias actuales en la formulación de alimentos. Sin embargo, un desafío importante radica en obtener datos de diversas fuentes y en integrarlos con otras fuentes de información relevantes, debido a la limitada interoperabilidad entre bases de datos de libre acceso.

Actualmente, existen diversas bases de datos¹ que contienen información relevante sobre la composición y etiquetado de alimentos, así como sobre las marcas de los productos alimenticios. Por ejemplo, algunas industrias alimenticias optan por compartir voluntariamente información sobre la composición de sus productos; un caso representativo es el proyecto de la USDA². Además, existen bases de datos construidas mediante métodos de colaboración abierta (conocidos en inglés como *crowdsourcing*), que permiten a los consumidores recopilar y compartir datos sobre la composición de alimentos mediante aplicaciones para dispositivos móviles o plataformas en línea. Estos métodos de recolección colaborativa han aumentado considerablemente la disponibilidad de datos públicos sobre ingredientes y aditivos en productos comerciales.

¹ I. Pravst, M. Hribar, K. Žmitek, B. Blažica, B. Koroušić Seljak, and A. Kušar, “Branded Foods Databases as a Tool to Support Nutrition Research and Monitoring of the Food Supply: Insights From the Slovenian Composition and Labeling Information System,” *Frontiers in Nutrition*, vol. 8, Jan. 2022

² FoodData Central, “Global Branded Food Products Database Documentation and Download User Guide.” Online, April 2023

Estas fuentes de datos resultan de interés para el estudio, ya que contienen información sobre ingredientes y aditivos empleados en productos comercializados. Es importante notar que la investigación en el diseño de alimentos ha tendido a enfocarse en la innovación culinaria y gastronómica, utilizando principalmente recetas caseras. Si bien este enfoque ha contribuido al desarrollo de nuevos conceptos gastronómicos, ha prestado relativamente poca atención a los productos comerciales que emplean ingredientes específicos, como los aditivos, los cuales desempeñan un papel esencial en la estabilidad y en las propiedades fisicoquímicas de los alimentos procesados.

De acuerdo con el *Codex Procedural Manual* (15ª edición)³, un aditivo alimentario se define como:

“Cualquier sustancia que normalmente no se consuma como alimento por sí misma y que normalmente no se utilice como un ingrediente típico del alimento, tenga o no valor nutritivo, cuya adición intencionada a los alimentos con un propósito tecnológico (incluido el organoléptico) en la fabricación, procesamiento, preparación, tratamiento, envasado, embalaje, transporte o almacenamiento de dicho alimento dé lugar, o pueda razonablemente esperarse que dé lugar (directa o indirectamente), a que dicha sustancia o sus subproductos se conviertan en un componente del alimento o afecten de otra manera las características de dicho alimento. El término no incluye contaminantes ni sustancias añadidas a los alimentos para mantener o mejorar sus cualidades nutricionales.”

³ Codex Alimentarius, “Glossary of Terms: Food Additive.” <https://www.fao.org/fao-who-codexalimentarius/glossary/en/>, 2024. [En línea; consultado el 04 de diciembre de 2024]

Además de los aditivos, en la industria alimentaria también se integran sustancias como los *ingredientes funcionales*, que se añaden con el objetivo de proporcionar beneficios específicos para la salud más allá de los requerimientos nutricionales básicos. Ejemplos comunes incluyen probióticos, prebióticos, péptidos bioactivos y compuestos antioxidantes. Asimismo, están los *nutrientes añadidos*, que comprenden vitaminas, minerales, aminoácidos o lípidos esenciales incorporados a los alimentos para fortalecer o enriquecer su perfil nutricional.

Esta orientación hacia las recetas caseras en muchos estudios de innovación alimentaria implica la omisión de ingredientes comúnmente usados en formulaciones comerciales, como los aditivos, lo cual limita la comprensión integral de las propiedades funcionales de los productos disponibles en el mercado.

2.2 De los métodos de modelado

Para aprovechar los datos textuales sobre ingredientes y aditivos, es esencial emplear técnicas de *Procesamiento de Lenguaje Natural* (PLN) que permitan transformar esta información en un espacio vectorial interpretable, donde las palabras puedan representarse cuantitativamente. A tal fin, se propone generar vectores distribuidos (*word embeddings*) para capturar las relaciones semánticas entre ingredientes y aditivos.

En este trabajo, se elige el modelo Word2Vec para la generación de *word embeddings*, dada su capacidad para captar patrones semánticos en textos. Word2Vec ofrece ventajas en términos de precisión en las relaciones semánticas cuando se entrena con vocabularios específicos, como es el caso de los ingredientes alimenticios, permitiendo una representación contextualizada de los términos.

Para garantizar la limpieza de los datos antes de la generación de los *embeddings*, se propone realizar un ajuste (*fine-tuning*) de un modelo de reconocimiento de entidades nombradas (NER, por sus siglas en inglés) con el objetivo de clasificar y segmentar automáticamente ingredientes y aditivos. Esto facilitará el análisis de cada componente en modelos separados y permitirá un estudio detallado de sus propiedades y relaciones dentro de un entorno controlado de aprendizaje automático.

El modelo NER, denominado GLiNER, ha sido seleccionado para esta tarea debido a su naturaleza como un modelo de reconocimiento de entidades nombradas generalizado; esto quiere decir que, a diferencia de modelos NER especializados en un solo tipo de entidad, GLiNER fue entrenado para identificar una amplia gama de entidades en distintos contextos, lo que permite reconocer términos específicos del ámbito alimentario, como ingredientes y aditivos, con alta precisión. Esta flexibilidad es especialmente útil para categorizar y clasificar ingredientes y aditivos en textos variados, permitiendo un análisis estructurado de cada componente en el contexto de la industria alimentaria.

Finalmente, se plantea emplear tres enfoques de clasificación—regresión logística, máquinas de soporte vectorial (SVM) y una red neuronal— para evaluar el desempeño de estos modelos sobre el conjunto de datos. La comparación de métricas de desempeño entre estos modelos permitirá identificar el enfoque más robusto y adaptarlo al análisis de los datos textuales, lo cual es clave para establecer una metodología confiable en el contexto de la ingeniería de alimentos.

A través del uso de bases de datos colaborativas y del PLN, es posible aprovechar la información contenida en los etiquetados de productos comerciales. La incorporación de modelos como Word2vec y GLiNER en el procesamiento de datos textuales facilita la exploración y representación precisa de ingredientes y aditivos en el mercado actual, superando limitaciones previas asociadas al enfoque en recetas caseras.

3 Problema

Contents

3.1	Desarrollo del problema	33
3.1.1	Pregunta de investigación	36
3.2	Objetivos	36
3.2.1	Objetivo general	36
3.2.2	Objetivos específicos	36
3.2.3	Hipótesis	37
3.2.4	Hipótesis Nula	37
3.2.5	Novedad científica, tecnológica o aportación	37
3.2.6	Futuras líneas de trabajo	38

3.1 Desarrollo del problema

El diseño de productos alimenticios industriales enfrenta múltiples desafíos debido a la complejidad propia de los alimentos, los cuales presentan diversas características organolépticas y descriptivas como sabor, color, textura, olor, entre otras¹. Además, el diseño tradicional de productos alimenticios se ha basado en *enfoques secuenciales y metodologías de ensayo y error*, lo que conlleva un proceso iterativo que puede resultar costoso y prolongado². El diseño de alimentos, desde la generación de ideas hasta la optimización, enfrenta riesgos de fallas que incrementan costos y retrasos, limitaciones críticas en un entorno industrial que exige innovación constante³.

La complejidad en el desarrollo de alimentos radica en la necesidad de considerar simultáneamente una multitud de factores interrelacionados. Estos incluyen la composición química, las estructuras internas (como emulsiones y geles), las propiedades fisicoquímicas, los procesos de preparación, la vida útil y las aplicaciones del producto final. Predecir con precisión cómo un cambio en uno de ellos afectará al producto final se vuelve complejo.

¹ D. Park, K. Kim, S. Kim, M. Spranger, and J. Kang, "FlavorGraph: a large-scale food-chemical graph for generating food representations and recommending food pairings," *Sci Rep*, vol. 11, Dec. 2021

² X. Zhang, T. Zhou, L. Zhang, K. Y. Fung, and K. M. Ng, "Food Product Design: A Hybrid Machine Learning and Mechanistic Modeling Approach," *Ind Eng Chem Res*, vol. 58, pp. 16743–16752, Sept. 2019

³ M. Al-Sarayreh, M. Gomes Reis, A. Carr, and M. M. dos Reis, "Inverse design and AI/Deep generative networks in food design: A comprehensive review," *Trends in Food Science and Technology*, vol. 138, pp. 215–228, Aug. 2023

En este contexto, la incorporación de modelos basados en Aprendizaje de Máquina (*Machine Learning*) y Aprendizaje Profundo (*Deep Learning*) se presenta como una opción para ofrecer nuevos marcos de análisis. Estos enfoques permiten analizar y comprender las complejas relaciones entre las múltiples variables involucradas en el diseño de alimentos, proporcionando un marco para la innovación en el ámbito de los alimentos. Sin embargo, la implementación de estos enfoques enfrenta un obstáculo significativo: *la obtención y estructuración de datos sobre las características de los alimentos*.

En este sentido, los datos de entrada para un modelo son útiles cuando proporcionan información estructurada y de calidad sobre el alimento o producto final a diseñar, como fórmulas químicas, propiedades (organolépticas, nutricionales, físicas, químicas, fisicoquímicas, económicas, culturales, etc.), así como mediciones o datos textuales que describan ingredientes y procesos específicos.

Existen diversas fuentes de *datos estructurados* que ofrecen información relevante sobre alimentos. Entre ellas, se destacan las bases de datos de composición de alimentos, como FOODB, USDA FoodData Central, la base de datos de Nueva Zelanda (NZFCD), la de Australia (AFCD), la Nacional de Suecia, la de Malasia (MYFCD) y la de los Países Bajos. Estas fuentes, junto con My Food Data, Food Data Central e INFOODS, proporcionan detalles sobre la composición de los alimentos, incluyendo los compuestos y nutrientes que contienen. Además, hay bases de datos especializadas en productos alimenticios, como CalorieKing, USDA FoodData Central y OpenFoodFacts. También existen conjuntos de datos específicos sobre la composición química, fisicoquímica y características organolépticas, como Super Natura II y FlavorDB, así como bases de datos sobre aditivos, como Codex Alimentarius y Food Additive Database⁴.

Por otro lado, existen fuentes de *datos no estructurados* provenientes de sitios web tales como las recetas de alimentos que contienen información a partir de las propiedades visuales del producto final (por ejemplo, a través de fotografías), los ingredientes, las cantidades, las unidades, los procesos y relaciones entre ingredientes, el flujo y orden de las instrucciones, los datos nutricionales, las preferencias y valoraciones de los consumidores. Estos recursos están en forma de *datos no estructurados* y requieren un procesamiento antes de poder ser utilizados en un marco de inteligencia artificial. Estos datos han demostrado ser útiles para analizar las preferencias alimentarias según características regionales de alimentos, guiar preferencias de consumo, generar recetas automáticamente e identificar ingredientes esenciales en diversas preparaciones, como se expone en el **Capítulo 4**.

⁴ M. Al-Sarayreh, M. Gomes Reis, A. Carr, and M. M. dos Reis, "Inverse design and AI/Deep generative networks in food design: A comprehensive review," *Trends in Food Science and Technology*, vol. 138, pp. 215–228, Aug. 2023

Sin embargo, existen algunas limitaciones en relación con las fuentes de datos mencionadas anteriormente. Por ejemplo, una limitación de los conjuntos de *datos estructurados* de compuestos químicos y moléculas de sabor es que el número de alimentos es reducido, con solo 936 (siendo aún menos excluyendo aquellos alimentos que son preparaciones). En cuanto a los *datos no estructurados* obtenidos de recetas, se enfrentan a la limitación de que los procesos, ingredientes y cantidades difieren significativamente de los procedimientos industriales. Además, los ingredientes listados en productos comerciales incluyen no solo alimentos, sino también aditivos, que son esenciales en los procesos de elaboración de productos alimenticios tecnológicos e industriales.

Aunque los datos más completos disponibles son los relacionados con la composición nutricional, es importante destacar que las características nutricionales por sí solas, no determinan el diseño de alimentos. Es necesario adoptar una visión integral que incorpore una gama más amplia de características valiosas sobre la formulación de alimentos.

Acceder a formulaciones de productos alimenticios en el ámbito industrial es casi imposible, debido a que las formulaciones y procedimientos industriales no son de conocimiento público. Sin embargo, una de las fuentes que permite acercarse a la comprensión de la formulación de un producto es el listado de ingredientes presente en el etiquetado de alimentos y bebidas. Según la Organización de las Naciones Unidas para la Alimentación y la Agricultura (FAO), el etiquetado de alimentos es uno de los medios más relevantes para transmitir información sobre los ingredientes, la calidad y el valor nutricional⁵.

En este estudio, se plantea que explorar los datos no estructurados de las etiquetas de productos alimenticios puede ofrecer una solución innovadora para abordar estas limitaciones. Estos datos contienen información valiosa sobre ingredientes y aditivos, que podría ser fundamental para identificar patrones y relaciones útiles en el diseño de nuevos productos.

Para validar esta propuesta, se diseñaron tres escenarios experimentales que evalúan distintas técnicas de procesamiento de datos. Estos escenarios permiten analizar cómo las técnicas propuestas impactan en las métricas clave de desempeño de los modelos, proporcionando evidencia para validar la hipótesis planteada.

⁵ Organización de las Naciones Unidas para la Alimentación y la Agricultura (FAO), "Etiquetado de alimentos," 2024. [En línea; consultado el 30 de marzo de 2024]

3.1.1 *Pregunta de investigación*

¿Existe una diferencia significativa en las métricas de desempeño (*accuracy*, *precision*, *recall* y *f1-score*) de los modelos de clasificación de alimentos entre los distintos escenarios experimentales definidos en el estudio?

3.2 *Objetivos*

3.2.1 *Objetivo general*

Evaluar el impacto de diferentes escenarios de procesamiento de datos, implementados mediante técnicas de Procesamiento de Lenguaje Natural (NLP), en las métricas de desempeño de modelos de clasificación de alimentos, utilizando análisis estadísticos adecuados para determinar diferencias significativas.

3.2.2 *Objetivos específicos*

- Establecer un escenario de control (*baseline*) con el uso de los datos originales sin preprocesar, sin segmentar idiomas, ni utilizar modelos como GPT o GLiNER, con el propósito de comparar los resultados y evaluar el impacto del modelo propuesto en este estudio.
- Utilizar un modelo *Generative Pre-trained Transformer (GPT)* para la generación de resumen *summarization* y estructuración de datos no estructurados, con el objetivo de generar un conjunto de datos estructurados para el ajuste (*fine-tuning*).
- Reentrenar al modelo llamado *GLiNER* con el conjunto de datos generado con GPT para ajustar el Reconocimiento de Entidades Nombradas (NER), diferenciando entre ingredientes y aditivos alimenticios.
- Generar nuevas listas de ingredientes alimenticios basadas en el reconocimiento ajustado de entidades nombradas, para entrenar un modelo que genere representaciones vectoriales de palabras (*word embeddings*).
- Identificar y segmentar los diferentes idiomas presentes en los textos, asegurando la homogeneidad del conjunto de datos antes de su procesamiento.
- Probar, al menos, dos técnicas para integrar la representación vectorial de ingredientes en listas de ingredientes.
- Probar y evaluar técnicas de balanceo de datos para mejorar la representatividad de las clases en el conjunto de datos.

- Realizar el preprocesamiento de datos no estructurados mediante técnicas como tokenización, eliminación de *stop words* y otras herramientas de limpieza, para preparar un conjunto de datos inicial.
- Integrar los datos estructurados en un modelo de Machine Learning capaz de clasificar alimentos, con el objetivo de apoyar el proceso de diseño y desarrollo de productos alimenticios.

3.2.3 Hipótesis

El modelo propuesto, que implementa técnicas de Procesamiento de Lenguaje Natural (NLP) en un escenario completo, tiene un impacto **significativo** en las métricas de desempeño (*accuracy*, *precision*, *recall* y *f1-score*) de los modelos de clasificación de alimentos, en comparación con los escenarios alternativos.

3.2.4 Hipótesis Nula

El modelo propuesto, que implementa técnicas de Procesamiento de Lenguaje Natural (NLP) en un escenario completo, **no tiene un impacto significativo** en las métricas de desempeño (*accuracy*, *precision*, *recall* y *f1-score*) de los modelos de clasificación de alimentos, en comparación con los escenarios alternativos.

3.2.5 Novedad científica, tecnológica o aportación

Este estudio introduce un enfoque innovador en el uso de datos *no estructurados* en la industria alimentaria mediante técnicas avanzadas de Procesamiento de Lenguaje Natural (PLN). A continuación, se destacan las principales aportaciones de esta investigación:

- **Obtención y mejora de datos sobre aditivos alimentarios:**
 - Implementación de *web scraping* en Wikipedia para recopilar información sobre aditivos alimentarios, generando un conjunto de datos útil para clasificaciones posteriores.
- **Modelos de PLN adaptados al contexto alimentario:**
 - Entrenamiento y ajuste del modelo *GLiNER* para el Reconocimiento de Entidades Nombradas (NER), logrando la clasificación de ingredientes y aditivos en listados textuales.
 - Integración de un modelo *Generative Pre-Trained Transformer (GPT)* para transformar y estructurar datos no estructurados, mejorando la organización y análisis de información alimentaria.

- **Mejora de datos alimentarios:**

- Aplicación de técnicas como segmentación de idiomas, limpieza de texto y balanceo de datos para generar conjuntos de datos consistentes y representativos.
- Construcción de representaciones semánticas avanzadas (*word embeddings*) para ingredientes y aditivos, proporcionando un marco analítico enriquecido para el estudio de relaciones entre componentes.

- **Modelo de clasificación de alimentos:**

- Diseño, implementación y evaluación de tres enfoques de clasificación: Máquinas de Soporte Vectorial (SVM), Regresión Logística Múltiple y Redes Neuronales, demostrando cómo el procesamiento avanzado de datos mejora las métricas de desempeño (*accuracy*, *precision*, *recall*, y *f1-score*).

- **Aplicación en bases de datos colaborativas y la industria alimentaria:**

- Automatización en la clasificación de ingredientes y aditivos en bases de datos tipo *crowdsourcing*, como OpenFoodFacts.
- Potencial para reducir la dependencia de entradas manuales y mejorar la precisión, completitud y organización de datos alimentarios en estas bases, facilitando su uso en investigación y aplicaciones industriales.

3.2.6 Futuras líneas de trabajo

- **Integración de fuentes estructuradas y no estructuradas:** Este estudio establece las bases para combinar datos estructurados (como bases de datos sobre composición de alimentos y aditivos) con datos no estructurados (como listados de ingredientes en etiquetas comerciales). Este enfoque podría ampliar las posibilidades de análisis y predicción en el diseño de productos alimenticios.

4 Estado del arte

Contents

4.1	Introducción	39
4.2	Aplicaciones de Inteligencia Artificial (IA) en el ámbito de alimentos	40

4.1 Introducción

LA INTELIGENCIA ARTIFICIAL (IA) ha revolucionado el ámbito alimentario al optimizar la formulación, desarrollo y producción de alimentos. En la actualidad, estas tecnologías se estudian y aplican en el diseño de productos, facilitando desde la identificación de combinaciones de ingredientes, hasta la replicación de texturas y sabores específicos. La IA ayuda a superar las limitaciones de los métodos tradicionales, abriendo nuevas posibilidades en la creación de alimentos personalizados, sostenibles y adaptados a las demandas del mercado¹.

Los algoritmos genéticos y los modelos de predicción son ejemplos de cómo la IA contribuye a optimizar cada etapa del proceso, desde la selección de ingredientes hasta la evaluación de características sensoriales. Herramientas de Procesamiento del Lenguaje Natural (PLN) y sistemas de recomendación están transformando cómo las empresas adaptan formulaciones y productos a preferencias individuales y restricciones dietéticas, mejorando la experiencia de las y los consumidores.

Además, herramientas de IA demuestran el potencial para explorar variaciones culinarias y crear alternativas innovadoras, expandiendo la capacidad del sector para responder a las preferencias cambiantes de los consumidores. Estos desarrollos consolidan el papel de la IA como un recurso esencial para empresas que buscan mejorar la eficiencia y la creatividad en la industria alimentaria.

¹ M. Al-Sarayreh, M. Gomes Reis, A. Carr, and M. M. dos Reis, "Inverse design and AI/Deep generative networks in food design: A comprehensive review," *Trends in Food Science and Technology*, vol. 138, pp. 215–228, Aug. 2023

Un ejemplo destacado en el ámbito de la industria alimentaria es el de “Giuseppe”², la IA desarrollada por NotCo, que utiliza modelos generativos y una capa de discernimiento para replicar productos de origen animal con ingredientes vegetales, promoviendo alternativas sostenibles y nutritivas sin sacrificar la calidad sensorial del producto final. A continuación se profundiza en los estudios que han explorado modelos aplicando la IA.

² NotCo, “Conoce a Giuseppe, la tecnología que lo hace realidad.” <https://notco.com/mx/sobre/giuseppe-geek>. [En línea; consultado el 28 de julio de 2024]

4.2 Aplicaciones de Inteligencia Artificial (IA) en el ámbito de alimentos

Distintos estudios han empleado modelos de aprendizaje automático (ML, por sus siglas en inglés) y aprendizaje profundo (DL, por sus siglas en inglés) para desarrollar propuestas para el diseño de alimentos. Un ejemplo, es el enfoque propuesto por Zhang y colaboradores³, en el cual se integran algoritmos genéticos (AG) para optimizar el diseño de productos alimenticios. En esta metodología, el modelo de ML predice las calificaciones sensoriales de los productos, mientras que los AG abordan la optimización del diseño mediante la selección óptima de ingredientes y condiciones de procesamiento. Esta combinación de técnicas permite una aproximación integral al ajuste de parámetros que potencian tanto la calidad sensorial, como la eficiencia en la producción de alimentos.

³ X. Zhang, T. Zhou, L. Zhang, K. Y. Fung, and K. M. Ng, “Food Product Design: A Hybrid Machine Learning and Mechanistic Modeling Approach,” *Ind Eng Chem Res*, vol. 58, pp. 16743–16752, Sept. 2019

El proceso de optimización se formula como un problema de “caja gris”, en el cual se integran tanto modelos mecanísticos, que describen relaciones físico-químicas específicas del alimento, como modelos de ML, que capturan la percepción sensorial del consumidor. La *función objetivo* de esta optimización es maximizar q , que representa la calidad sensorial, evaluada en términos de atributos como sabor, textura y color. Para esto, se define un conjunto de *variables de diseño*, incluyendo la selección de ingredientes alimenticios (x) y las condiciones de procesamiento (pd), las cuales son ajustadas por los AG para alcanzar los valores óptimos de q .

Las restricciones de diseño limitan los valores de ciertas características físicas del producto, representadas como Z , que se calculan a partir de modelos mecanísticos. Estas restricciones se manejan mediante funciones de penalización que guían a los AG en la búsqueda de soluciones viables, asegurando que las características del producto final, como la textura o el color, cumplan con los parámetros establecidos.

La integración de los modelos de ML y mecanísticos en un enfoque de optimización que permite que el algoritmo genético explore soluciones posibles, penalizando aquellas que no cumplen con las restricciones establecidas y favoreciendo configuraciones de ingredientes y procesos que maximicen la calificación sensorial. Como puede observarse en la Figura 4.1, este proceso de optimización se aplica en el diseño de galletas con chispas de chocolate, donde el algoritmo genético demostró funcionar para encontrar una combinación de ingredientes y condiciones de horneado que alcanzó una calificación sensorial óptima de 92.8 sobre 100.

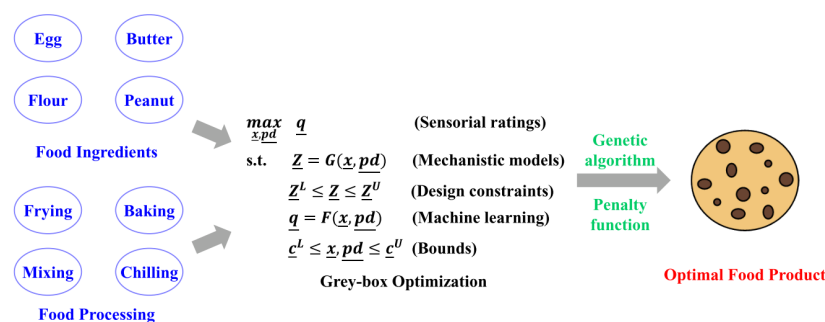


Figure 4.1: Modelo para el diseño de productos alimenticios híbrido. El objetivo es determinar la mejor combinación de ingredientes y condiciones de procesamiento para producir un alimento. Figura retomada de X. Zhang, T. Zhou, L. Zhang, K. Y. Fung, and K. M. Ng, "Food Product Design: A Hybrid Machine Learning and Mechanistic Modeling Approach".

Un enfoque adicional que complementa las aproximaciones tradicionales de diseño de alimentos es el desarrollado por Chaudhari y colaboradores⁴, quienes propusieron un algoritmo basado en técnicas de *web scraping* para extraer recetas de sitios web de cocina, almacenarlas en una base de datos y permitir búsquedas basadas en ingredientes. Este sistema utiliza Python y MongoDB para recopilar recetas automáticamente, facilitando a los chefs la búsqueda de platos que se ajusten a los ingredientes disponibles o a restricciones dietéticas específicas.

El algoritmo toma como entrada una lista de ingredientes y devuelve las recetas que contienen dichos ingredientes, lo que permite a los chefs preparar platillos en función de las necesidades de sus clientes, como en la Figura 4.2. Además, la base de datos resultante puede utilizarse para integrar recomendaciones personalizadas y optimización de recetas basadas en las preferencias nutrimentales. Este tipo de enfoque automatizado reduce la necesidad de búsquedas manuales.

Otro enfoque valioso es el introducido por RecipeScope, una herramienta desarrollada por Chang y colaboradores⁵, que ofrece una perspectiva en el análisis de recetas mediante un flujo computacional que utiliza técnicas de PLN y visualizaciones interactivas para estudiar

⁴ S. Chaudhari, A. R., V. G. Tekkur, P. G. L., and S. R. Karki, "Ingredient/Recipe Algorithm Using Web Mining and Web Scraping for Smart Chef," in *Proceedings of the IEEE CONECCT 2020: 6th International Conference on Electronics, Computing and Communication Technologies*, 2020

⁵ M. Chang, L. V. Guillaín, H. Jung, V. M. Hare, J. Kim, and M. Agrawala, "RecipeScope: An interactive tool for analyzing cooking instructions at scale," in *Conference on Human Factors in Computing Systems - Proceedings*, Association for Computing Machinery, Apr. 2018

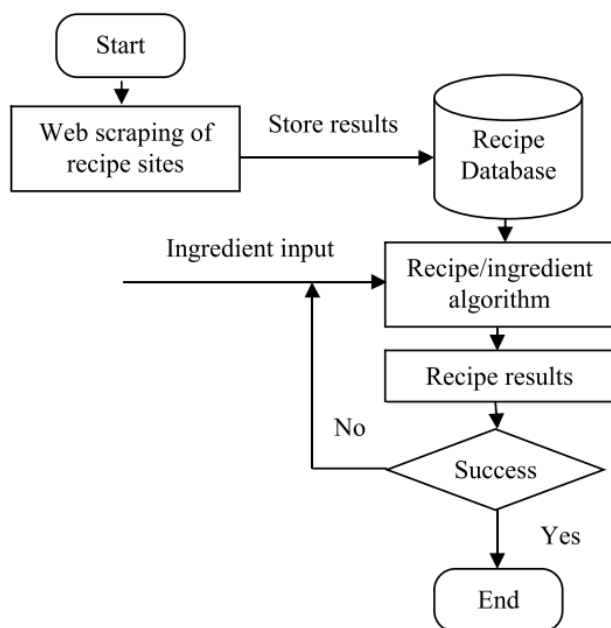


Figure 4.2: Diagrama de flujo del algoritmo de búsqueda de recetas basado en ingredientes. Figura retomada de S. Chaudhari, A. R., V. G. Tekkur, P. G. L., and S. R. Karki, "Ingredient/Recipe Algorithm Using Web Mining and Web Scraping for Smart Chef," in Proceedings of the IEEE CONECCT 2020: 6th International Conference on Electronics, Computing and Communication Technologies, 2020.

la estructura de recetas de cocina a gran escala. En lugar de centrarse únicamente en la optimización de ingredientes o valores nutricionales, RecipeScape permite la identificación de *patrones de uso de ingredientes y técnicas culinarias de recetas*, lo que proporciona un enfoque basado en datos para la creación e interpretación de platillos.

A través del uso de técnicas de análisis de árboles y puntajes que permiten analizar distancias entre recetas, esta herramienta integra automáticamente las recetas en agrupamientos (*clusters*) basados en similitudes estructurales, lo que facilita la comparación entre diferentes métodos de preparación de un mismo platillo. Además, RecipeScape permite a las personas usuarias identificar recetas "estándar" o "exóticas" y explorar variaciones creativas de un platillo, lo que lo convierte en una herramienta para el diseño de alimentos. La integración de RecipeScape en el análisis de alimentos se alinea con las metodologías de DL y ML utilizadas en otros enfoques, como el trabajo de Lei y colaboradores⁶, en el que se entrenaron modelos para optimizar recetas en función de nutrientes.

Un estudio adicional que explora el uso de técnicas de PLN para el diseño y generación de recetas es el desarrollado por Pan y colaboradores⁷. En este trabajo, se emplearon modelos de *word embeddings* a partir de *Skip-gram* para medir la similitud entre ingredientes y ayudar a realizar sustituciones cuando ciertos ingredientes no están disponibles. Además, se empleó una técnica de

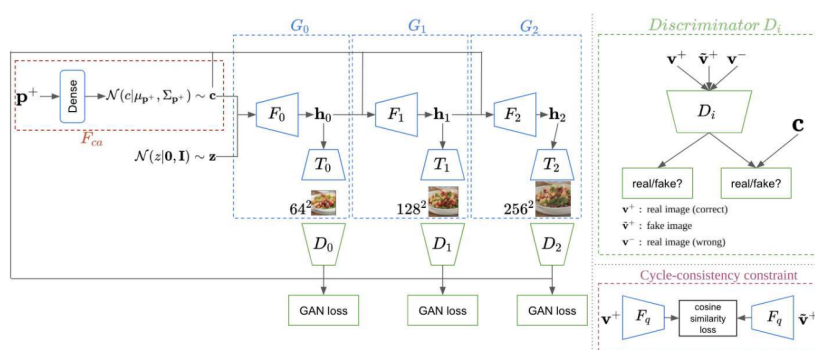
⁶ Z. Lei, A. U. Haq, M. Dorraki, D. Zhang, and D. Abbott, "Composing recipes based on nutrients in food in a machine learning context," *Neurocomputing*, vol. 415, pp. 382–396, 2020

⁷ Y. Pan, Q. Xu, and Y. Li, "Food recipe alternation and generation with natural language processing techniques," in *Proceedings - 2020 IEEE 36th International Conference on Data Engineering Workshops, ICDEW 2020*, pp. 94–97, Institute of Electrical and Electronics Engineers Inc., Apr. 2020

representación de lenguaje a partir de N-gram y el modelo de redes neuronales recurrentes (RNN) con capas de memoria a largo plazo (LSTM) para generar nuevas recetas manteniendo el estilo culinario original. Los resultados mostraron que los modelos basados en RNN/LSTM generaban recetas más coherentes en comparación con los modelos tradicionales, lo que destaca el potencial del PLN en la generación automática de nuevas combinaciones de alimentos.

De manera similar, Han y colaboradores⁸ presentaron CookGAN, como se observa en la Figura 4.3, se trata de un modelo generativo profundo diseñado para la síntesis de imágenes fotorrealistas de platillos a partir de listas de ingredientes. Este enfoque se centra en la generación visual de alimentos mediante redes generativas adversarias (GAN) y un modelo de atención que determina qué ingredientes son clave en la apariencia del platillo. CookGAN incorpora una restricción de consistencia cíclica, que asegura que las imágenes generadas correspondan fielmente a los ingredientes dados, abordando las complejidades visuales asociadas con la variedad de métodos de preparación y apariencia de los alimentos.

Este modelo ofrece una dimensión visual al proceso de diseño de alimentos, ya que permite visualizar nuevas combinaciones de ingredientes antes de implementarlas físicamente. Además, complementa otras herramientas como RecipeScape, al proporcionar una representación visual de los platillos, lo que facilita la experimentación con nuevas recetas y combinaciones de alimentos.



⁸ F. Han, R. Guerrero, and V. Pavlovic, "Cookgan: Meal image synthesis from ingredients," 2020

Figure 4.3: Marco del modelo generativo con múltiples ramas: El modelo consta de tres generadores (G_0, G_1, G_2) que crean imágenes en diferentes resoluciones ($64^2, 128^2, 256^2$). A cada generador le corresponde un discriminador (D_0, D_1, D_2) que evalúa la autenticidad de las imágenes, clasificándolas como reales o falsas. El proceso está regulado por una pérdida de consistencia cíclica y una pérdida de similitud coseno, garantizando coherencia entre las imágenes generadas y las reales. Figura retomada de F. Han, R. Guerrero, and V. Pavlovic, "Cookgan: Meal image synthesis from ingredients".

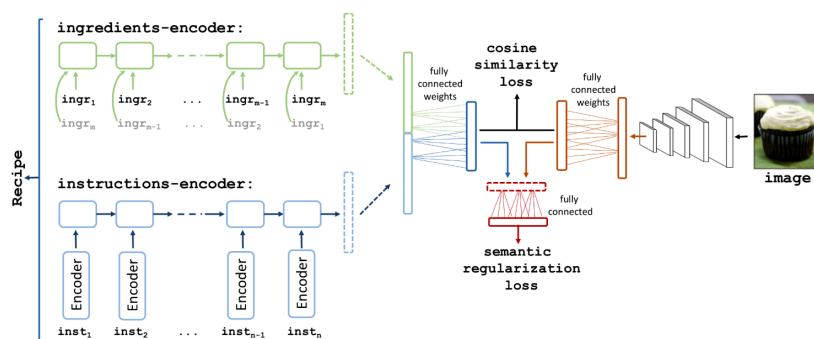
Por su lado, Lei y colaboradores⁹, desarrollaron un enfoque basado en DL para generar recetas optimizadas en función de los nutrientes. En este enfoque, se utilizó un *autoencoder* para representar recetas junto con métodos tradicionales como la factorización de matrices no negativas

⁹ Z. Lei, A. U. Haq, M. Dorraki, D. Zhang, and D. Abbott, "Composing recipes based on nutrients in food in a machine learning context," *Neurocomputing*, vol. 415, pp. 382–396, 2020

(NMF) y los mínimos cuadrados regularizados en dos pasos (TRLS). A través de estos métodos, se identificaron combinaciones de alimentos que no solo optimizan el sabor, sino también la calidad nutricional de los platillos. La incorporación de la norma de Frobenius y matrices dispersas no suaves en la NMF permitió mejorar la estabilidad del modelo y reducir el riesgo de sobreajuste durante el entrenamiento.

En este caso, los modelos se entrenaron para proyectar alimentos y nutrientes en un espacio de características, permitiendo predecir cuáles combinaciones de ingredientes son más propensas a ser saludables y agradables al paladar. El uso de un enfoque basado en DL y la integración de matrices de nutrientes permitieron que las combinaciones no solo estuvieran basadas en el sabor, sino también en los beneficios para la salud, a partir de un enfoque nutricional.

En línea con estos enfoques, el estudio de Marín y colaboradores¹⁰ aporta una nueva dimensión al integrar imágenes y recetas mediante DL. Su enfoque con Recipe1M+, como se observa en la Figura 4.4 y Figura 4.5, se trata de un conjunto de datos multimodal que combina más de un millón de recetas con 13 millones de imágenes de alimentos, permitió entrenar redes neuronales que capturan la relación entre las características visuales de los platillos y sus ingredientes e instrucciones. El sistema genera embeddings conjuntos de recetas e imágenes, facilitando la recuperación de recetas a partir de imágenes y viceversa, abordando un problema fundamental en el diseño y la adaptación de alimentos.



Este enfoque permite la creación de nuevas combinaciones de alimentos mediante *aritmética semántica* en el espacio de embeddings. Por ejemplo, la transformación de "pizza de pollo" en "ensalada de pollo" se puede realizar mediante *operaciones vectoriales*. Esto destaca cómo se puede explorar nuevas combinaciones y adaptaciones de platillos, complementando los métodos de diseño actuales.

¹⁰ J. Marin, A. Biswas, F. Ofli, N. Hynes, A. Salvador, Y. Aytar, I. Weber, and A. Torralba, "Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images," 2019

Figure 4.4: Modelo de incrustación neuronal conjunta con regularización semántica: Este modelo aprende un espacio de incrustación compartido entre imágenes de alimentos y recetas de cocina. El encoder de ingredientes (arriba) procesa una secuencia de ingredientes, mientras que el encoder de instrucciones (abajo) procesa las instrucciones de la receta. Ambos encoders generan representaciones que se combinan mediante capas completamente conectadas. Figura retomada de J. Marin, A. Biswas, F. Ofli, N. Hynes, A. Salvador, Y. Aytar, I. Weber, and A. Torralba, "Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images".

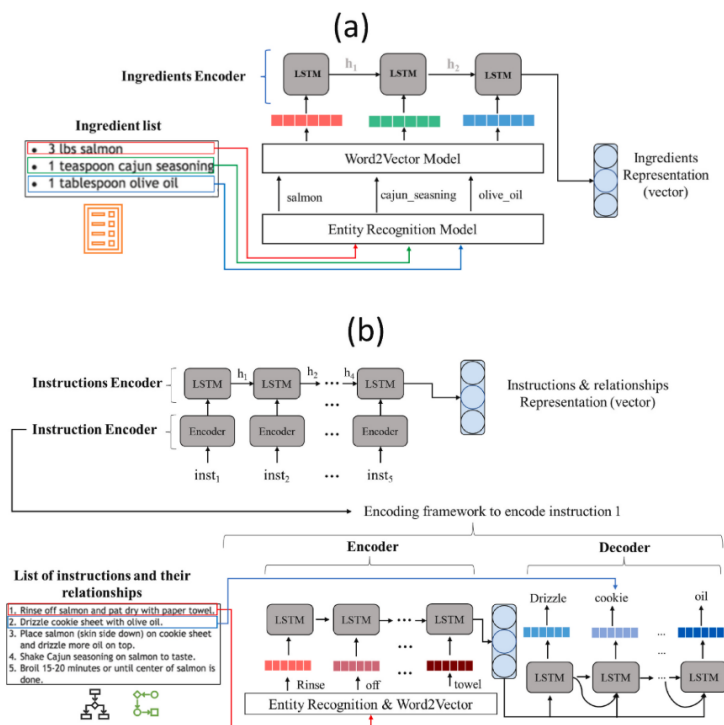


Figure 4.5: Modelo de codificación de recetas: Este diagrama muestra dos marcos de trabajo. (a) Codificación de ingredientes, que utiliza reconocimiento de entidades, vectores numéricos y un modelo LSTM para representar los ingredientes de una receta. (b) Codificación de instrucciones, que usa LSTM para procesar secuencias de instrucciones y sus relaciones, prediciendo el siguiente paso en función del anterior. Figura retomada de M. Al-Sarayreh, M. Gomes Reis, A. Carr, and M. M. dos Reis, "Inverse design and AI/Deep generative networks in food design: A comprehensive review".

Por otro lado, Anderson¹¹ analizó diversas técnicas de ML y DL aplicadas a los sistemas de recomendación de alimentos. Las metodologías analizadas incluyen filtrado colaborativo, sistemas basados en contenido, enfoques híbridos, redes de grafos y modelos de embeddings.

El filtrado colaborativo es una técnica que sugiere ítems (alimentos, recetas, restaurantes) basándose en patrones de comportamiento similares entre personas usuarias. Este enfoque emplea algoritmos como el de factorización de matrices (SVD - Singular Value Decomposition) para encontrar relaciones latentes entre personas usuarias e ítems, lo que permite hacer recomendaciones personalizadas sin requerir un análisis detallado del contenido de los ítems. Sin embargo, esta metodología presenta limitaciones importantes, como el problema de arranque en frío (*cold start*), donde las recomendaciones son ineficaces para nuevas personas usuarias o nuevos ítems que carecen de interacciones previas.

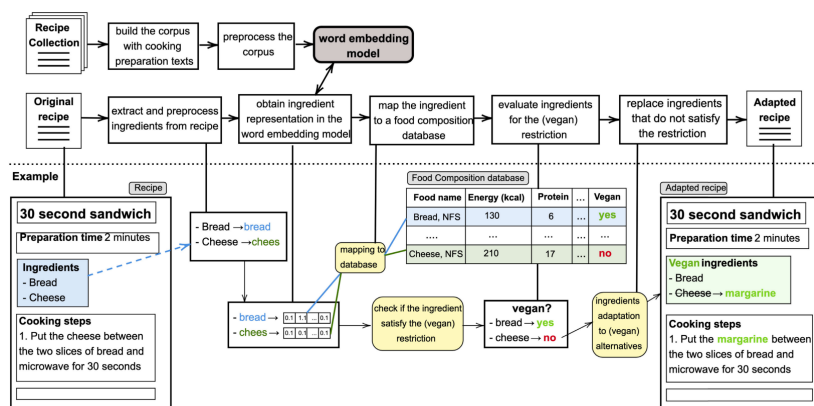
Los modelos basados en contenido se apoyan en el uso de técnicas como el PLN, que permite analizar las descripciones textuales de las recetas o platos. Un ejemplo de esta aplicación es el uso de técnicas como bag of words o TF-IDF para representar descripciones de alimentos y generar recomendaciones en función de los ingredientes

¹¹ C. Anderson, "A survey of food recommenders," 2018

que la persona usuaria ha preferido previamente. Los enfoques híbridos combinan las ventajas del filtrado colaborativo y los sistemas basados en contenido para superar las limitaciones de cada uno por separado.

Además de estos enfoques híbridos, un enfoque innovador en el uso de PLN para el diseño y adaptación de alimentos es el propuesto por Morales-Garzón y colaboradores¹². En este trabajo, se generaron *word embeddings* entrenados a partir de un corpus de recetas, no solo basado en listas de ingredientes, sino también en instrucciones de cocina, lo que permite capturar de forma precisa las relaciones entre ingredientes y técnicas culinarias. Este modelo no supervisado permitió la adaptación automática de recetas a las preferencias nutricionales y dietéticas de las personas usuarias, como la creación de versiones vegetarianas o veganas de platillos.

El método de Morales-Garzón y colaboradores, como se observa en la Figura 4.6, es capaz de sugerir sustituciones de ingredientes manteniendo la coherencia semántica y culinaria del platillo original, aprovechando la información capturada en el *espacio vectorial* generado por el modelo. Además, la integración de una base de datos nutricional permitió mejorar las adaptaciones considerando no solo el sabor, sino también el valor nutricional, un aspecto clave en el diseño de alimentos.



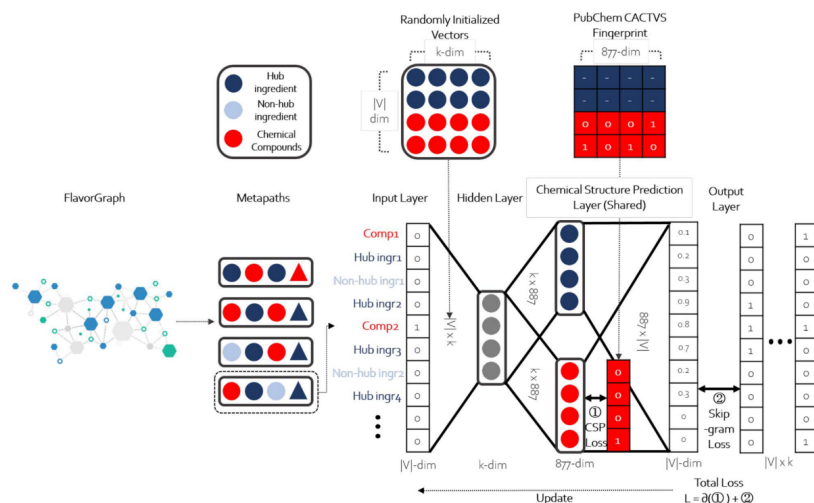
Este enfoque complementa otros métodos de diseño de alimentos basados en DL, al proporcionar una solución no supervisada para la adaptación de recetas y la creación de propuestas personalizadas de alimentos. La capacidad de capturar relaciones semánticas y culinarias entre ingredientes amplía las posibilidades de creación y adaptación de platillos, reduciendo la necesidad de intervenciones manuales.

