Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA
Unidad Iztapalapa

DETECCIÓN DE PARÁFRASIS UTILIZANDO ALGORITMOS DE APRENDIZAJE MAQUINAL

Para obtener el grado de:
Maestro en Ciencias y Tecnologías de la Información

Presenta:
Lic. David Luna Luna

Asesores:
Dr. Benjamín Moreno Montiel
Dr. René Mac Kinney Romero

Sinodales:

Presidenta : **Dra. Gemma Bel Enguix**
Secretario : **Dr. Ricardo Marcelín Jiménez**
Vocal : **Ing. Luis Fernando Castro Careaga**

CDMX, México
Agosto 2022

RESUMEN

La detección de paráfrasis es una importante área de estudio, dentro del procesamiento del lenguaje natural (PLN), con muchas aplicaciones como: la generación de respuestas automáticas, mejoras en el desempeño en traducciones automáticas, ranking de consultas, asignación de autoría y detección de plagio. El fenómeno lingüístico de la paráfrasis ha sido estudiado, desde un enfoque computacional, proponiendo modelos de inteligencia artificial que codifican el texto y pronostican el grado de similitud semántica. Todo modelo propuesto es guiado por datos y mucho del desempeño de los modelos depende de la calidad y cantidad de los datos que se utilizan para realizar la fase de entrenamiento. En este trabajo presentamos la creación de un nuevo corpus de paráfrasis extrayendo segmentos discursivos de un repositorio de tesis académicas. También proponemos una serie de modelos, basados en algoritmos de aprendizaje maquinal, que analizan y clasifican un par de textos en las clases *paráfrasis* y *no paráfrasis*. Además, utilizamos el modelo que obtuvo el mejor desempeño para generar una representación visual de la comparación de pares de documentos y ver como se distribuye la similitud semántica entre ellos. El corpus que generamos cuenta con un total de 1,203,964 pares de oraciones de los documentos que procesamos y el desempeño del mejor modelo obtenido alcanza una exactitud, en promedio, de 94 % (sobre los datos del nuevo corpus). La comparación de documentos, página por página, muestra correctamente como se distribuye la similitud semántica y las regiones donde se concentra la mayor similitud encontrada.

***Palabras clave**:* Aprendizaje Maquinal; Paráfrasis; Corpus; Aprendizaje profundo;

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1.1. Motivación

Es natural pensar que el autor de una obra literaria merece el crédito y ser reconocido por su trabajo. Sin embargo, en muchos escenarios se puede saber relativamente poco del grado de *originalidad* de la obra que el autor nos presenta. A la humanidad, a través del tiempo, le ha sido dificil desarrollar mecanismos que promuevan el *compartir el conocimiento* sin afectar a la propiedad intelectual de una obra literaria. En la antigüedad clásica [1], y debido principalmente a la ausencia de mecanismos de control de transmisión textual o bibliográfica, nacieron conceptos como el de *propiedad colectiva* donde se mantenía una ideología "comunitaria" de la literatura. El concepto de *propiedad colectiva* se ve reflejado en el famoso dicho de Quinto Aurelio Símaco, escritor del siglo IV d. C. "*Oratio publicata, res libeta est*". "Una vez publicado, el discurso no pertenece a nadie". Que sin duda genera un fenómeno que desata una serie de enfrentamientos, polémicas y "batallas textuales" entre los filólogos, hombres de letras, catedráticos o escritores de prestigio.
Son varios los ejemplos de aquellos enfrentamientos. Por ejemplo, Platón fue acusado de plagiar al mismo Pitágoras, después de haber adquirido un texto de Filolao (discípulo de Pitágoras) mediante el pago de diez mil denarios. Aulo Gelio, en el siglo II, reprodujo en sus *Noches Áticas* los siguientes versos de Timón:

> *Tú, Platón, puesto que ansiabas el saber, por una suma desorbitada de denarios compraste un librito que te enseñó a escribir el Timeo.*

También Teopompo de Quíos en su obra *La escuela de Platón*, afirma que:

[1] La Antigüedad clásica se puede localizar temporalmente, de forma restringida, en el momento de plenitud de las civilizaciones griega y romana (siglo V a. C. al siglo II d. C.) o, de forma amplia, en toda su duración (siglo VIII a. C. al siglo V d. C.)