¹² A. Morales-Garzón, J. Gómez-Romero, and M. J. Martín-Bautista, "A word embedding-based method for unsupervised adaptation of cooking recipes," *IEEE Access*, vol. 9, pp. 27389–27404, 2021

Figure 4.6: Adaptación de recetas para restricciones dietéticas: Este diagrama muestra el proceso para ajustar recetas según restricciones, como el veganismo. Los ingredientes se analizan y los que no cumplen con la restricción son reemplazados por alternativas adecuadas, como cambiar queso por margarina en un sándwich. Figura retomada de A. Morales-Garzón, J. Gómez-Romero, and M. J. Martín-Bautista, "A word embedding-based method for unsupervised adaptation of cooking recipes"

Otro avance significativo también es el desarrollado en FlavorGraph¹³, que utiliza grafos a gran escala para modelar las relaciones entre ingredientes y compuestos químicos, permitiendo mejorar tanto la representación de alimentos como las recomendaciones de combinaciones alimenticias, como se observa en la Figura 4.7. En este enfoque, se utiliza el modelo *metapath2vec* con una capa adicional de aprendizaje químico, lo que permite generar representaciones vectoriales tanto de alimentos, como de compuestos químicos. El uso de skip-gram en este proceso permite captar relaciones complejas entre ingredientes basadas en co-ocurrencias en recetas y similitudes químicas. Este modelo ha superado otros enfoques en la tarea de agrupación y clasificación de alimentos, además de descubrir nuevas combinaciones basadas en datos químicos y estadísticos. La integración de estos enfoques permite mejorar la precisión de las recomendaciones culinarias al vincular el análisis químico con el comportamiento en recetas.

El uso de redes de grafos que modelan las relaciones entre ingredientes y recetas permite descubrir sustituciones y combinaciones. Los modelos de embeddings, como word2vec, representan alimentos en un espacio vectorial, capturando relaciones semánticas entre ingredientes y permitiendo recomendaciones novedosas.



¹³ D. Park, K. Kim, S. Kim, M. Spranger, and J. Kang, "FlavorGraph: a large-scale food-chemical graph for generating food representations and recommending food pairings," *Sci Rep*, vol. 11, Dec. 2021

Figure 4.7: Incrustación de grafos con metapath2vec+CSP en FlavorGraph: Este modelo combina grafos de ingredientes y compuestos químicos, representando sus interacciones mediante metapaths y predicciones de estructuras químicas. Se entrenan capas ocultas con vectores de ingredientes y compuestos, optimizando la predicción de las relaciones mediante pérdidas de CSP y Skip-gram, para mejorar la comprensión de las conexiones en el FlavorGraph. Figura retomada de D. Park, K. Kim, S. Kim, M. Spranger, and J. Kang, "FlavorGraph: a large-scale food-chemical graph for generating food representations and recommending food pairings".

El DL, particularmente con redes neuronales convolucionales (CNN) y redes recurrentes (RNN), tiene un gran potencial. Las CNN se aplican al análisis de imágenes de alimentos, permitiendo sugerir versiones saludables de un plato. Las RNN y los modelos de memoria a largo plazo (LSTM) son útiles para procesar secuencias en recetas, mejorando

las recomendaciones o generando recetas automáticamente. Además, los sistemas generativos, como las redes generativas adversarias (GAN), están siendo explorados para crear nuevas combinaciones de alimentos.

De acuerdo con Anderson¹⁴, los resultados muestran que los enfoques híbridos y basados en embeddings superan las metodologías tradicionales. El aprendizaje profundo (DL), especialmente en el análisis de imágenes y la generación automática de recetas, está emergiendo como una herramienta clave en la personalización de recomendaciones culinarias. Sin embargo, la escasez de datos sigue siendo un desafío.

Actualmente, NotCo, con su sistema de IA llamado Giuseppe, ha desarrollado un conjunto patentado de algoritmos que estudian y aprenden de un gran universo de combinaciones de plantas, permitiendo replicar productos de origen animal a nivel molecular, o incluso mejorar sus propiedades. Al procesar y analizar estas infinitas combinaciones de ingredientes vegetales, Giuseppe ha logrado acumular una gran cantidad de información química, molecular, física y nutricional¹⁵.

Giuseppe es capaz de generar una amplia variedad de formulaciones para un producto deseado, buscando afinidades en sabor, textura, perfil nutricional y funcionalidad.

Para expandir sus capacidades, NotCo ha integrado modelos de inteligencia artificial generativa, incluidos modelos de lenguaje de gran tamaño (*Large Language Models*, LLM), para explorar combinaciones innovadoras de sabores y aromas. No obstante, reconocen que los LLM generales pueden producir combinaciones que, aunque creativas, no siempre son adecuadas para el desarrollo de alimentos. Para resolver este desafío, NotCo ha implementado una "capa de discernimiento" en Giuseppe, una herramienta que filtra y selecciona las ideas generadas, evaluando su viabilidad y ajustándose mejor a las realidades del desarrollo alimentario¹⁶.

¹⁴ C. Anderson, "A survey of food recommenders," 2018

¹⁵ NotCo, "Conoce a Giuseppe, la tecnología que lo hace realidad." <https://notco.com/mx/sobre/giuseppe-geek>. [En línea; consultado el 28 de julio de 2024]

¹⁶ M. Wolf, "From data-scraping to discernment layer: How notco's giuseppe ai has evolved over the past decade," 2024. [En línea; consultado el 28 de octubre de 2024]

5 Marco teórico

Contents

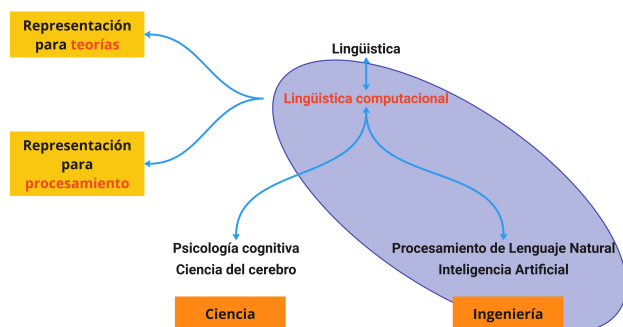
5.1	Introducción	49
5.2	El lenguaje y la ambigüedad	51
5.3	Representaciones vectoriales del lenguaje	53
5.4	Fundamentos matemáticos en la representación del lenguaje	56
5.4.1	Probabilidad y espacio latente en el modelado de lenguaje	57
5.5	Modelos de lenguaje iterativos basados en redes neuronales	62
5.5.1	Contexto local y global	62
5.6	Arquitecturas de modelos de lenguaje neuronales	63
5.6.1	Word2Vec	65
5.6.2	Transición hacia la arquitectura de Transformers	71
5.6.3	Transformers	78
5.7	Tareas específicas	91
5.7.1	Summarization	91
5.7.2	Named Entity Recognition (NER)	91
5.7.3	Text Classification	92
5.8	ANOVA: Análisis de Varianza	92
5.8.1	Modelo estadístico	92
5.8.2	Formulación de la hipótesis	92
5.8.3	Fórmulas del ANOVA	93
5.8.4	Criterio de decisión	94

5.1 Introducción

LA LINGÜÍSTICA es la disciplina que se dedica al estudio del lenguaje como un sistema complejo, abarcando desde las reglas gramaticales que estructuran los idiomas, hasta la manera en que los seres humanos perciben y organizan su experiencia del mundo a través

del lenguaje¹. Dentro de este campo, la *lingüística computacional* (LC) se enfoca en la formalización y modelización de estas reglas mediante herramientas matemáticas y computacionales, permitiendo una comprensión estructurada del lenguaje. Así, desde la LC se han desarrollado *marcos formales* para describir las reglas del lenguaje, las cuales se organizan en diferentes capas, como la morfología, la sintaxis y la semántica. Cada una de estas capas requiere marcos formales y diferentes modelos computacionales para su análisis. Además de la lingüística, la Neurociencia y la Psicología también juegan un papel importante en el entendimiento del lenguaje, al estudiar cómo los seres humanos procesan y comprenden el lenguaje en el cerebro y en el comportamiento humano².

Por otro lado, el *Procesamiento de Lenguaje Natural* (PLN) es una disciplina de la ingeniería cuyo objetivo es construir tecnologías capaces de procesar el lenguaje humano con el fin de resolver tareas útiles³; como rama de la Inteligencia Artificial (IA), el PLN aplica diversas técnicas, entre las que se incluyen el modelado estadístico, el aprendizaje automático (Machine Learning o ML por sus siglas en inglés), el aprendizaje profundo (Deep Learning o DL por sus siglas en inglés) y el análisis sintáctico y semántico. Estas metodologías permiten descomponer y analizar el lenguaje en sus componentes esenciales⁴, facilitando una comprensión de su *estructura y significado*.



Para que los algoritmos de ML y DL puedan trabajar con datos de texto, éstos deben de transformarse en alguna *representación matemática*. Este proceso convierte el texto crudo en una forma numérica apropiada para su análisis⁵.

A continuación, se profundiza en uno de los desafíos más complejos en el tratamiento del lenguaje desde una perspectiva computacional: *la ambigüedad lingüística*. Analizar esta problemática es fundamental para entender cómo los modelos de PLN deben abordar las múltiples interpretaciones posibles de una misma expresión, tanto a nivel sintáctico, como semántico.

¹ J. Tsujii, "Natural Language Processing and Computational Linguistics," *Computational Linguistics*, vol. 47, pp. 707–727, 12 2021

² J. Tsujii, "Natural Language Processing and Computational Linguistics," *Computational Linguistics*, vol. 47, pp. 707–727, 12 2021

³ DeepLearningIA, "A complete guide to natural language processing," 2023. [En línea; consultado el 11 de enero de 2024]

⁴ S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing A Comprehensive Guide to Building Real-World NLP Systems*. 2020
Figure 5.1: Relación entre disciplinas que estudian en lenguaje. La Figura original fue adaptada al idioma español de J. Tsujii, "Natural Language Processing and Computational Linguistics," *Computational Linguistics*, vol. 47, pp. 707–727, 12 2021

⁵ S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing A Comprehensive Guide to Building Real-World NLP Systems*. 2020

5.2 El lenguaje y la ambigüedad

El lenguaje humano es un *sistema estructurado* de comunicación que integra combinaciones complejas de sus unidades lingüísticas como: fonemas, morfemas, palabras, frases, oraciones y textos⁶. Una característica fundamental en el estudio del lenguaje es la *ambigüedad*, presente en algunas de estas unidades lingüísticas. Por ejemplo, las palabras polisémicas pueden tener múltiples significados, mientras que las oraciones pueden interpretarse de diferentes maneras según su estructura y contexto. En el cerebro humano, existen procesos cognitivos que facilitan la interpretación del significado correcto de las palabras en su contexto. En el ámbito computacional, esta tarea se denomina desambiguación del sentido de las palabras (Word-Sense Disambiguation o WSD, por sus siglas en inglés)⁷. Sin embargo, la desambiguación, es decir, la resolución de ambigüedades relacionadas con el significado y sentido de las palabras, representa uno de los mayores desafíos en el PLN. Esto se debe a que exige que las máquinas no solo procesen, sino que también “comprendan” el *contexto* en el que se producen las expresiones⁸.

Existen tipos diferentes de ambigüedad y cada uno de ellos presenta sus propios desafíos para la desambiguación. Entre estos tipos se incluyen la ambigüedad léxica, que se refiere a palabras con múltiples significados; la ambigüedad sintáctica, que surge de diferentes formas de interpretar la estructura gramatical de una oración; la ambigüedad semántica, que se relaciona con múltiples interpretaciones del significado de una frase; la ambigüedad pragmática, que depende del contexto social y situacional; y la ambigüedad discursiva, que involucra la interpretación de enunciados dentro de un discurso más amplio⁹.

Dado que la ambigüedad se relaciona directamente con *significado*, es importante entender cómo se aborda este concepto no solo desde la lingüística, sino también en los sistemas computacionales. Históricamente, para interpretar el *significado* de las palabras, la mayoría de los enfoques computacionales han recurrido al uso de recursos taxonómicos como WordNet¹⁰, que es una base de datos léxica del idioma inglés. WordNet ofrece una extensa clasificación y relaciones entre las palabras, proporcionando una estructura útil para la representación del significado.

A pesar de las ventajas de este recurso, existen problemas presentes en las *representaciones discretas* de palabras que se basan en relaciones taxonómicas¹¹. Un problema es la pérdida de matices; por ejemplo, sinónimos como “rápido”, “veloz” y “ágil” pueden tener connotaciones

⁶ J. Tsujii, “Natural Language Processing and Computational Linguistics,” *Computational Linguistics*, vol. 47, pp. 707–727, 12 2021

⁷ O. Pramod, “Disambiguating words: A deep dive into word sense disambiguation,” December 2023. Published in Towards Data Science

⁸ R. Socher, F. Chaubard, and R. Mundry, “Cs 224d: Deep learning for nlp,” Lecture Notes, 2016. Spring 2016, Stanford University

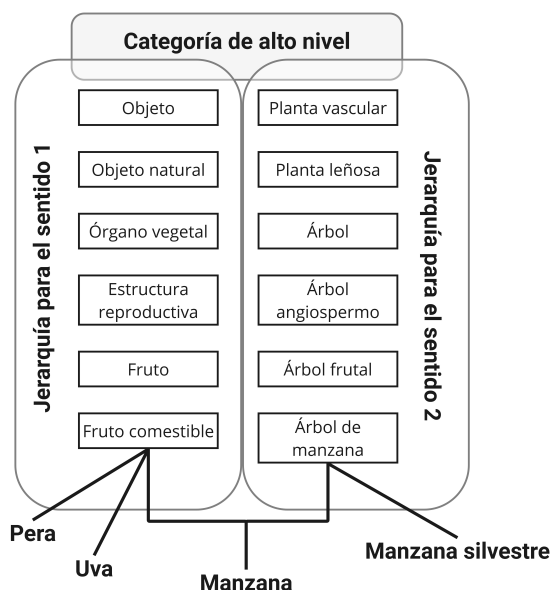
⁹ Stanford Encyclopedia of Philosophy, “Ambiguity,” 2021. First published Mon May 16, 2011; substantive revision Sat May 22, 2021

¹⁰ Princeton University, “WordNet: A Lexical Database for English.” <https://wordnet.princeton.edu/>. [En línea; consultado el 02 de mayo de 2024],

¹¹ A. Finch, E. Black, Y.-S. Hwang, and E. Sumita, “Using lexical dependency and ontological knowledge to improve a detailed syntactic and semantic tagger of English,” in *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, (Sydney, Australia), pp. 215–222, Association for Computational Linguistics, July 2006

ligeramente diferentes que no son capturadas por estas relaciones taxonómicas. Además, estos sistemas a menudo no se actualizan con suficiente rapidez para incluir neologismos, lo que limita su utilidad en contextos dinámicos¹².

WordNet organiza las palabras y sus significados en unidades discretas llamadas *synsets*, donde cada synset representa un significado específico y distinto de una palabra, como se ejemplifica en la Figura 5.2. Estas unidades se relacionan entre sí mediante conexiones semánticas claramente definidas, como hipónimos¹³, hiperónimos¹⁴ y antónimos¹⁵ que estructuran el léxico de manera taxonómica. Sin embargo, debido a que WordNet emplea estas *representaciones discretas*, no logra capturar las variaciones sutiles y los matices contextuales que pueden existir entre sinónimos o diferentes usos de una misma palabra. Esto significa que, mientras WordNet proporciona una estructura organizada para representar el significado, también enfrenta limitaciones al no reflejar con precisión las transiciones graduales y las connotaciones particulares que las palabras pueden tener en distintos contextos.



Estas deficiencias han motivado el desarrollo de enfoques más dinámicos y robustos, como las *representaciones vectoriales* del lenguaje. A continuación, se explora cómo los modelos basados en *representaciones vectoriales* ofrecen una solución alternativa a arquitecturas como WordNet, proporcionando una forma *continua y distribuida* de capturar la semántica y sintaxis, permitiendo una mayor flexibilidad en el análisis del *significado* en diversas tareas de procesamiento del lenguaje natural.

¹² S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. 2020

¹³ Los hipónimos son palabras que denotan un subconjunto más específico de un concepto más general. Ejemplo: "gato" es un hipónimo de "animal".

¹⁴ Los hiperónimos son palabras que representan un conjunto general o categoría que engloba conceptos más específicos. Ejemplo: "animal" es un hiperónimo de "gato".

¹⁵ Los antónimos son palabras con significados opuestos, como "alto" y "bajo".

Figure 5.2: Representación de la taxonomía de WordNet para dos sentidos diferentes de una palabra y cómo estos sentidos se organizan jerárquicamente dentro de categorías más amplias. La Figura original fue adaptada al idioma español de A. Finch, E. Black, Y.-S. Hwang, and E. Sumita, "Using lexical dependency and ontological knowledge to improve a detailed syntactic and semantic tagger of English," in *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, (Sydney, Australia), pp. 215–222, Association for Computational Linguistics, July 2006

5.3 Representaciones vectoriales del lenguaje

En el PLN, uno de los factores más cruciales es **cómo se representan las palabras como entrada a los modelos**. Históricamente, las palabras se trataban como símbolos aislados¹⁶, sin tener en cuenta las *relaciones semánticas* entre ellas. Para superar estas limitaciones, hoy en día se utilizan vectores para representar palabras de manera que capturen sus *similitudes y diferencias* en un espacio matemático N -dimensional.

Este espacio podría ser lo suficientemente complejo como para codificar todas las *relaciones semánticas* entre las palabras. Las dimensiones de este espacio podrían representar diferentes aspectos del *significado*, como el tiempo (pasado, presente, futuro), la cantidad (singular o plural) o el género (masculino o femenino)¹⁷, como se verá más adelante.

Así, en la década de los 80, surge la teoría de *Parallel Distributed Processing* (PDP, por sus siglas en inglés), un marco computacional diseñado para modelar procesos cognitivos. Inspirada en el funcionamiento del cerebro humano, esta teoría propone que el procesamiento de la información se lleva a cabo de manera *paralela y distribuida*¹⁸, mediante la interacción de numerosos elementos de procesamiento simples, conocidos como unidades que se comunican entre sí a través de señales excitatorias e inhibitorias. Bajo esta hipótesis, **la identidad** de una palabra, por ejemplo, no se encuentra en una única unidad, sino que se distribuye en el patrón de activación de un gran número de unidades¹⁹.

La teoría PDP es fundamental debido a la introducción del concepto de *representación distribuida* (*Distributed Representation*), que sugiere que cualquier entidad individual puede ser representada a través de múltiples unidades de procesamiento. Este enfoque ha sido ampliamente adoptado en el estudio de redes neuronales y se ha propuesto para el análisis de textos en el campo del PLN para explicar cómo las palabras pueden ser representadas en modelos de lenguaje mediante vectores con valores continuos distribuidos, en lugar de estar limitados a 0 y 1, como ocurre en las *representaciones localistas*.

De acuerdo con Hinton²⁰, las teorías localistas y las teorías distribuidas representan dos enfoques opuestos dentro de la semántica. Las teorías localistas sugieren que cada concepto o palabra está representada por una única unidad específica en la red, lo que implica una representación aislada y discreta²¹. Por el contrario, el enfoque distribuido está más alineado con la perspectiva estructuralista, que sostiene que los conceptos no tienen un significado intrínseco, sino que

¹⁶ O también llamados “símbolos atómicos”, lo que significa que no existía información adicional sobre las relaciones entre las palabras más allá de su **identidad**.

¹⁷ R. Socher, F. Chaubard, and R. Mundry, “Cs 224d: Deep learning for nlp.” Lecture Notes, 2016. Spring 2016, Stanford University

¹⁸ En contraste con los enfoques secuenciales y centralizados.

¹⁹ D. E. Rumelhart, J. L. McClelland, and P. R. Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. Volume 1, The MIT Press, 1986

²⁰ G. E. Hinton, “Learning distributed representations of concepts,” in *Parallel distributed processing: Implications for psychology and neurobiology* (R. G. M. Morris, ed.), pp. 46–61, Clarendon Press/Oxford University Press, 1989. Reprinted from Proceedings of the Eighth Annual Conference of the Cognitive Science Society, Amherst, Massachusetts: Erlbaum Associates, 1986

²¹ M. A. Ralph, “Distributed versus localist representations: Evidence from a study of item consistency in a case of classical anomia,” *Brain and Language*, vol. 64, pp. 339–360, October 1 1998

se definen por sus *relaciones* con otros conceptos dentro de una red. En otras palabras, el significado de un concepto emerge de la *interacción* y la *conectividad* entre diferentes elementos en lugar de residir en una unidad aislada.

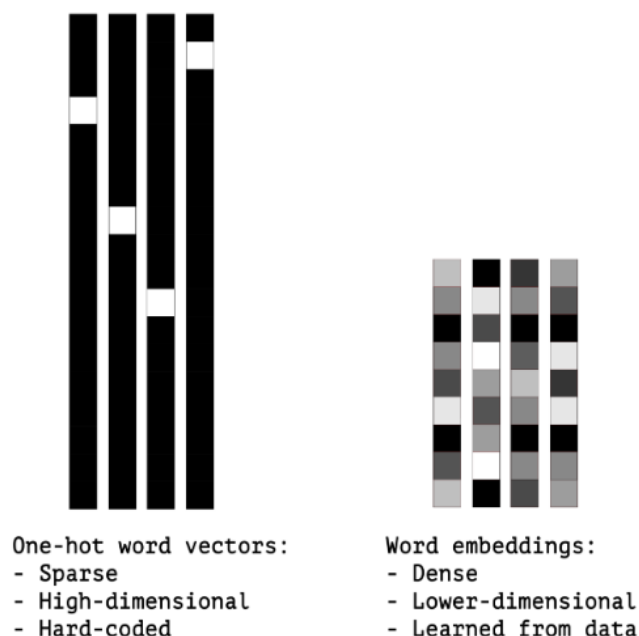


Figure 5.3: Diferencia entre la representación localista y distribuida. Figura retomada de J. Chen, "Intro to nlp part ii: Word embedding," 2021.

Este contraste entre las *representaciones localistas y distribuidas* es particularmente relevante en el campo del PLN, donde se han desarrollado representaciones vectoriales del lenguaje que siguen estos principios. Las *representaciones vectoriales* permiten mapear palabras a vectores matemáticos que capturan sus relaciones semánticas y contextuales dentro de un espacio continuo. Así, las representaciones localistas y distribuidas se pueden ver reflejadas en enfoques computacionales concretos que estructuran y codifican la información semántica de manera distinta.

Una arquitectura *localista* es un tipo de representación de texto en la que cada unidad representa un concepto o palabra de manera independiente. Un ejemplo común es el *one-hot encoding* (como se observa del lado izquierdo en la Figura 5.3), que trata a cada palabra de un corpus como un vector de alta dimensionalidad con un único elemento distinto de cero, $\mathbb{R}^{|V| \times 1}$, donde $|V|$ es el tamaño del vocabulario. Este enfoque, al igual que otras representaciones localistas, difícilmente permite capturar nociones de *similitud* entre palabras, ya

que los vectores correspondientes a palabras diferentes son ortogonales.

Por ejemplo, las representaciones vectoriales en este tipo de codificación serían:

$$v^a = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, v^{aba} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, v^{ába} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, v^{\text{zuzón}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

En este tipo de representación, los vectores de las palabras son completamente **independientes** entre sí, sin compartir componentes. Esto se traduce en que no existe una noción directa de *similitud* entre ellos.

Lo anterior está basado en una *propiedad fundamental del álgebra vectorial*, que relaciona el producto escalar entre dos vectores con el coseno del ángulo que forman:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos(\theta)$$

Donde:

- $\mathbf{a} \cdot \mathbf{b}$ es el producto punto entre dos vectores.
- $\|\mathbf{a}\|$ y $\|\mathbf{b}\|$ son las magnitudes de los vectores.
- θ es el ángulo entre los dos vectores.

En el caso de los vectores *one-hot*, que tienen 1 en una única posición y 0 en todas las demás, el producto escalar entre dos vectores diferentes siempre será 0, lo que implica que el ángulo entre ellos es de 90° , es decir, los vectores son *ortogonales*. Esto se puede observar en el siguiente ejemplo:

$$v^a \cdot v^{aba} = 0$$

Lo que significa que **no existe similitud** entre las palabras "a" y "aba" bajo esta representación, a pesar de que podrían estar relacionadas semánticamente. El producto escalar cero refleja la independencia lineal.

En contraste, las *representaciones distribuidas*, permiten que las palabras se representen como vectores en un espacio de menor dimensionalidad²², donde las relaciones semánticas se capturan en la cercanía o dirección de estos vectores. Sin embargo, para que esto

²² En comparación con representaciones localistas.

sea matemáticamente posible, es necesario que el espacio cumpla con ciertas características, las cuales se abordarán en los siguientes apartados.

En conclusión, la teoría PDP proporciona un marco teórico para entender la distinción entre las *representaciones localistas y distribuidas*, y su relación con propiedades del álgebra vectorial, como el principio de similitud basado en el coseno del ángulo. Las representaciones localistas, que asignan una unidad única a cada concepto, enfrentan dificultades cuando se trabaja con vectores de alta dimensionalidad²³. Entre estas limitaciones, destaca la incapacidad de identificar *relaciones* entre vectores, lo que genera problemas tanto en términos de eficiencia computacional, como de capacidad de generalización. Estas limitaciones han impulsado el desarrollo de técnicas más avanzadas que permiten la creación y optimización de *representaciones de menor dimensión* en comparación con representaciones localistas. Dichas representaciones, son cruciales para reducir la complejidad del espacio vectorial sin sacrificar la capacidad del sistema para captar las relaciones semánticas subyacentes.

²³ Este punto se trata más a fondo en Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems* (T. Leen, T. Dietterich, and V. Tresp, eds.), vol. 13, MIT Press, 2000.

5.4 Fundamentos matemáticos en la representación del lenguaje

El modelado del lenguaje natural se apoya en fundamentos matemáticos que permiten capturar y representar las complejidades del lenguaje humano. Entre los conceptos que apoyan estas representaciones destacan: la *probabilidad*, que facilita la estimación de la ocurrencia de palabras en una secuencia a partir de su contexto y el *espacio vectorial*, que proporciona una estructura geométrica en la que las palabras se representan mediante vectores, permitiendo modelar las relaciones semánticas y contextuales.

Por un lado, los *modelos probabilísticos*, como los modelos N-gram, utilizan principios de probabilidad para predecir la próxima palabra en una secuencia, basándose en el contexto previo de palabras. Este enfoque ha sido clave en los **modelos de lenguaje tradicionales**, ya que permite calcular la probabilidad de una secuencia de palabras de manera incremental, descomponiendo la *probabilidad conjunta* en una serie de *probabilidades condicionales*. A pesar de sus limitaciones en cuanto a la captura de contexto de largo alcance, estos modelos fueron fundamentales para las primeras aproximaciones en el PLN.

Por otro lado, con la evolución hacia los *modelos iterativos o neuronales*, se representan palabras y secuencias de manera más eficiente. Esto condujo al desarrollo de las representaciones distribuidas en espacios vectoriales, conocidas como *word embeddings*. Estos embeddings

proyectan palabras en un espacio latente, donde las relaciones semánticas y sintácticas entre las palabras pueden expresarse mediante distancias y direcciones en dicho espacio. Los principios matemáticos que subyacen a este tipo de representaciones aplican herramientas de álgebra lineal, optimización y geometría, lo que permite a los modelos capturar patrones complejos en el lenguaje de una manera que los enfoques probabilísticos tradicionales no podían.

Es importante aclarar que, aunque los word embeddings por sí solos (como los generados por *Word2Vec*, *GloVe* o *FastText*) no están diseñados para predecir palabras en una secuencia, muchos *modelos de lenguaje* modernos utilizan estos embeddings o generan propias representaciones vectoriales de manera contextualizada como una parte esencial del proceso. En estos modelos de lenguaje (como *GPT* o *BERT*), los embeddings son el punto de partida y luego el modelo ajusta las representaciones de palabras y predice secuencias, capturando relaciones contextuales complejas a lo largo del texto.

A continuación, en las secciones posteriores, se define qué es un *modelo de lenguaje*, se examina cómo los modelos de lenguaje probabilísticos descomponen la probabilidad de una secuencia de texto en probabilidades condicionales, un proceso clave en la generación y predicción del lenguaje. Además, se analizan las limitaciones de estos modelos de lenguaje probabilísticos. Asimismo, se introduce a las representaciones vectoriales distribuidas y los diferentes tipos de estas para dar continuidad a los modelos de lenguaje iterativos o neuronales.

5.4.1 Probabilidad y espacio latente en el modelado de lenguaje

Un *modelo de lenguaje* se define como una distribución de probabilidad p sobre un alfabeto Σ ²⁴, que representa el conjunto de todas las secuencias posibles de símbolos. Cada secuencia $x = x_1 \cdots x_T$ está compuesta por símbolos x_t provenientes de Σ y pueden tener longitud arbitraria. Un objetivo de PLN es estimar un modelo de lenguaje a partir de una familia paramétrica, es decir, un conjunto de modelos que dependen de ciertos parámetros ajustables. Este modelo debe ser capaz de asignar una alta probabilidad a las secuencias de texto que se asemejen a las producidas por seres humanos²⁵.

Un aspecto clave de los modelos de lenguaje es cómo descomponen la *probabilidad* de una secuencia completa. En lugar de calcular directamente la probabilidad conjunta $p(x)$, los modelos de lenguaje la dividen en una serie de probabilidades condicionales utilizando la *factorización autoregresiva*, donde la probabilidad de cada palabra depende de las palabras anteriores en la secuencia. Esto se expresa

²⁴ Un conjunto finito y no vacío de símbolos posibles, como palabras o caracteres de un vocabulario.

²⁵ L. Malagutti, A. Buinovskij, A. Svete, C. Meister, A. Amini, and R. Cotterell, "The role of n -gram smoothing in the age of neural networks," 2024.

matemáticamente como:

$$p(x) = p(\text{EOS} \mid x) \prod_{t=1}^{|x|} p(x_t \mid x_{<t}), \quad (5.1)$$

donde $\text{EOS}^{26} \notin \Sigma$ es un símbolo especial de fin de secuencia y $x_{<t} = x_1 \cdots x_{t-1}$ es el prefijo de la secuencia. Esta descomposición es fundamental en los modelos probabilísticos del lenguaje, como los N-gram, que fueron el estándar antes de la introducción de modelos neuronales.

Entre los primeros enfoques para modelar esta descomposición se encuentran los modelos de N-grams^{27,28}, que calculan la probabilidad de una palabra en función de las $N - 1$ palabras anteriores. Estos modelos simplifican el cálculo de $\hat{P}(w_t \mid w_1^{t-1})$ (ver la ecuación (5.1)) aproximando la probabilidad de una palabra basada en un contexto reducido, como se observa en la ecuación (5.2).

$$\hat{P}(w_t \mid w_1^{t-1}) \approx \hat{P}(w_t \mid w_{t-T+1}^{t-1}), \quad (5.2)$$

La ecuación (5.3) muestra un ejemplo concreto de un trigram ($N=3$), donde la probabilidad de una palabra depende únicamente de las dos palabras anteriores. Aunque estos modelos son fáciles de implementar y eficientes para predecir palabras en contextos locales, presentan limitaciones significativas. Entre ellas, su incapacidad para capturar dependencias de largo alcance y su tendencia a asignar probabilidades erróneas a combinaciones infrecuentes de palabras, lo que ocurre debido a la denominada "maldición de la dimensionalidad" (*Curse of dimensionality*)²⁹, que surge cuando el número de posibles combinaciones de palabras crece exponencialmente a medida que el tamaño del vocabulario y las secuencias aumentan. Como resultado, el modelo enfrenta dificultades para generalizar, ya que muchas combinaciones posibles no tienen suficientes datos para ser adecuadamente representadas.

$$\hat{P}(w_t \mid w_1^{t-1}) \approx \hat{P}(w_t \mid w_{t-2}, w_{t-1}), \quad (5.3)$$

A pesar de que los modelos N-gram intentan mitigar la maldición de la dimensionalidad limitando el contexto a un número fijo de palabras, esta reducción no es suficiente para resolver completamente el problema³⁰. Si bien logran reducir el espacio de búsqueda, siguen enfrentando el problema de "sparsity", ya que muchas combinaciones de palabras no se observan en los datos de entrenamiento, lo que provoca predicciones poco confiables para secuencias infrecuentes.

Es importante destacar que esta *factorización autoregresiva* (5.1) no es exclusiva de los modelos N-gram. Si bien los N-grams emplean un

²⁶ Significa: "End of Sequence".

²⁷ Los modelos N-gram son los modelos probabilísticos más simples y funcionan tomando las últimas $N - 1$ palabras de una secuencia para predecir la próxima palabra. Ejemplos incluyen el modelo unigram ($N=1$), bigram ($N=2$) o trigram ($N=3$).

²⁸ D. Jurafsky and J. H. Martin, *Speech and Language Processing*. Copyright © 2024. All rights reserved., 2024. Draft of August 20, 2024

²⁹ Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems* (T. Leen, T. Dietterich, and V. Tresp, eds.), vol. 13, MIT Press, 2000

³⁰ L. Malagutti, A. Buinovskij, A. Svete, C. Meister, A. Amini, and R. Cotterell, "The role of n -gram smoothing in the age of neural networks," 2024

contexto limitado, los *modelos neuronales* modernos también utilizan esta descomposición, pero con la capacidad de considerar un contexto mucho más amplio. En estos últimos modelos, la probabilidad condicional $p(x_t|x_{<t})$ puede depender de toda la secuencia previa, lo que les permite capturar dependencias más largas dentro del texto.

La transición hacia *modelos iterativos o neuronales*, como el propuesto por Bengio y colaboradores³¹, trajo consigo el uso de *representaciones distribuidas* para abordar problemas que limitaban a los modelos tradicionales, tal como se mencionó previamente. Además, en lugar de computar y almacenar *información global*³² acerca de un gran conjunto de posibles secuencias (que podrían ser billones), los *modelos iterativos* permiten aprender en cada paso del algoritmo.

A través de técnicas como la retropropagación de errores³³ (*backpropagation*), el modelo ajusta sus parámetros basándose en el error calculado para cada muestra de entrenamiento. Esto contrasta con los enfoques tradicionales de modelos de lenguaje, como los unigramas y bigramas, que calculan probabilidades de secuencias de palabras de manera independiente o en pares, lo que puede resultar ineficiente al no capturar correctamente la dependencia contextual entre palabras. Los *modelos neuronales*, en cambio, aprenden a capturar mejor estas dependencias complejas, lo que permite predecir con mayor precisión la probabilidad de una palabra dentro de su contexto³⁴.

En modelos neuronales, cada palabra del vocabulario se representa mediante un vector de características en un espacio de menor dimensionalidad³⁵, $\mathbb{R}^m$. Estos vectores distribuidos, también conocidos como *word embeddings*, permiten capturar relaciones semánticas y sintácticas entre las palabras, mapeándolas a un espacio latente donde las palabras con significados similares tienden a estar más cerca unas de otras.

Este espacio latente, representado por los *embeddings*, ofrece una forma compacta de capturar las características del lenguaje. En él, la proximidad geométrica entre vectores refleja similitudes semánticas y contextuales. La capacidad de los modelos neuronales para generar estas representaciones vectoriales ha sido clave en el desarrollo de los modelos de lenguaje modernos, como los modelos de lenguaje de gran tamaño (Large Language Models, LLMs)³⁶.

El concepto de *espacio vectorial latente* explica esta reducción de dimensionalidad. A diferencia de los enfoques clásicos, donde las dimensiones son explícitas y observables (como los métodos basados en Descomposición de Valores Singulares (*Singular Value Decomposition*))

³¹ Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems* (T. Leen, T. Dietterich, and V. Tresp, eds.), vol. 13, MIT Press, 2000

³² La información global se refiere a la necesidad de mantener un conjunto de datos con todas las posibles secuencias y sus probabilidades, lo cual se vuelve impráctico cuando se manejan grandes cantidades de datos y vocabulario.

³³ D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, pp. 1-21, 1988

³⁴ R. Socher, F. Chaubard, and R. Mundry, "Cs 224d: Deep learning for nlp." Lecture Notes, 2016. Spring 2016, Stanford University

³⁵ "Menor dimensionalidad" en comparación con vectores dispersos como los *one-hot-encoding*, ya que se trata de vectores proyectados, aunque el espacio a donde se proyectan sigue siendo grande en términos de magnitud (dimensionalidad), pero es significativamente menor que los vectores dispersos del one-hot encoding.

³⁶ C. Wang, P. Nulty, and D. Lillis, "A comparative study on word embeddings in deep learning for text classification," in *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval, NLPPIR '20*, (New York, NY, USA), p. 37-46, Association for Computing Machinery, 2021

que aplican matrices basadas en una ventana de co-ocurrencia (como puede observarse en la Figura 5.4), las dimensiones en un espacio latente no corresponden directamente a características lingüísticas evidentes. Estas dimensiones latentes representan patrones que emergen del entrenamiento del modelo³⁷. Estos patrones latentes permiten que los *word embeddings* capturen no solo información *local* sobre las palabras individuales, sino también las relaciones *globales* y contextuales en un corpus extenso.

$$X = \begin{matrix} & \begin{matrix} I & like & enjoy & deep & learning & NLP & flying & . \end{matrix} \\ \begin{matrix} I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{matrix} & \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

³⁷ E. Tiu, "Understanding latent space in machine learning: Learn a fundamental, yet often 'hidden,' concept of deep learning," *Towards Data Science*, February 2020. Published in Towards Data Science

Figure 5.4: Se muestra una matriz de co-ocurrencia X generada a partir de un corpus de texto. Las filas y las columnas representan palabras del vocabulario y cada celda contiene un valor que indica la frecuencia con la que cada par de palabras aparecen juntas dentro de una ventana de contexto definida.

Dentro de este marco, se pueden distinguir dos enfoques fundamentales en la construcción de *word embeddings*: los independientes del contexto (*context-independent*), también llamados incrustaciones estáticas³⁸ y los dependientes del contexto (*context-dependent*), también conocidos como incrustaciones contextualizadas^{39,40}. Las incrustaciones independientes del contexto generan representaciones estáticas, donde una palabra tiene un único vector asociado, independientemente de cómo se use en una oración. Estos vectores se suelen entrenar en grandes corpus de texto y estar disponibles como modelos preentrenados, como se observa en la Figura 5.6.

³⁸ *Static Word Embeddings*.

³⁹ *Contextualized Word Embeddings*.

⁴⁰ V. Hofmann, J. B. Pierrehumbert, and H. Schütze, "Dynamic contextualized word embeddings," 2021

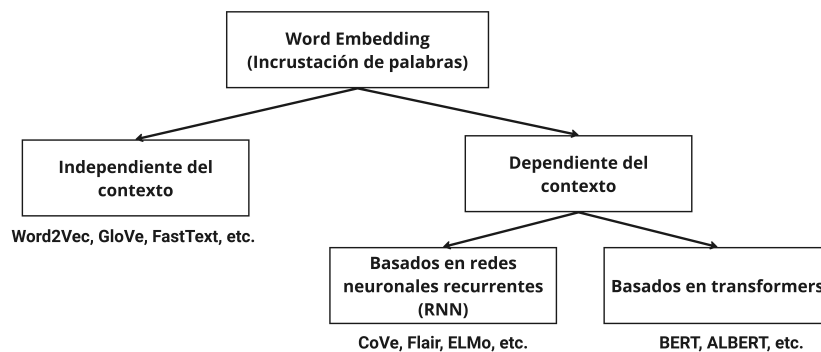


Figure 5.5: Taxonomía de las incrustaciones de texto (*word embeddings*). La Figura original fue adaptada al idioma español de Wang, 2020 y Chiusano, 2021.

Siguiendo la taxonomía de incrustaciones propuesta por Chiusano⁴¹ y como puede observarse en la Figura 5.5, las *incrustaciones independientes de contexto* se pueden dividir en métodos que emplean o no técnicas de aprendizaje automático. Los enfoques más simples, como *Bag-of-Words* o *Term Frequency-Inverse Document Frequency (TF-IDF)*, no utilizan técnicas de aprendizaje automático y se basan en estadísticas directas de frecuencia de palabras. Por otro lado, los modelos que emplean aprendizaje automático, como *Word2Vec*, *GloVe* y *FastText*, generan *representaciones distribuidas* de las palabras.

⁴¹ F. Chiusano, "Two minutes nlp — word embeddings models you should know tf-idf, word2vec, glove, fasttext, elmo, cove, bert, roberta, etc.," December 2021. Published in Towards Data Science

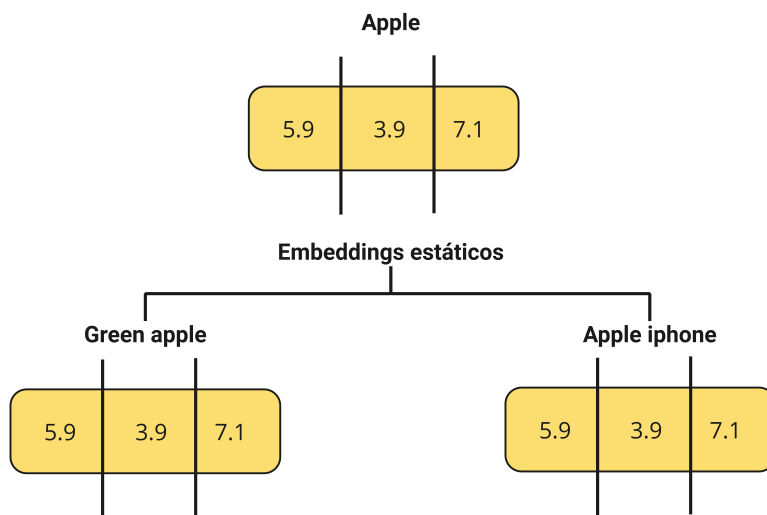


Figure 5.6: Word embedding independiente de contexto.

Por otro lado, las *incrustaciones dependientes del contexto* generan diferentes representaciones para la misma palabra según el contexto en que se encuentra. Esta capacidad de ajustar la representación en función del entorno contextual mejora la captura de relaciones semánticas. La capacidad de estos modelos para ajustar las incrustaciones mediante ajuste fino (*fine-tuning*) en tareas específicas también destaca la flexibilidad y potencia del espacio vectorial latente, que se adapta y especializa según el dominio de la aplicación.

Los métodos para generar *word embeddings* definen cómo se mapean las palabras a un espacio latente, donde cada palabra se representa como un vector. Cada enfoque varía en la manera en que se determina la dimensionalidad de este espacio y cómo se capturan las relaciones lingüísticas entre palabras. Sin embargo, todos ellos se basan en la *hipótesis distribucional de la lingüística*⁴², que establece que las palabras que aparecen en contextos similares tienden a tener significados similares. Esta hipótesis es crucial, ya que permite que las relaciones semánticas entre palabras se reflejen en la proximidad de sus vectores dentro del espacio latente, facilitando una representación de sus

⁴² S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing A Comprehensive Guide to Building Real-World NLP Systems*. 2020

similitudes y diferencias.

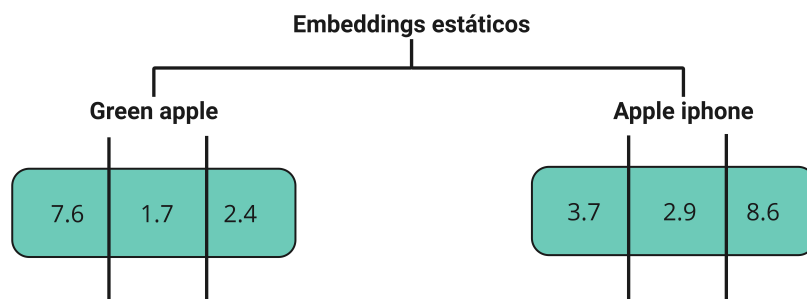


Figure 5.7: Word embedding dependiente de contexto.

En este sentido, a continuación se analiza el *modelado de contexto*, que hace referencia a cómo los *modelos basados en redes neuronales* logran capturar las variaciones del significado de las palabras según el contexto en el que se generan los vectores, sin embargo, ¿qué significa “contexto” en el ámbito del procesamiento de lenguaje natural?.

5.5 Modelos de lenguaje iterativos basados en redes neuronales

En el ámbito del PLN, el *contexto* en el que una palabra aparece es crucial para determinar su *significado*. Este contexto puede ser entendido de dos maneras: como *contexto local* o como *contexto global*, ambos importantes para capturar diferentes matices semánticos en los modelos de representación de palabras. A continuación se lleva a cabo la distinción entre estos tipos de contexto y sus implicaciones en los *modelos de lenguaje* basados en redes neuronales.

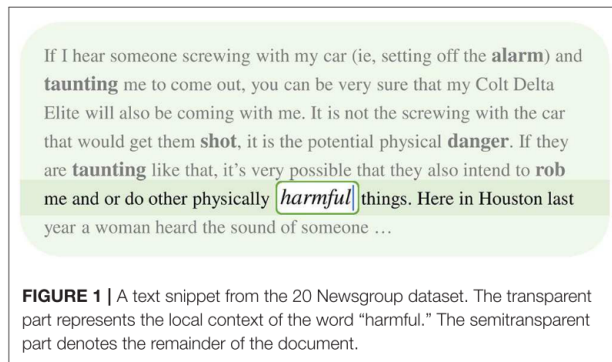
5.5.1 Contexto local y global

El *contexto local* se refiere a las palabras que rodean directamente a una palabra objetivo dentro de una ventana de tamaño limitado. Por otro lado, el *contexto global* abarca unidades semánticas más grandes, como un párrafo, un documento o incluso un conjunto de documentos. Este tipo de contexto es fundamental para capturar el tema general o el significado global en el que las palabras se encuentran inmersas⁴³.

Como se analizó previamente, los modelos N-gramas son un ejemplo clásico de un enfoque basado en *contexto local*. Estos modelos se centran en una ventana fija de palabras para predecir la siguiente palabra o desambiguar su significado. Sin embargo, al depender únicamente de un contexto limitado, los N-gramas no capturan las relaciones de largo alcance entre palabras que pueden ser críticas para el entendimiento semántico global.

⁴³ H. J. W. G. W. Z. C. Meng, Yu and J. Han, “Unsupervised word embedding learning by incorporating local and global contexts,” *Frontiers in Big Data*, vol. 3, p. 9, 2020

Ambos tipos de contexto, *local* y *global*, se utilizan de manera complementaria^{44,45}. Al combinar estos dos tipos de información, los modelos de lenguaje neuronales logran representar con mayor precisión el significado de las palabras en diferentes contextos. El *contexto local* es esencial para capturar detalles específicos y relaciones inmediatas entre palabras, mientras que el *contexto global* proporciona una comprensión más amplia de los patrones semánticos generales y las asociaciones a lo largo de un corpus extenso. Modelos modernos integran ambos tipos de contexto, utilizando redes neuronales profundas que consideran tanto la información contextual inmediata como las relaciones más amplias dentro del texto.



⁴⁴ E. Huang, R. Socher, C. Manning, and A. Ng, "Improving word representations via global context and multiple word prototypes," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (H. Li, C.-Y. Lin, M. Osborne, G. G. Lee, and J. C. Park, eds.), (Jeju Island, Korea), pp. 873–882, Association for Computational Linguistics, July 2012

⁴⁵ H. J. W. G. W. Z. Z. C. Meng, Yu and J. Han, "Unsupervised word embedding learning by incorporating local and global contexts," *Frontiers in Big Data*, vol. 3, p. 9, 2020

Figure 5.8: Ejemplificación del contexto local y global. Un fragmento de texto del conjunto de datos "20 Newsgroup". La parte transparente representa el contexto local de la palabra "harmful". La parte semitransparente denota el resto del documento. Figura retomada de H. J. W. G. W. Z. Z. C. Meng, Yu and J. Han, "Unsupervised word embedding learning by incorporating local and global contexts," *Frontiers in Big Data*, vol. 3, p. 9, 2020.

5.6 Arquitecturas de modelos de lenguaje neuronales

Los modelos de lenguaje han experimentado una evolución significativa, desde los **enfoques probabilísticos clásicos**⁴⁶, hasta los modernos LLMs. En esta sección se presentan las arquitecturas para modelar el lenguaje utilizadas en este proyecto, comenzando con una que fue pionera en el uso de redes neuronales.

La primera transición del **modelado probabilístico del lenguaje** hacia el uso de **redes neuronales** ocurrió con la propuesta del modelo *Feedforward Neural Network Language Model (NNLM)*⁴⁷, que propuso aprender representaciones distribuidas de palabras (word embeddings) dentro de un modelo de lenguaje basado en un *aprendizaje iterativo*. El principal objetivo del NNLM era predecir la siguiente palabra en una secuencia basada en las palabras *precedentes*, permitiendo así el cálculo de la probabilidad de la siguiente palabra utilizando una ventana de contexto más amplia. Como se muestra en la Figura 5.9, esta arquitectura se entrena maximizando la probabilidad de las secuencias de palabras en el corpus de entrenamiento, lo cual se formaliza a través de la siguiente función de costo:

⁴⁶ Como los modelos N-gram.

⁴⁷ Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems* (T. Leen, T. Dietterich, and V. Tresp, eds.), vol. 13, MIT Press, 2000

$$L = \frac{1}{T} \sum_t \log f(w_t, w_{t-1}, \dots, w_{t-n+1}; \theta) + R(\theta), \quad (5.4)$$

donde f es la función que calcula la probabilidad condicional de la siguiente palabra y $R(\theta)$ es un término de regularización que penaliza la magnitud de los parámetros del modelo para evitar el sobreajuste.

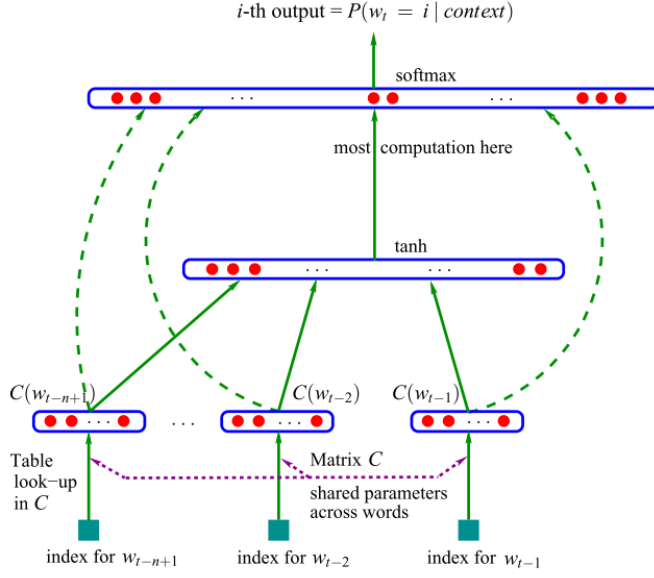


Figure 5.9: Arquitectura de la primera red neuronal profunda para PLN propuesta por Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems* (T. Leen, T. Dietterich, and V. Tresp, eds.), vol. 13, MIT Press, 2000.

Finalmente, la capa de salida del NNLM utiliza una función softmax para calcular la distribución de probabilidad sobre las palabras del vocabulario, según la siguiente ecuación:

$$\hat{P}(w_t | w_{t-1}, \dots, w_{t-n+1}) = \frac{e^{v_{w_t}}}{\sum_i e^{v_i}}, \quad (5.5)$$

En esta fórmula, v_{w_t} es el puntaje asignado a la palabra w_t por la red y se utiliza la función exponencial para normalizar estos puntajes en una distribución de probabilidad, garantizando que todas las probabilidades sumen 1. Esta ecuación es una representación directa del softmax aplicada a los logits v_i generados por la red neuronal.

En la arquitectura del modelo en la Figura 5.9 el proceso de generación de estos puntajes v_i involucra múltiples capas y transformaciones no lineales. La arquitectura incluye una capa de activación tanh, aplicada después de multiplicar los embeddings de entrada x por matrices de pesos $W^{(1)}$ y $W^{(2)}$. La ecuación resultante que describe este proceso es la siguiente:

$$\hat{y} = \text{softmax}(W^{(2)} \tanh(W^{(1)}x + b^{(1)}) + W^{(3)}x + b^{(3)}). \quad (5.6)$$

Esta fórmula describe cómo el modelo transforma los embeddings de entrada a través de capas ocultas antes de generar los puntajes para cada palabra en el vocabulario y finalmente aplicar softmax para obtener la distribución de probabilidad.

Si bien el *Feedforward Neural Network Language Model* fue un hito importante en la aplicación de redes neuronales al modelado del lenguaje, su principal limitación radicaba en la ineficiencia al momento de escalar a grandes corpus y vocabularios. Esto se debía a que, al incluir capas ocultas no lineales y trabajar con distribuciones de probabilidad completas sobre un vocabulario extenso, el costo computacional era considerable. Además, el NNLM estaba enfocado en predecir explícitamente la siguiente palabra en una secuencia, lo que implicaba altos costos de cálculo a medida que el tamaño del vocabulario aumentaba.

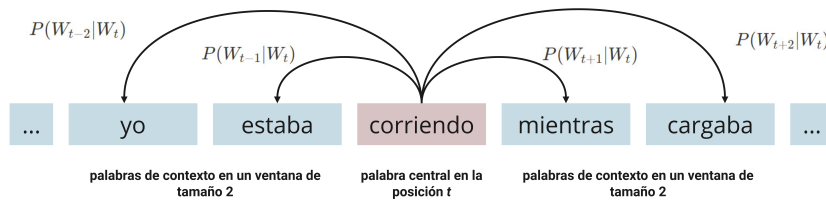
5.6.1 Word2Vec

A medida que surgieron las limitaciones de los primeros *modelos neuronales de lenguaje*, como el NNLM, el modelo de Word2Vec, propuesto por Mikolov y colaboradores⁴⁸, representó un avance significativo en el aprendizaje de *incrustaciones de palabras* (*word embeddings*), especialmente cuando se trabaja con grandes cantidades de datos. En lugar de predecir la siguiente palabra en una secuencia como en el NNLM, Word2Vec se centró en aprender directamente las relaciones semánticas entre palabras a partir de su contexto.

Este modelo se entrena utilizando un gran corpus de texto, donde cada palabra de un vocabulario fijo se representa como un vector. La técnica consiste en recorrer cada posición t en el texto, donde se tiene una palabra central c y un conjunto de palabras de contexto o . Para entrenar el modelo, se utiliza la similitud entre los vectores de las palabras c y o con el objetivo de calcular la probabilidad de que o aparezca dado c (o viceversa) (ver Figura 5.10). Los vectores de las palabras se ajustan iterativamente para maximizar esta probabilidad, logrando que palabras con contextos similares tengan representaciones vectoriales cercanas en el espacio vectorial. El resultado de este proceso es una representación estática de cada palabra en el vocabulario, lo que caracteriza a Word2Vec como un modelo *independiente de contexto*. La ventana de contexto utilizada en Word2Vec es solo una herramienta para aprender las representaciones de las palabras, mientras que los modelos de lenguaje modernos ajustan continuamente las representaciones de las palabras según el contexto en el que aparecen, lo que los convierte

⁴⁸ T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space." <http://arxiv.org/abs/1301.3781>, 2013

en dependientes de contexto.



Una diferencia clave entre Word2Vec y modelos anteriores como el NNLM es que Word2Vec elimina las capas ocultas no lineales, lo que da lugar a una arquitectura más simple y eficiente computacionalmente. Word2Vec utiliza dos modelos principales para aprender las representaciones de palabras: *Continuous Bag of Words (CBOW)* y *Skip-gram*. Ambas arquitecturas procesan una palabra central dentro de una ventana de contexto fija de tamaño m , pero lo hacen de manera distinta. En CBOW, el modelo predice la palabra central dada una ventana de contexto, mientras que en Skip-gram, la palabra central se utiliza para predecir las palabras de contexto.

Una característica distintiva de Word2Vec es que las *relaciones* entre palabras pueden manipularse mediante *operaciones algebraicas* en los vectores de palabras⁴⁹. A continuación, se describe el funcionamiento del modelo Skip-gram.

Skip-gram en Word2Vec

Se considera un vocabulario finito $\mathcal{V}$. Sean C, O variables aleatorias que representan un par (desconocido) de $C \in \mathcal{V}$ (una *palabra central*), y $O \in \mathcal{V}$ (una *palabra de contexto*, que aparece en el contexto de la palabra central). Se emplean c, o para denotar valores específicos de estas variables aleatorias. Además, sean $U \in \mathbb{R}^{|\mathcal{V}| \times d}$ y $V \in \mathbb{R}^{|\mathcal{V}| \times d}$ matrices de parámetros. Nótese que cada palabra en $\mathcal{V}$ está asociada a una fila única de U y una de V ⁵⁰; Este mapeo puede considerarse resultado de un ordenamiento arbitrario del conjunto $\mathcal{V}$.

El modelo *word2vec* es un modelo probabilístico que se define de la siguiente manera, donde u_w denota la fila de U correspondiente a la palabra $w \in \mathcal{V}$ (e igualmente para V):

$$p_{U,V}(o|c) = \frac{\exp(u_o^\top v_c)}{\sum_{w \in \mathcal{V}} \exp(u_w^\top v_c)} \quad (5.7)$$

La ecuación (5.7) corresponde a la función *softmax*, que asigna un puntaje a cada palabra en el vocabulario $\mathcal{V}$, donde estos puntajes son

Figure 5.10: Se ilustra el concepto de una ventana de contexto utilizada en modelos PLN. La palabra central (en color rojo) se encuentra en la posición t , mientras que las palabras de su contexto inmediato dentro de una ventana de tamaño 2 (en color azul), incluyen w_{t-2} , w_{t-1} , w_{t+1} , w_{t+2} . Las flechas representan las relaciones de probabilidad condicional $P(w|w_t)$ utilizadas para modelar la dependencia entre la palabra central y las palabras de su contexto. La imagen original fue adaptada al idioma español de C. Manning, J. Hewitt, F. Chaubard, M. Fang, G. Genthial, R. Mundra, and R. Socher, "Note 1: Introduction and word2vec." johnhew@cs.stanford.edu, 2023. CS 224n: Natural Language Processing with Deep Learning, Winter 2023.

⁴⁹ T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space." <http://arxiv.org/abs/1301.3781>, 2013

⁵⁰ C. Manning, J. Hewitt, F. Chaubard, M. Fang, G. Genthial, R. Mundra, and R. Socher, "Note 1: Introduction and word2vec." johnhew@cs.stanford.edu, 2023. CS 224n: Natural Language Processing with Deep Learning, Winter 2023

el resultado de productos punto entre los vectores incrustados de la palabra central c y las palabras de contexto o . Luego, la función *softmax* convierte estos puntajes en una distribución de probabilidad, de manera que las palabras con puntajes más altos (aquellas cuyo valor de incrustación tiene una mayor similitud con el vector de la palabra central) obtienen una mayor probabilidad de ser seleccionadas como palabras de contexto.

Likelihood

La verosimilitud del modelo *Skip-gram* se construye como el producto de todas las probabilidades de las palabras de contexto dentro de una ventana fija alrededor de una palabra central w_t para cada posición $t = 1, \dots, T$ del corpus. Formalmente, la verosimilitud está dada por:

$$L(\theta) = \prod_{t=1}^T \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} \mid w_t; \theta) \quad (5.8)$$

donde θ son los parámetros del modelo (las matrices U y V) y m es el tamaño de la ventana de contexto. Esta verosimilitud mide cuán bien el modelo predice las palabras de contexto para cada palabra central w_t .

Función de pérdida

Para entrenar el modelo, en lugar de maximizar directamente la verosimilitud, se minimiza el logaritmo negativo de la verosimilitud (es decir, la *pérdida de entropía cruzada*). Esto se expresa como la función de pérdida:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} \mid w_t; \theta) \quad (5.9)$$

Esta función de pérdida se optimiza durante el entrenamiento, lo que implica que se ajustan los parámetros U y V para maximizar la probabilidad de predecir correctamente las palabras de contexto, dado cada palabra central w_t .

Entrenamiento de Word2Vec

Para estimar los valores de los parámetros U y V , el modelo optimiza minimizando la función de *pérdida de entropía cruzada* respecto a la verdadera distribución $P^*(O|C)$:

$$\min_{U,V} \mathbb{E}_{o,c} [-\log p_{U,V}(o|c)]. \quad (5.10)$$

Esta ecuación se interpreta la minimización con respecto a los parámetros U y V de la esperanza del logaritmo negativo de la probabilidad $p_{U,V}(o|c)$, donde o y c son muestras de las distribuciones de O y V , respectivamente⁵¹.

Para entrenar *word2vec* en la práctica, (1) se construyen las matrices U y V , (2) se calcula la esperanza de la ecuación (5.10) y (3) se realiza la operación de minimización. En el paso (1) al principio las matrices se inicializan de manera aleatoria y se actualizan gradualmente a medida que el algoritmo aprende.

Para calcular el paso (2), se considera D un conjunto de documentos $\{d\}$, donde cada documento es una secuencia de palabras $w_1^{(d)}, \dots, w_m^{(d)}$, con todas $w \in \mathcal{V}$. Sea $k \in \mathbb{N}_{++}$ un tamaño de ventana positivo. La variable aleatoria C (palabra central) y O (palabra de contexto) se relacionan con este conjunto de datos de la siguiente manera: O toma el valor de cada palabra w_i en cada documento y para cada w_i , las palabras de contexto son $\{w_{i-k}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+k}\}$. Así, la ecuación (5.10) se reescribe como:

$$L(U, V) = \sum_{d \in D} \sum_{i=1}^m \sum_{j=1}^k -\log p_{U,V}(w_{i-j}^{(d)} | w_i^{(d)}), \quad (5.11)$$

Nótese que la suma se realiza sobre: (1) todos los documentos del conjunto D , (2) todas las palabras de cada documento y (3) todas las palabras de contexto que rodean a la palabra central, para calcular la probabilidad de cada palabra de contexto dato el centro de la ventana.

Estimación basada en gradiente

El objetivo es estimar los parámetros U y V , comenzando con una suposición inicial aleatoria e iterando en la dirección que localmente optimiza la suposición. Este proceso se realiza mediante métodos basados en el gradiente. Inicialmente, los valores $U^{(0)}$ y $V^{(0)}$ se generan aleatoriamente y luego se realizan varias iteraciones del siguiente procedimiento:

$$U^{(i+1)} = U^{(i)} - \alpha \nabla_U L(U^{(i)}, V^{(i)}). \quad (5.12)$$

donde α es la tasa de aprendizaje. Este procedimiento debe interpretarse como una actualización del valor de U en la iteración $i+1$, basado en el valor de U en la iteración anterior, más un paso (proporcional a α) en la dirección que minimiza localmente la función objetivo $L(U, V)$, especificada en la ecuación (5.11).

Un aspecto crucial es que el cálculo de $L(U, V)$ es computacionalmente costoso, ya que requiere recorrer todo el conjunto de datos. En

⁵¹ C. Manning, J. Hewitt, F. Chaubard, M. Fang, G. Genthial, R. Mundra, and R. Socher, "Note 1: Introduction and word2vec." johnhew@cs.stanford.edu, 2023. CS 224n: Natural Language Processing with Deep Learning, Winter 2023

lugar de calcular el objetivo de manera exacta, en la práctica se realiza una optimización basada en gradientes *estocásticos*, donde se aproxima $L(U, V)$ utilizando un subconjunto de muestras en cada paso de la ecuación (5.12). Este procedimiento se realiza mediante el muestreo de documentos $d_1, \dots, d_\ell \sim D$ y se calcula:

$$\hat{L}(U, V) = \sum_{d_1, \dots, d_\ell} \sum_{i=1}^m \sum_{j=1}^k -\log p_{U, V}(w_{i-j}^{(d)} | w_i^{(d)}), \quad (8)$$

El gradiente ajusta los parámetros calculando la derivada parcial de la función de pérdida con respecto al parámetro v_c para una instancia única de una palabra central c :

$$\nabla_{v_c} \hat{L}(U, V) = \sum_{d \in D} \sum_{i=1}^m \sum_{j=1}^k -\nabla_{v_c} \log p_{U, V}(w_{i-j}^{(d)} | w_i^{(d)}), \quad (9)$$

intuitivamente, el gradiente total es la suma de las gradientes individuales de todos los términos en la suma. Ahora se calcula el gradiente de la probabilidad. Para simplificar la notación, se denota $w_{i-j}^{(d)}$ como o (la palabra de contexto) y $w_i^{(d)}$ como c (la palabra central).

Ahora, se se toma un solo término de la suma y se separa aplicando las reglas del logaritmo:

$$\nabla_{v_c} \log p_{U, V}(o | c) = \nabla_{v_c} \log \left(\frac{\exp(u_o^\top v_c)}{\sum_{w=1}^n \exp(u_w^\top v_c)} \right) \quad (5.13)$$

$$= \nabla_{v_c} \log \exp(u_o^\top v_c) - \nabla_{v_c} \log \sum_{w=1}^n \exp(u_w^\top v_c), \quad (5.14)$$

Parte A.

$$\nabla_{v_c} \log \exp(u_o^\top v_c) = \nabla_{v_c} u_o^\top v_c \quad (5.15)$$

$$= u_o \quad (5.16)$$

La derivada parcial de $u_o^\top v_c$ con respecto a $v_{c,i}$ es:

$$\nabla_{v_{c,i}} u_o^\top v_c = \nabla_{v_{c,i}} \sum_j u_{o,j} v_{c,j} = u_{o,i}$$

Esto se debe a que solo un término de la suma depende de $v_{c,i}$. Específicamente, la derivada parcial $\nabla_{v_{c,i}} u_{o,j} v_{c,j}$ es igual a $u_{o,i}$ para $j = i$, ya que u_o no depende de v_c . Al agrupar estas derivadas de cada

componente, se obtiene el vector completo $[u_{o,1}, \dots, u_{o,d}] = u_o$, tal como se mencionó previamente.

Es importante notar que la forma del gradiente es u_o , no $u_o^\top$. Esto sigue una convención de notación matemática donde el gradiente de una función vectorial debe tener la misma forma que el objeto respecto al cual se deriva⁵².

Parte B.

$$-\nabla_{v_c} \log \sum_{w=1}^n \exp(u_w^\top v_c) = \frac{1}{\sum_{w=1}^n \exp(u_w^\top v_c)} \nabla_{v_c} \sum_{x=1}^n \exp(u_x^\top v_c) \quad (5.17)$$

$$= \frac{1}{\sum_{w=1}^n \exp(u_w^\top v_c)} \sum_{x=1}^n \nabla_{v_c} \exp(u_x^\top v_c) \quad (5.18)$$

$$= \frac{1}{\sum_{w=1}^n \exp(u_w^\top v_c)} \sum_{x=1}^n \exp(u_x^\top v_c) \nabla_{v_c} u_w^\top v_c \quad (5.19)$$

$$= \frac{1}{\sum_{w=1}^n \exp(u_w^\top v_c)} \sum_{x=1}^n \exp(u_x^\top v_c) u_w \quad (5.20)$$

Al combinar las partes A y B, se obtiene la siguiente expresión:

$$u_o - \frac{1}{\sum_{w=1}^n \exp(u_w^\top v_c)} \sum_{x=1}^n \exp(u_x^\top v_c) u_w = u_o - \sum_{x=1}^n \frac{\exp(u_x^\top v_c)}{\sum_{w=1}^n \exp(u_w^\top v_c)} u_w \quad (5.21)$$

$$= u_o - \sum_{x=1}^n p_{U,V}(x | c) u_w \quad (5.22)$$

$$= u_o - \mathbb{E}[u_w] \quad (5.23)$$

$$= \text{"observado"} - \text{"esperado"}. \quad (5.24)$$

Esta ecuación (5.24) captura el concepto clave: el vector u_o que corresponde a la palabra que realmente se observó, luego se resta de este vector lo que el modelo esperaba en promedio. Esta "esperanza" es la suma ponderada de los vectores u_x de todas las palabras en el vocabulario, donde el peso de cada vector es la probabilidad asignada por el modelo $p_{U,V}(x | c)$.

En consecuencia, el vector v_c se actualiza para ser "más similar" al vector de la palabra que fue realmente observada u_o que al promedio ponderado de las palabras que el modelo esperaba en función de sus probabilidades.

⁵² C. Manning, J. Hewitt, F. Chaubard, M. Fang, G. Genthial, R. Mundra, and R. Socher, "Note 1: Introduction and word2vec." johnhew@cs.stanford.edu, 2023. CS 224n: Natural Language Processing with Deep Learning, Winter 2023

5.6.2 Transición hacia la arquitectura de Transformers

Antes de la llegada de los Transformers, los modelos basados en **Redes Neuronales Recurrentes (RNN)** representaban una de las herramientas más utilizadas en el PLN y *secuencias temporales*. Desde enfoques iniciales como se vio anteriormente con Word2Vec hasta la evolución de las RNN, la investigación se ha enfocado en nuevas formas de capturar dependencias en secuencias de *longitud variable*. Las RNN surgieron como una solución clave para procesar secuencias de datos, ya que son capaces de preservar información a lo largo del tiempo mediante el uso de un estado oculto recurrente.

A continuación, se analiza cómo se desarrollaron las RNN y sus extensiones más avanzadas como las Long Short-Term Memory (LSTM) y Gated Recurrent Unit (GRU), diseñadas para abordar los problemas presentes en el aprendizaje de dependencias a largo plazo. Sin embargo, a pesar de estos avances, las RNN y sus variantes aún enfrentaban limitaciones que abrieron el camino hacia la invención de arquitecturas *no secuenciales* como los *Transformers*.

Funcionamiento de las RNN

El principal componente de una RNN es su capacidad para *reutilizar los pesos a lo largo del tiempo*. En cada paso de tiempo t , el estado oculto h_t se calcula combinando dos entradas: el estado oculto anterior h_{t-1} y el vector de entrada actual x_t , que representa el input en ese momento. Esta operación se realiza mediante una función no lineal, como la tangente hiperbólica o la sigmoide, después de multiplicar cada entrada por su correspondiente matriz de pesos, $W^{(hh)}$ y $W^{(hx)}$, respectivamente:

$$h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_t) \quad (5.25)$$

El hecho de que los mismos pesos se reutilicen a lo largo de cada paso de tiempo permite que el número de parámetros del modelo *no dependa de la longitud de la secuencia*, una característica que mitiga el problema de la dimensionalidad visto anteriormente. Además, las RNN pueden almacenar información de pasos de tiempo muy anteriores, aunque en la práctica se ven afectadas por problemas como la desaparición (*vanishing gradient*) y explosión (*explosion gradient*) de gradientes.

Una vez calculado el estado oculto h_t , se utiliza para predecir la siguiente palabra en una secuencia mediante la función softmax, que genera una distribución de probabilidad sobre todo el vocabulario:

$$\hat{y}_t = \text{softmax}(W^{(S)}h_t) \quad (5.26)$$

Función de pérdida en RNN

El entrenamiento de una RNN implica minimizar una función de pérdida que generalmente se define como la *entropía cruzada*. Esta mide la discrepancia entre la probabilidad predicha $\hat{y}_t$ y la palabra real y_t . La entropía cruzada para un paso de tiempo t se define como:

$$J^{(t)}(\theta) = - \sum_{j=1}^{|V|} y_{t,j} \times \log(\hat{y}_{t,j}) \quad (5.27)$$

donde $|V|$ es el tamaño del vocabulario, $y_{t,j}$ es el valor verdadero para la palabra j (1 si es la palabra correcta, o en caso contrario), y $\hat{y}_{t,j}$ es la probabilidad predicha para esa palabra.

Al extender esta función a lo largo de una secuencia completa de longitud T , se obtiene la pérdida total del corpus:

$$J = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) \quad (5.28)$$

Una de sus principales fortalezas de RNN es su capacidad para procesar secuencias de longitud variable, lo que las hace flexibles para tareas *secuenciales*. Además, el tamaño del modelo no aumenta con la longitud de la secuencia, ya que los mismos pesos se reutilizan en cada paso de tiempo, lo cual es beneficioso en términos de eficiencia computacional. Asimismo, el uso repetido de los mismos pesos en cada paso asegura una simetría, donde todas las entradas se procesan de manera consistente a lo largo de la secuencia.

Sin embargo, las RNN también presentan ciertas limitaciones. Una de las principales desventajas es que la computación es *secuencial*, lo que impide la *paralelización*, haciendo que el entrenamiento sea más lento en comparación con otros modelos. En la práctica, también es difícil que las RNN accedan a información de muchos pasos anteriores debido a problemas como la *desaparición y explosión de gradientes*, que afectan negativamente el rendimiento de la red al intentar aprender dependencias a largo plazo.

El uso de memoria en una RNN puede ser considerablemente elevado, ya que la red debe almacenar vectores de entrada, pesos y estados ocultos para cada palabra en una secuencia. Además, para corpus de gran tamaño, la memoria requerida para almacenar las matrices de pesos W puede crecer de manera significativa. Sin embargo, el tamaño de estas matrices no escala con el tamaño del corpus, lo que

diferencia a las RNN de los modelos de lenguaje tradicionales, en los que el tamaño de los parámetros sí crece con el tamaño del conjunto de datos.

Problemas con el gradiente

Uno de los principales desafíos en las RNN es el problema del *vanishing gradient* (gradiente que desaparece), el cual ocurre cuando, al retropropagar el error a través de muchos pasos de tiempo, los gradientes se vuelven extremadamente pequeños. Como resultado, la red tiene dificultades para aprender dependencias a largo plazo, ya que la contribución de los pasos de tiempo distantes se vuelve insignificante. Por el contrario, el *gradient explosion* (explosión de gradientes) surge cuando los gradientes crecen exponencialmente con cada paso de tiempo, llevando a inestabilidad en el entrenamiento y haciendo que los pesos se actualicen de manera desproporcionada.

Matemáticamente, estos problemas se derivan de la propagación del gradiente a través de productos de matrices de pesos. Considerando una cadena de multiplicación de los Jacobianos de cada paso de tiempo, la norma del gradiente puede reducirse o amplificarse exponencialmente dependiendo del valor de los parámetros de las matrices de pesos. Si el producto de estas normas es menor que uno, los gradientes tienden a desaparecer. Si es mayor que uno, los gradientes tienden a explotar⁵³.

Para mitigar el problema del *gradient explosion* (explosión de gradientes), se utiliza una técnica llamada *gradient clipping* (recorte de gradientes), que limita los gradientes a un valor máximo cuando su magnitud excede un umbral predefinido. Esta solución, introducida por Pascanu, Mikolov y Bengio⁵⁴, asegura que los gradientes no crezcan exponencialmente y mantengan la estabilidad durante el entrenamiento. El algoritmo de recorte ajusta los gradientes a una magnitud controlada si superan un *valor crítico*, evitando actualizaciones desproporcionadas en los pesos.

Por otro lado, para solucionar el problema del *vanishing gradient* (gradiente que desaparece), se recomienda el uso de técnicas de inicialización de pesos que estabilicen el flujo de gradientes. Además, las funciones de activación como las *Rectified Linear Units (ReLU)* son eficaces para evitar la disminución de los gradientes, ya que presentan derivadas constantes, lo que permite que los gradientes fluyan a través de la red sin reducirse excesivamente⁵⁵.

⁵³ M. Mohammadi, R. Mundry, R. Socher, L. Wang, and A. Kamath, "Cs224n: Natural language processing with deep learning." Course Lecture Notes, Stanford University, Winter 2019, 2019. Lecture Notes: Part V Language Models, RNN, GRU and LSTM

⁵⁴ R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," 2013

⁵⁵ M. Mohammadi, R. Mundry, R. Socher, L. Wang, and A. Kamath, "Cs224n: Natural language processing with deep learning." Course Lecture Notes, Stanford University, Winter 2019, 2019. Lecture Notes: Part V Language Models, RNN, GRU and LSTM

Unidades Recurrentes con Puertas (GRU)

Las *Gated Recurrent Units* (GRU) son una variante de las RNN diseñadas para capturar dependencias a largo plazo mediante el uso de puertas que controlan el flujo de información. A diferencia de las RNN estándar, las GRU utilizan dos mecanismos principales: la *puerta de actualización* y la *puerta de reinicio*, lo que les permite manejar la información relevante y mitigar el problema del gradiente que desaparece.

La puerta de actualización, descrita por la ecuación:

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1}), \quad (5.29)$$

controla cuánto del estado oculto anterior h_{t-1} se llevará al siguiente estado h_t . Por otro lado, la puerta de reinicio:

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}), \quad (5.30)$$

decide cuánta información del estado anterior debe olvidarse. La nueva memoria $\tilde{h}_t$, dada por:

$$\tilde{h}_t = \tanh(r_t \odot Uh_{t-1} + Wx_t), \quad (5.31)$$

combina la entrada actual con el estado anterior para generar una nueva representación. Finalmente, el estado oculto se calcula como:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t, \quad (5.32)$$

permitiendo que las GRU retengan o actualicen información de manera controlada.

Este diseño mejora la capacidad de las RNN para manejar secuencias largas y aprender dependencias a largo plazo, al evitar el olvido excesivo de información previa o la sobresaturación de nuevas entradas. Las GRU son, por lo tanto, más eficientes en la captura de información relevante en comparación con las RNN tradicionales.

Memoria a Largo Plazo (LSTM)

Las redes *Long Short-Term Memories* (LSTM)⁵⁶ son una extensión de las RNN diseñadas para resolver el problema del *vanishing gradient* y permitir que las redes aprendan dependencias a largo plazo en secuencias temporales. A diferencia de las RNN tradicionales o las GRU, las LSTM introducen un sistema de memoria más complejo, compuesto por tres puertas clave: la puerta de entrada, la puerta de olvido y la puerta de salida.

El funcionamiento de una LSTM se describe mediante las siguientes ecuaciones:

⁵⁶ S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997

$$i_t = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1}) \quad (5.33)$$

Puerta de entrada, controla cuánta nueva información se almacena en la celda de memoria.

$$f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) \quad (5.34)$$

Puerta de olvido, decide cuánta información de la celda de memoria previa se debe conservar.

$$o_t = \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) \quad (5.35)$$

Puerta de salida, regula qué información de la celda de memoria debe ser expuesta al estado oculto.

$$\tilde{c}_t = \tanh(W^{(c)}x_t + U^{(c)}h_{t-1}) \quad (5.36)$$

Nueva memoria candidata que será agregada a la memoria de la celda.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (5.37)$$

Memoria de la celda, que integra la nueva memoria y la memoria previa, ajustada por las puertas de entrada y olvido.

$$h_t = o_t \odot \tanh(c_t) \quad (5.38)$$

Estado oculto final, que es la salida de la LSTM.

Las LSTM son capaces de mantener la información relevante durante más pasos de tiempo en comparación con las RNN tradicionales, debido a la celda de memoria que controla de manera más precisa qué información debe ser olvidada o recordada. Esta característica las hace útiles para tareas donde la memoria a largo plazo es importante, como la traducción automática, el reconocimiento de voz y el modelado de lenguaje natural.

Modelo Seq2Seq y mecanismo de atención

Como se ha descrito anteriormente, el procesamiento de secuencias ha sido un desafío dentro del campo del PLN. Los modelos *Sequence-to-Sequence* (Seq2Seq)⁵⁷ surgieron como una solución efectiva para diversas tareas de secuencias, tales como la traducción automática, mediante el uso de RNNs o LSTM. Estos modelos se caracterizan por emplear dos componentes principales: un *codificador* (encoder) y un *decodificador* (decoder). A lo largo de los años, se han introducido mecanismos avanzados como la *atención* (attention mechanism)⁵⁸ para mejorar estos

⁵⁷ I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014

⁵⁸ D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016

modelos y, más recientemente, el diseño de arquitecturas como los *Transformers* ha revolucionado el estado del arte en este campo.

Modelo Seq2Seq: Funcionamiento general

El modelo Seq2Seq está diseñado para transformar una secuencia de entrada en una secuencia de salida de *longitud variable*. El modelo opera mediante un proceso de dos fases. En la primera fase, el codificador recibe la secuencia de entrada y la convierte en una representación interna, conocida como el *vector de contexto*. Este vector, que encapsula la información de la entrada, es luego utilizado por el decodificador para generar la secuencia de salida.

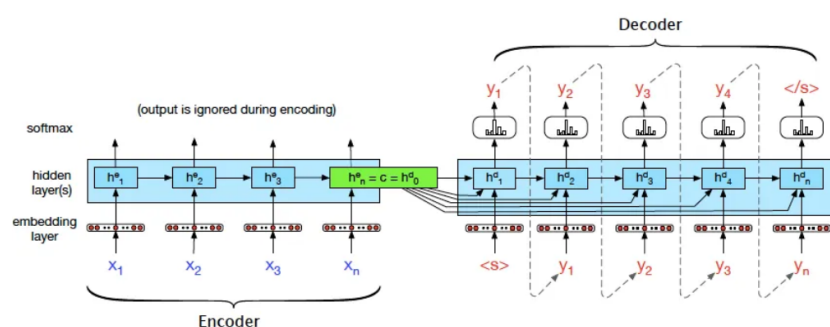


Figure 5.11: Arquitectura de un modelo secuencia a secuencia (*sequence-to-sequence, Seq2Seq*), compuesto por un codificador (*encoder*) y un decodificador (*decoder*). El el codificador, se procesan las entradas $x_1, x_2, x_3, \dots, x_n$, a través de una capa de *embeddings* seguida por capas ocultas (*hidden layers*), produciendo una representación codificada de la secuencia. Figura retomada de K. Chilamkurthy, "Nlp theory and code: Encoder-decoder models (part 11/30)," *Medium*, March 2021. Published in Nerd For Tech

Una característica clave de los modelos Seq2Seq es que las representaciones intermedias no dependen de la longitud fija de la secuencia, lo que los hace útiles para tareas donde *la longitud de la salida difiere de la entrada*. No obstante, estos modelos enfrentan desafíos cuando se procesan secuencias largas, debido a que comprimen toda la información en un único vector C , lo que puede ocasionar la pérdida de detalles importantes.

Codificador y Decodificador

El *codificador* en un modelo Seq2Seq generalmente consiste en una RNN o una LSTM, que procesa la secuencia de entrada de forma secuencial. En cada paso, la célula recurrente del codificador actualiza su estado oculto, capturando la información sobre la secuencia procesada hasta ese punto. El estado oculto final, denominado *vector de contexto*, representa la secuencia de entrada completa y es transferido al decodificador.

El *decodificador*, que también es típicamente una LSTM o una RNN, utiliza este vector de contexto como punto de partida para generar la

secuencia de salida. En cada paso, el decodificador produce una palabra o símbolo de la secuencia de salida y luego retroalimenta esta salida al sistema para predecir el siguiente elemento. Este proceso continúa hasta que se genera la secuencia completa. Aunque esta estructura es poderosa, su principal limitación radica en que el vector de contexto, que condensa toda la información de la secuencia de entrada, puede no ser suficiente para representar secuencias largas con detalle.

Mecanismo de Atención

El *mecanismo de atención* fue propuesto como una solución a las limitaciones de los modelos Seq2Seq, especialmente cuando se trabaja con secuencias largas. En lugar de confiar en un único vector de contexto fijo, la atención permite que el decodificador acceda *dinámicamente* a diferentes partes de la secuencia de entrada durante el proceso de generación de la salida.

En esencia, *la atención* asigna un peso a cada parte de la secuencia de entrada, lo que indica su relevancia en el paso actual del decodificador. Estos pesos, calculados en función de la similitud entre el estado del decodificador y las representaciones de la secuencia de entrada, permiten que el modelo “se enfoque” en las partes relevantes de la entrada en cada paso de la generación de la salida. Este mecanismo no solo mejora la precisión del modelo en secuencias largas, sino que también permite una alineación implícita entre las palabras de entrada y las palabras de salida.

Los *Transformers*, introducidos por Vaswani y colaboradores⁵⁹ en 2017, representaron una solución innovadora a estos problemas. En lugar de depender de RNNs o LSTMs, los Transformers utilizan exclusivamente mecanismos de *autoatención* (self-attention) para procesar toda la secuencia de entrada en *paralelo*. Esta arquitectura elimina la necesidad de procesar las entradas *secuencialmente*, lo que permite una mayor eficiencia en el entrenamiento y en la inferencia.

El mecanismo de *autoatención* en los Transformers permite que cada token de la secuencia de entrada considere todos los otros tokens simultáneamente, asignando pesos a las relaciones entre ellos. Esto permite capturar dependencias a largo plazo y relaciones complejas y lo hace de manera paralelizable.

Aprendizaje por transferencia (Transfer learning)

El aprendizaje por transferencia es una técnica en la que un modelo entrenado en una tarea es reutilizado o ajustado para mejorar el

⁵⁹ A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017

rendimiento en una tarea diferente, pero relacionada. En lugar de entrenar un modelo desde cero, se aprovechan los conocimientos adquiridos previamente por el modelo en una gran cantidad de datos, lo que reduce significativamente el tiempo de entrenamiento y la necesidad de grandes cantidades de datos para la nueva tarea. Esta técnica es particularmente útil en problemas donde los datos disponibles son limitados, y ha sido ampliamente adoptada en campos como el PLN y Visión por Computadora (*Computer Vision*)⁶⁰.

Característica	Seq2Seq	Transformers
Arquitectura básica	Basado en <i>RNNs</i> , <i>LSTMs</i> o <i>GRUs</i> . Utiliza un codificador que genera un vector de contexto y un decodificador que genera la salida secuencialmente.	Basado en <i>autoatención</i> . Puede utilizar solo codificadores, solo decodificadores, o ambos. Emplea capas de <i>autoatención</i> para procesar toda la secuencia a la vez.
Mecanismo de atención	La <i>atención</i> se aplica entre el codificador y decodificador, permitiendo al decodificador centrarse en partes relevantes de la entrada durante cada paso de la salida.	La <i>autoatención</i> es el núcleo de la arquitectura. Cada token puede prestar atención a todos los demás tokens dentro de la misma secuencia en cada capa, tanto en el codificador como en el decodificador.
Procesamiento	Procesa la entrada <i>secuencialmente</i> , lo que impide la paralelización. Cada token debe ser procesado en orden.	Procesa la entrada en <i>paralelo</i> , permitiendo que cada token acceda a información de todos los demás tokens simultáneamente.
Positional encoding vs. recurrencia	Maneja el orden de las palabras mediante la <i>recurrencia</i> de las <i>RNNs</i> , donde cada token se procesa en función del anterior.	No usa recurrencia. Usa <i>Positional Encoding</i> para mantener el orden de los tokens en la secuencia.

⁶⁰ L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers*. O'Reilly Media, Inc., revised ed., 2022

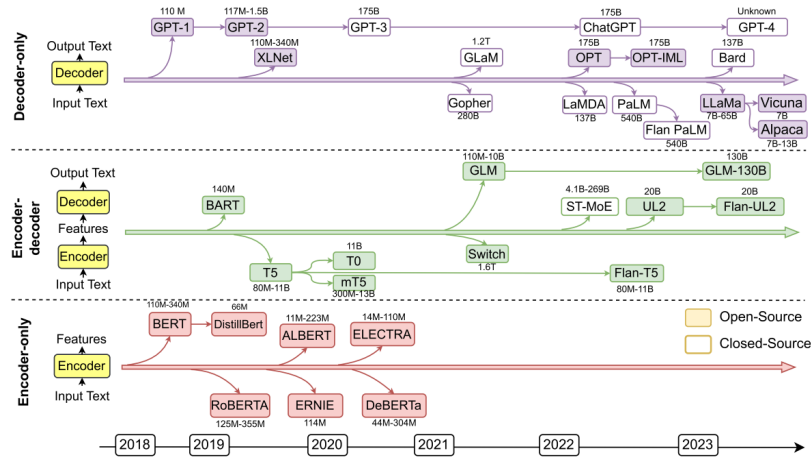
Table 5.1: Elaboración propia. Comparación entre características de modelos Seq2Seq y Transformers

5.6.3 Transformers

Como se discutió anteriormente, el Transformer se fundamenta en la arquitectura *encoder-decoder*. Sin embargo, a pesar de la diversidad de modelos existentes en la actualidad, la mayoría pueden agruparse en tres grandes categorías: (a) Transformers basados exclusivamente en encoders (*encoder-only*); (b) Transformers basados exclusivamente en decoders (*decoder-only*); y (c) Transformers que emplean la arquitectura completa de codificador-decodificador (*encoder-decoder*)⁶¹.

Desde la introducción de los Transformers, han surgido numerosas variantes, entre las que destacan los modelos basados únicamente en encoders, como BERT, DistilBERT, RoBERTa, XLM, XLM-RoBERTa, ALBERT, ELECTRA y DeBERTa. Estos modelos están orientados principalmente a tareas de clasificación, etiquetado y comprensión de secuencias. Por otro lado, los modelos basados exclusivamente en decoders incluyen GPT, GPT-2, CTRL, GPT-3 y GPT-Neo/GPT-J-6B, y se especializan en tareas de generación de texto. Finalmente, los modelos que emplean la arquitectura completa de encoder-decoder, como BART, M2M-100 y BigBird, están diseñados para tareas que requieren tanto comprensión como generación, como la traducción automática y la síntesis de texto.

⁶¹ L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers*. O'Reilly Media, Inc., revised ed., 2022



La Figura 5.12 ilustra una línea de tiempo que recoge algunos de los modelos de lenguaje más representativos desarrollados entre 2018 y 2023, destacando la evolución de estas arquitecturas en la última década⁶².

Arquitectura del modelo

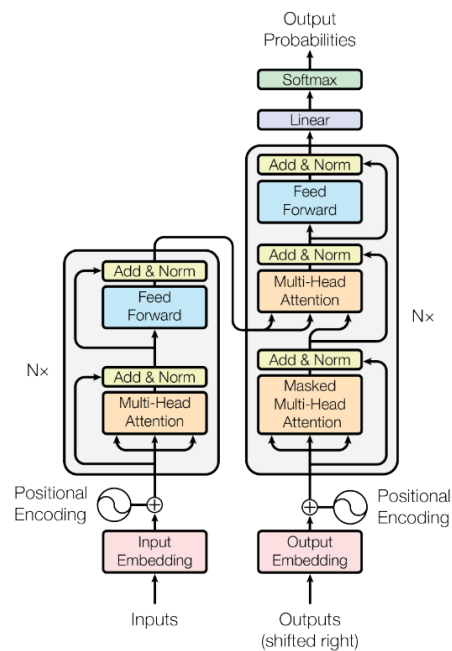


Figure 5.12: Línea de tiempo con algunos de los modelos o arquitecturas de modelos de lenguaje representativos desde el 2018 a 2023. Figura retomada de S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying large language models and knowledge graphs: A roadmap," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, p. 3580–3599, July 2024.