> *"se descubrirá que la mayoría de sus diálogos son inútiles y falsos: la mayoría plagios, procedentes de las diatribas de Aristipo y algunos, incluso, de las de Antístenes y muchos de las de Brisón de Heraclea."* [2]

Podemos pensar que tales acusaciones tenían como función principal, aparte de afectar la reputación del autor, de enmudecer a un rival ideológico. Sin embargo, son esas acusaciones las que, de fondo o de manera oculta, atienden a la demanda de *originalidad*, es decir, la imitación por sí sola no es suficiente en un contexto creativo y de erudición si lo que se quiere y desea es el desarrollo de nuevas ideas en cualquier ámbito social y cultural. De modo que contar con mecanismos que promovieran la generación de fuentes literarias con un mayor grado de *originalidad*, se convirtió en una necesidad que debía ser atendida en interés de la literatura.
Más adelante con la creación de las bibliotecas e invención de la imprenta, se crearon nuevos mecanismos de control de transmisión de textos que ayudaron a repeler el concepto de *propiedad colectiva*. La asignación de autoría, de un texto literario, se convirtió en una labor fundamental para la clasificación y el almacenamiento de la propiedad intelectual. Se comenzó a asignar el calificativo de *plagio* a textos y obras literarias que carecieran de *originalidad* o que eran, básicamente, una copia exacta de algún texto previamente revisado. También se comenzaron a crear autoridades literarias capaces de "enjuiciar" la *originalidad* de un texto, y además, con la autoridad suficiente para desprestigiar y retirar el crédito (al menos simbólico) del plagiario descubierto, llevándolo a una desaprobación social y, en el peor de los casos, al silencio textual [1].

Hoy en día, en la era digital en la que nos encontramos, la forma en que se transmiten textos u obras literarias ha cambiado. Ahora la mayor parte de la producción literaria y textual es en formato digital, lo que implica que se tengan grandes cantidades de datos e información literaria disponible para cualquier tipo usuario y en cualquier momento a través de internet, haciendo que el concepto de *propiedad colectiva* vuelva a tomar gran ímpetu en la sociedad actual.
En el ámbito académico, y gracias en mayor medida al internet, es más sencillo acceder a publicaciones de investigadores de todo el mundo. *ResearchGate*, una plataforma enfocada a compartir, descubrir y discutir investigaciones de científicos e investigadores de todo el mundo, reporta que cuenta con más de 135 millones de publicaciones, de más de 20 millones de investigadores, disponibles en su plataforma [3].
En el año 2018 se realizó un estudio [2] para estimar el número de registros con los que contaban 12 motores de búsqueda de textos académicos en internet. *Google Scholar*[4], siendo el que proporciona, por mucho, el mayor volumen de información académica de entre los 12 evaluados, resultó tener alrededor de 389 millones de registros disponibles

[2]Traducción por Antoni Piqué Angordans, en Los Megáricos: Presentación y traducción de los textos. Barcelona: Universitat de València, 1989, pág. 105.

[3]Madisch, D. and Hofmayer, D., 2021. ResearchGate - About. [online] ResearchGate. Available at: https://www.researchgate.net/about [Accessed 7 April 2021].

[4]https://scholar.google.com/

(ya en ese año). Le siguió *Microsoft Academic* con 170 millones. Y el menor de todos, *Semantic Scholar*, con 40 millones de registros.
Podemos imaginar lo sencillo que puede ser en estos días realizar el proceso de *plagio* de un texto si tomamos en cuenta el factor de accesibilidad (que nos brindan ese tipo de plataformas) y si lo mezclamos con el hecho de que es muy sencillo realizar un copia de un texto entero, debido a que las funciones *copy-paste* son estándar en todo artefacto tecnológico actual. Es decir, debido a la sencillez de acceder a textos ajenos y de copiarlos, el concepto de *propiedad colectiva* es, de hecho, mucho más factible y reproducible que en la antigüedad clásica.

El uso de frases, sentencias o segmentos discursivos completos, sin el permiso o autorización del autor es sin duda un problema social de mayor dificultad hoy en día. Este problema, que recordemos no es nuevo, ahora adquiere otra forma y medios para realizar su cometido. Puede ser visto tanto en el ámbito académico como profesional y tener un impacto socio-cultural relevante.
Un método muy común para cometer plagio sobre un texto o segmento discursivo, es realizar una paráfrasis del mismo (o parafrasear el texto). Un texto es paráfrasis de otro si ambos tienen distinta forma pero aproximadamente el mismo significado semántico [3] o ambos transmiten la misma información pero tienen distintas estructuras léxico-sintácticas [4].
Cada vez es más necesario tener herramientas, en casi cualquier ámbito social, que sean capaces de identificar si alguna persona ha hecho o realizado algún tipo de plagio a nivel documental. En ese sentido la detección de paráfrasis puede ser muy útil ya que ambos, el plagio y paráfrasis, son conceptos que están directamente relacionados. Si bien el ser humano puede identificar la paráfrasis de forma casi automática, crear un algoritmo para que la computadora la detecte de forma automática no es una tarea tan trivial. Sin embargo, se puede plantear el problema de detección de paráfrasis como un problema de clasificación con técnicas de Inteligencia Artificial (IA) y así intentar resolver el problema de detectar paráfrasis creando herramientas tecnológicas que ayuden a detectar el plagio de textos para proceder, con evidencias tangibles e infalibles, en el proceso de denuncia ante las instituciones encargadas de asegurar que se ejerzan los derechos del autor.