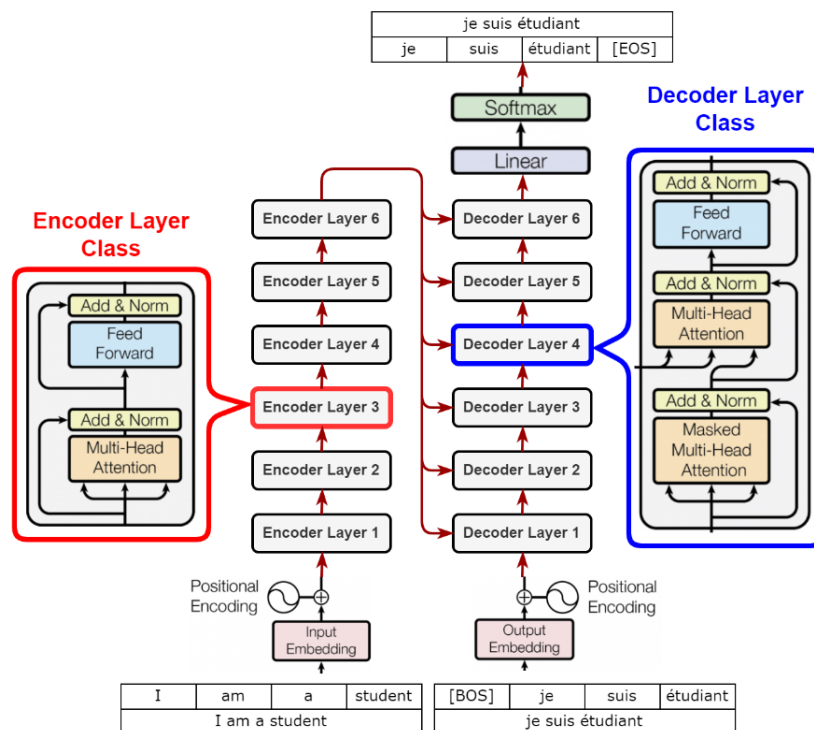
⁶² S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying large language models and knowledge graphs: A roadmap," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, p. 3580–3599, July 2024.

Figure 5.13: Arquitectura del modelo Transformer. Figura tomada de A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

Encoder

El *encoder* es un componente fundamental del modelo Transformer, encargado de procesar una secuencia de entrada y generar una secuencia de representaciones intermedias. Dicha secuencia de entrada, representada como $(x_1, x_2, \dots, x_n)$, es transformada en una serie de representaciones continuas $(z_1, z_2, \dots, z_n)$ que contienen información semántica y posicional de los tokens. Este conjunto de representaciones intermedias, z , es utilizado posteriormente por el decodificador para generar la secuencia de salida⁶³.

La arquitectura del encoder se organiza en N capas idénticas apiladas que comparten la misma estructura interna, como se observa en la Figura 5.14. Cada una de estas capas contiene los siguientes componentes principales: un mecanismo de autoatención (*self-attention*), una capa (*feed-forward layer*) y un mecanismo de normalización y conexiones residuales.



⁶³ A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017

Figure 5.14: Diagrama de la arquitectura del modelo Transformer, que ilustra la estructura de capas de codificación (Encoder Layer Class) y decodificación (Decoder Layer Class). A la izquierda, el bloque de codificación consta de seis capas con módulos de atención múltiple (Multi-Head Attention) y una red de avance (Feed Forward) con normalización. A la derecha, el bloque de decodificación también incluye seis capas, donde cada capa tiene atención enmascarada (Masked Multi-Head Attention) para procesar secuencias de salida de manera autoregresiva. Entre los bloques de codificación y decodificación, se produce la atención cruzada, permitiendo que el decodificador se alinee con la salida del codificador. Este diseño permite la traducción de una oración en inglés ("I am a student") a francés ("je suis étudiant"). Figura retomada de R. M. Go, *Deep Learning Bible - Part K: Transformer from scratch*, ch. K1.2. How it works, step by step. Mintrobot, 2024.

Self-Attention

El primer componente dentro de cada capa del encoder es el mecanismo de *autoatención*. Este permite que el modelo considere, para cada token de entrada, todos los otros tokens en la secuencia,

asignándoles diferentes pesos en función de su relevancia contextual. Esto se traduce en que cada representación de token en la secuencia puede integrar información global de todos los otros tokens, a diferencia de enfoques tradicionales de secuencias que solo tienen en cuenta información local.

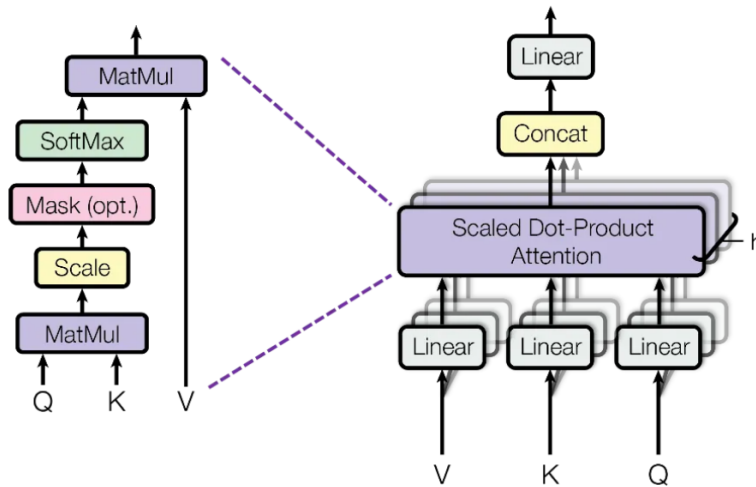
El mecanismo de autoatención se basa en dos técnicas esenciales: (1) la atención escalada por producto punto y (2) la atención de múltiples cabezas⁶⁴.

(1) Scaled Dot-Product Attention

La idea nuclear de la *autoatención* es que, en lugar de utilizar un embedding fijo para cada token de la secuencia, se calcula una nueva representación de cada token basada en una combinación ponderada de todas las representaciones de los demás tokens. Es decir, una secuencia de embeddings de tokens $x_1, x_2, \dots, x_n$ genera una secuencia de nuevos embeddings $x'_1, x'_2, \dots, x'_n$, donde cada x'_i se define como una combinación lineal de todos los x_j en la secuencia⁶⁵:

$$x'_i = \sum_{j=1}^n w_{ij} v_j \quad (5.39)$$

Aquí, los coeficientes w_{ij} son los *pesos de atención* (*attention weights*) que determinan la relevancia de cada x_j para el token x_i . Estos pesos se calculan utilizando el mecanismo de *atención escalada por producto punto* (*scaled dot-product attention*). El proceso consta de cuatro pasos principales, como se ilustra en la Figura 5.15.



⁶⁴ A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017

⁶⁵ L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers*. O'Reilly Media, Inc., revised ed., 2022

Figure 5.15: Esquema de la atención escalada por producto punto (izquierda) y atención de múltiples cabezas (derecha). Figura retomada de A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

El procedimiento de la *atención escalada por producto punto* es el siguiente:

Primero, los embeddings de los tokens de entrada se proyectan en tres vectores distintos: *queries* (Q), *keys* (K) y *values* (V). El objetivo es medir la similitud entre cada Q y todas las K mediante la multiplicación de matrices $QK^\top$, como se muestra en la operación de **MatMul** en la Figura 5.15 (lado izquierdo). Este producto produce una matriz de puntajes de atención de tamaño $n \times n$, donde n es el número de tokens en la secuencia. Las consultas y claves con mayor similitud generan puntajes más altos ⁶⁶.

En segundo lugar, los puntajes de atención obtenidos se escalan dividiéndolos por $\sqrt{d_k}$, donde d_k es la dimensión de los vectores clave. Este escalado evita que los valores resultantes sean demasiado grandes, lo que podría causar inestabilidad en los gradientes durante el entrenamiento ⁶⁷. Además, en algunos casos se aplica una máscara opcional (**Mask (opt.)**) para bloquear ciertos tokens, como ocurre en tareas autoregresivas.

A continuación, los puntajes escalados se normalizan utilizando la función *softmax*, como se observa en el bloque **SoftMax** de la Figura 5.15. La función *softmax* convierte los puntajes de atención en probabilidades, asegurando que los pesos de atención w_{ij} en cada fila sumen 1. La ecuación para el cálculo de la atención es:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V \quad (5.40)$$

Este paso permite que el modelo distribuya su atención de manera ponderada entre los diferentes tokens de la secuencia, asignando mayor peso a los tokens más relevantes dentro del contexto.

Finalmente, los pesos de atención obtenidos se utilizan para combinar los vectores de valor (*values*). Esta operación se realiza mediante la multiplicación de los pesos de atención por los valores V , resultando en una nueva representación ponderada para cada token. La operación final es la segunda multiplicación de matrices (**MatMul**) indicada en la figura. De este modo, la representación x'_i se calcula como una combinación lineal de los valores v_j , ponderados por los pesos de atención w_{ij} , como se describe en la ecuación (5.39).

La Figura 5.15 (lado derecho) muestra cómo el mecanismo de *atención escalada por producto punto* se extiende al mecanismo de *multi-head attention*. Cada cabeza de atención calcula sus propios pesos y produce sus propias representaciones, las cuales luego se concatenan y se proyectan de nuevo al espacio de la dimensión original del modelo.

⁶⁶ L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers*. O'Reilly Media, Inc., revised ed., 2022

⁶⁷ A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017

(2) Multi-Head Attention

El uso de una única cabeza de atención puede limitar la capacidad del modelo para capturar diferentes tipos de relaciones entre los tokens. Para resolver este problema, el mecanismo de autoatención se extiende a un esquema de *atención multi-cabeza* (*multi-head attention*). En este esquema, Q , K y V se dividen en múltiples subconjuntos, cada uno de los cuales es procesado de forma independiente por una cabeza de atención distinta.

Cada una de estas cabezas aplica su propia versión del mecanismo de atención escalada por producto punto, capturando distintos aspectos de las relaciones contextuales entre los tokens. Al final, las salidas de todas las cabezas de atención se concatenan y se proyectan nuevamente en el espacio de la dimensión original del modelo (d_{model}) mediante una transformación lineal. La ecuación⁶⁸ que describe este proceso es:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (5.41)$$

donde head_i es el resultado de aplicar la atención escalada sobre el subconjunto i -ésimo de las proyecciones de Q , K y V , y W^O es la matriz de proyección final.

Feed-Forward Layer

Después del mecanismo de atención multi-cabeza, cada capa del encoder incluye una red completamente conectada o *feed-forward layer*. Esta red se aplica de manera independiente a cada token de la secuencia y consiste en dos transformaciones lineales separadas por una función de activación no lineal (usualmente *ReLU*). Formalmente, este proceso se puede describir como:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (5.42)$$

donde W_1 y W_2 son las matrices de pesos de las capas lineales y b_1 y b_2 son los respectivos sesgos.

Esta capa sirve para procesar de manera más profunda las representaciones de los tokens obtenidas del mecanismo de atención, introduciendo una no linealidad adicional en el modelo..

Adding Layer Normalization

En cada subcapa del encoder, tanto en el mecanismo de autoatención como en la capa *feed-forward*, se utiliza una *conexión residual*, es decir, la entrada de la subcapa se suma a su salida. Posteriormente, el resultado

⁶⁸ A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017

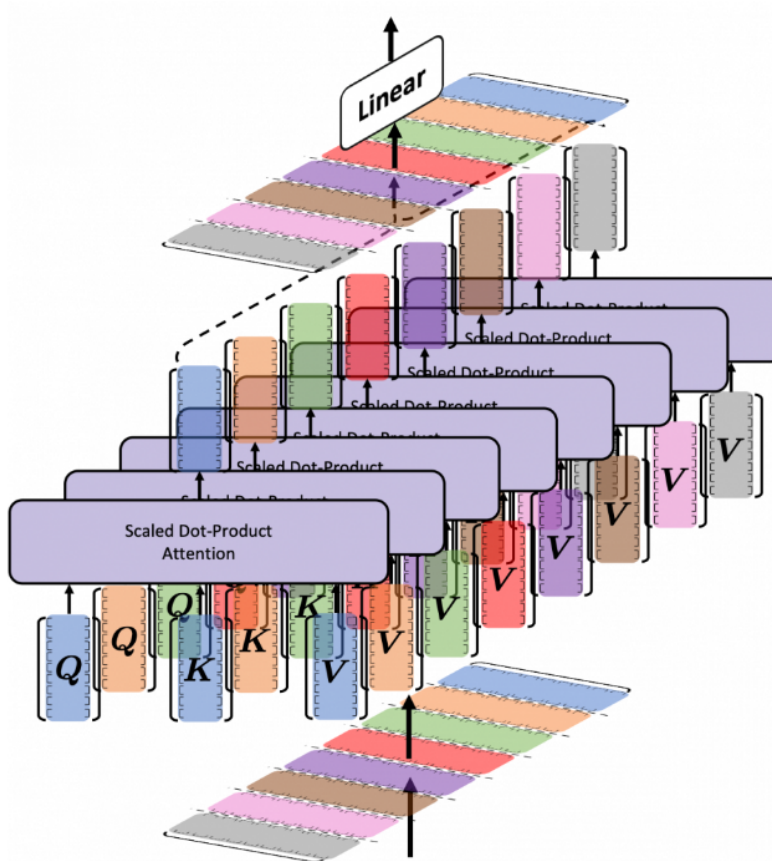


Figure 5.16: Esquema del mecanismo de *Atención Multi-Cabeza* en el modelo Transformer. La secuencia de entrada se proyecta en matrices de *queries* (Q), *keys* (K) y *values* (V), que luego se procesan de manera paralela en varias cabezas de atención. Cada cabeza de atención aplica el mecanismo de atención escalada por producto punto (*Scaled Dot-Product Attention*), enfocándose en diferentes partes de la secuencia. Las salidas de todas las cabezas se concatenan y proyectan a través de una capa lineal, generando una representación final que se utiliza en las siguientes capas del modelo. Figura retomada de R. M. Go, *Deep Learning Bible - Part K: Transformer from scratch*, ch. K1.2. How it works, step by step. Mintrobot, 2024.

se normaliza utilizando la técnica de normalización por capas (*layer normalization*) para estabilizar el proceso de entrenamiento. Este proceso es formalizado mediante la expresión:

$$\text{LayerNorm}(x + \text{Sublayer}(x)) \quad (5.43)$$

La *LayerNorm* ayuda a mitigar problemas asociados a la distribución de activaciones y a mejorar la convergencia durante el entrenamiento. En este modelo, se emplean tanto la normalización por capas como las *conexiones residuales* (*skip connections*). La normalización por capas se utiliza para ajustar cada entrada del batch a una media cero y varianza unitaria, mientras que las conexiones residuales permiten pasar tensores no procesados a la siguiente capa y sumarlos a los tensores procesados.

Existen dos configuraciones principales para colocar la normalización por capas dentro del modelo Transformer:

Post-layer normalization: Utilizada en el artículo original del Transformer, coloca la normalización por capas después de las conexiones residuales. Aunque efectiva, es difícil de entrenar desde cero debido a la divergencia de gradientes, por lo que se requiere un *learning rate warm-up* para estabilizar el proceso de entrenamiento⁶⁹.

Pre-layer normalization: Común en la literatura reciente, coloca la normalización por capas antes de las conexiones residuales. Es mucho más estable durante el entrenamiento y no suele requerir un *learning rate warm-up*, lo que ha llevado a su adopción en muchas implementaciones modernas de Transformers⁷⁰.

Positional embeddings

Como se mencionó anteriormente, el mecanismo de *multi-head attention* se basa en una suma ponderada de los valores (V) de los tokens, lo que lo hace invariante a la posición de los mismos. Esto implica que, por sí sola, la capa de atención no puede capturar el orden de los tokens en la entrada, ya que esta información se pierde durante el cálculo de la atención. Para resolver esto, se introducen los *positional embeddings*, que codifican de manera explícita la información posicional y se suman a los *token embeddings* al inicio del proceso de transformación. De este modo, el modelo puede considerar tanto el contenido semántico de los tokens como su ubicación relativa dentro de la secuencia, proporcionando al Transformer la capacidad de capturar el orden y las relaciones posicionales entre los tokens.

Los *positional embeddings* sinusoidales son una de las formas más

⁶⁹ L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers*. O'Reilly Media, Inc., revised ed., 2022

⁷⁰ L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers*. O'Reilly Media, Inc., revised ed., 2022

comunes de representar la posición de los tokens en la secuencia. Estos embeddings se definen mediante funciones seno y coseno que varían en función de la posición del token pos y de la dimensión i dentro del vector de representación. Las fórmulas⁷¹ que describen estos embeddings posicionales son:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \quad (5.44)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \quad (5.45)$$

Aquí, d_{model} es la dimensionalidad de los embeddings, pos es la posición del token en la secuencia y i es la dimensión específica del vector de representación. Estas funciones sinusoidales crean un patrón periódico que varía a lo largo de las diferentes dimensiones de los embeddings, permitiendo que el modelo aprenda tanto dependencias locales como relaciones a largo plazo entre los tokens.

La elección de 10000 en el denominador es clave para el manejo de las frecuencias dentro de los embeddings. El término $10000^{2i/d_{\text{model}}}$ asegura que cada dimensión del embedding utilice una frecuencia distinta. Las dimensiones más bajas (i pequeño) tienen frecuencias más altas, lo que permite al modelo capturar relaciones posicionales de corto alcance, mientras que las dimensiones más altas (i grande) capturan relaciones a mayor escala debido a las frecuencias más bajas. Esto garantiza que el modelo pueda detectar patrones a diferentes escalas dentro de una secuencia⁷².

Además de proporcionar una representación posicional, las funciones seno y coseno tienen propiedades importantes para el proceso de entrenamiento. Las derivadas de $\sin(x)$ y $\cos(x)$ son $\cos(x)$ y $-\sin(x)$, respectivamente. Estas derivadas bien definidas permiten un flujo de gradientes estable durante el entrenamiento, lo que es crucial para la retropropagación. Al utilizar la constante $10000^{2i/d_{\text{model}}}$, se moderan las variaciones en las frecuencias de las funciones sinusoidales, evitando gradientes desaparecidos o explosivos, lo cual asegura un aprendizaje eficiente y estable a lo largo de las capas profundas del Transformer⁷³.

Otro aspecto destacado de los *positional embeddings* sinusoidales es su capacidad para generalizar a secuencias de longitud mayor que las vistas durante el entrenamiento. Debido a la naturaleza periódica de las funciones seno y coseno, el modelo es capaz de extrapolar posiciones no observadas previamente, permitiendo que el Transformer maneje

⁷¹ A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017

⁷² F. Preetham, "Math behind positional embeddings in transformer models," May 2024

⁷³ F. Preetham, "Math behind positional embeddings in transformer models," May 2024

secuencias largas sin necesidad de modificar los embeddings o realizar un nuevo entrenamiento.

Existen diversas formas de implementar los *positional embeddings*. Además de los embeddings sinusoidales, llamadas *representaciones de posición absolutas* basadas en patrones estáticos, como se mencionó anteriormente, también existen las *representaciones de posición relativa*⁷⁴.

Recientemente, algunos modelos como DeBERTa han adoptado *representaciones de posición relativas*. En lugar de codificar la posición absoluta de un token en la secuencia, estas representaciones se enfocan en capturar las relaciones entre las posiciones de los tokens dentro de su contexto. Esto refleja de manera más precisa las dependencias contextuales entre los tokens y permite que el modelo adapte su atención en función de la posición relativa de los tokens en la secuencia. Para implementar este enfoque, se requieren modificaciones adicionales en el mecanismo de atención, ya que las posiciones relativas no pueden representarse simplemente sumando un embedding de posición al vector de entrada. Este tipo de representación es particularmente útil en tareas donde la relación entre los tokens es más importante que su posición absoluta dentro de la secuencia⁷⁵.

Decoder

El *decoder* es el segundo componente clave del modelo Transformer, responsable de generar la secuencia de salida $(y_1, y_2, \dots, y_m)$, basándose en las representaciones intermedias z producidas por el encoder. A diferencia del encoder, el decoder está diseñado para procesar una secuencia de salida generada de manera autoregresiva, lo que significa que en cada paso de tiempo t , la predicción y_t se basa tanto en las representaciones del encoder como en las predicciones anteriores del decoder.

La estructura interna del decoder se organiza en N capas apiladas, como se observa en la Figura 5.18. Cada una de estas capas sigue un proceso similar para generar una salida adecuada. El primer paso es la incorporación del *embedding* de entrada del decoder, seguido de la adición de una codificación posicional sinusoidal para incluir información de posición dentro de la secuencia de entrada. A continuación, se aplican dos máscaras. La primera máscara, conocida como *Self Padding Mask*, se crea a partir de las entradas del encoder, lo que permite ignorar tokens de relleno (padding) en la secuencia. La segunda máscara, llamada *Look Ahead Mask*, se genera a partir de las entradas del decoder, evitando que el modelo vea información futura durante la generación de la secuencia.

⁷⁴ L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers*. O'Reilly Media, Inc., revised ed., 2022

⁷⁵ L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers*. O'Reilly Media, Inc., revised ed., 2022

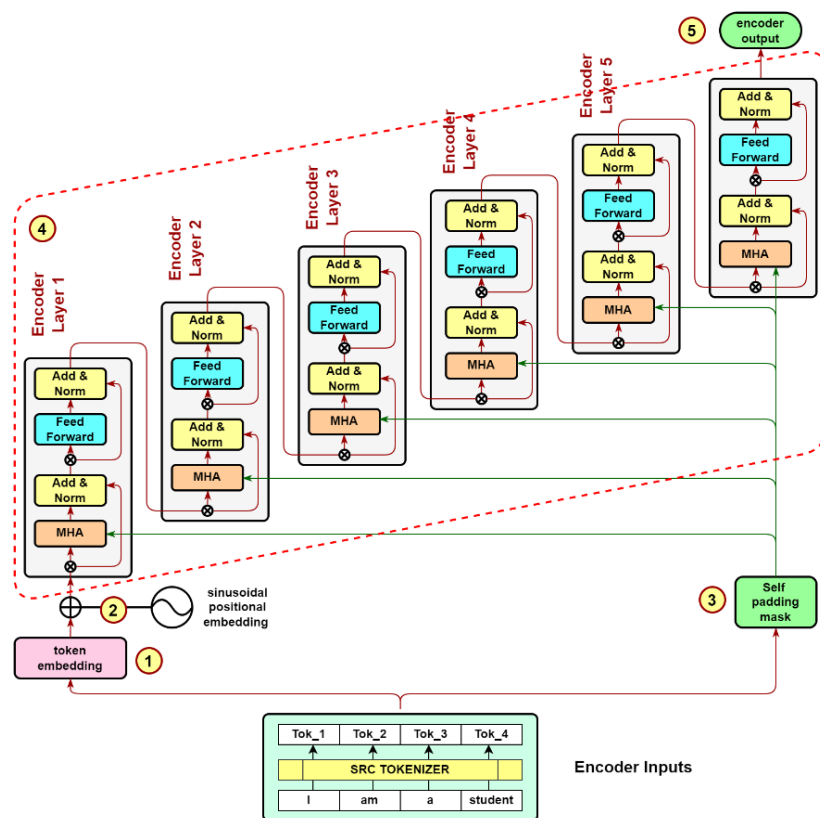


Figure 5.17: Esquema del proceso completo en el encoder de un Transformer. (1) Los tokens de entrada se convierten en *embeddings* a través de un tokenizer. (2) Se suman *positional embeddings* sinusoidales para incorporar la información de posición en la secuencia. (3) Se aplica una máscara de padding (*Self Padding Mask*) para ignorar tokens vacíos durante la autoatención. (4) La secuencia de tokens pasa a través de N capas del encoder, cada una compuesta de bloques de autoatención de múltiples cabezas (*MHA*), redes *feed-forward*, y conexiones residuales con normalización (*Add & Norm*). (5) La salida del encoder es una representación codificada de la secuencia de entrada, que será utilizada por el decodificador. Imagen tomada de R. M. Go, *Deep Learning Bible - Part K: Transformer from scratch*, ch. K1.2. How it works, step by step. Mintrobot, 2024.

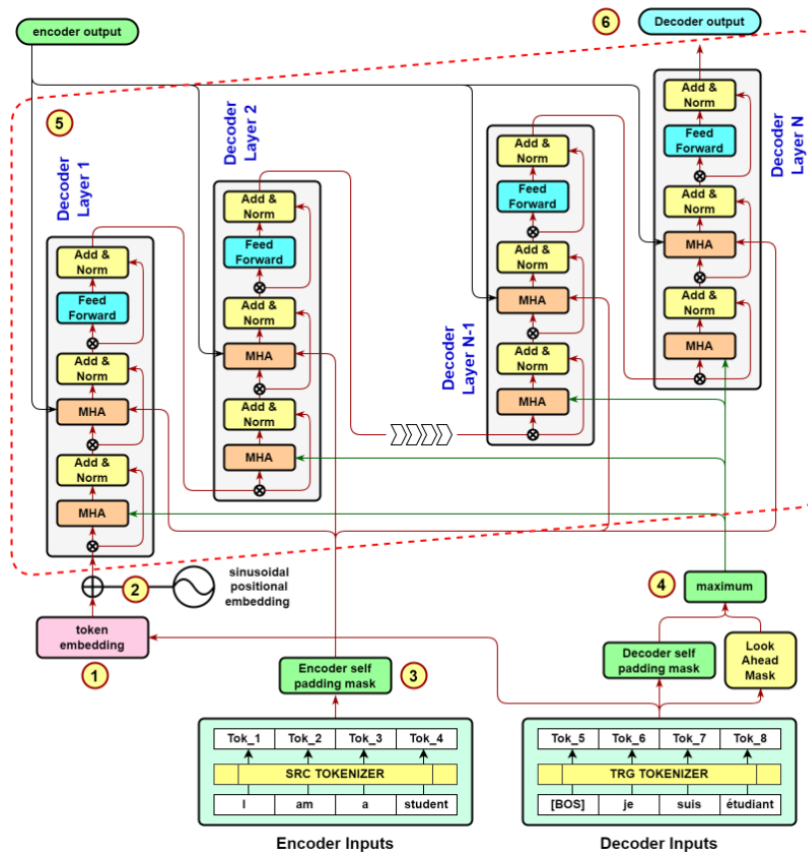


Figure 5.18: Esquema del proceso en el decodificador con N capas en un Transformer. El decodificador genera la salida en los siguientes pasos: (1) Se incorpora el embedding del token de entrada del decodificador. (2) Se añade la codificación posicional sinusoidal para incorporar la información de posición en la secuencia de entrada del decodificador. (3) Se crea una máscara de padding a partir de las entradas del codificador para ignorar los tokens de relleno. (4) Se genera una máscara de anticipación (*Look Ahead Mask*) a partir de las entradas del decodificador, evitando que el modelo vea información futura durante la generación. (5) Las N capas del decodificador están apiladas, donde cada capa realiza operaciones de autoatención multi-cabeza y atención entre el encoder y el decodificador. (6) Finalmente, la salida de la última capa del decodificador se utiliza para predecir el próximo token de la secuencia de salida. Imagen tomada de R. M. Go, *Deep Learning Bible - Part K: Transformer from scratch*, ch. K1.2. How it works, step by step. Mintrobot, 2024.

Dentro de cada capa del decoder, el proceso continúa con la primera atención multi-cabeza. En este caso, las (Q), (K) y (V) son iguales y provienen de la entrada del decoder o de la salida del decoder anterior. Esta atención está enmascarada para garantizar que la generación sea autoregresiva, es decir, que el modelo solo considere los tokens generados previamente. La salida de esta atención multi-cabeza se pasa a través de una conexión residual, que suma la entrada de la subcapa con su salida y luego se normaliza mediante *LayerNorm*.

Después de la primera atención, se aplica una segunda atención multi-cabeza, conocida como *Encoder-Decoder Attention*. En esta etapa, las consultas (Q) provienen de la salida de la primera atención multi-cabeza del decodificador, mientras que las claves (K) y valores (V) provienen de las representaciones generadas por el encoder. Este mecanismo de atención permite al decoder enfocarse en la información relevante de la secuencia de entrada durante la generación de la secuencia de salida. Al igual que en la atención anterior, se añade una conexión residual seguida de normalización por capas (*LayerNorm*).

Posteriormente, cada capa del decoder incluye una red *feed-forward* que procesa cada token de manera independiente. Esta red consiste en dos transformaciones lineales separadas por una función de activación no lineal, generalmente *ReLU*. Al igual que las subcapas de atención, la red *feed-forward* también se normaliza con una capa residual y *LayerNorm*.

El decoder se apila en $N = 6$ capas idénticas⁷⁶, cada una de ellas utilizando conexiones residuales alrededor de cada subcapa, seguida de normalización por capas (*LayerNorm*). Esta arquitectura asegura una propagación estable de los gradientes y mejora la convergencia durante el entrenamiento. En cuanto a la autoatención, esta se modifica mediante una máscara especial que previene que los tokens futuros influyan en la predicción de un token dado, asegurando que las predicciones se basen únicamente en la secuencia generada hasta el momento. Este mecanismo es crucial para mantener la naturaleza autoregresiva del decodificador.

Finalmente, la salida de la última capa del decoder se utiliza para predecir el próximo token en la secuencia, o bien se pasa a la siguiente capa del decoder si hay más capas apiladas. Este proceso continúa hasta que se genera la secuencia completa de salida.

El decoder también emplea una máscara de autoatención (*Look Ahead Mask*), que bloquea el acceso a información futura, lo cual es vital para

⁷⁶ De acuerdo con el ejemplo de artículo original.

evitar que el modelo prediga de manera trivial los tokens subsecuentes basándose en los futuros. Esto asegura que la predicción en una posición i dependa únicamente de los tokens generados previamente.

5.7 Tareas específicas

Es fundamental conocer las distinciones entre los *diferentes tipos de modelos* basados en Transformers para seleccionar la arquitectura más adecuada según la *tarea específica*. Aunque comparten una estructura base, los modelos resuelven distintos problemas: los modelos *autoregresivos* como GPT son ideales para la generación de texto, mientras que los *autoencoding* como BERT son más efectivos para tareas de clasificación y comprensión contextual. Modelos *multimodales* permiten integrar texto con otros tipos de datos, como imágenes y los *retrieval-based* acceden a información externa para mejorar el rendimiento en tareas como respuesta a preguntas. Además, modelos como los *seq-to-seq* son cruciales para aplicaciones como la traducción automática. A continuación, se describen las tres tareas principales de este estudio:

5.7.1 Summarization

La tarea de *summarization* o resumen automático se refiere a generar una versión más corta y condensada de un texto, preservando la información más relevante. Modelos basados en la arquitectura *Generative Pre-trained Transformer*, (GPT), un modelo de tipo *autoregresivo*, han demostrado un excelente desempeño en esta tarea. Estos modelos están preentrenados para generar secuencias de texto previendo el siguiente token basándose en el contexto anterior, lo que les permite generar resúmenes coherentes y bien estructurados. En el caso de la tarea de resumen, el modelo toma un texto extenso como entrada y genera una versión reducida, manteniendo la coherencia y los detalles clave del contenido original.

5.7.2 Named Entity Recognition (NER)

La tarea de *Named Entity Recognition* (NER) consiste en identificar y clasificar entidades nombradas en un texto, como personas, organizaciones, lugares, fechas, entre otras. Un modelo ampliamente utilizado para esta tarea es *BERT*, el cual, por ser un modelo de *autoencoding*, permite capturar representaciones profundas y contextuales de las secuencias de texto. Además, la variante *GLNER* de BERT, que está específicamente diseñada para mejorar el reconocimiento de entidades, ha mostrado resultados sobresalientes en NER. Este enfoque permite que el modelo procese el contexto completo

de una oración, lo que facilita la identificación precisa de entidades dentro de la misma.

5.7.3 *Text Classification*

La tarea de *text classification* o clasificación de texto se refiere a asignar etiquetas a fragmentos de texto en función de su contenido. Un enfoque popular para esta tarea es el uso de *Word2Vec*, una técnica que genera representaciones distribuidas (embeddings) de palabras basadas en el contexto en el que aparecen. A diferencia de los modelos transformers, *Word2Vec* genera embeddings de palabras estáticos, pero sigue siendo útil en tareas de clasificación de texto al entrenar modelos que asocien palabras con categorías específicas. En este caso, se entrena un modelo sobre los embeddings para aprender patrones que permitan categorizar correctamente nuevos textos basados en las similitudes semánticas de las palabras.

5.8 *ANOVA: Análisis de Varianza*

El Análisis de Varianza (ANOVA) es una técnica estadística utilizada para comparar las medias de varias poblaciones a partir del estudio de sus varianzas. En este estudio, se utilizó un ANOVA de un solo factor para evaluar el impacto de los diferentes escenarios experimentales (*Baseline*, *Parcial* y *Completo*) sobre las métricas clave (*accuracy*, *precision*, *recall* y *f1-score*).

5.8.1 *Modelo estadístico*

El modelo estadístico para el ANOVA puede expresarse como:

$$y_{ij} = \mu + \tau_i + \epsilon_{ij}, \quad (5.46)$$

donde:

- y_{ij} : Observación j -ésima del grupo i .
- μ : Media global de todas las observaciones.
- τ_i : Efecto del tratamiento (escenario) i , es decir, la desviación de la media del grupo i respecto a la media global.
- ϵ_{ij} : Error aleatorio asociado a la observación y_{ij} , que se asume normalmente distribuido: $\epsilon_{ij} \sim N(0, \sigma^2)$.

5.8.2 *Formulación de la hipótesis*

Para evaluar si existen diferencias significativas entre los escenarios, se plantearon las siguientes hipótesis:

- **Hipótesis nula (H_0):** Todas las medias de los grupos son iguales, es decir:

$$H_0 : \mu_{\text{Baseline}} = \mu_{\text{Partial}} = \mu_{\text{Complete}} \quad (5.47)$$

- **Hipótesis alternativa (H_A):** Al menos una de las medias de los grupos es diferente, es decir:

$$H_A : \mu_i \neq \mu_j. \quad (5.48)$$

5.8.3 Fórmulas del ANOVA

Para realizar el ANOVA, se descompone la variabilidad total (SC_{Tot}) en dos componentes: la variabilidad entre grupos (SC_{Trt}) y la variabilidad dentro de los grupos (SC_{Error}). Las fórmulas utilizadas son:

Suma de cuadrados total (SC_{Tot})

$$SC_{Tot} = \sum_{i=1}^k \sum_{j=1}^n (y_{ij} - \bar{Y})^2 = \sum_{i=1}^k \sum_{j=1}^n (y_{ij})^2 - \frac{Y^2}{N}, \quad (5.49)$$

donde:

- y_{ij} : Observación individual.
- $\bar{Y}$: Media global.
- Y^2 : Cuadrado de la suma total de las observaciones.
- N : Número total de observaciones.

Suma de cuadrados entre grupos (SC_{Trt})

$$SC_{Trt} = \sum_{i=1}^k n \cdot (\bar{y}_i - \bar{Y})^2 = \frac{\sum_{i=1}^k n (\sum_{j=1}^n y_{ij})^2}{n} - \frac{Y^2}{N}, \quad (5.50)$$

donde:

- $\bar{y}_i$: Media del grupo i .
- n : Número de observaciones por grupo (asumiendo balanceo).

Suma de cuadrados residual (SC_{Error})

$$SC_{Error} = \sum_{i=1}^k \sum_{j=1}^n (y_{ij} - \bar{y}_i)^2 = \sum_{i=1}^k n \sum_{j=1}^n (y_{ij})^2 - \sum_{i=1}^k \frac{(\sum_{j=1}^n y_{ij})^2}{n}. \quad (5.51)$$

Relación entre las sumas de cuadrados

La relación entre las sumas de cuadrados es:

$$SC_{Tot} = SC_{Trt} + SC_{Error}. \quad (5.52)$$

5.8.4 Criterio de decisión

El estadístico se calcula como la razón entre el cuadrado medio entre tratamientos (SC_{Trt}) y el cuadrado medio del error (SC_{Error}):

$$F = \frac{CM_{Trt}}{CM_{Error}}, \quad (5.53)$$

donde:

- $CM_{Trt} = \frac{SC_{Trt}}{df_{Trt}}$, con $df_{Trt} = k - 1$.
- $CM_{Error} = \frac{SC_{Error}}{df_{Error}}$, con $df_{Error} = N - k$.

Se compara el valor obtenido de con el valor crítico de la distribución para un nivel de significancia dado ($\alpha = 0.05$) o se evalúa el valor :

- Si $p < \alpha$, se rechaza y se concluye que hay diferencias significativas entre los grupos.
- Si $p \geq \alpha$, no se rechaza .

6 Desarrollo metodológico

Contents

6.1	Motivación	95
6.2	Conjunto de datos: <i>OpenFoodFacts</i>	96
6.2.1	Acceso a los datos	97
6.2.2	Preparación de los datos	98
6.3	Motivación para el uso de <i>GLiNER</i>	101
6.3.1	El Modelo <i>GLiNER</i>	102
6.4	Motivación para generación de conjunto de datos y uso de Web Scrapping	104
6.4.1	Extracción de datos mediante Web Scrapping	104
6.4.2	Tarea de resumir el texto <i>summarization</i>	105
6.4.3	Proceso de tokenización y Reconocimiento de Entidades Nombradas (NER)	107
6.5	Ajuste (<i>Fine-tuning</i>) de <i>GLiNER</i>	109
6.6	Entrenamiento de Word2Vec para la clasificación de alimentos	112
6.7	Métricas de evaluación	117
6.7.1	Cálculo del tamaño de muestra para evaluación NER	123
6.8	Análisis estadístico de la hipótesis	124
6.8.1	Descripción de los escenarios	124
6.8.2	Variables de análisis	126
6.8.3	Metodología estadística	127
6.8.4	Definición de métricas de desempeño	128

6.1 Motivación

ESTE CAPÍTULO detalla el proceso metodológico implementado para cumplir con los objetivos del proyecto, centrados en proponer un flujo de trabajo para la clasificación de productos alimenticios a partir de listas de ingredientes extraídas de la base de datos colaborativa

OpenFoodFacts. Estas listas incluyen tanto ingredientes, como aditivos alimentarios, lo que introduce un desafío para el entrenamiento de los modelos de clasificación, ya que interesa explorar la separación de ambas entidades y categorizar correctamente estas dos clases antes de su procesamiento.

El principal desafío radica en que las listas de ingredientes obtenidas provienen de sistemas de Reconocimiento Óptico de Caracteres (OCR) y de introducir de forma manual los datos por parte de las personas usuarias debido a la naturaleza colaborativa de esta base de datos (*crowdsourcing*), lo que puede dar lugar a situaciones como la captura incompleta de caracteres o la transcripción literal de los ingredientes tal como aparecen en las etiquetas. Esto conduce a que los ingredientes y aditivos se mezclen en un solo conjunto. Dado que es de interés analizar ambas entidades por separado, fue necesario desarrollar un método para limpiar y estructurar estos datos, diferenciando de manera clara entre alimentos y aditivos.

Con base en lo anterior, se realizó una clasificación de entidades mediante un ajuste *fine-tuning* de un modelo de Reconocimiento de Entidades Nombradas (*Named Entity Recognition, NER por sus siglas en inglés*) llamado *GLiNER*. Para entrenar este modelo, se creó un conjunto de datos específico, obteniendo información mediante *web scraping* de páginas web de Wikipedia, basada en el listado oficial de aditivos alimentarios del Codex Alimentarius y haciendo uso de la API (*Application Programming Interface*) de OpenAI para estructurar los datos.

Finalmente, los datos fueron utilizados para entrenar un modelo de *word embeddings* basado en Word2Vec, que permitió clasificar los alimentos. El flujo de trabajo se desarrolló en tres etapas, las cuales comprenden desde la extracción de los datos, hasta la etapa de clasificación. Este proceso se detalla más adelante y se ilustra en la Figura 6.6.

6.2 Conjunto de datos: *OpenFoodFacts*

El presente trabajo utiliza el conjunto de datos proporcionado por la base de datos colaborativa *OpenFoodFacts*¹, la cual recopila información sobre productos alimenticios de todo el mundo. Este repositorio es de acceso abierto y contiene información relevante sobre ingredientes, aditivos alimentarios, etiquetas nutricionales, imágenes del producto, entre otros datos relevantes. Para este estudio, los datos fueron obtenidos a través de un SDK² (*Software Development Kit*, por sus siglas en inglés) que se conecta a la API pública de *OpenFoodFacts*,

¹ OpenFoodFacts, “Open food facts - world food products database.” <https://world.openfoodfacts.org/>, 2024. [En línea; consultado el 29 de marzo de 2024]

² OpenFoodFacts, “Open food facts python sdk.” <https://github.com/openfoodfacts/openfoodfacts-python>, 2016–2024. MIT License

y que permite acceder a los datos. Adicionalmente, *OpenFoodFacts* ofrece exportaciones completas de su base de datos en varios formatos, incluidos JSONL, CSV, RDF y dumps de MongoDB.

6.2.1 Acceso a los datos

El uso de la base de datos de *OpenFoodFacts* está regulado por un conjunto de licencias que promueven la reutilización abierta de la información, bajo condiciones que garantizan el respeto a los derechos de autor y la transparencia en el uso de los datos. La base de datos tiene la licencia de **Open Data Commons Open Database License (ODbL)**, lo que permite reutilizar la información siempre que se otorgue crédito a *OpenFoodFacts* y cualquier obra derivada mantenga esta misma licencia. Los contenidos individuales de la base de datos, como los datos sobre productos alimenticios, están sujetos a la licencia **Database Contents License (DbCL)**, mientras que las imágenes de los productos están disponibles bajo la licencia **Creative Commons Attribution-ShareAlike (CC BY-SA)**. Esta última licencia permite la reutilización de las imágenes siempre que se mantenga la atribución al autor original y se respeten los derechos sobre elementos gráficos que pudieran estar sujetos a copyright.

Es importante destacar que *OpenFoodFacts* no garantiza la exactitud ni la completitud de los datos presentes en su plataforma, dado que la información es recopilada de forma colaborativa por una comunidad global de personas usuarias y puede contener errores derivados de la entrada manual de datos o de imprecisiones en las etiquetas de los productos.

Con el SDK de Python oficial de *OpenFoodFacts*, se interactuó con la API de *OpenFoodFacts* para acceder a los datos de productos alimenticios. Para facilitar el procesamiento, el conjunto de datos fue descargado en formato ZIP el 29 de marzo de 2024, conteniendo un archivo CSV. Este archivo incluye más de 3 millones de registros de productos alimenticios y 206 columnas con información diversa, como nombres, categorías, ingredientes y detalles de los propietarios de las marcas.

Una vez descargado el archivo CSV comprimido, se utilizó la biblioteca Pandas en Python para filtrar y extraer solo las columnas relevantes para este estudio.

El archivo CSV original contiene 3,130,011 de registros. Sin embargo, tras aplicar filtros y seleccionar solo las columnas necesarias, el conjunto de datos se redujo a un DataFrame con 3 columnas relevantes para la clasificación de productos alimenticios. De estas, se

utilizó exclusivamente la columna (`ingredients_text`), que presenta 2,191,292 registros sin datos³ (70%) y 938,719 registros con datos (30%).

Nombre de variable	Tipo de variable	Tipo de dato
<code>code</code>	id	string
<code>ingredients_text</code>	texto	string
<code>main_category_en</code>	texto	string

³ Es decir, aquellas con datos faltantes, representados como "nan".

Table 6.1: Descripción de variables del conjunto de datos.

6.2.2 Preparación de los datos

Durante la inspección de los datos textuales (`ingredients_text`), se observó que los textos estaban mezclados entre varios idiomas. Es decir, un registro podría contener información en inglés, mientras que el siguiente podría estar en idioma alemán o algún otro. En consecuencia, para la detección de los idiomas en el conjunto de datos, se utilizó el modelo preentrenado de *FastText*⁴, desarrollado por Facebook y especializado en la identificación de idiomas. Este modelo fue descargado y cargado en el entorno de trabajo para procesar el campo `ingredients_text` y determinar el idioma de cada registro. El modelo preentrenado, almacenado en el archivo `lid.176.bin`, se obtuvo desde los repositorios oficiales de *FastText*.

⁴ F. A. Research, "Fasttext pretrained language identification model." <https://dl.fbaipublicfiles.com/fasttext/supervised-models/lid.176.bin>, 2024. [En línea; consultado el 29 de marzo de 2024]

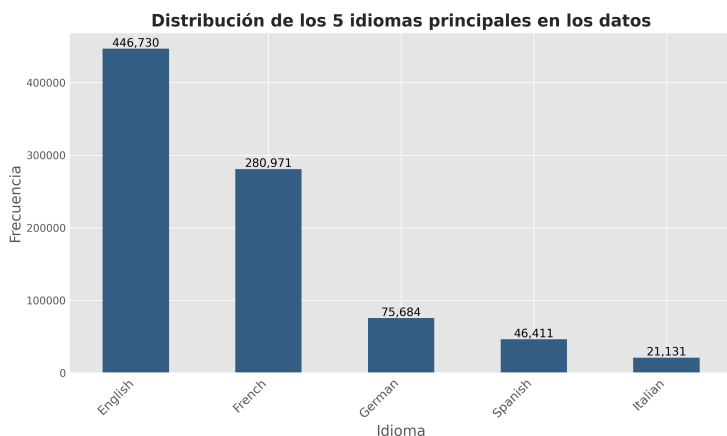


Figure 6.1: Distribución de los cinco idiomas principales en los datos.

Una vez detectados los códigos de idioma en el conjunto de datos mediante *FastText*, se procedió a convertir dichos códigos a sus nombres descriptivos correspondientes. Para realizar esta conversión, se empleó la biblioteca *langcodes*⁵, que permite mapear los códigos de idioma a sus nombres comunes. En total, se identificaron 128 idiomas, de los cuales los cinco más comunes se presentan en la Figura 6.1.

⁵ L. Technologies, "Langcodes: A library for working with language codes." <https://pypi.org/project/langcodes/>, 2024. [En línea; consultado el 29 de marzo de 2024]

Con base en este análisis, se decidió trabajar únicamente con los datos en inglés, lo que redujo el conjunto de datos a 446,730 registros.

Por otro lado, la variable *y* (categorías) de los registros anteriores incluye un total de 13,292 categorías diferentes de productos. Sin embargo, muchas de estas categorías presentaron solo un registro o un número muy bajo de observaciones, por lo que se seleccionó una *muestra de 10 categorías* con un total de 42,027 **registros**, conformando así el conjunto de datos final.

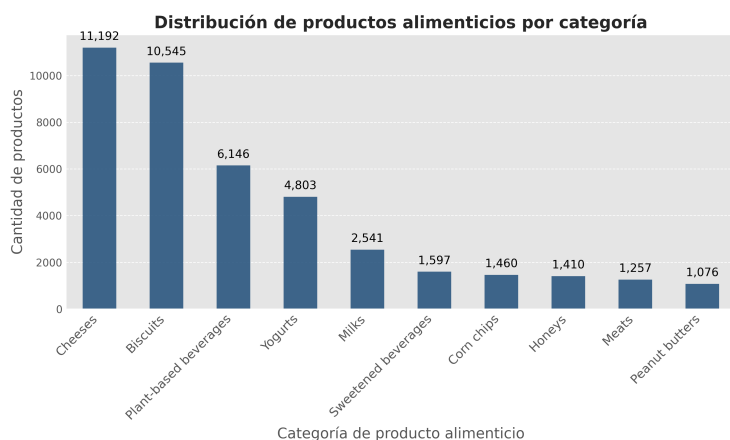


Figure 6.2: Frecuencia de las categorías y del conjunto de datos final.

Las categorías a clasificar fueron: *Cheeses*, *Biscuits*, *Plant-based beverages*, *Yogurts*, *Milks*, *Sweetened beverages*, *Corn chips*, *Honey's*, *Meats* y *Peanut butters*, como se observa en la Figura 6.2. Esta selección se basó en la frecuencia de registros de cada categoría dentro del conjunto de datos, eligiendo aquellas con mayor cantidad de registros. Asimismo, se descartaron algunas categorías de acuerdo con los intereses y objetivos definidos por el equipo de trabajo.

Hasta este punto, el conjunto de datos presentaba un reto adicional: las listas de ingredientes incluían tanto alimentos, como aditivos. Esta situación es particularmente importante ya que, para entrenar los modelos de clasificación es de interés separar los aditivos de los ingredientes principales. Aquí entra el uso de un modelo NER, *GLiNER*.

A continuación se presenta el pseudocódigo que describe el flujo de procesamiento de los datos obtenidos desde *OpenFoodFacts* hasta la generación del conjunto de datos final, antes de que se aplique el modelo ajustado de *GLiNER*. Este proceso también se ilustra en la Figura 6.3.

Algoritmo 1: Flujo de procesamiento de datos de OpenFood-Facts en Pseudocódigo

Datos: ruta_entrada_dataset, ruta_salida_filtrada

Resultado: Datos procesados guardados en ruta_salida_filtrada

Procedimiento PROCESARDATOS(*ruta_entrada_dataset*,
ruta_salida_filtrada):

(1): Descarga y extracción de datos

Instalar el paquete openfoodfacts;
dataset ← Descargar datos usando ProductDataset("csv");
Descomprimir el archivo descargado;
Cargar los datos desde la ruta ruta_entrada_dataset;

(2): Filtrado de columnas

datos_originales ← Cargar dataset desde
ruta_entrada_dataset;
columnas_seleccionadas ← ['ingredients_text',
'main_category_en'];
datos_filtrados ← Filtrar datos_originales por
columnas_seleccionadas;

(3): Eliminación de datos faltantes

datos_filtrados ← Eliminar filas con valores faltantes en
'ingredients_text';

Función DETECTARIDIOMA(*texto*):

si *texto* es una cadena **entonces**
 codigo_idioma ← fasttext.detect(texto);
 devolver codigo_idioma
 en otro caso
 devolver 'desconocido'

(4.1): Detección del idioma

datos_filtrados['idioma'] ← Aplicar DETECTARIDIOMA a la
columna 'ingredients_text';

Función OBTENERNOMBREIDIOMA(*codigo*):

 nombre_idioma ← langcodes.getLanguageName(codigo);
 devolver nombre_idioma **devolver** 'Desconocido'

(4.2): Filtrado por idioma

datos_filtrados['nombre_idioma'] ← Aplicar
OBTENERNOMBREIDIOMA a la columna 'idioma';
datos_filtrados_ingles ← Filtrar datos_filtrados donde
'nombre_idioma' = 'Inglés';

(5): Filtrado por categorías de productos

categorias ← ['Cheeses', 'Bread', 'Yogurts', 'Milks', 'Corn
chips', 'Honeys', 'Milk Chocolate', 'Peanut butters', 'Meats',
'Biscuits', 'Sweetened beverages', 'Plant-based beverages'];
datos_finales ← Filtrar datos_filtrados_ingles donde
'main_category_en' esté en categorias;

Guardado de datos finales

Guardar datos_finales en ruta_salida_filtrada;
devolver datos_finales;

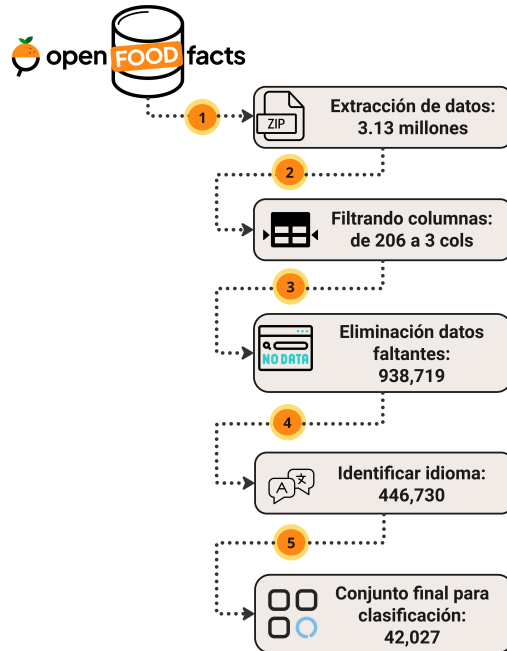


Figure 6.3: **Flujo de la Etapa I.** El procesamiento de datos comenzó con (1) la extracción de ≈ 3.13 millones de registros desde la base de datos de Open Food Facts. (2) Posteriormente, se realizó un filtrado de columnas, reduciendo el número de columnas de 206 a 3 en el DataFrame. (3) A continuación, se eliminaron los registros que contenían datos faltantes, lo que resultó en un total de 938,719 registros. (4) Se procedió a identificar el idioma, seleccionando únicamente los registros en inglés, lo que redujo el conjunto de datos a 446,730 registros. (5) Finalmente, se obtuvo un conjunto de datos preparado para clasificación con 42,027 registros. Este proceso fue diseñado para garantizar datos utilizables para los siguientes pasos del análisis. Elaboración propia.

6.3 Motivación para el uso de GLiNER

Debido a que las listas de ingredientes contenían tanto alimentos como aditivos mezclados, se requirió un método para identificarlos y categorizar estas entidades correctamente, ya que tienen propiedades y roles diferentes dentro de la formulación de alimentos.

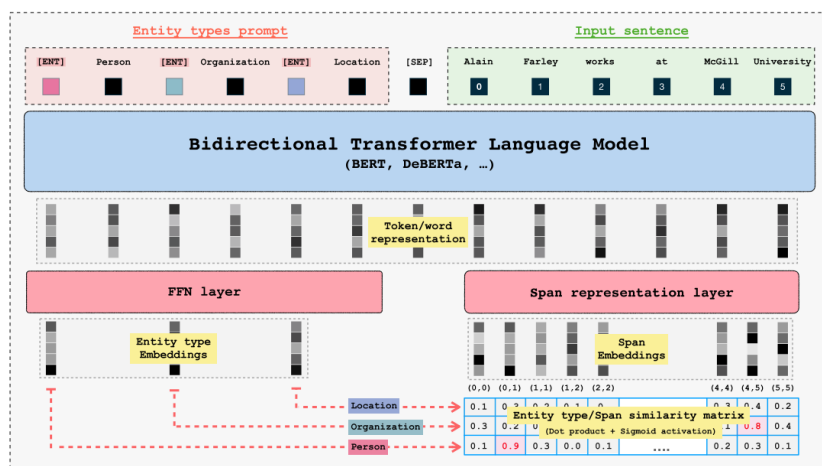
Basado en la técnica de Transferencia de Conocimiento (*Transfer Learning*), que consiste en reutilizar conocimientos adquiridos durante el entrenamiento de un modelo en una tarea inicial para resolver un problema relacionado, se recurrió a un enfoque de NER para abordar este problema. En particular, se utilizó el modelo preentrenado llamado *GLiNER*, que fue ajustado mediante un proceso conocido como *fine-tuning*. Este proceso consistió en especializar *GLiNER* para la tarea específica de *reconocer aditivos* dentro de las listas de ingredientes. Para ello, se utilizó un conjunto de datos creado específicamente para este propósito.

Con el fin de comprender mejor cómo *GLiNER* fue adaptado para esta tarea específica, es importante primero explorar las características principales de este modelo, lo cual se detalla a continuación.

6.3.1 El Modelo GLiNER

GLiNER (*Generalist Model for Named Entity Recognition*)⁶ es un modelo NER diseñado para identificar cualquier tipo de entidad. A diferencia de los modelos NER tradicionales, que están limitados a un conjunto predefinido de entidades, GLiNER utiliza modelos de transformers bidireccionales, como BERT o DeBERTa, lo que le permite reconocer entidades basándose en instrucciones en lenguaje natural, sin restricciones a categorías preestablecidas.

El enfoque de GLiNER para el Reconocimiento de Entidades abiertas (*Open NER*)⁷ consiste en comparar representaciones de tipos de entidad con representaciones de fragmentos textuales (spans) en un espacio latente común, en lugar de tratar esta tarea como una generación de texto. Esta estrategia resuelve los problemas de escalabilidad de los modelos NER autorregresivos, ya que el procesamiento de contexto bidireccional genera representaciones más ricas y precisas. GLiNER se organiza en tres componentes fundamentales que trabajan en conjunto para lograr la identificación de entidades, como puede observarse en la Figura 6.4.



Primero, el modelo emplea un codificador bidireccional preentrenado, como BERT o DeBERTa, que se encarga de generar representaciones para cada token del texto de entrada. En la Figura 6.4, este proceso se visualiza en el componente central azul, etiquetado como *Bidirectional Transformer Language Model*. En este paso, cada palabra de la oración ("Alain Farley works at McGill University") se transforma en una representación vectorial que captura el contexto bidireccional, es decir, teniendo en cuenta tanto las palabras anteriores, como las posteriores.

⁶ U. Zaratiana, N. Tomeh, P. Holat, and T. Charnois, "Gliner: Generalist model for named entity recognition using bidirectional transformer," 2023.

⁷ Open NER (Open Named Entity Recognition) se refiere a sistemas NER que están diseñados para ser utilizados en un entorno abierto, sin estar limitados a un dominio específico o conjunto de entidades predefinido. En un contexto tradicional, los sistemas de NER identifican entidades como personas, organizaciones, fechas y ubicaciones en un texto, pero suelen estar entrenados con un conjunto limitado de categorías.

Figure 6.4: Arquitectura del modelo GLiNER. Figura retomada de U. Zaratiana, N. Tomeh, P. Holat, and T. Charnois, "Gliner: Generalist model for named entity recognition using bidirectional transformer," 2023.

Luego, las representaciones de los tokens se pasan al módulo de representación de spans, que calcula las representaciones de fragmentos de texto que abarcan uno o más tokens. En la imagen, esta parte corresponde a la capa de *Span representation*, donde se combinan las representaciones de los tokens para generar embeddings de spans, como el caso del span “(0,1)”, que corresponde a las palabras “Alain Farley”. Este módulo es crucial para identificar entidades que pueden estar formadas por más de una palabra.

Finalmente, el módulo de representación de entidades calcula los embeddings de los tipos de entidad que se desean extraer, los cuales son posteriormente comparados con los embeddings de los spans en una matriz de similitud. En la Figura 6.4, este proceso se ilustra en la sección de *Entity type/Span similarity matrix*, donde se evalúa la compatibilidad entre los embeddings de las entidades, como “Person” y los embeddings de los spans generados previamente. El grado de coincidencia se determina utilizando un producto punto seguido de una activación sigmoide, lo que permite identificar si el span corresponde a una entidad particular. En el ejemplo de la Figura 6.4, el span “(0,1)”, que representa “Alain Farley”, muestra una alta coincidencia con la entidad “Person”, lo que indica que el modelo ha identificado correctamente esta entidad en el texto.

Actualmente, este modelo se encuentra disponible a través de Hugging Face. *GLiNER* proporciona diversas versiones de modelos preentrenados que pueden utilizarse según los requisitos de tamaño de parámetros e idioma. Estos modelos se pueden descargar desde Hugging Face⁸ o el repositorio de GitHub⁹ del proyecto, donde también se encuentra disponible la documentación.

⁸ https://huggingface.co/urchade/gliner_base

⁹ <https://github.com/urchade/GLiNER>

A continuación se muestra una lista de los modelos disponibles de *GLiNER*, junto con sus características:

Versión	Nombre del Modelo	Parámetros	Idioma	Licencia
v0	urchade/gliner_base	209M	Inglés	cc-by-nc-4.0
v0	urchade/gliner_multi	209M	Multilingüe	cc-by-nc-4.0
v1	urchade/gliner_small-v1	166M	Inglés	cc-by-nc-4.0
v1	urchade/gliner_medium-v1	209M	Inglés	cc-by-nc-4.0
v1	urchade/gliner_large-v1	209M	Inglés	cc-by-nc-4.0
v2	urchade/gliner_small-v2	166M	Inglés	apache-2.0
v2	urchade/gliner_medium-v2	209M	Inglés	apache-2.0
v2	urchade/gliner_large-v2	209M	Inglés	apache-2.0
v2.1	urchade/gliner_small-v2.1	166M	Inglés	apache-2.0
v2.1	urchade/gliner_medium-v2.1	209M	Inglés	apache-2.0
v2.1	urchade/gliner_multi-v2.1	209M	Multilingüe	apache-2.0

Table 6.2: Modelos disponibles de *GLiNER* y sus características.

6.4 Motivación para generación de conjunto de datos y uso de Web Scrapping

El entrenamiento del modelo *GLiNER* para la identificación de aditivos alimentarios en listas de ingredientes requiere de conjuntos de datos especializados y estructurados. La distinción adecuada entre ingredientes y aditivos condujo a generar un conjunto de datos especializado en aditivos alimentarios. Ante la falta de fuentes de datos preexistentes que proporcionaran la información sobre este tema, surgió la motivación de emplear técnicas de *web scraping* para obtener datos de aditivos alimentarios.

Wikipedia fue seleccionada como la fuente primaria para extraer datos. Esta técnica permitió recopilar descripciones específicas de los aditivos, abarcando tanto los sinónimos de los aditivos hasta características de clase funcional y sus aplicaciones en la industria alimentaria, lo que resultó en un conjunto de datos alineado con los requerimientos del modelo. Crear un conjunto de datos propio a partir de fuentes confiables asegura que el modelo se entrene con información de interés.

6.4.1 Extracción de datos mediante Web Scrapping

Se empleó el *Codex Alimentarius*¹⁰, un conjunto de normas, directrices y códigos de prácticas alimentarias cuyo propósito es proteger la salud del consumidor y asegurar la aplicación de prácticas justas en el comercio de alimentos. En particular, en lo que respecta a los aditivos alimentarios, se hace referencia a la *Norma General del Codex para los Aditivos Alimentarios (GSFA)*, la cual establece las condiciones en las que pueden utilizarse aditivos, independientemente de si han sido previamente regulados por el Codex o no¹¹. Además, a través del sitio web oficial del Codex, es posible acceder al listado completo de aditivos alimentarios autorizados¹².

Este *Codex Alimentarius* se tomó como referencia para identificar los nombres de los aditivos alimentarios aprobados internacionalmente. Cabe mencionar que el proceso de *web scraping* no se realizó directamente sobre la página web del Codex Alimentarius, sino sobre los datos de *Wikipedia*, donde se llevó a cabo la extracción de datos de los artículos correspondientes a cada aditivo.

El proceso fue semi-automatizado. Inicialmente, se identificaron manualmente los nombres de los aditivos en el Codex y se buscaron sus artículos en Wikipedia. Con las URL de estos artículos, se generó un archivo JSON que contenía las direcciones de los aditivos alimentarios,

¹⁰ FAO/WHO Food Standards, "Codex Alimentarius: Food Additive Index," 2023. Updated up to the 46th Session of the Codex Alimentarius Commission (2023)

¹¹ Food and Agriculture Organization of the United Nations (FAO) and World Health Organization (WHO), "Codex General Standard for Food Additives (GSFA) Online Database." <https://www.fao.org/fao-who-codexalimentarius/codex-texts/dbs/gsfa/en/>. [En línea; consultado el 07 de agosto de 2024]

¹² <https://bit.ly/4fdtnJD>

lo que facilitó la automatización del *web scraping* mediante Python. Este proceso se llevó a cabo respetando los términos de uso y las licencias *Creative Commons* de Wikipedia, no se hizo uso de información privada, se respetó la propiedad intelectual y no se causaron daños al servidor, garantizando que la extracción de datos cumpliera con las normativas legales¹³.

¹³ Se puede consultar el Anexo 1 donde se describen los términos de uso de la página web de Wikipedia.

6.4.2 Tarea de resumir el texto *summarization*

Dada la cantidad de información obtenida a través del *web scraping* para cada aditivo, fue necesario realizar un resumen de cada uno de éstos. Para ello, se utilizó la API de OpenAI que, mediante un esquema de *prompting*¹⁴, generó resúmenes concisos y relevantes de cada aditivo, lo que facilitó la posterior estructuración de los datos. Esta fase fue clave para asegurar que la información estuviera enfocada en el ámbito alimentario.

Posteriormente, los textos resumidos pasaron por un proceso adicional de extracción de entidades, también utilizando *prompting* con la API de OpenAI. Este enfoque permitió identificar los aditivos y las clases funcionales dentro de las descripciones textuales, facilitando su categorización. Los datos resultantes fueron tokenizados y preparados para su integración en el modelo *GLiNER* y utilizados en el ajuste (*fine-tuning*).

El siguiente código muestra el *prompt* utilizado para la tarea de resumen de texto (*summarization*):

Código para generar resumen (*summarization*)

```
1 {"openai_model": "gpt-4o-mini",
2  "messages": [
3    {"role": "system", "content": "You are an assistant that
4      summarizes texts."},
5    {"role": "user", "content": "Please, summarize the following
6      text. Focus on synonyms, E number (European Union identifiers),
7      food additive functional classes, and food industry
8      applications: {text}"},
9    {"role": "user", "content": "Make sure to highlight food
10     industry applications and do not add any other applications."},
11    {"role": "user", "content": "Build a paragraph no longer than
12     250 words."},
13    {"role": "user", "content": "Do not include any historical data
14     or chemical compounds and pharmaceutical applications in the
15     paragraph."}
16  ]
17 }
```

¹⁴ Proceso de proporcionar una instrucción, pregunta o declaración a un modelo de lenguaje con la finalidad de generar una respuesta o realizar una tarea específica. El "prompt" es el input que se da al modelo para guiar su salida.

Los parámetros utilizados para este modelo de OpenAI son los siguientes, junto con los valores asignados:

- **openai_model:** Este parámetro define el modelo de OpenAI que se utilizará. En este caso, el valor es "gpt-4o-mini".
- **batch_size:** Especifica el número de tareas o solicitudes procesadas simultáneamente. Un valor de 10 indica que se procesarán 10 textos en cada lote.
- **temperature:** Controla la aleatoriedad de las respuestas generadas. Un valor bajo, como 0.1, produce respuestas más coherentes y deterministas.
- **top_p:** Ajusta el muestreo por probabilidad acumulada (*nucleus sampling*). Un valor de 0.9 permite cierta variabilidad, priorizando las respuestas más probables.
- **frequency_penalty:** Penaliza la repetición de términos ya utilizados en la respuesta. Un valor de 0.8 reduce la repetición de palabras, mejorando la diversidad del texto generado.
- **presence_penalty:** Similar al parámetro anterior, evita que el modelo repita palabras que ya han aparecido en el contexto. Un valor de 0.8 favorece la inclusión de nueva información en las respuestas.
- **messages:** Es una lista de mensajes de interacción entre el sistema y la persona usuaria. Incluye:
 - **system:** Indica al modelo que actúe como un asistente especializado en generar resúmenes de texto.
 - **user:** Define las instrucciones específicas de la persona usuaria, como la necesidad de enfocarse en sinónimos, identificadores E (números de la Unión Europea), clases funcionales de aditivos y aplicaciones en la industria alimentaria. También se detallaron restricciones adicionales, como limitar el resumen a 250 palabras y evitar información no relevante, como datos históricos o aplicaciones farmacéuticas.

Posteriormente, para estructurar los datos se utilizó el siguiente *prompt*, que genera una estructura en formato JSON para la extracción de entidades:

Prompt para la extracción de entidades

```
1 {"base_prompt": "Given a passage, your task is to extract entities.
  The output should be a valid JSON object in the following
  format: {"text": "", "additive": [], "functional_class": []}.
  Ensure that all extracted entities are presented exactly as
  they appear in the text, including plurals and any other
  formatting."}
```

El código anterior muestra el formato necesario para que el modelo devuelva las entidades (aditivos y clases funcionales), respetando su aparición en el texto original. Esta identificación fue fundamental para permitir que, en pasos posteriores, se pueda mapear con precisión las entidades dentro del texto y determinar su posición exacta. Ahora, el siguiente ejemplo se presenta el resultado generado a partir del proceso de resumen y de la extracción de entidades anteriormente descritos:

Resultado de un resumen con extracción de entidades

```
1 {"summary_1": "Acesulfame potassium, also known as Ace K or E950 in
  the European Union, is a synthetic calorie-free sugar
  substitute widely used in the food industry. It is 200 times
  sweeter than sucrose and often blended with other sweeteners
  like sucralose or aspartame to achieve a more sucrose-like
  taste and mask any bitter aftertaste. Its stability under heat
  and varying pH conditions makes it suitable for baking and
  products requiring long shelf lives. Acesulfame K's smaller
  particle size compared to sucrose allows for uniform mixtures
  with other sweeteners, enhancing its application in various
  food products. Commonly found in carbonated drinks alongside
  other sweeteners, it is also used extensively in protein shakes
  .", "additive": ["acesulfame potassium", "Ace K", "E950", "
  sucrose", "sucralose", "aspartame"],
2   "functional class": ["sweetener", "sweeteners", "sugar
  substitute"]
3 "summary_2: ..."}

```

En este ejemplo, el resultado muestra cómo el modelo generó un resumen conciso sobre el aditivo “Acesulfame potassium”. Se incluyen detalles sobre su uso en la industria alimentaria, características y propiedades que lo hacen adecuado para diversos productos alimenticios, como bebidas carbonatadas y productos horneados.

6.4.3 *Proceso de tokenización y Reconocimiento de Entidades Nombradas (NER)*

Con la identificación de las entidades previa, el siguiente paso fue descomponer el texto en unidades básicas o “tokens”, lo que permitió analizar cada palabra, símbolo o número de manera individual. Este

proceso de generar tokens o segmentación del texto es importante para identificar de manera precisa las posiciones específicas de las entidades dentro del texto, lo que facilita el procesamiento posterior por parte del modelo. De esta forma, las entidades, como los aditivos alimentarios o las clases funcionales, pueden ser mapeadas en su posición exacta dentro del texto original. En el ejemplo mostrado, cada palabra o símbolo ha sido tokenizado y las entidades relevantes han sido identificadas con su respectiva posición en el texto, como se observa a continuación.

Tokenización

```
1 {"Acesulfame", "potassium", ",", "also", "known", "as", "Ace", "K",
  "or", "E950", "in", "the", "European", "Union", ",", "is", "a",
  "synthetic", "calorie", "-", "free", "sugar", "substitute", "
  widely", "used", "in", "the", "food", "industry", ".", "It", "
  is", "200", "times", "sweeter", "than", "sucrose", "and", "
  often", "blended", "with", "other", "sweeteners", "like", "
  sucralose", "...", "limits", "."}
```

Posteriormente, se utilizó la biblioteca SpaCy, retomada de una función del repositorio “gliner-finetune”¹⁵ para aplicar NER mediante patrones personalizados. Esto permitió identificar y etiquetar entidades clave, como los aditivos alimentarios y sus clases funcionales, dentro del texto tokenizado.

¹⁵ W. J. Mattingly, “gliner-finetune.” <https://github.com/wjbmattingly/gliner-finetune>, 2024. [En línea; consultado el 05 de junio de 2024]

Tokenización e identificación de posición

```
1 [{"Acesulfame", "potassium", ",", "also", "known", "as", "Ace", "K",
  "or", "E950", "in", "the", "European", "Union", ",", "is", "a",
  "synthetic", "calorie", "-", "free", "sugar", "substitute", "
  widely", "used", "in", "the", "food", "industry", ".", "It", "
  is", "200", "times", "sweeter", "than", "sucrose", "and", "
  often", "blended", "with", "other", "sweeteners", "like", "
  sucralose", "...", "limits", "."}, {"ner": [[6, 7, "ADDITIVE"],
  [9, 9, "ADDITIVE"], [21, 22, "FUNCTIONAL CLASS"], [36, 36, "
  ADDITIVE"], [42, 42, "FUNCTIONAL CLASS"], [44, 44, "ADDITIVE"],
  [46, 46, "ADDITIVE"], [51, 51, "ADDITIVE"], [64, 64, "
  FUNCTIONAL CLASS"], [92, 93, "ADDITIVE"], [99, 99, "FUNCTIONAL
  CLASS"], [130, 131, "ADDITIVE"]]]}
```

Finalmente, se creó un conjunto de datos que incluye un total de 372 aditivos con la estructura previamente descrita. Este conjunto de datos se ha compartido en la plataforma Hugging Face¹⁶, facilitando el acceso y reutilización por parte de la comunidad investigadora para tareas relacionadas con el procesamiento de lenguaje natural y análisis de aditivos alimentarios.

¹⁶ <https://huggingface.co/datasets/patymun/additives>

6.5 Ajuste (*Fine-tuning*) de GLiNER

Una vez disponible el conjunto de datos, se procedió a realizar el ajuste (*fine-tuning*) del modelo GLiNER. Para ello, se utilizó la versión preentrenada del modelo disponible en la plataforma Hugging Face, específicamente el modelo llamado `urchade/gliner_small`. Este modelo se ajustó para la tarea de reconocimiento de entidades dentro del dominio de los aditivos alimentarios.

Entrenamiento y evaluación (*Train/Test*)

Durante el proceso de ajuste del modelo GLiNER, se realizó una división del conjunto de datos en subconjuntos de entrenamiento (*train*) y evaluación (*test*). El conjunto de datos completo constaba de 372 aditivos alimentarios. Para asegurar que el modelo fuera entrenado y evaluado de manera adecuada, los datos se dividieron en un 90% para entrenamiento y un 10% para evaluación.

Esta separación se realizó de forma aleatoria. El conjunto de entrenamiento fue utilizado para ajustar los pesos del modelo, mientras que el conjunto de evaluación permitió monitorear el rendimiento del modelo en datos no vistos durante el entrenamiento.

Configuración del entorno de trabajo

A continuación se estableció el entorno de trabajo utilizando PyTorch como biblioteca que gestiona las operaciones de cálculo y procesamiento durante el entrenamiento y evaluación del modelo. Se habilitó la paralelización de tokenizadores mediante la variable de entorno (`tokenizers_parallelism`). Posteriormente, se cargó el modelo preentrenado desde Hugging Face y se configuró un objeto de tipo (`DataCollator`) para manejar los datos. El `DataCollator` organizó las secuencias en lotes, aplicando el *padding* dinámico para que todas las secuencias en un lote tuvieran la misma longitud y preparó las etiquetas para las tareas de etiquetado secuencial como el reconocimiento de entidades.

Hiperparámetros del modelo

Se definió el número de pasos de entrenamiento, el tamaño de lotes, el número de épocas de entrenamiento, se especificó la tasa de aprendizaje, el decaimiento de los pesos y la estrategia de evaluación.

A continuación se presenta el pseudocódigo de la fase II, que corresponde a lo descrito en la sección anterior y al flujo representado en la Figura 6.5.

Algoritmo 2: Etapa II: Flujo de procesamiento de datos y entrenamiento con GLiNER (Parte 1)

Datos: ruta_archivo_datos, modelo_openai, configuración, datos aditivos

Resultado: Estructura de datos procesada y modelo ajustado entrenado

Procedimiento PROCESARDATOS(*ruta_archivo_datos*, *modelo_openai*, *configuración*):

(1): Cargar variables de entorno y configuración

Procedimiento CARGARCLAVEAPI(*ruta_archivo_env*):

api_key ← Cargar la clave API desde el archivo .env;

si *api_key* *válida* **entonces**

 Inicializar cliente OpenAI con la clave API ;

devolver *cliente OpenAI*

en otro caso

 Error "Clave API no encontrada"

cliente ← CARGARCLAVEAPI () ;

config ← Cargar configuración desde archivo JSON ;

(2): Procesar aditivos y descripciones

Cargar *wiki_additives.json* y extraer URLs de Wikipedia ;

para cada *url* ∈ *urls* **hacer**

name ← Extraer nombre del aditivo desde la URL;

description ← Scraping de la descripción ;

si *description* *válida* **entonces**

 Guardar descripción ;

en otro caso

 Añadir a la lista de URLs fallidas ;

Guardar descripciones en archivos JSON ;

(3): Generación de resúmenes de las descripciones

Procedimiento GENERARRESUMENES(*descripciones*, *modelo_openai*, *configuración*):

summaries ← Diccionario vacío para guardar los resúmenes ;

para cada *batch* ∈ *lotes_de_descripciones* **hacer**

para cada *description* ∈ *batch* **hacer**

formatted_messages ← Formatear mensajes con la descripción (prompting) ;

response ← Enviar petición a la API de OpenAI ;

summary ← Extraer resumen de la respuesta ;

 Guardar resumen en el diccionario ;

 Guardar los resúmenes en el archivo JSON ;

devolver *summaries*

summaries ← GENERARRESUMENES(*descripciones*, *modelo_openai*, *configuración*);

Algoritmo 3: Etapa II: Flujo de procesamiento de datos y entrenamiento con GLiNER (Continuación)

Datos: ruta_archivo_datos, modelo_openai, configuración, datos aditivos

Resultado: Estructura de datos procesada y modelo entrenado

Procedimiento PROCESARDATOS():

(4.1): Generación de estructura de datos con OpenAI

base_prompt ← Cargar prompt base desde archivo ;

para cada (key, text) ∈ archivo_datos **hacer**

prompt ← base_prompt con texto interpolado ;
 response ← Generar entidades usando la API de OpenAI ;
 Guardar entidades extraídas en estructura de datos ;

Guardar resultados procesados en structure_output.json ;

(4.2): Procesamiento para tokenizar e identificar entidades

Procedimiento PROCESARTOKENSENTIDADES(loader_results, progress, task):

processed_results ← lista vacía para almacenar los resultados procesados ;

para cada (i, result) ∈ enumerate(loader_results, 1) **hacer**

para cada (key, value) ∈ result.items() **hacer**

si value contiene 'text' **entonces**

processed_example ← procesar usando process_example(value) ;
 Añadir processed_example a processed_results ;

devolver processed_results

processed_results ← loader_results, progress, task;

Guardar resultados procesados en processed_structure.json ;

(5): Entrenamiento del modelo GLiNER

Procedimiento ENTRENARGLINER(dataset_entrenamiento, dataset_prueba):

Cargar modelo GLiNER desde Hugging Face ;
 Definir TrainingArguments con hiperparámetros ;
 Inicializar el entrenador con los argumentos ;
 Entrenar el modelo con el dataset de entrenamiento ;
 Guardar modelo entrenado en models/ ;

dataset ← Cargar y dividir datos en entrenamiento y prueba ;

ENTRENARGLINER(dataset_entrenamiento, dataset_prueba) ;

(6): Guardado de resultados finales

Guardar resultados en Hugging Face usando la API ;

devolver modelo entrenado, estructura de datos procesada

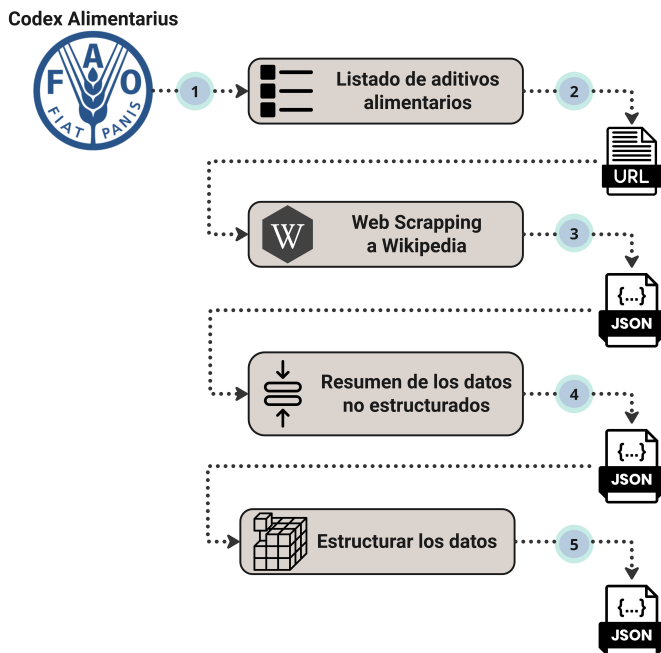


Figure 6.5: **Flujo de la Etapa 2.** El procesamiento de datos comienza con (1) la generación del listado de aditivos alimentarios basado en el *Codex Alimentarius*, el cual se traduce en un conjunto de URLs para cada aditivo; (2) posteriormente, se realizó un proceso de *Web Scrapping* sobre este listado, obteniendo como resultado un archivo en formato JSON; (3) con los datos obtenidos, se generó un resumen para cada aditivo, centrado en su relevancia dentro de la industria alimentaria; (4) finalmente, se estructuraron los datos mediante diversos microprocesos que incluyeron la identificación y extracción de entidades clave, la tokenización y la determinación de las posiciones de dichas entidades en el texto. El resultado final fue un conjunto de datos en formato JSON, el cual fue compartido con la comunidad de *HuggingFace* para su uso. Elaboración propia.

6.6 Entrenamiento de Word2Vec para la clasificación de alimentos

Una vez ajustado el modelo, se utilizó el conjunto de datos de la Etapa I, compuesto por un total de 42,027 registros. Se realizó una limpieza de los textos presentes en el campo `ingredients_text`, empleando la biblioteca *nlTK* para eliminar “stop words”, es decir, aquellas palabras que no aportan valor semántico en este análisis, como artículos, preposiciones y algunas conjunciones.

Un reto específico fue el tratamiento de los ingredientes anidados en las listas, ya que muchos ingredientes contenían descripciones adicionales entre paréntesis o corchetes, lo que indicaba los componentes de un ingrediente compuesto. A continuación, se muestra un ejemplo de cómo los textos originales fueron procesados para obtener listas más limpias y manejables:

Ejemplo de limpieza de ingredientes

- 1 Texto original:
- 2 Asiago cheese (cultured milk, enzymes, red peppers, salt), provolone cheese (cultured pasteurized milk, enzymes, salt), cellulose (to prevent caking), natamycin (a mold inhibitor).
- 3
- 4 Lista procesada:
- 5 ['Asiago cheese', 'provolone cheese', 'cellulose', 'natamycin']

Continuando con el proceso de limpieza, se transforma el texto a minúsculas y se elimina cualquier carácter no alfabético, excepto las comas, que son relevantes para estructurar los datos.

Se eliminan caracteres especiales adicionales que podrían interferir con el procesamiento del texto, como guiones, asteriscos o signos de puntuación que no son necesarios para la interpretación. Finalmente, se aplica un proceso de lematización, que reduce las palabras a su forma base o raíz, lo que permite normalizar los términos para mejorar la coherencia en el análisis posterior. Una vez completado este procesamiento, el texto se reconstruye en forma de cadena y está listo para su uso en etapas posteriores del modelo.

El modelo **Word2Vec** fue entrenado utilizando estas nuevas listas limpias de ingredientes, sin la interferencia de los aditivos, que habían sido previamente identificados y separados por el modelo *GLiNER*. Este enfoque permitió una clasificación de los alimentos, alineada con los objetivos del proyecto.

Entrenamiento de vectores

En la práctica, se entrenó un modelo utilizando la biblioteca de *Gensim*. Se entrenó el modelo utilizando los siguientes parámetros: un corpus compuesto por 42,027 listas de ingredientes, una dimensión vectorial de 100, una ventana de contexto de tamaño 5 y el método de skip-gram para aprender las representaciones semánticas de las palabras.

Algoritmo 4: Etapa III: Entrenamiento de modelos de clasificación (Parte 1)

Datos: Listas de ingredientes, etiquetas (de la Etapa I)

Resultado: Generación de word embeddings y vectores de ingredientes

Procedimiento PROCESARDATOS():

(1.1): Limpieza de las listas de ingredientes

Retomar el conjunto de datos preprocesados de la Etapa I;

Convertir a minúsculas, eliminar caracteres especiales y

tokenizar las listas de ingredientes;

lista_limpia $\leftarrow$ procesamiento de limpieza finalizado;

(1.2): Entrenamiento del modelo Word2Vec

Entrenar modelo Word2Vec con los siguientes parámetros:

- sentences $\leftarrow$ listas de ingredientes limpias (lista_limpia) tokenizadas
- vector_size = 100 (Dimensionalidad de los word embeddings)
- window = 5 (Ventana de contexto para predecir palabras)
- sg = 1 (Usar Skip-gram)

w2v $\leftarrow$ Entrenar modelo con los parámetros anteriores;

Obtener lista de palabras en el vocabulario del modelo

w2v.wv.key_to_index;

(2): Crear vectores de las listas de ingredientes

ingredient_vectors $\leftarrow$ Crear un vector para cada lista de ingredientes calculando el promedio de sus word embeddings;

para cada lista $\in$ listas de ingredientes **hacer**

vector $\leftarrow$ Calcular el promedio de los word embeddings de los ingredientes en la lista;

Agregar vector a ingredient_vectors;

(3): División de los datos en entrenamiento y prueba

Usar el método de división estratificada train_test_split para separar los datos en conjunto de entrenamiento y prueba;

(X_train, X_test, y_train, y_test) $\leftarrow$ ingredient_vectors, etiquetas, test_size=0.2, stratify=etiquetas;

Algoritmo 5: Etapa III: Entrenamiento de modelos de clasificación (Parte 2)

Datos: Vectores de ingredientes, etiquetas

Resultado: Modelos entrenados para clasificar

Procedimiento ENTRENAR_MODELOS():

(4): Entrenamiento y evaluación de los modelos de clasificación

(4.1): Modelo de Regresión Logística

Inicializar el modelo de Regresión Logística multiclase

lrmodel_multi;

GridSearchCV para ajustar los hiperparámetros;

Entrenar lrmodel_multi con X_train y y_train;

y_pred ← Predecir con lrmodel_multi en X_test;

Evaluar el modelo usando accuracy, precisión, recall, f1-score, y matriz de confusión;

(4.2): Modelo LDA (Análisis Discriminante Lineal)

Inicializar modelo LDA lda_model;

Entrenar lda_model con X_train y y_train;

lda_score ← Evaluar el rendimiento de lda_model con validación cruzada k-fold y calcular métricas de rendimiento en X_test;

(4.3): Modelo SVM (Máquinas de Soporte Vectorial)

Inicializar modelo SVM sv_model con kernel lineal;

Realizar búsqueda de hiperparámetros (GridSearchCV) para ajustar el modelo;

Entrenar sv_model con X_train y y_train;

sv_score ← Evaluar sv_model en X_test usando validación cruzada y calcular métricas de rendimiento;

(4.4): Modelo de Red Neuronal

Inicializar una red neuronal secuencial model con:

- Capa de entrada de dimensión igual a los embeddings (input_dim = 100)
- Capa densa con 64 neuronas y activación tanh
- Capa de salida con 10 neuronas y activación sigmoid para la clasificación multiclase

model ← Crear modelo con la estructura anterior;

Compilar el modelo con el optimizador adam y la función de pérdida categorical_crossentropy;

Métrica utilizada: accuracy;

(5): Comparación de modelos

Comparar el rendimiento de todos los modelos en términos de accuracy, f1-score, precision y recall;

Elegir el modelo con mejor rendimiento para la tarea de clasificación de ingredientes;

Fin del procedimiento

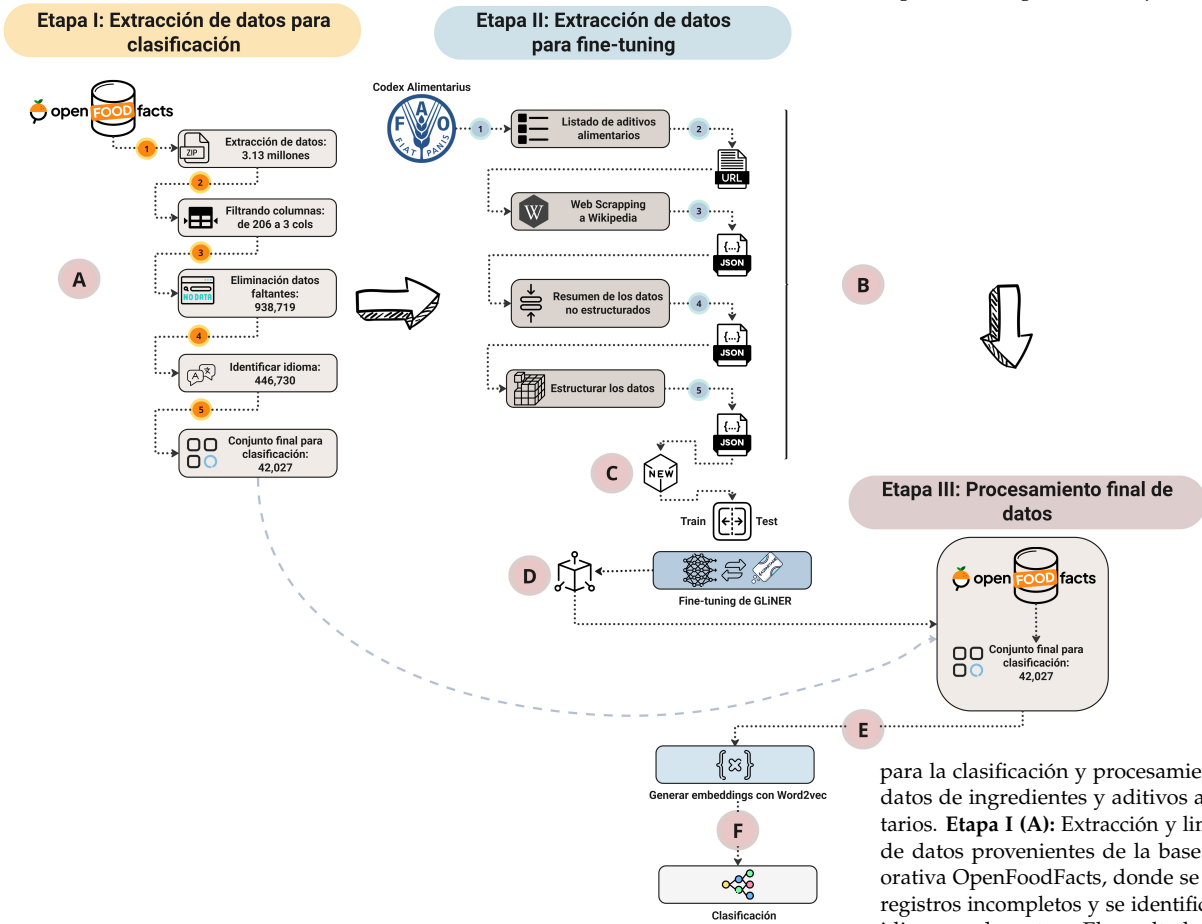


Figure 6.6: Diagrama del flujo de trabajo

para la clasificación y procesamiento de datos de ingredientes y aditivos alimentarios. **Etapa I (A):** Extracción y limpieza de datos provenientes de la base colaborativa OpenFoodFacts, donde se filtran registros incompletos y se identifican los idiomas relevantes. El resultado es un conjunto final de 42,027 registros para clasificación. **Etapa II (B):** Extracción de datos mediante técnicas de web scrapping en Wikipedia y con el apoyo del listado de aditivos del Codex Alimentarius. (C) El resultado de lo anterior es un nuevo conjunto de datos necesario para el *fine-tuning* del modelo GLiNER. Posteriormente este conjunto de datos se separa en *train* y *test*, se pasa por el modelo preentrenado y (D) genera un nuevo modelo. **Etapa III (E):** Aplicar el nuevo modelo a los datos de la etapa I, seguido de la generación de embeddings con Word2Vec para representar semánticamente los alimentos. Finalmente, (F) los vectores generados son utilizados para la clasificación de productos alimenticios.

6.7 Métricas de evaluación

La evaluación del desempeño en tareas de Reconocimiento de Entidades Nombradas (NER) es un paso fundamental para determinar la calidad de los modelos y su aplicabilidad en tareas posteriores. Existen dos enfoques principales para esta evaluación: **a nivel de token** y **a nivel de entidad**. Aunque ambos métodos buscan medir la capacidad del modelo para identificar correctamente entidades en un texto, se diferencian en su granularidad y en cómo abordan los errores.

Enfoque a nivel de token

El enfoque de evaluación a nivel de token mide el rendimiento considerando cada palabra o token en el texto de manera independiente. Este método se basa en el etiquetado mediante esquemas como IOB (*Inside, Outside, Beginning*) o BIOES (*Beginning, Inside, Outside, End, Single*), que permiten estructurar la delimitación de entidades en secuencias. Las métricas comunes, como **precisión**, **recall** y **F1-score**, se calculan comparando las etiquetas asignadas por el modelo con las etiquetas de referencia (*gold standard*) para cada token.

Aunque este enfoque es útil para identificar errores específicos en la asignación de etiquetas, presenta limitaciones en tareas prácticas. Por ejemplo, un modelo que identifica correctamente algunos tokens de una entidad pero no todos puede recibir una puntuación elevada, lo que no refleja la falta de precisión a nivel de la entidad completa. Por tanto, este enfoque no capta completamente la utilidad del modelo en aplicaciones que dependen de la identificación exacta de entidades.

Enfoque a nivel de entidad

El enfoque a nivel de entidad evalúa el rendimiento del modelo considerando las entidades nombradas como unidades completas. Esto incluye verificar no solo el tipo de entidad (en este caso, INGREDIENT o ADDITIVE), sino también su delimitación exacta en el texto. Una entidad predicha se considera correcta únicamente si coincide completamente con la delimitación y el tipo de la entidad en la referencia, es por esto que esta forma de evaluación se considera “más apegada a la realidad práctica”.

Para este enfoque, se emplean métricas similares a nivel de token, como **precisión**, **recall** y **F1-score**, pero aplicadas a entidades completas. Este método refleja mejor la capacidad del modelo para realizar tareas prácticas que dependen de la extracción precisa de entidades.

Además, a lo largo del tiempo se han desarrollado varios marcos

y esquemas de evaluación en competencias y desafíos, como CoNLL, ACE, MUC y SemEval, cada uno con métricas y criterios específicos diseñados para abordar distintos retos asociados con el Reconocimiento de Entidades Nombradas¹⁷. Por ejemplo, algunas métricas incluyen ajustes para manejar casos complejos como entidades solapadas o jerarquías de entidades. Estos esquemas han ayudado a establecer estándares más rigurosos y comprensivos para la evaluación de modelos de NER.

¹⁷

Comparación entre enfoques

Ambos enfoques tienen aplicaciones prácticas y contribuyen al desarrollo de modelos de NER, pero su utilidad depende del objetivo del sistema. Mientras que el nivel de token proporciona información detallada para depuración y mejora del modelo, el nivel de entidad refleja de manera más directa el impacto en aplicaciones del mundo real.

Ejemplos de evaluación de un sistema NER

A continuación se presentan distintos escenarios que se pueden presentar durante la evaluación de entidades:

I. Coincidencia exacta entre el sistema y el estándar dorado

Golden Standard		System Prediction	
Surface String	Entity Type	Surface String	Entity Type
in	O	in	O
New	B-LOC	New	B-LOC
York	I-LOC	York	I-LOC
.	O	.	O

Table 6.3: Ejemplo de coincidencia perfecta entre predicción y estándar dorado.

II. El sistema predice una entidad inexistente

Golden Standard		System Prediction	
Surface String	Entity Type	Surface String	Entity Type
an	O	an	O
Awful	O	Awful	B-ORG
Headache	O	Headache	I-ORG
in	O	in	O

Table 6.4: Ejemplo donde el sistema predice una entidad que no existe en el estándar dorado.

III. El sistema omite una entidad

Golden Standard		System Prediction	
Surface String	Entity Type	Surface String	Entity Type
in	O	in	O
Palo	B-LOC	Palo	O
Alto	I-LOC	Alto	O
,	O	,	O

Table 6.5: Ejemplo donde el sistema no predice una entidad que sí está en el estándar dorado.

Considerando solo estos tres escenarios, se puede realizar una evaluación básica clasificando los resultados en términos de **falsos positivos (FP)**, **verdaderos positivos (TP)**, **falsos negativos (FN)** y **verdaderos negativos (TN)**. Con estas clasificaciones, es posible calcular métricas como la **precisión**, el **recall** y la **puntuación F1** para cada tipo de entidad nombrada.

Sin embargo, este enfoque simplifica los escenarios y descarta casos de coincidencias parciales, o situaciones donde el sistema identifica correctamente la cadena superficial (*surface string*)¹⁸ de la entidad nombrada, pero asigna un tipo incorrecto. Estos escenarios pueden requerir una evaluación adicional, considerando el rendimiento a nivel de la entidad completa.

A continuación, según Batista¹⁹, en las Tablas 6.6, 6.7 y 6.8 se agregan otros escenarios que pueden ocurrir durante la evaluación de NER:

¹⁸ El término *surface string* (cadena superficial) se refiere al texto literal o a las palabras específicas de una entidad nombrada tal como aparecen en el texto original. Es la representación exacta de cómo se encuentra la entidad en el texto.

¹⁹ D. Batista, "Named-entity evaluation metrics based on entity-level," 2018. consultado el 08 de diciembre de 2024

IV. El sistema asigna el tipo de entidad incorrecto

Golden Standard		System Prediction	
Surface String	Entity Type	Surface String	Entity Type
I	O	I	O
live	O	live	O
in	O	in	O
Palo	B-LOC	Palo	B-ORG
Alto	I-LOC	Alto	I-ORG
,	O	,	O

Table 6.6: Ejemplo donde el sistema asigna el tipo de entidad incorrecto.

V. El sistema delimita incorrectamente el string de la entidad

Golden Standard		System Prediction	
Surface String	Entity Type	Surface String	Entity Type
Unless	O	Unless	B-PER
Karl	B-PER	Karl	I-PER
Smith	I-PER	Smith	I-PER
resigns	O	resigns	O

Table 6.7: Ejemplo donde el sistema delimita incorrectamente los límites del string de la entidad.

VI. El sistema se equivoca en la delimitación y el tipo de la entidad

Golden Standard		System Prediction	
Surface String	Entity Type	Surface String	Entity Type
Unless	O	Unless	B-ORG
Karl	B-PER	Karl	I-ORG
Smith	I-PER	Smith	I-ORG
resigns	O	resigns	O

Table 6.8: Ejemplo donde el sistema se equivoca tanto en la delimitación como en el tipo de entidad.

Estos escenarios ilustran diversos tipos de errores comunes en sistemas NER. Mientras que los ejemplos simples como los mostrados en las Tablas 6.3, 6.4 y 6.5 se pueden evaluar en términos de **falsos positivos**, **verdaderos positivos** y **falsos negativos**, escenarios más complejos como delimitaciones incorrectas o tipos de entidades mal asignados requieren análisis adicionales.

Es posible incorporar estas situaciones en métricas de evaluación extendidas que consideren tanto la coincidencia parcial, como los errores en la delimitación de las entidades. Esto puede realizarse mediante el diseño de nuevas métricas que penalicen de manera proporcional según el tipo de error cometido.

Métricas de evaluación basadas en diferentes categorías de errores

En MUC-5 (*Fifth Message Understanding Conference*)²⁰, se introdujeron métricas específicas para evaluar el desempeño de los modelos mediante la comparación de sus respuestas con las anotaciones del estándar de referencia (*gold standard*). Estas métricas categorizan los resultados en función de diferentes tipos de coincidencias y errores, definidos de la siguiente manera:

1. **Correcto (COR):** La salida del modelo coincide completamente con el estándar (tanto límites como tipo).

²⁰ N. Chinchor and B. Sundheim, "MUC-5 evaluation metrics," in *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27, 1993*, 1993

2. **Incorrecto (INC):** La salida del modelo no coincide con el estándar y no puede considerarse equivalente. Son identificadas, pero con errores en límites o el tipo.
3. **Parcial (PAR):** La salida del modelo y el estándar son similares pero no idénticos, por ejemplo, coinciden parcialmente en los límites o el tipo de entidad.
4. **Faltante (MIS):** Una anotación del estándar (*golden annotation*) no es identificada por el sistema.
5. **Espurio (SPU):** El sistema produce una respuesta que no está presente en las anotaciones del estándar.

Extensiones de las métricas de evaluación

Para complementar las métricas de evaluación basadas en diferentes categorías de errores, en SemEval-2013²¹, y siguiendo la evaluación definida previamente en MUC, se introdujeron cuatro enfoques específicos para medir la *precisión*, el *recall* y la *puntuación F1*:

- **Evaluación estricta:** Coincidencia exacta tanto en los límites como en el tipo de entidad.
- **Coincidencia exacta de límites:** Evaluación basada únicamente en los límites de la entidad, independientemente del tipo de entidad asignado.
- **Coincidencia parcial de límites:** Evaluación que considera solapamientos parciales entre los límites de las entidades, independientemente del tipo de entidad.
- **Coincidencia de tipo:** Se requiere algún grado de solapamiento entre la entidad etiquetada por el modelo y la entidad del estándar, pero se evalúa únicamente el tipo de entidad.

La puntuación final se calcula utilizando la **medida F1 micro-promediada**, que agrupa todas las entidades sin distinción. Esta métrica tiene la ventaja de tomar en cuenta todos los posibles errores cometidos por un sistema NER, proporcionando una visión integral del desempeño del modelo²².

Adicionalmente, se calculan las métricas de *precisión*, *recall* y *puntuación F1* para cada tipo de entidad, y luego se reportan como **métricas macro-promediadas**. Este enfoque permite evaluar el nivel de dificultad asociado con el reconocimiento de cada tipo de entidad, ya que no todas las entidades tienen la misma frecuencia. Al hacerlo, se puede identificar con mayor claridad qué tipos de entidades representan un mayor desafío para el sistema NER.

²¹ I. Segura-Bedmar, P. Martínez, and M. Herrero-Zazo, "SemEval-2013 task 9 : Extraction of drug-drug interactions from biomedical texts (DDIExtraction 2013)," in *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)* (S. Manandhar and D. Yuret, eds.), (Atlanta, Georgia, USA), pp. 341–350, Association for Computational Linguistics, June 2013

²² I. Segura-Bedmar, P. Martínez, and M. Herrero-Zazo, "SemEval-2013 task 9 : Extraction of drug-drug interactions from biomedical texts (DDIExtraction 2013)," in *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)* (S. Manandhar and D. Yuret, eds.), (Atlanta, Georgia, USA), pp. 341–350, Association for Computational Linguistics, June 2013

Evaluación de NER integral

Los cinco tipos de errores y los cuatro esquemas de evaluación interactúan de la siguiente manera, ejemplificado en el contexto de ingredientes y aditivos:

Escenario	Entidad (estándar)	Surface String (Standard)	Entity Type (Prediction)	Surface String (Prediction)	Type	Partial	Exact	Strict
I	ingredient	"sugar"	ingredient	"sugar"	COR	COR	COR	COR
II	-	-	ingredient	"zero calories"	SPU	SPU	SPU	SPU
III	additive	"acesulfame k"	-	-	MIS	MIS	MIS	MIS
IV	ingredient	"salt"	additive	"salt"	INC	COR	COR	INC
V	additive	"xanthan gum"	additive	"of xanthan gum"	COR	PAR	INC	INC
VI	additive	"xanthan gum"	ingredient	"of xanthan gum"	INC	PAR	INC	INC

Table 6.9: Interacción entre los tipos de errores y los esquemas de evaluación.

Cálculo de métricas

Para calcular las métricas estándar de **precisión**, **recall** y **F1-score**, se define lo siguiente:

Posible (POS): Representa todas las entidades que deberían haber sido identificadas según el estándar de referencia. Matemáticamente se calcula como:

$$\text{POSIBLE (POS)} = (\text{COR} + \text{INC} + \text{PAR}) + \text{MIS} = (TP) + FN$$

- TP (True Positives): Entidades identificadas correctamente.
- FN (False Negatives): Entidades que el sistema no detectó.

Actual (ACT): Representa todas las entidades que el sistema predijo, independientemente de si son correctas o no. Matemáticamente, se calcula como:

$$\text{ACTUAL (ACT)} = (\text{COR} + \text{INC} + \text{PAR}) + \text{SPU} = (TP) + FP$$

- FP (False positives): Entidades predichas incorrectamente (que no deberían estar).

Las métricas se calculan según el esquema de evaluación deseado:

Coincidencia Exacta (Strict, Exact)

Este esquema evalúa las entidades que coinciden perfectamente tanto en los *límites* como en el *tipo*.

$$\text{Precisión} = \frac{\text{COR}}{\text{ACT}} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{\text{COR}}{\text{POS}} = \frac{TP}{TP + FN}$$

Coincidencia Parcial (Partial, Type)

Este esquema permite considerar coincidencias parciales entre los límites o el tipo. Las coincidencias parciales (*PAR*) se ponderan como 0.5.

$$\text{Precisión} = \frac{\text{COR} + 0.5 \times \text{PAR}}{\text{ACT}}$$

$$\text{Recall} = \frac{\text{COR} + 0.5 \times \text{PAR}}{\text{POS}}$$

F1-score

El **F1-score** combina la precisión y el recall en una única métrica:

$$\text{F1-score} = 2 \cdot \frac{\text{Precisión} \cdot \text{Recall}}{\text{Precisión} + \text{Recall}}$$

Resumen de Resultados

La Tabla 6.10 muestra un resumen de resultados obtenidos en la evaluación de un sistema NER. Se puede observar que esquemas menos estrictos (Tipo y parcial) tiene valores altos de precisión, recall y F1-score porque no requiere coincidencias exactas. Esquemas más estrictos (Exacto y Estricto) reducen las métricas porque exigen mayor precisión en las predicciones.

Medida	Tipo	Parcial	Exacta	Estricto
Correcto	3	3	3	2
Incorrecto	2	2	2	3
Parcial	0	2	0	0
Perdido	1	1	1	1
Espurio	1	1	1	1
Precisión	0.5	0.66	0.5	0.33
Recall	0.5	0.66	0.5	0.33
F1-score	0.5	0.66	0.5	0.33

Table 6.10: Resultados de precisión, recall y F1-score para diferentes esquemas de evaluación.

6.7.1 Cálculo del tamaño de muestra para evaluación NER

Para una población de $N = 42,027$, un nivel de confianza del 95% ($Z = 1.96$), y un margen de error del 8% ($e = 0.08$), se calcula el tamaño de la muestra utilizando la fórmula para poblaciones finitas:

$$n = \frac{N \cdot Z^2 \cdot p \cdot (1 - p)}{e^2 \cdot (N - 1) + Z^2 \cdot p \cdot (1 - p)} \quad (6.1)$$

Sustituyendo los valores:

$$n = \frac{42027 \cdot (1.96)^2 \cdot 0.5 \cdot (1 - 0.5)}{(0.08)^2 \cdot (42027 - 1) + (1.96)^2 \cdot 0.5 \cdot (1 - 0.5)} \quad (6.2)$$

1. Calcular Z^2 :

$$Z^2 = 1.96^2 = 3.8416$$

2. Calcular $p \cdot (1 - p)$:

$$p \cdot (1 - p) = 0.5 \cdot (1 - 0.5) = 0.25$$

3. Calcular el numerador:

$$\text{Numerador} = 42027 \cdot 3.8416 \cdot 0.25 = 40321.54$$

4. Calcular el denominador:

$$\text{Denominador} = (0.08)^2 \cdot (42027 - 1) + 3.8416 \cdot 0.25$$

$$\text{Denominador} = 0.0064 \cdot 42026 + 0.9604 = 268.97$$

5. Finalmente, el tamaño de la muestra es:

$$n = \frac{40321.54}{268.97} \approx 150$$

Por lo tanto, el tamaño de la muestra es:

$$n \approx 150$$

6.8 *Análisis estadístico de la hipótesis*

6.8.1 *Descripción de los escenarios*

Para evaluar el desempeño de las técnicas propuestas y contrastarlas con lo planteado en la hipótesis, se definieron tres escenarios experimentales que reflejan diferencias en la aplicación de los procesamientos propuestos en este proyecto. Este apartado se justifica como un medio para establecer una base que permita analizar los resultados obtenidos y validar el impacto de los procesamientos propuestos en este proyecto. Específicamente, se estableció un escenario de línea base y los resultados de aplicar el modelo. Además, conforme a los resultados descritos específicamente en el apartado 7.3.1, por la pérdida de información derivada de los resultados de la clasificación con el modelo NER, se decidió incluir un tercer escenario que no considera la aplicación del modelo ajustado NER.

Escenario de línea base (Baseline)

Este escenario establece una línea base sin aplicar las técnicas de procesamiento de datos propuestas. Los modelos se entrenaron y evaluaron utilizando datos sin preprocesar, sin segmentación de idiomas, sin separación entre ingredientes y aditivos. Representa el punto de referencia inicial para las comparaciones con los escenarios propuestos.

Escenario parcial

En este escenario, surge como resultado de aplicar el modelo propuesto, excepto la clasificación llevada a cabo por el modelo ajustado NER. Incluye procesos como:

- Segmentación de idiomas.
- Limpieza y estructuración de texto.
- Generación de representaciones vectoriales (*word embeddings*).
- Balanceo de clases utilizando la técnica SMOTE (*Synthetic Minority Oversampling Technique*).

El objetivo de este escenario es observar el impacto parcial de estas metodologías en las métricas de desempeño de los modelos y compararlo tanto con el escenario de línea base como con el escenario del modelo completo.

Escenario completo

Este escenario implementa todas las técnicas propuestas. Las técnicas incluyen:

- El ajuste de modelo NER mediante datos estructurados generados por modelos generativos (GPT).
- Clasificación de ingredientes y aditivos.
- Segmentación de idiomas.
- Limpieza y estructuración de texto.
- Balanceo de clases utilizando la técnica SMOTE (*Synthetic Minority Oversampling Technique*).
- Generación de representaciones vectoriales (*word embeddings*).

6.8.2 Variables de análisis

Variable independiente

Los escenarios de procesamiento de datos (*línea base, parcial y completo*), que representan las técnicas de procesamiento de lenguaje natural (NLP) aplicadas.

Variable dependiente

La métricas de desempeño del modelo de clasificación, que son: *accuracy, precision, recall* y *f1-score*.

En este estudio, cada modelo fue ejecutado 15 veces utilizando diferentes configuraciones y particiones de los datos, lo que permite evaluar la robustez de los resultados y minimizar el impacto del azar en el desempeño reportado. Esto asegura que las medias y desviaciones estándar calculadas reflejen el comportamiento promedio de los modelos bajo cada escenario experimental.

El número de 15 repeticiones por modelo y escenario fue seleccionado para garantizar la estabilidad y robustez de los resultados estadísticos. Este número permite:

- Obtener estimaciones confiables de las métricas clave (*accuracy, precision, recall, f1-score*), minimizando el impacto del ruido estadístico.
- Cumplir con los requisitos mínimos para aplicar pruebas estadísticas como el ANOVA, asegurando una adecuada potencia estadística (0.8) y control del error tipo I (0.05).

De acuerdo con la fórmula de cálculo del tamaño de muestra para el ANOVA:

$$n = \frac{2 \cdot \sigma^2 \cdot (z_{1-\alpha/2} + z_{1-\beta})^2}{\Delta^2}, \quad (6.3)$$

donde:

- n : número de observaciones por grupo,
- σ^2 : (0.04) varianza estimada de los datos,
- $z_{1-\alpha/2}$: valor crítico de la distribución normal para un nivel de significancia α (1.96 para $\alpha = 0.05$),
- $z_{1-\beta}$: (0.84) valor crítico para el poder estadístico,
- Δ : (0.2) tamaño del efecto esperado entre grupos.

sustituyendo los valores:

$$n = \frac{2 \cdot 0.04 \cdot (1.96 + 0.84)^2}{0.2^2}, \quad (6.4)$$

realizando las operaciones:

$$n = \frac{2 \cdot 0.04 \cdot (2.8)^2}{0.04}, \quad (6.5)$$

$$n = \frac{2 \cdot 0.04 \cdot 7.84}{0.04}, \quad (6.6)$$

$$n = \frac{0.6272}{0.04} = 15.68. \quad (6.7)$$

Por lo tanto, se considera:

$$n = 15.$$

Esto asegura un tamaño de muestra adecuado con un nivel de confianza del 95% y un poder estadístico del 80%.

Estas métricas se presentan en el Capítulo 7 en Tabla 7.19, Tabla 7.21 y Tabla 7.23 como medias μ con sus desviaciones estándar $\pm\sigma$, calculadas como:

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i \quad (6.8)$$

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2} \quad (6.9)$$

donde n es el número de corridas realizadas y x_i representa el valor de la métrica en la i -ésima corrida.

6.8.3 Metodología estadística

Se aplicaron dos enfoques estadísticos principales para evaluar el impacto de los escenarios experimentales en cada una de las métricas clave de desempeño: **ANOVA de un solo factor** (*one-way*) y la prueba no paramétrica de **Kruskal-Wallis**. El factor considerado en ambos análisis es el **escenario experimental**, que incluye tres niveles: *Baseline*, *Parcial* y *Completo*.

Adicionalmente, se calcularon medidas de tamaño del efecto, como η^2 y ω^2 , para evaluar la magnitud de las diferencias entre los escenarios en los casos donde el ANOVA identificó diferencias significativas.

1. Comprobación de los supuestos previos:

- **Normalidad:** La distribución normal de los residuos fue evaluada mediante la prueba de Shapiro-Wilk y visualizaciones de QQ-plot. En aquellos casos donde los resultados indicaron un rechazo de la hipótesis de normalidad ($p < 0.05$), se utilizó la prueba no paramétrica de Kruskal-Wallis como alternativa al ANOVA.
- **Homogeneidad de varianzas:** Se empleó el test de Levene para verificar que las varianzas fueran similares entre los escenarios experimentales.

2. Análisis de impacto en las métricas clave:

- Para las métricas clave (*accuracy*, *precision*, *recall*, *f1-score*), se realizó un ANOVA de un solo factor cuando se cumplieron los supuestos de normalidad y homogeneidad de varianzas.
- En los casos donde no se cumplieron los supuestos de normalidad, se aplicó la prueba de Kruskal-Wallis para identificar diferencias significativas entre los escenarios experimentales, eliminando la dependencia de los supuestos paramétricos.

3. Análisis post-hoc:

- Para las diferencias significativas identificadas mediante el ANOVA, se utilizó la prueba post-hoc de Tukey HSD, que permite determinar qué pares de escenarios presentan diferencias significativas.
- En el caso de Kruskal-Wallis, se aplicó un análisis post-hoc basado en comparaciones por pares con corrección de Bonferroni para controlar el error de tipo I.

6.8.4 Definición de métricas de desempeño

Se evaluaron las métricas: *accuracy*, *precision*, *recall* y *f1-score*, además del tiempo de ejecución. Para garantizar una evaluación exhaustiva y equilibrada, las métricas *precision*, *recall* y *f1-score* fueron ponderadas en función de las clases, generando así un promedio ponderado. Las métricas se definieron de la siguiente manera:

Matriz de confusión: En clasificación multiclase, la matriz de confusión representa las frecuencias de predicciones correctas e incorrectas para cada clase. Para cada clase i , la matriz muestra las verdaderas predicciones en la diagonal (donde la clase real es igual a la predicha), mientras que las predicciones incorrectas aparecen en las demás celdas. Cada fila de la matriz representa las instancias reales de

la clase y cada columna muestra las predicciones para esa clase. Los conceptos clave en la matriz de confusión son:

TP (True Positives): Los casos que fueron correctamente clasificadas como pertenecientes a la clase i . **FP (False Positives):** Los casos que fueron incorrectamente clasificadas como pertenecientes a la clase i , pero en realidad pertenecían a otra clase. **FN (False Negatives):** Los casos que pertenecían a la clase i , pero fueron incorrectamente clasificadas como pertenecientes a otra clase. **TN (True Negatives):** Los casos que no pertenecían a la clase i y fueron correctamente clasificadas como no pertenecientes a dicha clase. Lo anterior puede observarse en la Figura 6.7.

MATRIZ DE CONFUSIÓN - $MC(i, j)$ de $N \times N$

CLASE I					
Real = 1	TP	FN	FN	FN	FN
Real = 2	FP	TN	TN	TN	TN
Real = ...	FP	TN	...	TN	TN
Real = N-1	FP	TN	TN	TN	TN
Real = N	FP	TN	TN	TN	TN
	Predicted = 1	Predicted = 2	Predicted = ...	Predicted = N-1	Predicted = N

CLASE N-I					
Real = 1	TN	TN	TN	FP	TN
Real = 2	TN	TN	TN	FP	TN
Real = ...	TN	TN	...	FP	TN
Real = N-1	FN	FN	FN	TP	FN
Real = N	TN	TN	TN	FP	TN
	Predicted = 1	Predicted = 2	Predicted = ...	Predicted = N-1	Predicted = N

Figure 6.7: Ejemplo de una matriz de confusión multiclase NN . Cada celda de la matriz representa la clasificación de las instancias según las clases reales (en filas) y las clases predichas (en columnas). Las celdas en azul corresponden a los verdaderos positivos (TP), donde la clase predicha coincide con la clase real. Las celdas verdes representan los falsos positivos (FP), donde una clase fue incorrectamente predicha como la clase actual. Las celdas amarillas indican los falsos negativos (FN), donde las instancias de la clase actual fueron clasificadas incorrectamente en otras clases. Las celdas restantes corresponden a los verdaderos negativos (TN), que son todas las instancias correctamente clasificadas como no pertenecientes a la clase actual. Esta estructura se repite para cada clase en problemas de clasificación multiclase. Figura retomada de W. B. Bustamante, "Calculando la precisión en un modelo de clasificación multiclase," 2021. Consultado: octubre, 2024.

Accuracy: En el contexto multiclase, la precisión global (accuracy) se calcula como la proporción de predicciones correctas sobre el total de predicciones realizadas. Se define como:

$$\text{Accuracy} = \frac{\sum_{i=1}^C \text{TP}_i}{\sum_{i=1}^C (\text{TP}_i + \text{FP}_i + \text{FN}_i)} \quad (6.10)$$

donde C es el número de clases y TP_i representa los verdaderos positivos para la clase i .

Precision: La precisión para una clase específica i se define como la proporción de verdaderos positivos para esa clase sobre todas las instancias predichas para dicha clase. Se calcula como:

$$\text{Precision}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i} \quad (6.11)$$

Para obtener una precisión global en problemas multiclase, se puede calcular un promedio ponderado en función del número de instancias por clase.

Recall: El *recall* para una clase específica i se define como la proporción de verdaderos positivos para esa clase sobre el total de instancias reales de esa clase. Se calcula como:

$$\text{Recall}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i} \quad (6.12)$$

Al igual que la precisión, puede calcularse un promedio ponderado en problemas multiclase.

F1-Score: El *f1-score* es la media armónica entre la precisión y el *recall*, proporcionando un equilibrio entre ambas métricas. Para una clase específica i , se define como:

$$\text{F1-Score}_i = 2 \cdot \frac{\text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \quad (6.13)$$

En problemas multiclase, también puede calcularse un promedio ponderado de los valores de *f1-score* por clase.

7 Resultados y discusión

Contents

7.1	Introducción	131
7.2	Identificación de idiomas y calidad de los datos	133
7.3	Desempeño del Modelo GLiNER ajustado . . .	133
7.3.1	Evaluación de las métricas de rendimiento del modelo ajustado.	134
7.4	Resultados de Word2Vec en la clasificación de alimentos	134
7.4.1	Generación de representaciones vectoriales	140
7.4.2	Descripción del uso de las listas de ingredientes depuradas	143
7.5	Evaluación de los algoritmos de clasificación .	145
7.6	Resultados de desempeño para la clasificación en los diferentes escenarios	153
7.6.1	Resultados del escenario de línea base	153
7.6.2	Resultados del escenario parcial . . .	153
7.6.3	Resultados del escenario completo . .	154
7.6.4	Comparación de métricas entre escenarios	155
7.6.5	Validación de supuestos	156
7.6.6	Resultados de ANOVA de una vía, tamaño del efecto y Post Hoc	158
7.6.7	Resultados de Kruskal-Wallis y Post Hoc	159
7.7	Discusión	160

7.1 Introducción

EN ESTE CAPÍTULO se presentan los resultados relacionados con la identificación de idiomas y la calidad de los datos de entrada. Los análisis iniciales revelaron que una proporción de los registros fue identificada incorrectamente, lo que permitió detectar filtraciones

de texto en otros idiomas. Este hallazgo subraya la importancia de garantizar la limpieza y consistencia de los datos antes de su procesamiento.

En cuanto al desempeño del modelo GLiNER para la separación entre ingredientes y aditivos, la metodología de evaluación permitió una interpretación integral de los resultados. Además de considerar los tokens predichos correcta e incorrectamente, se analizaron los errores relacionados con los límites y el tipo de entidad. Los resultados muestran que la precisión del modelo, es decir, su capacidad para distinguir entre ingredientes y aditivos, es alta. Sin embargo, el recall es más bajo, lo que indica una omisión de entidades, tanto de ingredientes como de aditivos, en ambos casos.

Un hallazgo clave es que el modelo aún omite una cantidad considerable de textos, independientemente de si corresponden a ingredientes o aditivos. Esto resalta la necesidad de considerar un escenario alternativo que omita esta etapa del modelo para mejorar el desempeño.

Por otro lado, los resultados obtenidos con Word2Vec para el análisis separado de ingredientes y aditivos mostraron hallazgos relevantes en términos de similitudes y relaciones generadas. La configuración del entrenamiento, en particular el ajuste de frecuencias para aumentar o reducir el vocabulario, demostró ser un parámetro crítico que no debe ser ignorado en el momento de entrenar el corpus.

En cuanto a la combinación de vectores, no se observaron diferencias considerables entre las estrategias de suma y promedio. Por ello, se optó por continuar el análisis con vectores promediados, dada su simplicidad y consistencia. Además, las pruebas realizadas con submuestreo y SMOTE indicaron que SMOTE es la mejor opción, ya que ofrece mejores métricas de desempeño.

Con base en estos análisis, se definieron tres escenarios experimentales: un escenario "completo" con las características descritas en el capítulo metodológico (6), un escenario "parcial" y su respectiva "línea base". Estos escenarios permitieron diseñar un experimento estadístico basado en la comparación de varianzas mediante ANOVA. Los resultados obtenidos proporcionaron evidencia estadísticamente significativa para rechazar la hipótesis nula y aceptar la hipótesis planteada en este proyecto.

7.2 Identificación de idiomas y calidad de los datos

Como se mencionó en el capítulo anterior, en el subapartado 6.2.2, se identificó que los textos contenían una mezcla de varios idiomas. Por esta razón, incluso después de filtrar los textos para incluir únicamente aquellos en *inglés*, se realizó una evaluación utilizando una muestra representativa, calculada con un nivel de confianza del 95% y un margen de error del 5%. El detalle de este cálculo puede consultarse en el Anexo 8.3.1.

Los resultados obtenidos fueron 373 casos correctos y 8 casos incorrectos. La proporción de casos incorrectos es:

$$\text{Proporción de error} = \frac{8}{381} \approx 0.0215 \text{ o } 2.15\% \quad (7.1)$$

Este resultado indica que el 2.15% de los registros fueron identificados incorrectamente. Algunos de los idiomas presentes en la muestra fueron: alemán, danés, francés, italiano, español y sueco. Esto quiere decir que la identificación automática del idioma es precisa, aunque existe una pequeña tasa de error.

7.3 Desempeño del Modelo GLiNER ajustado

De acuerdo con la metodología de evaluación descrita en el capítulo anterior, en el apartado 6.7, se presentan a continuación los resultados obtenidos para la evaluación del modelo NER ajustado.

Para la evaluación se utilizó la biblioteca ‘nervaluate’¹, desarrollada para su uso en Python. A continuación, en la Tabla 7.1, se presentan los resultados generales obtenidos considerando todas las entidades evaluadas:

Evaluación	Correcto	Incorrecto	Parcial	Perdido	Espurio	Posible	Actual	Precisión	Recall	F1-score
Entidades (ent_type)	936	84	0	282	4	1302	1024	0.9141	0.7189	0.8048
Parcial (partial)	997	0	23	282	4	1302	1024	0.9849	0.7746	0.8672
Estricto (strict)	916	104	0	282	4	1302	1024	0.8945	0.7035	0.7876
Exacto (exact)	997	23	0	282	4	1302	1024	0.9736	0.7657	0.8573

En términos generales, el modelo tiene una precisión alta (91.41%) en la identificación de entidades, lo que significa que la mayoría de las entidades predichas son correctas. Sin embargo, el *recall* es más bajo (71.89%), lo que indica que el modelo no está capturando todas las entidades presentes en el estándar.

Los esquemas más relajados (como parcial) reportan métricas más altas (F-1score de 86.71%) porque consideran coincidencias parciales. En cambio, los esquemas más estrictos (exacto y extricto) penalizan más

¹ MantisAI, “Nervaluate: Full named-entity evaluation metrics based on semeval’13,” 2023

Table 7.1: Resultados generales para las diferentes configuraciones de evaluación del modelo NER ajustado.

los errores en los límites y el tipo de entidad, reduciendo el F1-score a 78.76%.

A continuación en la Tabla 7.2 y en la Tabla 7.3 se presentan los resultados de la evaluación para las entidades de ingredientes y aditivos, respectivamente.

Evaluación	Correcto	Incorrecto	Parcial	Perdido	Espurio	Posible	Actual	Precisión	Recall	F1-score
Entidades (ent_type)	648	40	0	187	3	875	691	0.9378	0.7406	0.8276
Parcial (partial)	671	0	17	187	3	875	691	0.9834	0.7766	0.8678
Estricto (strict)	632	56	0	187	3	875	691	0.9146	0.7223	0.8072
Exacto (exact)	671	17	0	187	3	875	691	0.9711	0.7669	0.8570

Table 7.2: Resultados para la etiqueta **INGREDIENTE**.

Los resultados de la Tabla 7.2 para identificar ingredientes muestran un buen desempeño en esta categoría, con la precisión elevada en todos los esquemas (hasta 97.11% en el esquema exacto). El *recall* para esta categoría es significativamente más alto (74.06% – 77.66%) en comparación con la etiqueta **aditivo**, lo que sugiere que el modelo es más efectivo al identificar **ingredientes**.

Aunque la *precisión* es alta, el *recall* moderado implica que el modelo omite algunas entidades de tipo **ingrediente** (187 entidades perdidas en todos los esquemas).

Evaluación	Correcto	Incorrecto	Parcial	Perdido	Espurio	Posible	Actual	Precisión	Recall	F1-score
Entidades (ent_type)	288	44	0	95	1	427	333	0.8649	0.6745	0.7579
Parcial (partial)	326	0	6	95	1	427	333	0.9880	0.7705	0.8658
Estricto (strict)	284	48	0	95	1	427	333	0.8529	0.6651	0.7474
Exacto (exact)	326	6	0	95	1	427	333	0.9790	0.7635	0.8579

Table 7.3: Resultados para la etiqueta **ADITIVO**.

Los valores de *precisión* son más bajos para esta etiqueta, especialmente en el esquema más estricto 85.29% comparado con **ingrediente** con 91.46%. El *recall* es significativamente menor (67.45% – 77.05%), indicando que el modelo enfrenta más desafíos para capturar correctamente los **aditivos**.

7.3.1 Evaluación de las métricas de rendimiento del modelo ajustado.

Durante la identificación de entidades, hubo una pérdida de información, debido a que al momento de procesar las listas de ingredientes, hubo ingredientes que el modelo no pudo clasificar, estos ingredientes no se contemplaron en las nuevas listas tanto de ingredientes, como de aditivos.

7.4 Resultados de Word2Vec en la clasificación de alimentos

El vocabulario utilizado constaba de $N = 42,027$ registros. El modelo Word2Vec se entrenó con este corpus, utilizando un tamaño de vector de

100 y una ventana de contexto de 5. Se ajustó un parámetro que permite ignorar palabras con una frecuencia menor a un valor determinado. Por defecto, este parámetro tiene un valor de 5; sin embargo, se realizaron pruebas con otros valores. Al establecer el valor en 2, se obtuvieron 4,185 palabras; con 3, se obtuvieron 2,983 palabras; con 4, se obtuvieron 2,402 palabras; y con la frecuencia de 5, se redujo a 2,047 palabras.

Durante las pruebas de este hiperparámetro, se observó que al disminuir la frecuencia, aumentaba el listado de palabras incorrectamente escritas. Mantener la frecuencia en 5 aseguró una menor cantidad de palabras y omitió aquellas que podrían ser errores de la base de datos provenientes de la técnica de OCR.

Al analizar las 2,047 palabras, se encontraron 58 errores (que representan el 2.83%) relacionados con palabras de idiomas distintos al inglés y posibles defectos en la escritura, como puede observarse en algunos ejemplos de la Tabla 7.4.

Palabras incorrectas			
""	az car	fraise	lait
aceite vegetal	azucar	glukossirap	laktase
acide citrique	graines de basilic	habaneros	le cocoa
antioxydant	e	huile v g tale	le cream
ar	beurre	eau	le kosher gelatin
ar me	echinacea	joghurt	le natural flavor
ar naturel	farine de bl	koffein	panax ginsengrotekstrakt
aroma	fd c	konservaringsmedel	queso
aromaer	fd c blue lake	konserveringmiddel	queso quesadilla
arome naturel	fettarmer	konserveringsmidler	...

Table 7.4: Ejemplos de palabras entrenadas en el modelo Word2Vec con errores.

Para eliminar este ruido en el conjunto de datos, se identificaron 58 casos de palabras incorrectas y se creó un diccionario en Python para traducir o corregir dichas palabras en el corpus original. Después de este procesamiento, se reentrenó el modelo Word2Vec, reduciendo el vocabulario a 2,000 palabras. Esta vez, todas las palabras estaban correctamente en inglés y el modelo fue calculado utilizando los mismos parámetros descritos previamente.

Resultados relacionados con la similitud

Estos vectores viven en un nuevo espacio vectorial y conforme a lo comentado en las secciones anteriores, con Word2Vec es posible identificar palabras que el modelo considera similares. En relación con la similitud de los ingredientes, se presentan los resultados de los dos ejemplos en la Tabla 7.5.

Ingrediente (apple)	Ingrediente (sugar)
orange (0.9032)	polydextrose (0.7913)
grape (0.8997)	sweetened condensed milk (0.7721)
lemon (0.8899)	fructose (0.7710)
beet (0.8795)	semisweet chocolate (0.7695)
pear (0.8765)	natural cocoa (0.7602)
pineapple (0.8709)	cocoa drop (0.7552)
mango (0.8579)	hydrogenated palm kernel oil (0.7538)
pomegranate (0.8545)	corn syrup (0.7527)
papaya (0.8484)	apple powder (0.7522)
carrot (0.8421)	butter fat (0.7514)

Table 7.5: Similitud de ingredientes con *apple* y *sugar* según el modelo Word2Vec.

La Tabla 7.5 ilustra los resultados obtenidos mediante el modelo Word2Vec, que permite identificar ingredientes semánticamente similares a *apple* y *sugar* según su contexto en el conjunto de datos analizados. Los ingredientes con mayor similitud a *apple* son principalmente frutas, tales como *orange*, *grape*, *lemon* y *pear*. Este comportamiento sugiere que el modelo ha logrado capturar las relaciones semánticas subyacentes entre ingredientes de la misma categoría, agrupando de manera efectiva frutas en un espacio vectorial donde estos términos están cercanos.

Por otro lado, los ingredientes más similares a *sugar* incluyen no solo edulcorantes y azúcares como *polydextrose*, *fructose* y *corn syrup*, sino también productos altamente procesados y con un alto contenido de azúcar, como *sweetened condensed milk* y *semisweet chocolate*. Además, aparecen otros ingredientes utilizados en la producción de alimentos procesados, como *hydrogenated palm kernel oil* y *cocoa drop*. Este patrón sugiere que el modelo también ha capturado relaciones semánticas entre ingredientes utilizados comúnmente en productos que contienen azúcar o en la fabricación de productos dulces.

Los valores de similitud proporcionados entre paréntesis indican el grado de proximidad entre los ingredientes en el espacio vectorial, con valores más cercanos a 1 que denotan una mayor similitud. En el caso de "apple", los ingredientes más similares, como *orange* (0.9032) y *grape* (0.8997), son aquellos que comparten características similares dentro de la categoría de frutas. En cuanto a *sugar*, ingredientes relacionados con edulcorantes, tales como *polydextrose* (0.7913) y *fructose* (0.7710), muestran altos grados de similitud, lo que indica que el modelo ha agrupado correctamente ingredientes con propiedades similares en términos de uso y composición.

Los resultados presentados en la Tabla 7.6 muestran los ingredientes que el modelo Word2Vec considera menos similares a *apple* y *sugar*. A diferencia de las tablas que reflejan similitud semántica, esta lista

Ingrediente (apple)	Ingrediente (sugar)
organic pork (0.1228)	organic pork (0.2160)
organic ground beef (-0.0030)	organic ground beef (0.0823)
lactose (-0.0359)	bison (0.0231)
chocolate (-0.0668)	organic chia seed (-0.0366)
corn flour (-0.0726)	organic ginger (-0.0506)
glucose syrup (-0.0749)	organic apple (-0.0521)
potato starch (-0.0757)	organic whole milk (-0.0564)
malt extract (-0.0856)	organic cream (-0.0633)
red lake (-0.0861)	organic brown rice (-0.0710)
skim milk (-0.0861)	organic lemon (-0.0742)

Table 7.6: No similitud de ingredientes con *apple* y *sugar* según el modelo Word2Vec.

de ingredientes representa aquellos que se encuentran en el extremo opuesto del espacio vectorial, es decir, ingredientes con los que el modelo no ha encontrado asociaciones significativas en las listas de ingredientes analizadas.

Para el caso de *apple*, los ingredientes menos similares incluyen productos como *organic pork* (0.1228), *organic ground beef* (-0.0030) y *lactose* (-0.0359), que están lejos en el espacio vectorial dado que representan productos de origen animal o lácteos, categorías de alimentos que no comparten un contexto de uso o composición con la manzana. En términos más generales, estos ingredientes pertenecen a grupos alimenticios distintos a las frutas, lo que explica la baja similitud captada por el modelo.

En el caso de *sugar*, los ingredientes con menor similitud incluyen *organic pork* (0.2160), *organic ground beef* (0.0823) y *bison* (0.0231). Nuevamente, estos son productos de origen animal, lo que sugiere que el modelo identifica correctamente la falta de relación directa entre productos cárnicos y el azúcar, que suele estar asociado a alimentos procesados, edulcorantes o ingredientes de origen vegetal. Esto confirma que el modelo es capaz de distinguir entre categorías alimentarias con poca o ninguna conexión contextual en el conjunto de datos.

El caso de los alimentos orgánicos

Un fenómeno observado en los resultados fue que los ingredientes con el prefijo *organic* tienden a agruparse independientemente de su categoría alimentaria, lo que revela un posible sesgo semántico inducido por la presencia de términos que no reflejan directamente las propiedades intrínsecas de los ingredientes. En este caso, la etiqueta *organic* parece actuar como un factor importante en la agrupación de los ingredientes dentro del espacio vectorial generado por el modelo.

Este comportamiento sugiere que el modelo está priorizando la

coincidencia de la etiqueta *organic* por encima de otras características más relevantes. Ingredientes como *organic chia seed*, *organic ginger*, *organic sunflower oil* y *organic cane sugar syrup* presentan una alta similitud entre ellos, pese a que pertenecen a categorías alimentarias distintas.

Sin embargo, al analizar los ingredientes agrupados por el modelo se identificaron algunas características compartidas. Por ejemplo en la Tabla 7.7 se muestran 5 casos de alimentos con la etiqueta *organic*. El primero, *organic chia seed* se asocia con ingredientes como *ginger*, *sunflower oil* y *cane sugar syrup*, que suelen utilizarse en la elaboración de alimentos saludables y naturales. Estos ingredientes no solo comparten la etiqueta *organic*, sino que también forman parte de productos que se enfocan en dietas integrales, naturales y funcionales, como barras energéticas o productos libres de gluten.

De forma similar, *organic cocoa butter* aparece relacionado con ingredientes como *dark chocolate chip*, *liquid egg*, y *cardamom*, que son comunes en la repostería saludable y gourmet. En este caso, el modelo agrupa ingredientes utilizados en la elaboración de postres o chocolates, destacando el uso compartido en productos con perfiles de sabor naturales.

Por otro lado, *organic sunflower seed* se asocia con ingredientes como *nonfat yogurt*, *spirulina*, y *wheat grass*, que son comunes en productos orientados hacia la nutrición deportiva o suplementos alimenticios. Este tipo de agrupaciones muestra cómo el modelo detecta patrones en ingredientes utilizados en la industria de alimentos funcionales y naturales, más allá de su etiqueta *organic*.

Finalmente, en el caso de *organic whole oat flour* y *organic apple juice*, los ingredientes similares, como *rolled oat*, *brown rice flour*, *pumpkin seed*, y *nonfat yogurt*, sugieren un enfoque en la producción de alimentos horneados saludables o bebidas funcionales. Estos ingredientes se encuentran comúnmente en productos enfocados en el bienestar y la salud.

Otra forma de analizar la similitud

A continuación se presentan los resultados obtenidos mediante la función `doesn't_match` de Word2Vec, aplicada a varias listas de ingredientes alimentarios. El análisis se resume con algunos ejemplos en la Tabla 7.8. El análisis permitió identificar las palabras que no encajan semánticamente en cada lista de ingredientes alimentarios proporcionada. Los resultados revelan interesantes patrones que

Ingrediente base	Ingredientes similares
<i>organic chia seed</i>	organic ginger, organic evaporated cane sugar, organic graham flour, organic sunflower oil, organic rolled oat, organic whole oat flour, organic cocoa powder, organic cane sugar syrup, organic vanilla, organic brown rice syrup
<i>organic cocoa butter</i>	organic dark chocolate chip, organic liquid egg, organic cardamom, organic light brown sugar, organic pumpkin, organic almond, organic flaxseed, organic almond flour, organic quinoa flake, organic sweet potato
<i>organic sunflower seed</i>	organic nonfat yogurt, organic spirulina, organic wheat grass, organic shredded coconut, organic pumpkin seed, organic almond flour, organic barley grass, organic quinoa, organic whole ground corn, organic almond meal
<i>organic whole oat flour</i>	organic rolled oat, organic brown rice flour, organic nutmeg, organic graham flour, organic almond meal, organic expeller pressed oil, organic semi sweet chocolate chip, organic pear, organic quinoa, organic peach
<i>organic apple juice</i>	organic spirulina, organic wheat grass, organic orange, organic barley grass, organic pumpkin seed, coconut milk powder, organic nonfat yogurt, organic broccoli, organic peach puree, organic soy sauce

Table 7.7: Ejemplificación de agrupación de ingredientes similares según el modelo Word2Vec para ingredientes con el prefijo "organic".

reflejan las relaciones contextuales captadas por el modelo, así como algunas limitaciones derivadas de la naturaleza de los datos de entrenamiento.

Lista de palabras	Palabra que no encaja
salt, sugar, natural flavor, water	water
milk, cheese, butter, beef	beef
honey, molasses, corn syrup, vinegar	vinegar
flour, sugar, butter, yeast	sugar
apple, orange, strawberry, chicken	strawberry
tomato, garlic, onion, chocolate	chocolate

Table 7.8: Resultados del análisis de las palabras que no encajan en cada lista de ingredientes.

En el primer caso, la lista compuesta por *salt*, *sugar*, *natural flavor*, y *water* mostró que la palabra *water* es la que no encaja con los otros ingredientes. Este resultado es coherente, ya que *water* es un líquido, mientras que los demás ingredientes son sólidos. Esta distinción entre estados físicos podría haber influido en la decisión del modelo.

En la segunda lista, formada por *milk*, *cheese*, *butter* y *beef*, la palabra *beef* fue seleccionada como la que no encaja. Este resultado es adecuado, ya que *milk*, *cheese* y *butter* son productos lácteos, mientras que *beef* es un producto cárnico, lo que sugiere que el modelo ha capturado

correctamente las diferencias en categoría alimentaria.

Para la tercera lista, compuesta por *honey*, *molasses*, *corn syrup* y *vinegar*, el modelo identificó *vinegar* como la palabra que no encaja. Este resultado es esperado, dado que *honey*, *molasses* y *corn syrup* son endulzantes, mientras que *vinegar* es un ácido que se utiliza en contextos culinarios completamente distintos.

En la cuarta lista, *flour*, *sugar*, *butter* y *yeast*, el modelo seleccionó *sugar* como la palabra que no encaja, lo cual es un resultado interesante. Aunque *sugar* es un ingrediente común en la panadería, no cumple la misma función estructural que *flour*, *butter* y *yeast*, lo que probablemente llevó al modelo a considerarla diferente.

En la quinta lista, compuesta por *apple*, *orange*, *strawberry* y *chicken*, el modelo eligió *strawberry* como la palabra que no encaja, a pesar de que *chicken* parece ser la opción más diferente, al ser un producto cárnico. Este resultado sugiere que *strawberry* podría no estar tan bien representada en el conjunto de datos del modelo en comparación con *apple* y *orange*, lo que explica esta decisión.

Finalmente, en la sexta lista, *tomato*, *garlic*, *onion* y *chocolate*, la palabra *chocolate* fue seleccionada como la que no encaja. Este resultado es lógico, ya que *tomato*, *garlic* y *onion* son ingredientes comúnmente utilizados en salsas y comidas saladas, mientras que *chocolate* es un ingrediente dulce.

Presencia de aditivos alimentarios

Es importante señalar que, incluso en esta etapa del análisis, se observa la presencia de ciertos aditivos alimentarios que no fueron identificados de manera adecuada por el modelo ajustado de GLiNER. Esta situación evidencia que, a pesar de los ajustes realizados en el modelo de reconocimiento de entidades nombradas, persisten limitaciones en la capacidad del sistema para detectar y clasificar todos los aditivos presentes en las listas de ingredientes. Esta observación subraya la necesidad de continuar refinando los métodos de identificación, con especial énfasis en mejorar la precisión del reconocimiento de aditivos.

7.4.1 Generación de representaciones vectoriales

Para visualizar los word embeddings, se utilizaron los algoritmos PCA (Principal Component Analysis) y t-SNE (t-distributed Stochastic Neighbor Embedding) a través de la herramienta interactiva en línea TensorFlow Embedding Projector. Esta herramienta permite explorar visualmente los embeddings en 2D y 3D, ofreciendo una forma

de analizar la distribución y agrupamiento de las representaciones vectoriales. La proyección tridimensional por PCA puede observarse en la Figura 7.1.

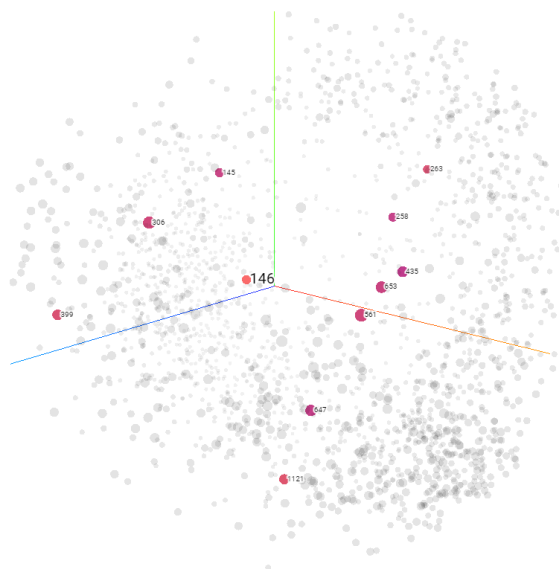


Figure 7.1: Visualización de la palabra "apple" (índice 146) en el *TensorFlow Embedding Projector*, utilizando una proyección tridimensional por PCA. En el gráfico, se ha resaltado "apple" y sus 10 palabras más cercanas en el espacio vectorial original, calculando la distancia mediante la similitud coseno. Esta visualización permite identificar las palabras que comparten mayor similitud semántica con "apple", lo que se refleja en la proximidad de los puntos en el gráfico y que coinciden con lo mostrado en la Tabla 7.5 para la misma palabra.

Para analizar el comportamiento de agrupamiento, se empleó t-SNE, una técnica de reducción de dimensionalidad que permite visualizar datos de alta dimensión en un espacio de menor dimensionalidad, preservando principalmente las relaciones locales entre puntos cercanos. Aunque no es un algoritmo de clusterización, t-SNE es efectivo para identificar patrones y estructuras de agrupamiento en los datos debido a su capacidad para representar relaciones no lineales.

En este estudio, se realizó un análisis utilizando t-SNE con los parámetros de perplejidad (10, 20, 30, 40)² y una tasa de aprendizaje fija de 10. El proceso, que fue iterativo, se llevó a cabo durante 100,000 iteraciones. El índice seleccionado fue el 146 correspondiente a la palabra *apple*, junto con sus 10 puntos más cercanos en el espacio original basados en la distancia de coseno. El objetivo fue evaluar si la cercanía entre estos puntos se mantenía consistente en la representación de baja dimensionalidad, permitiendo identificar de forma clara las relaciones semánticas entre los términos.

El análisis de los resultados de t-SNE con perplejidades de 10 a 40 muestra cómo las agrupaciones de los ingredientes se vuelven más coherentes y estables a medida que aumenta la perplejidad. Con una perplejidad de 10, los ingredientes están dispersos sin una clara coherencia en las agrupaciones. En este nivel de perplejidad, las

² El valor de perplejidad más apropiado depende de la densidad del conjunto de datos. En términos generales, cuanto más grande y denso es el conjunto de datos, mayor debe ser el valor de la perplejidad. Los valores típicos de perplejidad suelen oscilar entre 5 y 50.

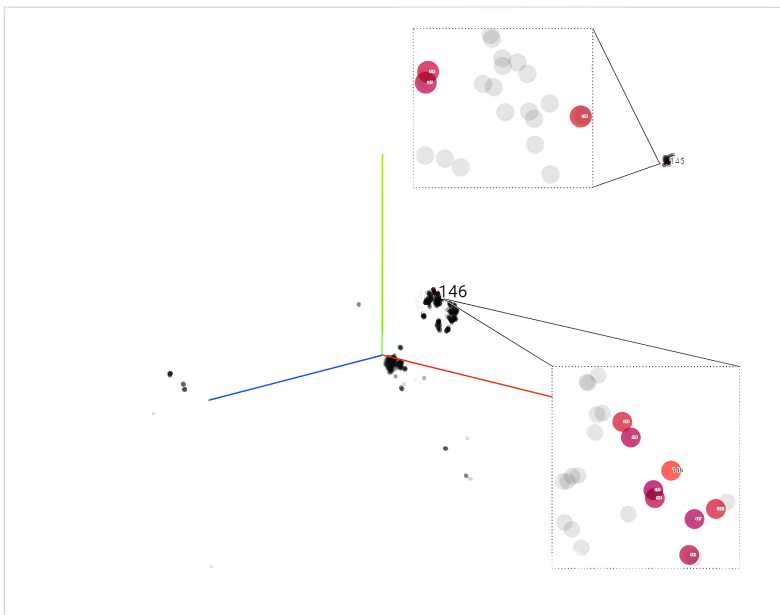


Figure 7.2: Proyección de vectores mediante t-SNE con perplejidad 10.

relaciones semánticas no son evidentes, como puede verse en la Figura 7.2 el cuadrante superior muestra tres ingredientes (*pineapple*, *mango*, *vainilla yogurt*), mientras que el cuadrante inferior muestra a los demás ingredientes más dispersos, aunque cercanos.

Con una perplejidad de 20, se observan agrupaciones más lógicas. Frutas como *apple*, *orange*, *lemon* y vegetales como *carrot* y *beet* comienzan a formar grupos, lo que sugiere que el modelo empieza a capturar mejor las relaciones semánticas entre estos ingredientes. A pesar de esto, *grape* y *vanilla yogurt* siguen mostrando un comportamiento aislado respecto del resto.

A perplejidad 30, las agrupaciones se consolidan aún más. Ingredientes como *grape*, *pear* y *papaya* se agrupan entre sí, mientras que *apple*, *orange*, *lemon*, *carrot* y *beet* continúan formando un grupo consistente, reflejando las similitudes en sus características compartidas. La persistente separación de *vanilla yogurt* y *pineapple* sugiere que estas categorías siguen siendo percibidas como distintas.

Finalmente, en la perplejidad 40 como se observa en la Fig 7.3, las agrupaciones son más claras y estables. *Vanilla yogurt* se agrupa con mango, mientras que *apple*, *orange*, *lemon*, *carrot* y *beet* mantienen su coherencia como un grupo. *Grape* se mantiene aislada, lo que indica que su perfil en los datos es lo suficientemente único como para mantenerse separada de las demás frutas. A medida que la perplejidad aumenta, las relaciones entre ingredientes se vuelven más precisas, lo

que sugiere que un valor de perplejidad entre 30 y 40 captura mejor las similitudes semánticas entre los ingredientes, reflejando su uso común y sus propiedades en la representación vectorial.

Este tipo de visualización ayudó a explorar visualmente las agrupaciones latentes en los embeddings y proporcionó una manera intuitiva de observar cómo los términos semánticamente similares se relacionan en el espacio vectorial.

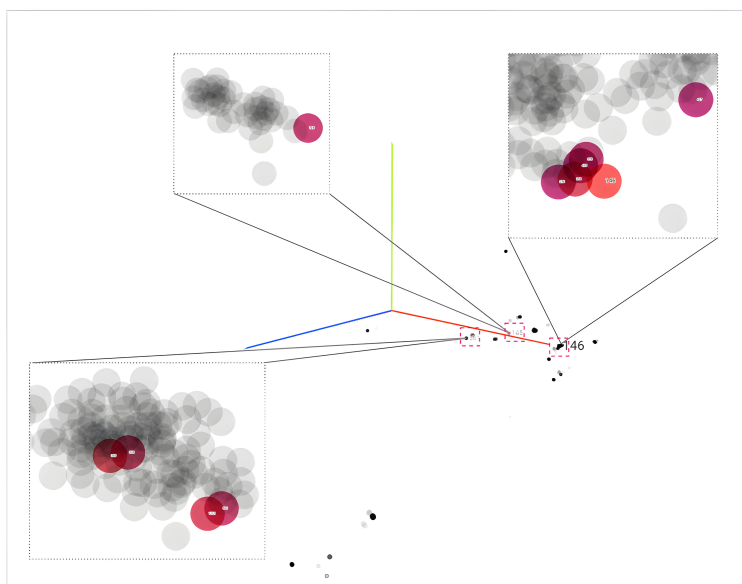


Figure 7.3: Proyección de vectores mediante t-SNE con perplejidad 40.

7.4.2 Descripción del uso de las listas de ingredientes depuradas

Tras la generación de los vectores correspondientes a cada ingrediente, es importante destacar que no todas las palabras fueron mapeadas al modelo, como se explicó previamente. Al entrenar el modelo con un parámetro de frecuencia mínima de 5, se experimentó una pérdida de información debido a la exclusión de palabras con baja frecuencia de aparición, quedando finalmente con 2,000 vectores correspondientes al vocabulario.

Dado que el análisis se realiza a nivel de listas completas de ingredientes y no de ingredientes individuales, se optó por dos opciones para generar un solo vector representativo para cada lista, con el fin de utilizarlo en un algoritmo de clasificación. La primera opción fue calcular el vector promedio de los ingredientes en cada lista. Este vector promedio, denotado como $\mathbf{v}_{avg}$, se obtiene a través del promedio de los vectores individuales de los ingredientes de la lista, de acuerdo con la

siguiente expresión:

$$\mathbf{v}_{\text{avg}} = \frac{1}{n} \sum_{i=1}^n \mathbf{v}_i \quad (7.2)$$

donde $\mathbf{v}_{\text{avg}}$ es el vector promedio, y n representa el número de ingredientes presentes en la lista. El valor de cada componente j del vector promedio se calcula como:

$$(\mathbf{v}_{\text{avg}})_j = \frac{1}{n} (v_{1j} + v_{2j} + \dots + v_{nj}) \quad (7.3)$$

Finalmente, el vector promedio $\mathbf{v}_{\text{avg}}$ se utiliza como entrada en una función de clasificación, denotada por f , que modela la relación entre la representación vectorial de los ingredientes y su categoría de clasificación:

$$y = f(\mathbf{v}_{\text{avg}}) \quad (7.4)$$

Asimismo, la segunda opción fue utilizar la suma de los vectores correspondientes a los ingredientes. A continuación, se suman los vectores correspondientes a cada ingrediente de la lista sin dividir por el número de ingredientes. La ecuación es la siguiente:

$$\mathbf{v}_{\text{sum}} = \sum_{i=1}^n \mathbf{v}_i \quad (7.5)$$

En este caso, $\mathbf{v}_{\text{sum}}$ representa la suma total de los vectores de los ingredientes y n es el número de ingredientes en la lista.

Tanto para la generación de vectores representativos mediante el promedio como mediante la suma, se identificaron 677 listas que no pudieron ser procesadas. En el caso del promedio, esto se debió a una división por cero, mientras que en el caso de la suma, se excluyeron aquellos índices cuyos vectores resultaron en cero. La exclusión de estos casos podría introducir ruido en la clasificación, ya que las etiquetas de dichas listas podrían pertenecer a cualquier categoría, lo que afectaría la precisión del modelo.

La Figura 7.4 ilustra tres escenarios posibles. En el primer escenario (Opción A), todos los ingredientes de un producto están representados dentro del vocabulario del modelo. En el segundo (Opción B), algunos ingredientes no están presentes en el vocabulario, lo que conduce a una representación parcial. Finalmente, en el tercer escenario (Opción C), ninguno de los ingredientes de la lista se encuentra en el vocabulario. Los casos correspondientes al escenario C fueron omitidos del conjunto de datos, resultando en 41,350 listas válidas para la clasificación.

El conjunto de datos resultante se dividió en una proporción de 80/20 para entrenamiento y prueba, con 33,080 listas en el conjunto

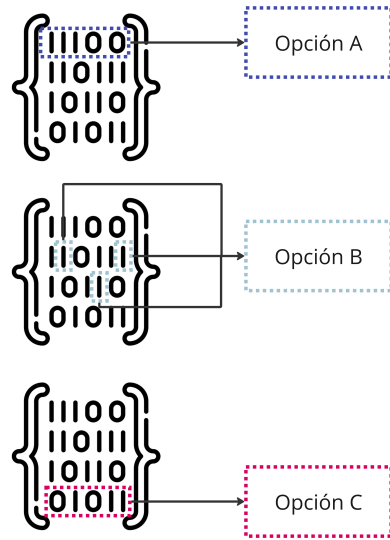


Figure 7.4: Descripción de escenarios posibles para la representación vectorial de listas de ingredientes.

de entrenamiento ($X_{\text{train}} = 33,080$, $y_{\text{train}} = 33,080$) y 8,270 listas en el conjunto de prueba ($X_{\text{test}} = 8,270$, $y_{\text{test}} = 8,270$). Se estratificaron las clases, debido a que se encuentran desbalanceadas como se observa en la Figura 7.5. A continuación, se presentan los resultados obtenidos en el entrenamiento y prueba de los modelos de clasificación.

En la Figura 7.5 se observa la distribución de las frecuencias de diferentes categorías de productos alimenticios. Los productos categorizados como *Cheeses* y *Biscuits* presentan las mayores frecuencias, con 11123 y 10483 registros, respectivamente.

En contraste, las categorías de *Peanut butters* y *Meats* muestran frecuencias considerablemente más bajas, con 1050 y 1202 menciones, lo que sugiere una menor representación en comparación con los productos lácteos y los productos de origen vegetal. Las categorías con frecuencias intermedias, como *Yogurts* (4667) y *Milks* (2513), reflejan una presencia significativa, pero no predominante en comparación con las principales categorías.

7.5 Evaluación de los algoritmos de clasificación

Se evaluaron tres algoritmos de clasificación: regresión logística, máquinas de soporte vectorial (SVM) y redes neuronales. A continuación, se detallan los resultados obtenidos y los ajustes realizados para cada modelo.

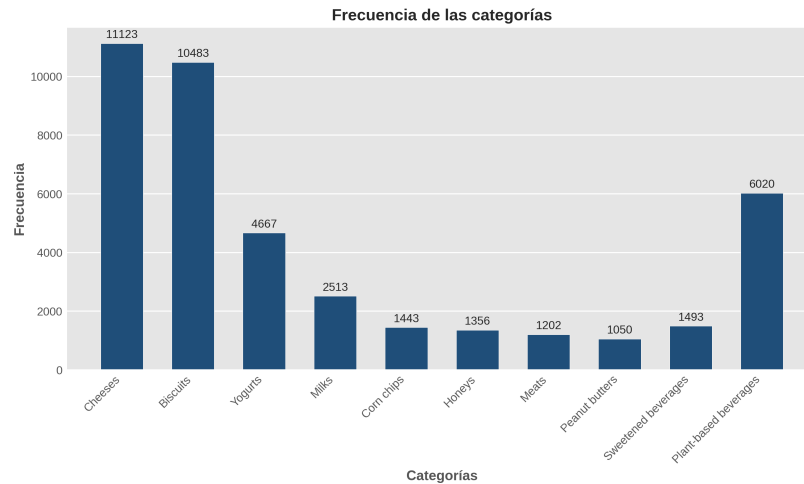


Figure 7.5: Distribución de frecuencias de productos alimenticios por categoría.

Regresión logística

Se utilizó un modelo de regresión logística multiclase *multinomial*. El solver `lbfgs` de la biblioteca de *scikit-learn* fue seleccionado por su eficiencia en problemas multiclase, así como por su compatibilidad con la regularización $L2$ por defecto. Además, este solver admite tanto entradas densas como dispersas, lo que lo hace adecuado para el conjunto de datos utilizado. Finalmente, el número máximo de iteraciones (`max_iter`) se fijó en 1,000 después de probar con mayor cantidad de iteraciones se obtuvo el mismo resultado.

Máquinas de soporte vectorial (SVM)

Para el entrenamiento del modelo de Máquinas de soporte vectorial (SVM), se aplicó una estandarización previa de los datos. Se implementó el modelo utilizando tanto el kernel lineal como el kernel de base radial (RBF), explorando el ajuste de hiperparámetros mediante validación cruzada.

En el caso del modelo con kernel lineal, se realizó una búsqueda de los mejores hiperparámetros utilizando `GridSearchCV` con validación cruzada de 5 pliegues. Se optimizó el parámetro de regularización C dentro del rango $[0.01, 0.1, 1, 5, 10]$, con el objetivo de encontrar el valor que maximizara la precisión del modelo. El uso de validación cruzada permitió evaluar el rendimiento del modelo de manera robusta, dividiendo el conjunto de datos en cinco subconjuntos, utilizando cuatro para entrenar y uno para validar, y repitiendo este proceso cinco veces. Como resultado, se identificó que el mejor valor para C fue 5, con una puntuación de validación cruzada de 0.8496.

Para abordar el desbalanceo de clases en el conjunto de datos, se empleó el parámetro “class_weight=balanced”, que ajusta automáticamente el peso de cada clase en función de su frecuencia inversa en los datos de entrenamiento. Esta estrategia garantiza que las clases minoritarias tengan mayor peso durante el entrenamiento, lo que ayuda al modelo a aprender mejor sus patrones y evita que se incline en exceso hacia las clases mayoritarias.

Posteriormente, el modelo óptimo fue evaluado en el conjunto de prueba, obteniendo una precisión del 85.31%. El uso de GridSearchCV fue clave para optimizar el ajuste del modelo, asegurando una evaluación exhaustiva del impacto de los diferentes valores de C en la capacidad predictiva del clasificador. Además, el parámetro “class_weight=balanced” se empleó para manejar el desbalance de clases en el conjunto de datos, mejorando la capacidad del modelo para generalizar sobre clases minoritarias.

Para el kernel RBF, se llevó a cabo un ajuste similar, aunque el hiperparámetro γ se calculó automáticamente.

Red neuronal

Para el entrenamiento de la red neuronal, al igual que con SVM, también se aplicó una estandarización previa a los datos. La arquitectura de la red consistió en dos capas densas, la primera con 64 neuronas (“tanh”) y la segunda con 10 neuronas (“sigmoid”) que se corresponden con la cantidad de categorías presentes en el conjunto de datos. Se utilizó el optimizador “adam” con una tasa de aprendizaje de 0.001, ajustada para mantener la estabilidad en el proceso de entrenamiento. Tras pruebas con diferentes configuraciones, el número de épocas óptimo se determinó en 150, dado que valores más altos no mostraron mejoras significativas en *accuracy*.

Tipo de capa	Dimensiones de salida	Param #
dense (Dense)	(None, 64)	6,464
dense_1 (Dense)	(None, 10)	650
Parámetros totales:		7,114
Parámetros entrenables:		7,114
Parámetros no entrenables:		0

Table 7.9: Arquitectura del Modelo de Red Neuronal

La siguiente tabla resume los resultados obtenidos, incluyendo el tiempo de ejecución para cada método. Estos resultados permiten comparar de manera directa los modelos en términos de precisión

y eficiencia computacional, proporcionando una evaluación de su rendimiento.

Método de clasificación	Accuracy	Precision	Recall	f1-score	Ejecution time
Regresión Logística [†]	0.85	0.86	0.85	0.85	≈ 2.1751 seg
Regresión Logística [‡]	0.85	0.86	0.85	0.85	≈ 9.1765 seg
SVM (Lineal) [†]	0.88	0.88	0.88	0.88	≈ 624.0573 seg
SVM (Lineal) [‡]	0.88	0.88	0.87	0.87	≈ 242.4733 seg
SVM (RBF) [†]	0.88	0.89	0.88	0.88	≈ 137.3460 seg
SVM (RBF) [‡]	0.87	0.88	0.87	0.87	≈ 119.8915 seg
Red neuronal [†]	0.88	0.92	0.83	N/A	≈ 7770.1959 seg
Red neuronal [‡]	0.88	0.91	0.83	N/A	≈ 10632.8641 seg

Table 7.10: Desempeño de los modelos de clasificación en términos de métricas clave y tiempo de ejecución. [†] Representa el promedio de vectores y [‡] la suma de vectores.

Regresión logística

Las clases como *Cheeses* y *Biscuits* muestran altos valores de precisión, recall y f1-score, lo que indica que el modelo es efectivo en la clasificación correcta de estas categorías. Esto se refleja también en la matriz de confusión e en la Figura 7.6, donde estas clases tienen altos valores en la diagonal, lo que indica que la mayoría de las predicciones fueron correctas.

Clases	Precision	Recall	f1-score
Cheeses	0.94	0.92	0.93
Biscuits	0.92	0.96	0.94
Plan-based beverage	0.80	0.76	0.78
Yogurts	0.85	0.81	0.83
Milks	0.83	0.73	0.77
Sweetened beverages	0.93	0.54	0.69
Corn chips	0.82	0.59	0.69
Honeys	0.97	0.77	0.86
Meats	0.69	0.45	0.55
Peanut butters	0.68	0.90	0.78

Table 7.11: Resultados obtenidos por clase para las métricas de desempeño de la regresión logística con multinomial con la técnica de **promedio de vectores**.

Las clases *Meats*, *Corn chips* y *Sweetened beverages* presentan un desempeño más bajo, especialmente en el f1-score (0.55, 0.69, 0.69, respectivamente). El recall bajo de 0.45 en *Meats* indica que muchos ejemplos de esta clase no fueron correctamente clasificados, lo cual también es evidente en la matriz de confusión con mayor cantidad de falsos negativos y algunos valores fuera de la diagonal.

La clase *Peanut butters* tiene un valor alto de recall (0.90), lo que indica que casi todos los ejemplos de esta clase fueron correctamente clasificados, aunque su precisión es más baja (0.68). Esto sugiere que, aunque se detectan la mayoría de los ejemplos de esta clase, el modelo también tiende a clasificar erróneamente algunos ejemplos de otras

clases como *Peanut butters*.

Matriz de confusión - Regresión logística

0	2045	28	44	24	13	1	9	0	2	59
1	7	2006	10	2	16	1	7	5	11	31
2	43	40	712	37	1	2	0	0	5	93
3	23	11	37	405	1	0	0	0	3	23
4	20	37	2	1	210	0	8	0	3	8
5	5	2	1	0	0	147	0	0	1	115
6	25	9	3	0	4	3	141	0	0	55
7	3	24	1	0	7	1	0	162	1	11
8	3	22	23	3	2	2	0	0	136	108
9	9	10	53	4	0	1	6	0	36	1085
	0	1	2	3	4	5	6	7	8	9

Predichos

Figure 7.6: Matriz de confusión resultado de la regresión logística con los vectores promediados.

Clases	Precision	Recall	f1-score
Cheeses	0.94	0.91	0.93
Biscuits	0.95	0.94	0.94
Plan-based beverages	0.83	0.75	0.79
Yogurts	0.83	0.81	0.82
Milks	0.81	0.78	0.79
Sweetened beverages	0.94	0.54	0.68
Corn chips	0.80	0.59	0.68
Honeys	0.95	0.81	0.87
Meats	0.64	0.49	0.55
Peanut butters	0.66	0.92	0.77

Table 7.12: Resultados obtenidos por clase para las métricas de desempeño de la regresión logística con multinomial y con las técnica de **suma de vectores**.

La Tabla 7.12 muestra los resultados obtenidos a partir de la técnica de suma de vectores para las lista de ingredientes. Como se puede observar, los valores no difieren significativamente de los reportados en la Tabla 7.11.

Máquinas de soporte vectorial

Conforme a los resultados, el kernel RBF generalmente ofrece un mejor rendimiento en comparación con el kernel lineal, especialmente en clases que presentan más desafíos, como *Meats* y *Plan-based beverages*, donde el kernel RBF obtiene un mayor recall y F1-Score.

La técnica de promedio de vectores parece ser más efectiva que la de suma en términos de equilibrio entre precisión y recall, como lo demuestra el rendimiento ligeramente superior en recall y F1-Score en varias clases.

Clases	Kernel Lineal (Prom de Vectores)			Kernel RBF (Prom de Vectores)		
	Precisión	Recall	F1-Score	Precisión	Recall	F1-Score
Cheeses	0.95	0.93	0.94	0.96	0.94	0.95
Biscuits	0.94	0.96	0.95	0.95	0.96	0.95
Plan-based beverages	0.85	0.80	0.83	0.87	0.84	0.85
Yogurts	0.87	0.84	0.86	0.90	0.86	0.88
Milks	0.88	0.80	0.83	0.87	0.82	0.84
Sweetened beverages	0.96	0.58	0.73	0.93	0.57	0.70
Corn chips	0.83	0.70	0.76	0.77	0.63	0.69
Honeys	0.96	0.84	0.90	0.99	0.85	0.92
Meats	0.72	0.49	0.58	0.72	0.53	0.61
Peanut butters	0.71	0.92	0.80	0.71	0.92	0.80

Table 7.13: Comparación de las métricas de desempeño obtenidas por clase para SVM con kernel lineal y kernel RBF utilizando la técnica de **promedio de vectores**.

A pesar de los buenos resultados generales, clases como *Sweetened beverages* y *Meats* siguen presentando desafíos en ambas configuraciones.

Clases	Kernel Lineal (Suma de Vectores)			Kernel RBF (Suma de Vectores)		
	Precisión	Recall	F1-Score	Precisión	Recall	F1-Score
Cheeses	0.95	0.93	0.94	0.96	0.93	0.94
Biscuits	0.94	0.96	0.95	0.95	0.96	0.95
Plan-based beverages	0.86	0.78	0.82	0.86	0.81	0.83
Yogurts	0.85	0.85	0.85	0.87	0.83	0.85
Milks	0.83	0.80	0.81	0.82	0.81	0.82
Sweetened beverages	0.94	0.55	0.69	0.98	0.54	0.69
Corn chips	0.85	0.60	0.71	0.77	0.62	0.69
Honeys	0.93	0.81	0.87	0.97	0.77	0.86
Meats	0.66	0.52	0.58	0.69	0.49	0.58
Peanut butters	0.69	0.91	0.79	0.68	0.92	0.78

Table 7.14: Comparación de las métricas de desempeño obtenidas por clase para SVM con kernel lineal y kernel RBF utilizando la técnica de **suma de vectores**.

Red neuronal

En la Tabla 7.15 se presentan los resultados obtenidos durante el entrenamiento y la validación del modelo de red neuronal utilizado con los vectores promediados. Se evaluaron métricas clave como precisión, *recall*, exactitud (*accuracy*) y la función de pérdida (*loss*) tanto en los datos de entrenamiento como en los datos de validación.

Métrica	Entrenamiento	Validación
Precisión	0.9361	0.9176
Recall	0.8725	0.8337
Accuracy	0.9029	0.8763
Loss	0.0533	0.0744
Tiempo total de entrenamiento:	7770.1959 seg	

Table 7.15: Resultados obtenidos del modelo de red neuronal tras 150 épocas. Las métricas incluyen precisión, *recall*, exactitud (*accuracy*) y pérdida (*loss*) tanto para los datos de entrenamiento como para los de validación para los **vectores promediados**. El tiempo total de entrenamiento fue de 7936.5440 segundos.

La Figura 7.7 presenta el comportamiento del modelo a lo largo de

las 150 épocas de entrenamiento. En la gráfica izquierda se observa una disminución continua de la función de pérdida, lo que refleja una mejora progresiva en el ajuste del modelo. Por su parte, la gráfica derecha muestra un aumento constante en la precisión, con una tendencia hacia la estabilización en las últimas épocas, lo que indica que el modelo ha alcanzado un rendimiento óptimo en este intervalo.

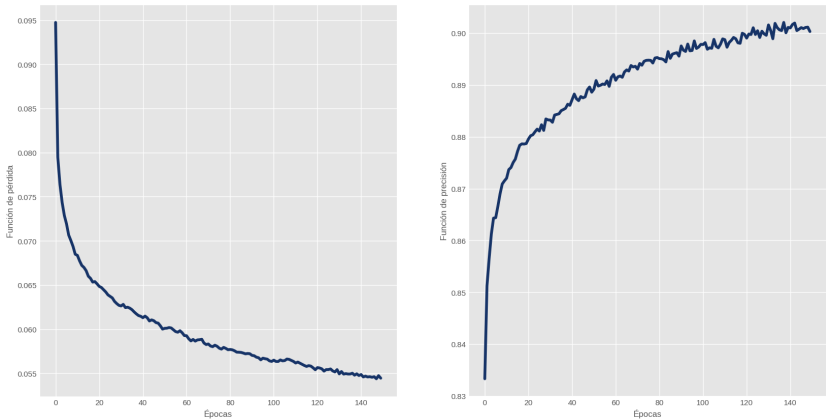


Figure 7.7: Comportamiento de la función de pérdida y precisión a lo largo de 150 épocas para el modelo de red neuronal. La gráfica de la izquierda ilustra la disminución en la función de pérdida, mientras que la gráfica de la derecha muestra el aumento en la precisión del modelo.

A continuación se presentan los resultados obtenidos del entrenamiento y la validación para la misma arquitectura de red neuronal pero con los vectores sumados. Prácticamente los resultados son bastante similares en términos de desempeño, como se observa en la Tabla 7.16.

Métrica	Entrenamiento	Validación
Precisión	0.9329	0.9144
Recall	0.8592	0.8328
Accuracy	0.8949	0.8776
Loss	0.0578	0.0738
Tiempo total de entrenamiento:		10632.8641 seg

Table 7.16: Resultados obtenidos del modelo de red neuronal tras 150 épocas. Las métricas incluyen precisión, *recall*, exactitud (*accuracy*) y pérdida (*loss*) tanto para los datos de entrenamiento como para los de validación para los **vectores sumados**. El tiempo total de entrenamiento fue de 7728.0316 segundos.

Mejora en el balance de clases en el conjunto de datos

Con el objetivo de mejorar el rendimiento del modelo, se implementaron técnicas de balanceo de clases para abordar la problemática de desbalance en el conjunto de datos. Para este fin, se utilizaron dos enfoques distintos: Submuestreo y SMOTE (*Synthetic Minority Over-sampling Technique*). En los análisis subsecuentes únicamente se consideró el promedio de los vectores para entrenar los

modelos, debido a que en el apartado anterior, los resultados no muestran diferencias entre dicha técnica y la suma de vectores.

El submuestreo es una técnica de reducción que busca equilibrar las clases disminuyendo la cantidad de ejemplos de las clases mayoritarias. Este enfoque permite reducir el sesgo hacia las clases con mayor representación, a la vez que disminuye el tamaño total del conjunto de datos, lo cual puede agilizar el proceso de entrenamiento. Sin embargo, el submuestreo puede conducir a la pérdida de información relevante, especialmente si los ejemplos eliminados contienen patrones esenciales para la generalización del modelo.

Por otro lado, SMOTE es una técnica de aumento de datos diseñada para generar ejemplos sintéticos en las clases minoritarias mediante la interpolación de instancias en el espacio de características. SMOTE tiene como objetivo reducir el desbalance sin duplicar muestras, lo cual ayuda a disminuir el riesgo de sobreajuste y permite que el modelo aprenda mejor las características distintivas de las clases menos representadas. Sin embargo, la generación de datos sintéticos puede introducir ruido en el conjunto de datos, especialmente si la interpolación no refleja adecuadamente las variaciones naturales de las clases minoritarias.

En la Tabla 7.17 se presentan los resultados de desempeño de varios modelos de clasificación tras aplicar el **submuestreo**, con un total de 1,076 registros por clase.

Método de clasificación	Accuracy	Precision	Recall	f1-score	Tiempo ejecución
Regresión Logística	0.78	0.79	0.78	0.78	≈ 0.4212 seg
SVM (Lineal)	0.82	0.83	0.82	0.82	≈ 28.2966 seg
SVM (RBF)	0.82	0.82	0.82	0.82	≈ 5.9135 seg
Red neuronal	0.81	0.86	0.74	N/A	≈ 7156.1524 seg

Table 7.17: Desempeño de los modelos de clasificación en términos de métricas clave y tiempo de ejecución con técnica del promedio de vectores y balanceo por submuestras de clases.

De manera similar, en la Tabla 7.18 se presentan los resultados de desempeño después de aplicar **SMOTE**, logrando una distribución de clases balanceada con 11,123 registros en cada clase.

Método de clasificación	Accuracy	Precision	Recall	f1-score	Tiempo ejecución
Regresión Logística	0.82	0.83	0.82	0.82	≈ 8.9630 seg
SVM (Lineal)	0.86	0.87	0.86	0.86	≈ 3394.1821 seg
SVM (RBF)	0.89	0.90	0.90	0.89	≈ 1227.4673 seg
Red neuronal	0.88	0.89	0.85	N/A	≈ 17044.7272 seg

Table 7.18: Desempeño de los modelos de clasificación en términos de métricas clave y tiempo de ejecución con técnica del promedio de vectores y balanceo por SMOTE.

7.6 Resultados de desempeño para la clasificación en los diferentes escenarios

7.6.1 Resultados del escenario de línea base

En la Tabla 7.19 se observan los valores para el escenario de línea base que se calcularon sin llevar a cabo ninguno de los procesos propuestos en el modelo de este proyecto. Los valores incluyen el promedio (μ), desviación estándar ($\pm\sigma$), error estándar (SE), e intervalo de confianza (IC) al 95%. La red neuronal fue ejecutada una sola vez, por lo que no es posible calcular SE ni IC .

Método de clasificación	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Regresión Logística	82.51 $\pm$ 0.37 SE : 0.0955 IC : (82.3228, 82.6972)	83.17 $\pm$ 0.36 SE : 0.0930 IC : (82.9877, 83.3523)	82.51 $\pm$ 0.37 SE : 0.0955 IC : (82.3228, 82.6972)	82.20 $\pm$ 0.39 SE : 0.0981 IC : (82.0077, 82.3923)
SVM (Lineal)	85.92 $\pm$ 1.12 SE : 0.0028 IC : (85.3624, 86.4966)	85.21 $\pm$ 0.96 SE : 0.0021 IC : (84.79315, 85.6422)	85.50 $\pm$ 0.83 SE : 0.0021 IC : (85.0873, 85.9157)	85.59 $\pm$ 0.81 SE : 0.0024 IC : (88.1102, 86.0854)
SVM (RBF)	82.32 $\pm$ 0.39 SE : 0.1007 IC : (82.1226, 82.5174)	83.34 $\pm$ 0.31 SE : 0.0800 IC : (83.1832, 83.4968)	82.32 $\pm$ 0.39 SE : 0.1007 IC : (82.1226, 82.5174)	81.94 $\pm$ 0.41 SE : 0.1059 IC : (81.7324, 82.1476)
Red neuronal	86.00 SE : N/A IC : N/A	91.00 SE : N/A IC : N/A	83.00 SE : N/A IC : N/A	N/A SE : N/A IC : N/A

Table 7.19: Desempeño de los modelos de clasificación del escenario de línea base.

En la Tabla 7.20 se observan los valores que representan el tiempo total de ejecución promedio en segundos, excepto para la Red Neuronal, que se corrió una sola vez.

Método de clasificación	Tiempo de ejecución (s)
Regresión Logística	$\approx$ 8.24
SVM (Lineal)	$\approx$ 1382.40
SVM (RBF)	$\approx$ 504.87
Red Neuronal	$\approx$ 43153.54

Table 7.20: Tiempos de ejecución promedio de los modelos de clasificación en el escenario de línea base.

7.6.2 Resultados del escenario parcial

Método de clasificación	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Regresión Logística	92.07 $\pm$ 0.29 SE : 0.0749 IC : (91.7930, 92.0870)	92.12 $\pm$ 0.29 SE : 0.0749 IC : (91.9730, 92.2670)	92.07 $\pm$ 0.29 SE : 0.0749 IC : (91.9330, 92.2070)	92.01 $\pm$ 0.29 SE : 0.0749 IC : (91.8590, 92.1610)
SVM (Lineal)	91.94 $\pm$ 0.29 SE : 0.0020 IC : (94.5907, 95.3950)	92.12 $\pm$ 1.16 SE : 0.0030 IC : (94.4346, 95.6181)	94.90 $\pm$ 0.66 SE : 0.0017 IC : (94.5639, 95.2414)	92.01 $\pm$ 0.84 SE : 0.0021 IC : (94.3782, 95.2359)
SVM (RBF)	91.74 $\pm$ 0.29 SE : 0.0749 IC : (91.5930, 91.8870)	91.87 $\pm$ 0.27 SE : 0.0697 IC : (91.7350, 92.0050)	91.74 $\pm$ 0.29 SE : 0.0749 IC : (91.5930, 91.8870)	91.66 $\pm$ 0.29 SE : 0.0749 IC : (91.5130, 91.8070)
Red neuronal	96.00 SE : N/A IC : N/A	96.00 SE : N/A IC : N/A	97.00 SE : N/A IC : N/A	N/A SE : N/A IC : N/A

Table 7.21: Desempeño de los modelos de clasificación. Estos valores se calcularon llevando a cabo los procesos propuestos en el modelo de este proyecto, excepto la separación a través del modelo ajustado NER. Los valores incluyen el promedio (μ), desviación estándar ($\pm\sigma$), error estándar (SE), e intervalo de confianza (IC) al 95%. La red neuronal fue ejecutada una sola vez, por lo que no es posible calcular SE ni IC .

Método de clasificación	Tiempo de ejecución (s)
Regresión Logística	≈ 3.68
SVM (Lineal)	≈ 1300.70
SVM (RBF)	≈ 120.73
Red Neuronal	≈ 35125.43

Table 7.22: Tiempos de ejecución promedio de los modelos de clasificación. Los valores están en segundos; el tiempo de la Red Neuronal corresponde a una única ejecución.

7.6.3 Resultados del escenario completo

En etapas previas, los modelos de clasificación se entrenaron únicamente con vectores representativos de ingredientes, sin incluir los aditivos. En esta sección, se exploran los resultados obtenidos al incorporar los vectores de aditivos, con el propósito de enriquecer la representación de los datos y mejorar el rendimiento del modelo.

Para este análisis, los vectores de ingredientes y aditivos fueron generados de forma independiente y luego combinados mediante el cálculo del promedio de ambos vectores, aprovechando que ambos modelos de *embeddings* comparten la misma dimensionalidad (100). Esta estrategia permite capturar información complementaria tanto de ingredientes como de aditivos, lo cual puede mejorar la discriminación entre clases.

El modelo Word2Vec de aditivos constó de 789, utilizando los mismos valores de parámetros de entrenamiento que el modelo Word2Vec desarrollado para ingredientes.

La Tabla 7.23 muestra los resultados de desempeño de los modelos de clasificación al utilizar vectores de ingredientes y aditivos combinados, como resultado de todo el modelo propuesto en este trabajo.

Método de clasificación	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Regresión Logística	86.86 ± 0.38 SE : 0.0022 IC : (87.6803, 88.5622)	89.04 ± 0.32 SE : 0.0022 IC : (89.0752, 89.9438)	86.86 ± 0.38 SE : 0.0025 IC : (87.2998, 88.2808)	87.35 ± 0.36 SE : 0.0022 IC : (87.8534, 88.7177)
SVM (Lineal)	88.12 ± 0.87 SE : 0.1084 IC : (89.2075, 89.6325)	89.50 ± 0.85 SE : 0.1162 IC : (91.4222, 91.8778)	87.79 ± 0.96 SE : 0.1084 IC : (89.0275, 89.4525)	88.28 ± 0.85 SE : 0.1084 IC : (88.9075, 89.3325)
SVM (RBF)	89.21 ± 0.30 SE : 0.0775 IC : (89.0581, 89.3619)	91.66 ± 0.29 SE : 0.0749 IC : (91.5132, 91.8068)	89.21 ± 0.34 SE : 0.0878 IC : (89.0379, 89.3821)	89.89 ± 0.32 SE : 0.0826 IC : (89.7281, 90.0519)
Red neuronal	89.00 SE : N/A IC : N/A	96.00 SE : N/A IC : N/A	85.00 SE : N/A IC : N/A	N/A SE : N/A IC : N/A

Table 7.23: Desempeño de los modelos de clasificación en términos de métricas clave con técnica del promedio de vectores y balanceo por SMOTE. Los valores incluyen el promedio (μ), desviación estándar ($\pm\sigma$), error estándar (SE), e intervalo de confianza (IC) al 95%. La red neuronal fue ejecutada una sola vez, por lo que no es posible calcular SE ni IC.

Método de clasificación	Tiempo de ejecución (s)
Regresión Logística	≈ 5.62
SVM (Lineal)	≈ 2089.21
SVM (RBF)	≈ 144.86
Red Neuronal	≈ 22144.20

Table 7.24: Tiempos de ejecución promedio de los modelos con técnica del promedio de vectores y balanceo por SMOTE. Los valores están en segundos; el tiempo de la Red Neuronal se refiere a una única ejecución.

Es importante destacar que el tiempo de ejecución de todos los algoritmos mencionados se midió utilizando una CPU en un entorno de Google Colab.

7.6.4 Comparación de métricas entre escenarios

La Figura 7.8 muestra las distribuciones de las métricas de evaluación (*Accuracy*, *Precision*, *Recall* y *F1score*) en los diferentes escenarios analizados: línea base, parcial y completo. Estas distribuciones permiten observar cómo varían los valores de cada métrica dependiendo del escenario, proporcionando información clave sobre el desempeño del modelo en condiciones específicas.

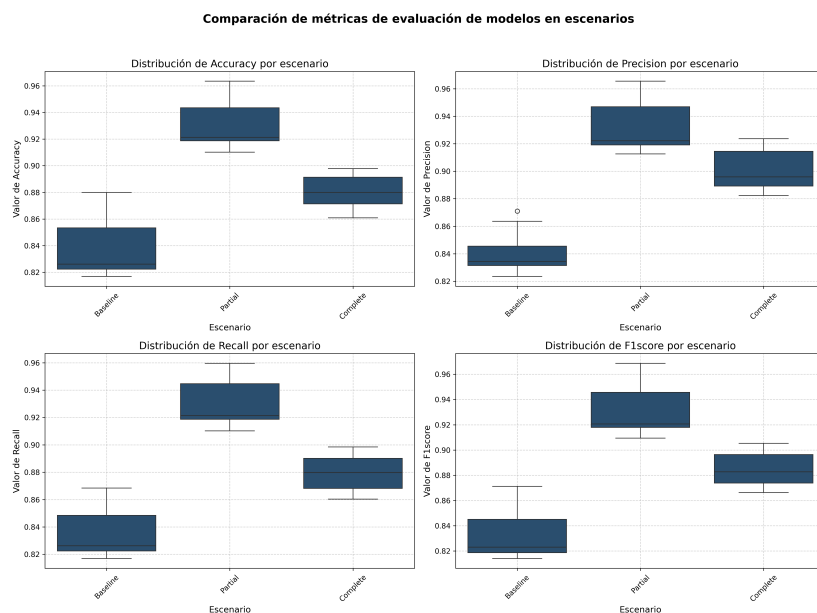


Figure 7.8: Diferencias entre métricas por escenario.

En particular, se observa que los valores de las métricas tienden a mejorar progresivamente desde el escenario *línea base* hasta el *completo*, con el escenario *parcial* presentando mejores valores. Además, la dispersión de los valores es mayor en el escenario *línea base*, indicando una menor estabilidad en el desempeño en comparación con los otros escenarios.

7.6.5 Validación de supuestos

Para garantizar la validez de los análisis realizados y confirmar el cumplimiento de los supuestos necesarios para los métodos estadísticos empleados, se evaluaron los siguientes aspectos: *la normalidad de los residuos y la homogeneidad de las varianzas*.

Normalidad de los residuos

La normalidad de los residuos se evaluó utilizando la prueba de Shapiro-Wilk, cuyos resultados se presentan en la Tabla 7.25. Según los valores obtenidos para las métricas evaluadas (*Accuracy*, *F1score*, *Precision* y *Recall*), los valores de p asociados a la estadística de Shapiro-Wilk son significativamente menores que 0.05, lo que indica una desviación significativa de la normalidad en todas las métricas.

Adicionalmente, los gráficos Q-Q mostrados en la Figura 7.9 proporcionan una visualización complementaria, confirmando la ausencia de normalidad en los residuos para cada una de las métricas y escenarios analizados. Complementariamente, la Figura 7.10 muestra histogramas de los residuos con curvas de densidad ajustadas, los cuales también revelan patrones de asimetría y desviaciones respecto a la distribución normal esperada.

Métrica	Shapiro_Stat	Shapiro_p	Normalidad
Accuracy	0.892483	1.973×10^{-8}	No
F1score	0.884340	7.665×10^{-9}	No
Precision	0.877704	3.656×10^{-9}	No
Recall	0.881184	5.373×10^{-9}	No

Table 7.25: Resultados de prueba de normalidad Shapiro-Wilk para las métricas.

Homogeneidad de varianzas

La homogeneidad de las varianzas entre los diferentes escenarios (línea base, parcial y completo) se verificó mediante la prueba de Levene. En la Tabla 7.26 se presentan los resultados correspondientes. Para todas las métricas evaluadas, los valores de p asociados a la estadística de Levene son mayores a 0.05, lo que permite concluir que no hay evidencia suficiente para rechazar la hipótesis nula de igualdad de varianzas. Por lo tanto, se asume homocedasticidad en los datos analizados.

Métrica	Escenarios comparados	Levene_stat	Levene_p	Homocedasticidad
Accuracy	Baseline, Partial, Complete	0.966853	0.382954	Sí
F1score	Baseline, Partial, Complete	0.800084	0.451458	Sí
Precision	Baseline, Partial, Complete	1.356376	0.261159	Sí
Recall	Baseline, Partial, Complete	0.319029	0.727413	Sí

Table 7.26: Resultados de la prueba de Levene para homocedasticidad.

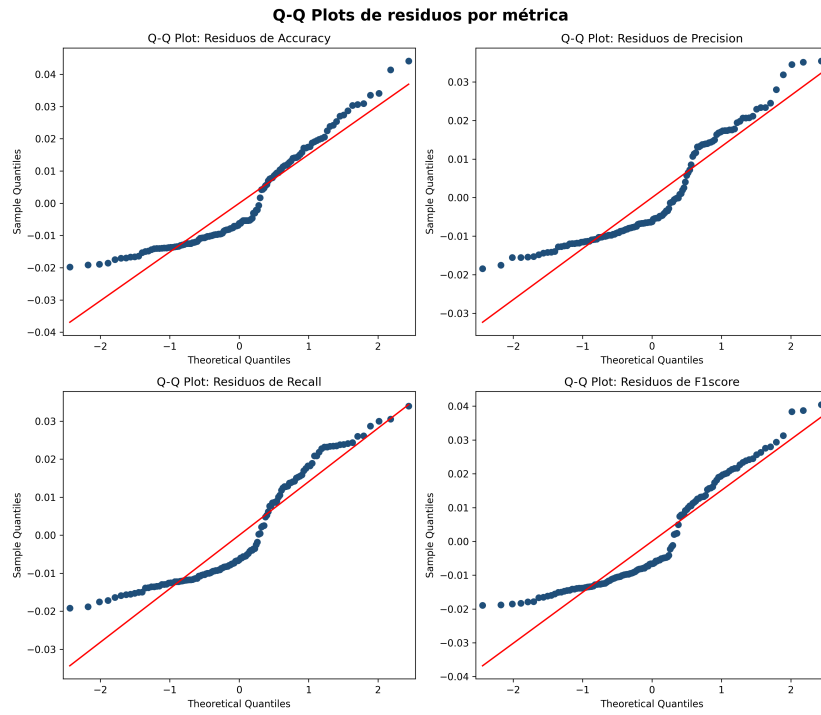


Figure 7.9: Gráficos Q-Q por métricas y escenario para observar la normalidad de los residuos.

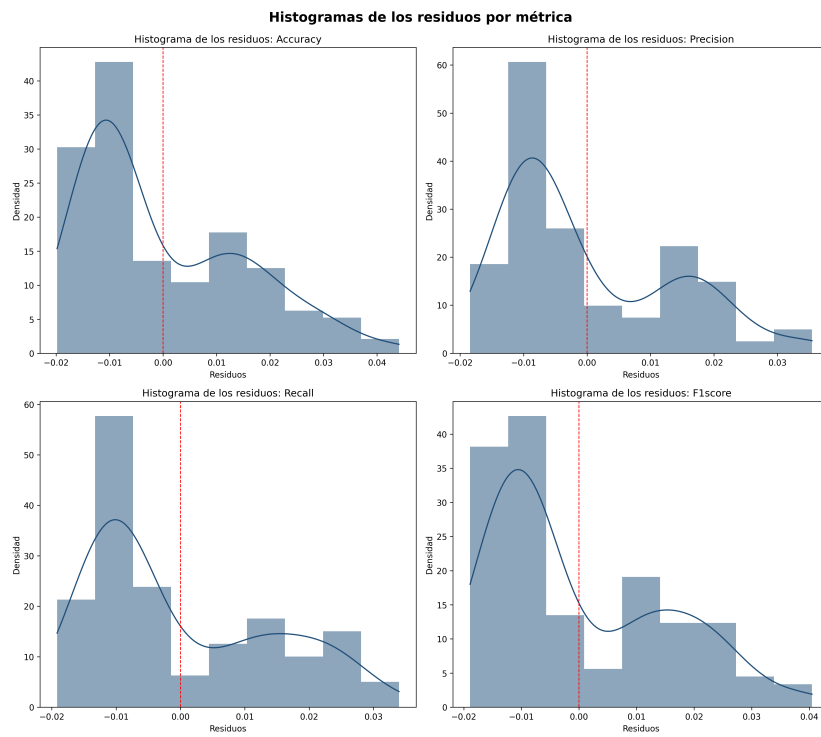


Figure 7.10: Histogramas de los residuos para las métricas *Accuracy*, *Precision*, *Recall* y *F1score*, con curvas de densidad ajustadas. Las líneas rojas indican el valor promedio de los residuos. Estos gráficos evidencian una desviación respecto a la distribución normal, mostrando patrones de asimetría y concentraciones de los residuos hacia valores específicos.

Asimetría y Curtosis

Como se observa en la Tabla 7.27, las métricas tienen valores de asimetría cercanos a 0, lo que indica que las distribuciones son aproximadamente simétricas. En cuanto a la curtosis, todas presentan valores negativos, lo que sugiere distribuciones platicúrticas, es decir, con colas más ligeras que una distribución normal.

Métrica	Asimetría	Curtosis
Accuracy	0.028167	-1.021952
Precision	-0.215263	-1.169481
Recall	0.061331	-1.086214
F1score	-0.124708	-1.050295

Table 7.27: Asimetría y Curtosis de las métricas de evaluación

Justificación para el uso de ANOVA de una vía y Kruskal-Wallis

En este estudio, los residuos de las métricas evaluadas han mostrado ser **homocedásticos**, pero no cumplen con el supuesto de normalidad. Sin embargo, el ANOVA es conocido por ser robusto a ligeras desviaciones de normalidad, especialmente cuando las muestras son equilibradas y el tamaño de cada grupo es mayor o igual a 15, como en este caso.

Dado que las métricas presentan distribuciones aproximadamente simétricas y el análisis de homocedasticidad confirmó igualdad de varianzas, el uso de un **ANOVA de una vía** se llevó a cabo como primera aproximación.

Complemento con Kruskal-Wallis

A pesar de las ventajas del ANOVA, su sensibilidad a la normalidad puede influir en los resultados. Por ello, se complementa con la prueba no paramétrica de **Kruskal-Wallis**, que no asume normalidad y compara las medianas de los grupos.

Esta combinación permite contrastar los resultados del ANOVA con una metodología robusta frente a violaciones de la normalidad. La prueba de Kruskal-Wallis también se complementa con análisis post-hoc (Conover) para identificar las diferencias específicas entre los grupos.

7.6.6 Resultados de ANOVA de una vía, tamaño del efecto y Post Hoc

Los resultados del ANOVA de una vía para las métricas evaluadas (Tabla 7.28) muestran diferencias estadísticamente significativas ($p < 0.05$) entre los escenarios en todas las métricas analizadas: *Accuracy*, *Precision*, *Recall* y *F1score*. Los valores del estadístico *F* son elevados para todas las métricas, indicando que las diferencias entre grupos

explican una proporción considerable de la variabilidad total de los datos.

Métrica	Suma de Cuadrados	df	F	p-valor
Accuracy	0.197	2	420.55	5.49×10^{-58}
Precision	0.194	2	539.53	2.94×10^{-64}
Recall	0.202	2	497.57	3.37×10^{-62}
F1score	0.207	2	444.50	2.30×10^{-59}

Table 7.28: Resultados del ANOVA de una vía para las métricas evaluadas

Para cuantificar la magnitud de las diferencias entre los grupos, se calcularon los tamaños del efecto, representados por η^2 (eta-squared) y ω^2 (omega-squared), cuyos resultados se presentan en la Tabla 7.29. Ambas medidas indican tamaños del efecto grandes para todas las métricas, con valores de η^2 y ω^2 superiores a 0.86, lo que sugiere que las diferencias observadas entre los escenarios son relevantes.

Métrica	Eta-squared (η^2)	Omega-squared (ω^2)
Accuracy	0.864353	0.861413
Precision	0.891005	0.888619
Recall	0.882889	0.880333
F1score	0.870715	0.867906

Table 7.29: Resultados del tamaño del efecto para cada métrica

Posteriormente, se realizó un análisis post-hoc utilizando el test de Tukey HSD para identificar cuáles de los pares de escenarios presentan diferencias significativas (Tabla 7.30). Para todas las métricas, se encontraron diferencias significativas (p -ajustada < 0.05) entre todos los pares de escenarios (*Baseline* vs. *Complete*, *Baseline* vs. *Partial*, y *Complete* vs. *Partial*). Estas diferencias se reflejan en los intervalos de confianza, que no incluyen el valor cero, lo que refuerza la conclusión de que las medias de los escenarios son distintas entre sí.

Métrica	Grupo 1	Grupo 2	Diferencia de medias	p-ajustada	Límite inferior	Límite superior	Rechazo
Accuracy	Baseline	Complete	0.0448	0.0	0.0372	0.0524	True
	Baseline	Partial	0.0935	0.0	0.0859	0.1012	True
	Complete	Partial	0.0487	0.0	0.0411	0.0564	True
Precision	Baseline	Complete	0.0616	0.0	0.0549	0.0683	True
	Baseline	Partial	0.091	0.0	0.0843	0.0977	True
	Complete	Partial	0.0294	0.0	0.0227	0.0361	True
Recall	Baseline	Complete	0.0451	0.0	0.038	0.0522	True
	Baseline	Partial	0.0947	0.0	0.0875	0.1018	True
	Complete	Partial	0.0495	0.0	0.0424	0.0566	True
F1score	Baseline	Complete	0.0526	0.0	0.045	0.0602	True
	Baseline	Partial	0.0958	0.0	0.0883	0.1034	True
	Complete	Partial	0.0432	0.0	0.0355	0.0508	True

Table 7.30: Resultados post-hoc de Tukey HSD para todas las métricas

7.6.7 Resultados de Kruskal-Wallis y Post Hoc

Los resultados de la prueba de Kruskal-Wallis (Tabla 7.31) muestran diferencias estadísticamente significativas ($p < 0.05$) entre los grupos para todas las métricas evaluadas: *Accuracy*, *Precision*, *Recall* y *F1score*. Los altos valores del estadístico H refuerzan la presencia de variaciones relevantes entre los escenarios.

El análisis post-hoc de Conover (Tabla 7.32) identifica pares de grupos con diferencias significativas. Para todas las métricas, las

Métrica	H-statistic	p-valor	Significancia
Accuracy	115.58	8.0×10^{-26}	Sí
Precision	113.79	2.0×10^{-25}	Sí
Recall	118.30	2.9×10^{-26}	Sí
F1score	118.42	1.9×10^{-26}	Sí

Table 7.31: Resumen de la prueba de Kruskal-Wallis para todas las métricas

comparaciones indican que las diferencias observadas son consistentes entre todos los grupos, con valores p extremadamente pequeños en la mayoría de las combinaciones. Esto confirma que las diferencias detectadas por Kruskal-Wallis reflejan efectos significativos entre los escenarios.

Métrica	Grupo 1	Grupo 2	p-valor
Accuracy	1	2	3.03×10^{-58}
	1	3	1.09×10^{-26}
	2	3	5.74×10^{-30}
Precision	1	2	1.48×10^{-55}
	1	3	7.35×10^{-29}
	2	3	5.19×10^{-24}
Recall	1	2	7.63×10^{-63}
	1	3	2.76×10^{-31}
	2	3	4.56×10^{-32}
F1score	1	2	4.71×10^{-63}
	1	3	1.70×10^{-31}
	2	3	3.62×10^{-32}

Table 7.32: Resultados post-hoc (Conover) para las métricas evaluadas

7.7 Discusión

El procesamiento de una base de datos generada mediante *crowdsourcing* presenta desafíos considerables en el procesamiento de datos. Si bien técnicas como el Reconocimiento Óptico de Caracteres (OCR) pueden ser beneficiosas para extraer información, la *calidad de los datos* obtenida no siempre es la mejor. La ausencia de garantías sobre la precisión ortográfica y la consistencia de los datos desde la fuente representa un obstáculo considerable. La calidad insuficiente de los datos de entrada puede llevar a una pérdida de información crucial y afectar negativamente el rendimiento de los modelos de Procesamiento de Lenguaje Natural (PLN).

La identificación y el correcto tratamiento de idiomas fue un punto crítico en este estudio. Aunque se llevó a cabo un preprocesamiento para eliminar palabras en idiomas distintos al inglés, el proceso no fue completamente efectivo, lo que repercutió en el vocabulario utilizado para entrenar el modelo *Word2Vec*. La eliminación manual de palabras no deseadas no es una solución viable en un flujo de trabajo automatizado. Por lo tanto, mejorar las técnicas de detección de idiomas mediante la implementación de modelos más robustos es relevante para futuros trabajos.

La selección del parámetro de frecuencia mínima en el entrenamiento de *Word2Vec* también tuvo implicaciones significativas. Al establecer una frecuencia mínima de aparición de las palabras, se redujo el vocabulario de manera considerable, eliminando tanto ruido, como posibles términos relevantes de baja frecuencia. Este ajuste fue necesario para mitigar el impacto de errores ortográficos y palabras en otros idiomas, pero también pudo contribuir a la pérdida de información valiosa.

El modelo ajustado de GLiNER, utilizado para el Reconocimiento de Entidades Nombradas (NER), no detectó algunos ingredientes, lo que llevó a su exclusión en las nuevas listas formadas. Además, la presencia persistente de aditivos en el vocabulario entrenado indica que el modelo no logró separar completamente las categorías de ingredientes y aditivos, lo que sugiere la necesidad de refinamiento en la etapa de preprocesamiento y etiquetado.

En cuanto al modelo *Word2Vec* para generar *word embeddings*, los resultados fueron satisfactorios en términos de las relaciones semánticas capturadas entre ingredientes. Se observaron agrupaciones basadas en métodos de elaboración, perfiles de sabor y categorías de alimentos, lo que indica que el modelo pudo aprender representaciones útiles. Sin embargo, la inclusión de etiquetas como "Organic" en los ingredientes generó agrupaciones que podrían no ser deseables. Estas etiquetas adicionales representan atributos que deberían ser tratados de manera separada, ya que su inclusión directa en el modelo de PLN puede distorsionar las relaciones semánticas que se desean capturar.

La visualización de los *word embeddings* mediante PCA y t-SNE permitió identificar hallazgos sobre la estructura de los datos, aunque el análisis no fue exhaustivo. La utilización de algoritmos de agrupamiento (*cluster*) especializados podría ofrecer una descripción más precisa del comportamiento de los vectores en espacios de alta dimensionalidad.

Las técnicas empleadas para generar vectores representativos de las listas de ingredientes de cada producto —suma y promedio de vectores— no mostraron diferencias significativas en el desempeño de los modelos de clasificación. Esto sugiere que nuevas metodologías para combinar *word embeddings* podrían explorarse para mejorar la representación y, por ende, el rendimiento de los clasificadores.

El desbalance en las clases etiquetadas afectó negativamente el rendimiento de los modelos de clasificación. Aunque tanto el *submuestreo*, como *SMOTE* se evaluaron como técnicas de balanceo,

únicamente SMOTE demostró ser útil para mejorar las métricas de desempeño, en particular para el SVM con kernel RBF. Sin embargo, la ampliación del conjunto de datos con más registros reales (o categorías adicionales) podría permitir el entrenamiento de modelos más robustos y generalizables.

Como se menciona en Pan³, la adopción de técnicas de PLN representa un enfoque viable para comparar la similitud entre diferentes ingredientes. Este marco permite avanzar hacia la sustitución de ingredientes como parte del desarrollo de nuevos productos mediante inteligencia artificial. Basándose en una medida de similitud, como las utilizadas en el estudio, es posible explorar nuevas metodologías para optimizar esta fase. No obstante, en este trabajo no se incluyeron arquitecturas neuronales como Redes Neuronales Recurrentes (RNN) o Memorias a largo y corto plazo (LSTM) para modelar el lenguaje, lo que deja abierta la posibilidad de que futuros trabajos incorporen estas u otras arquitecturas, principalmente aquellas basadas en codificadores y decodificadores, como menciona Lei⁴, principalmente por su naturaleza para procesar datos de alta dimensionalidad.

Además, siguiendo las observaciones del estudio, es crucial considerar que dos ingredientes considerados similares con base en las medidas actuales de similitud, no necesariamente serán intercambiables en contextos prácticos⁵. Esto se debe a cómo se calculan los vectores de representación (*word embeddings*), los cuales tienden a capturar similitudes basadas en contextos con vecinos semánticos similares. Por lo tanto, se requiere investigación adicional tanto para evaluar la efectividad de estas métricas de similitud como para explorar alternativas que puedan reflejar mejor las características específicas de los ingredientes en escenarios reales.

Por otro lado, el espacio latente generado por Word2Vec presenta ciertas limitaciones, como su dependencia del contexto lingüístico, lo que significa que las relaciones capturadas entre los ingredientes se basan estrictamente en los contextos en los que aparecen en el corpus textual. Esto puede resultar insuficiente cuando las similitudes semánticas deseadas no están adecuadamente reflejadas en el texto de origen. Además, para aplicaciones prácticas, es crucial complementar estas representaciones semánticas con información explícita sobre las características físicas o químicas de los ingredientes, como sabor, textura, comportamiento en procesos de cocción, entre otros, que son fundamentales para evaluar la viabilidad de sustituciones en contextos culinarios.

³ Y. Pan, Q. Xu, and Y. Li, "Food recipe alternation and generation with natural language processing techniques," in *Proceedings - 2020 IEEE 36th International Conference on Data Engineering Workshops, ICDEW 2020*, pp. 94–97, Institute of Electrical and Electronics Engineers Inc., Apr. 2020

⁴ Z. Lei, A. U. Haq, M. Dorraki, D. Zhang, and D. Abbott, "Composing recipes based on nutrients in food in a machine learning context," *Neurocomputing*, vol. 415, pp. 382–396, 2020

⁵ Y. Pan, Q. Xu, and Y. Li, "Food recipe alternation and generation with natural language processing techniques," in *Proceedings - 2020 IEEE 36th International Conference on Data Engineering Workshops, ICDEW 2020*, pp. 94–97, Institute of Electrical and Electronics Engineers Inc., Apr. 2020

En este sentido, el uso de un autoencoder puede ser una solución efectiva para procesar los vectores generados por Word2Vec, ya que permite aprender representaciones latentes refinadas que pueden combinar las relaciones semánticas capturadas en el texto con otros atributos adicionales. Este procesamiento no solo podría mejorar la identificación de sustitutos más adecuados para ingredientes, sino también facilitar la exploración de combinaciones novedosas al integrar dimensiones latentes que reflejen tanto propiedades lingüísticas como no lingüísticas.

Finalmente, la integración de los vectores de ingredientes y aditivos para la predicción de clases demostró ser fundamental para obtener mejores métricas de desempeño. Si bien en esta ocasión los modelos se construyeron de manera independiente para ingredientes y aditivos como parte de un enfoque exploratorio, en futuras investigaciones sería deseable desarrollar un modelo unificado que integre ambas entidades bajo términos de comparación y del análisis de resultados.

8 Conclusiones

Contents

8.1	Conclusiones finales	164
8.1.1	Principales hallazgos	164
8.1.2	Aportaciones del estudio	165
8.1.3	Limitaciones y futuras líneas de trabajo	165
8.2	Trabajo a futuro	165
8.2.1	Conclusión general	166
8.3	Anexos	167
8.3.1	Cálculo de la muestra para evaluación de idioma en las listas de ingredientes	167
8.3.2	Repositorio con código replicable . . .	167
8.3.3	Términos de uso de Wikipedia	168
8.3.4	Listado de URLs	168

8.1 Conclusiones finales

8.1.1 Principales hallazgos

- El estudio demostró que las métricas de desempeño (*Accuracy*, *Precision*, *Recall* y *F1-score*) mejoraron desde el contraste del escenario *línea base*, el *parcial*, el *completo*, destacando el impacto positivo de integrar técnicas como *word embeddings*, balanceo de clases (SMOTE) y el modelo de NER ajustado.
- ANOVA de una vía y Kruskal-Wallis confirmaron diferencias significativas entre los escenarios, con tamaños del efecto (η^2 y ω^2) superiores a 0.86, reflejando un impacto importante en las métricas clave.
- SMOTE mejoró significativamente el desempeño en clases minoritarias, demostrando su efectividad en datos desbalanceados como los de OpenFoodFacts.
- La integración de vectores de aditivos enriqueció las representaciones de los productos alimentarios, mejorando las métricas generales.

- El modelo SVM con kernel radial destacó en el desempeño, aunque no fue un factor de análisis principal.

8.1.2 *Aportaciones del estudio*

- Desarrollo de un marco que combina técnicas de PLN, como NER ajustado y *word embeddings*, para analizar etiquetas de productos alimenticios.
- Propuesta de un enfoque efectivo para balancear datos desbalanceados mediante SMOTE, aplicable a otros dominios.
- Evaluación de escenarios experimentales, aportando al análisis de datos no estructurados en la industria alimentaria.
- Aplicación práctica para enriquecer bases colaborativas, mejorando la calidad de los datos para aplicaciones industriales y de investigación.

8.1.3 *Limitaciones y futuras líneas de trabajo*

- Mejorar la calidad de los datos, con preprocesamiento más avanzado y adaptación a otros idiomas como el español.
- Integrar datos estructurados para análisis más completos y validar el modelo en otros dominios.
- Optimizar representaciones semánticas para capturar mejor las características de ingredientes y aditivos.

8.2 *Trabajo a futuro*

- Implementar corrección de errores ortográficos y segmentación de idiomas para mejorar la calidad de los datos.
- Reentrenar el modelo NER con datos más diversos y explorar alternativas como BERT para mejorar la clasificación.
- Desarrollar técnicas de *word embeddings*, combinando enfoques estáticos y dinámicos.
- Incorporar atributos adicionales, como perfiles nutricionales y fisicoquímicos, para enriquecer las predicciones del modelo.
- Diseñar una arquitectura ETL y sistemas OCR para automatizar la integración de datos desde etiquetas físicas.
- Usar clusterización y reducción de dimensionalidad para analizar relaciones entre ingredientes y mejorar la interpretabilidad.

8.2.1 Conclusión general

Este estudio demuestra el potencial de las técnicas de Procesamiento de Lenguaje Natural (PLN) para transformar datos no estructurados, como etiquetas de productos alimenticios, en información estructurada y útil para la industria alimentaria. Las técnicas implementadas, como los modelos de *word embeddings*, el balanceo de clases mediante SMOTE y el ajuste de modelos de reconocimiento de entidades nombradas (NER), lograron mejorar significativamente el desempeño de los modelos en tareas críticas como la identificación de ingredientes y aditivos, así como en la clasificación de productos alimenticios. Estas mejoras se reflejaron en métricas como *accuracy*, *precision*, *recall* y *F1-score*, con diferencias estadísticamente significativas entre escenarios experimentales.

El impacto positivo de estas técnicas no solo mejoró el análisis de datos no estructurados, sino que también habilitó aplicaciones prácticas en el diseño y desarrollo de nuevos productos alimenticios. La mejora en la calidad de los datos y la integración de datos no estructurados como una dimensión clave de análisis tienen el potencial de generar un modelo robusto, enfocado en la recombinación de ingredientes y en la exploración de nuevas posibilidades para la formulación de productos. Este enfoque podría ampliar significativamente las oportunidades para diseñar productos más innovadores y alineados con las tendencias y preferencias del mercado.

Además, la centralización de datos de calidad en bases de datos tipo crowdsourcing contribuye a mejorar la consistencia y escalabilidad de los datos, al tiempo que facilita su actualización continua. Esto asegura que las bases de datos alimentarias mantengan su relevancia, precisión y utilidad a lo largo del tiempo, ofreciendo un recurso confiable tanto para la industria, como para la investigación.

Finalmente, las técnicas propuestas no solo generan nuevas oportunidades para resolver desafíos en el ámbito alimentario, sino que también ofrecen un marco replicable para abordar problemas en otros sectores que requieren el análisis de datos no estructurados. En conclusión, este estudio confirma el papel fundamental del PLN como una herramienta estratégica para impulsar la innovación y la mejora continua en la industria alimentaria.

8.3 Anexos

8.3.1 Cálculo de la muestra para evaluación de idioma en las listas de ingredientes

Del conjunto de datos de $N = 42,027$ registros que fueron sometidos a identificación de idioma. Para estimar la proporción de registros identificados correctamente, se calculó una muestra representativa con un nivel de confianza del 95% y un margen de error del 5%.

El tamaño de la muestra (n) se calculó utilizando la fórmula para poblaciones finitas:

$$n = \frac{Z^2 \cdot p \cdot q \cdot N}{e^2(N - 1) + Z^2 \cdot p \cdot q} \quad (8.1)$$

Donde Z es el valor crítico para el nivel de confianza deseado (para 95%, $Z = 1.96$); p es la proporción esperada de éxito (se asume $p = 0.5$ para máxima variabilidad); $q = 1 - p$; e es el margen de error tolerado (en este caso, $e = 0.05$); y N es el tamaño de la población total.

Sustituyendo los valores:

$$n = \frac{(1.96)^2 \cdot 0.5 \cdot 0.5 \cdot 42,027}{(0.05)^2(42,027 - 1) + (1.96)^2 \cdot 0.5 \cdot 0.5} \quad (8.2)$$

Se calcula:

$$n = \frac{3.8416 \cdot 0.25 \cdot 42,027}{0.0025 \cdot 42,026 + 0.9604} \quad (8.3)$$

$$n = \frac{40,377.6}{105.065 + 0.9604} \quad (8.4)$$

$$n = \frac{40,377.6}{106.0254} \approx 381 \quad (8.5)$$

Por lo tanto, el tamaño de muestra necesario es aproximadamente $n = 381$ registros. Se tomó una muestra de $n = 381$ registros generados al azar.

8.3.2 Repositorio con código replicable

Para aquellas personas interesadas en explorar más a fondo los detalles técnicos del proyecto, incluyendo la implementación del modelo, el preprocesamiento de datos y el flujo de trabajo, se ha puesto a disposición el código fuente en un repositorio público.

El repositorio incluye la organización de archivos y directorios del proyecto; una guía para reproducir los resultados o aplicar el enfoque

a otros conjuntos de datos; documentación en el código y scripts y notebooks con ejemplos reproducibles.

El acceso al repositorio es https://github.com/patymunoz/food_additives_ner.

8.3.3 *Términos de uso de Wikipedia*

Wikipedia es una plataforma impulsada por la Fundación Wikimedia, una organización sin fines de lucro que busca empoderar a personas de todo el mundo para compartir conocimiento bajo licencias libres. Los términos de uso destacan que los usuarios son responsables de sus contribuciones y deben cumplir con las políticas de cada proyecto, como el Código Universal de Conducta. Además, los contenidos publicados deben ser licenciados bajo Creative Commons (CC BY-SA 4.0) y/o GFDL para garantizar su redistribución libre y atribución adecuada. Para más detalles sobre las licencias y derechos, revisa los términos completos en https://foundation.wikimedia.org/wiki/Policy:Terms_of_Use/es.

8.3.4 *Listado de URLs*

El listado completo de URLs utilizadas para descargar la información sobre los aditivos alimentarios está disponible en el siguiente enlace:

<https://gist.github.com/patymunoz/1adc7acce7e214f6bd858814c293c97b>

Bibliography

- [1] I. Pravst, M. Hribar, K. Žmitek, B. Blažica, B. Koroušić Seljak, and A. Kušar, “Branded Foods Databases as a Tool to Support Nutrition Research and Monitoring of the Food Supply: Insights From the Slovenian Composition and Labeling Information System,” *Frontiers in Nutrition*, vol. 8, Jan. 2022.
- [2] FoodData Central, “Global Branded Food Products Database Documentation and Download User Guide.” Online, April 2023.
- [3] Codex Alimentarius, “Glossary of Terms: Food Additive.” <https://www.fao.org/fao-who-codexalimentarius/glossary/en/>, 2024. [En línea; consultado el 04 de diciembre de 2024].
- [4] D. Park, K. Kim, S. Kim, M. Spranger, and J. Kang, “FlavorGraph: a large-scale food-chemical graph for generating food representations and recommending food pairings,” *Sci Rep*, vol. 11, Dec. 2021.
- [5] X. Zhang, T. Zhou, L. Zhang, K. Y. Fung, and K. M. Ng, “Food Product Design: A Hybrid Machine Learning and Mechanistic Modeling Approach,” *Ind Eng Chem Res*, vol. 58, pp. 16743–16752, Sept. 2019.
- [6] M. Al-Sarayreh, M. Gomes Reis, A. Carr, and M. M. dos Reis, “Inverse design and AI/Deep generative networks in food design: A comprehensive review,” *Trends in Food Science and Technology*, vol. 138, pp. 215–228, Aug. 2023.
- [7] Organización de las Naciones Unidas para la Alimentación y la Agricultura (FAO), “Etiquetado de alimentos,” 2024. [En línea; consultado el 30 de marzo de 2024].
- [8] NotCo, “Conoce a Giuseppe, la tecnología que lo hace realidad.” <https://notco.com/mx/sobre/giuseppe-geek>. [En línea; consultado el 28 de julio de 2024].
- [9] S. Chaudhari, A. R., V. G. Tekkur, P. G. L., and S. R. Karki, “Ingredient/Recipe Algorithm Using Web Mining and Web

- Scraping for Smart Chef,” in *Proceedings of the IEEE CONECCT 2020: 6th International Conference on Electronics, Computing and Communication Technologies*, 2020.
- [10] M. Chang, L. V. Guillaín, H. Jung, V. M. Hare, J. Kim, and M. Agrawala, “RecipeScape: An interactive tool for analyzing cooking instructions at scale,” in *Conference on Human Factors in Computing Systems - Proceedings*, Association for Computing Machinery, Apr. 2018.
 - [11] Z. Lei, A. U. Haq, M. Dorraki, D. Zhang, and D. Abbott, “Composing recipes based on nutrients in food in a machine learning context,” *Neurocomputing*, vol. 415, pp. 382–396, 2020.
 - [12] Y. Pan, Q. Xu, and Y. Li, “Food recipe alternation and generation with natural language processing techniques,” in *Proceedings - 2020 IEEE 36th International Conference on Data Engineering Workshops, ICDEW 2020*, pp. 94–97, Institute of Electrical and Electronics Engineers Inc., Apr. 2020.
 - [13] F. Han, R. Guerrero, and V. Pavlovic, “Cookgan: Meal image synthesis from ingredients,” 2020.
 - [14] J. Marin, A. Biswas, F. Ofli, N. Hynes, A. Salvador, Y. Aytar, I. Weber, and A. Torralba, “Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images,” 2019.
 - [15] C. Anderson, “A survey of food recommenders,” 2018.
 - [16] A. Morales-Garzón, J. Gómez-Romero, and M. J. Martín-Bautista, “A word embedding-based method for unsupervised adaptation of cooking recipes,” *IEEE Access*, vol. 9, pp. 27389–27404, 2021.
 - [17] M. Wolf, “From data-scraping to discernment layer: How notco’s giuseppe ai has evolved over the past decade,” 2024. [En línea; consultado el 28 de octubre de 2024].
 - [18] J. Tsujii, “Natural Language Processing and Computational Linguistics,” *Computational Linguistics*, vol. 47, pp. 707–727, 12 2021.
 - [19] DeepLearningIA, “A complete guide to natural language processing,” 2023. [En línea; consultado el 11 de enero de 2024].
 - [20] S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing A Comprehensive Guide to Building Real-World NLP Systems*. 2020.

- [21] O. Pramod, “Disambiguating words: A deep dive into word sense disambiguation,” December 2023. Published in Towards Data Science.
- [22] R. Socher, F. Chaubard, and R. Mundra, “Cs 224d: Deep learning for nlp,” Lecture Notes, 2016. Spring 2016, Stanford University.
- [23] Stanford Encyclopedia of Philosophy, “Ambiguity,” 2021. First published Mon May 16, 2011; substantive revision Sat May 22, 2021.
- [24] Princeton University, “WordNet: A Lexical Database for English.” <https://wordnet.princeton.edu/>. [En línea; consultado el 02 de mayo de 2024],.
- [25] A. Finch, E. Black, Y.-S. Hwang, and E. Sumita, “Using lexical dependency and ontological knowledge to improve a detailed syntactic and semantic tagger of English,” in *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, (Sydney, Australia), pp. 215–222, Association for Computational Linguistics, July 2006.
- [26] D. E. Rumelhart, J. L. McClelland, and P. R. Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. Volume 1, The MIT Press, 1986.
- [27] G. E. Hinton, “Learning distributed representations of concepts,” in *Parallel distributed processing: Implications for psychology and neurobiology* (R. G. M. Morris, ed.), pp. 46–61, Clarendon Press/Oxford University Press, 1989. Reprinted from Proceedings of the Eighth Annual Conference of the Cognitive Science Society, Amherst, Massachusetts: Erlbaum Associates, 1986.
- [28] M. A. Ralph, “Distributed versus localist representations: Evidence from a study of item consistency in a case of classical anomia,” *Brain and Language*, vol. 64, pp. 339–360, October 1 1998.
- [29] L. Malagutti, A. Buinovskij, A. Svete, C. Meister, A. Amini, and R. Cotterell, “The role of n -gram smoothing in the age of neural networks,” 2024.
- [30] D. Jurafsky and J. H. Martin, *Speech and Language Processing*. Copyright © 2024. All rights reserved., 2024. Draft of August 20, 2024.
- [31] Y. Bengio, R. Ducharme, and P. Vincent, “A neural probabilistic language model,” in *Advances in Neural Information Processing Systems* (T. Leen, T. Dietterich, and V. Tresp, eds.), vol. 13, MIT Press, 2000.

- [32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, pp. 1–21, 1988.
- [33] C. Wang, P. Nulty, and D. Lillis, "A comparative study on word embeddings in deep learning for text classification," in *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*, NLPPIR '20, (New York, NY, USA), p. 37–46, Association for Computing Machinery, 2021.
- [34] E. Tiu, "Understanding latent space in machine learning: Learn a fundamental, yet often 'hidden,' concept of deep learning," *Towards Data Science*, February 2020. Published in Towards Data Science.
- [35] V. Hofmann, J. B. Pierrehumbert, and H. Schütze, "Dynamic contextualized word embeddings," 2021.
- [36] F. Chiusano, "Two minutes nlp — word embeddings models you should know tf-idf, word2vec, glove, fasttext, elmo, cove, bert, roberta, etc.," December 2021. Published in Towards Data Science.
- [37] H. J. W. G. W. Z. Z. C. Meng, Yu and J. Han, "Unsupervised word embedding learning by incorporating local and global contexts," *Frontiers in Big Data*, vol. 3, p. 9, 2020.
- [38] E. Huang, R. Socher, C. Manning, and A. Ng, "Improving word representations via global context and multiple word prototypes," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (H. Li, C.-Y. Lin, M. Osborne, G. G. Lee, and J. C. Park, eds.), (Jeju Island, Korea), pp. 873–882, Association for Computational Linguistics, July 2012.
- [39] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space." <http://arxiv.org/abs/1301.3781>, 2013.
- [40] C. Manning, J. Hewitt, F. Chaubard, M. Fang, G. Genthial, R. Mundra, and R. Socher, "Note 1: Introduction and word2vec." johnhew@cs.stanford.edu, 2023. CS 224n: Natural Language Processing with Deep Learning, Winter 2023.
- [41] M. Mohammadi, R. Mundra, R. Socher, L. Wang, and A. Kamath, "Cs224n: Natural language processing with deep learning." Course Lecture Notes, Stanford University, Winter 2019, 2019. Lecture Notes: Part V Language Models, RNN, GRU and LSTM.
- [42] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," 2013.

- [43] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.
- [44] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” 2014.
- [45] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” 2016.
- [46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017.
- [47] L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers*. O’Reilly Media, Inc., revised ed., 2022.
- [48] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, “Unifying large language models and knowledge graphs: A roadmap,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, p. 3580–3599, July 2024.
- [49] F. Preetham, “Math behind positional embeddings in transformer models,” May 2024.
- [50] OpenFoodFacts, “Open food facts - world food products database.” <https://world.openfoodfacts.org/>, 2024. [En línea; consultado el 29 de marzo de 2024].
- [51] OpenFoodFacts, “Open food facts python sdk.” <https://github.com/openfoodfacts/openfoodfacts-python>, 2016–2024. MIT License.
- [52] F. A. Research, “Fasttext pretrained language identification model.” <https://dl.fbaipublicfiles.com/fasttext/supervised-models/lid.176.bin>, 2024. [En línea; consultado el 29 de marzo de 2024].
- [53] L. Technologies, “Langcodes: A library for working with language codes.” <https://pypi.org/project/langcodes/>, 2024. [En línea; consultado el 29 de marzo de 2024].
- [54] U. Zaratiana, N. Tomeh, P. Holat, and T. Charnois, “Gliner: Generalist model for named entity recognition using bidirectional transformer,” 2023.
- [55] FAO/WHO Food Standards, “Codex Alimentarius: Food Additive Index,” 2023. Updated up to the 46th Session of the Codex Alimentarius Commission (2023).

- [56] Food and Agriculture Organization of the United Nations (FAO) and World Health Organization (WHO), “Codex General Standard for Food Additives (GSFA) Online Database.” <https://www.fao.org/fao-who-codexalimentarius/codex-texts/dbs/gsfa/en/>. [En línea; consultado el 07 de agosto de 2024].
- [57] W. J. Mattingly, “gliner-finetune.” <https://github.com/wjbmattlingly/gliner-finetune>, 2024. [En línea; consultado el 05 de junio de 2024].
- [58] D. Batista, “Named-entity evaluation metrics based on entity-level,” 2018. consultado el 08 de diciembre de 2024.
- [59] N. Chinchor and B. Sundheim, “MUC-5 evaluation metrics,” in *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27, 1993*, 1993.
- [60] I. Segura-Bedmar, P. Martínez, and M. Herrero-Zazo, “SemEval-2013 task 9 : Extraction of drug-drug interactions from biomedical texts (DDIExtraction 2013),” in *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)* (S. Manandhar and D. Yuret, eds.), (Atlanta, Georgia, USA), pp. 341–350, Association for Computational Linguistics, June 2013.
- [61] MantisAI, “Nervaluate: Full named-entity evaluation metrics based on semeval’13,” 2023.