Considerando todo lo anterior, en el presente proyecto de investigación vamos a explorar e intentar resolver el problema de detección de paráfrasis, desde un perspectiva computacional y tecnológica, empleando técnicas de Inteligencia Artificial, Aprendizaje Maquinal (AM) y Minería de Datos (MD). Haciendo la aclaración de que la detección de plagio es una de las posibles aplicaciones de la detección de paráfrasis y no es materia de estudio en el presenta trabajo de investigación.
Combinaremos métodos y tecnologías actuales para crear modelos de IA que nos ayuden en la tarea de detectar paráfrasis de forma automática, y minaremos el repositorio de fuentes documentales TESIUAMI (en donde podemos encontrar las tesis de los alumno

egresados de la Universidad Autónoma Metropolitana *UAM*, Unidad Iztapalapa) para generar una base de datos de textos que usaremos como base experimental.

1.2. Hipótesis planteadas

Las hipótesis planteadas para este trabajo de investigación son las siguientes:

1. Dentro de la Unidad Iztapalapa de nuestra universidad existe un gran repositorio de textos académicos llamado TESIUAMI. Es posible crear un corpus de paráfrasis sobre los textos que se encuentran en dicho repositorio de forma automatizada empleando técnicas de minería de datos.

3. Para las computadoras el lenguaje humano es ambiguo y complejo de entender e interpretar en muchos aspectos. Entonces, es posible crear modelos que logren abstraer las estructuras semánticas y contextuales de segmentos discursivos desde una perspectiva computacional.

4. Las técnicas de aprendizaje profundo (Deep Learning, *DL*) han mostrado un desempeño muy bueno en tareas que tienen que ver con el lenguaje natural. Es posible configurar y entrenar una red neuronal que sea capaz de detectar la paráfrasis.

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar un sistema de detección de paráfrasis fusionando algoritmos de Minería de Datos y Aprendizaje Maquinal.

1.3.2. Objetivos particulares

+ Revisar el estado del arte sobre las principales técnicas de AM y Minería de Datos utilizadas en la detección de paráfrasis.

+ Crear un corpus de paráfrasis, de contexto científico, usando como fuente principal el repositorio TESIUAMI.

+ Planear la automatización del proceso de generación de un corpus de paráfrasis sobre nuestra base de datos experimental.

+ Programar algoritmos de AM y DL que ayuden en la detección de paráfrasis.

+ Ensamblar el primer prototipo del Sistema de detección de paráfrasis sobre un corpus de contexto científico, utilizando algoritmos de Minería Datos y Aprendizaje Maquinal.

1.4. Metodología

La metodología a seguir en este proyecto de investigación es la siguiente:

1. Revisión del estado del arte sobre la detección de paráfrasis desde una perspectiva computacional.
2. Definir las tecnologías y lenguaje de programación a usar para desarrollar el sistema de detección de paráfrasis.
3. Seleccionar los textos, dentro del repositorio TESIUAMI, que se van a utilizar para lograr tener una vista minable de los datos.
4. Explorar los métodos y formas de crear un corpus de paráfrasis de forma automática.
5. Seleccionar los algoritmos de AM y DL apropiados y adecuados para realizar la detección de paráfrasis.
6. Implementar los algoritmos antes seleccionados, usando las tecnologías antes seleccionadas.
7. Establecer un esquema de experimentación que nos ayude a evaluar el desempeño y nivel de generalización de nuestras implementaciones.
8. Implementar el primer prototipo del sistema para la detección de paráfrasis.

1.5. Justificación

Este proyecto de investigación se enfoca principalmente en estudiar, implementar y evaluar mecanismos que logren detectar paráfrasis de forma automática. El fenómeno lingüístico denominado "paráfrasis" es de gran interés en el campo del Procesamiento del Lenguaje Natural (*PLN*) dado que parafrasear un texto es una herramienta empleada frecuentemente en muchos contextos y por lo tanto de gran importancia. Es esta razón por la que, dentro del área del PLN, se han estado construyendo sistemas que detecten de forma automática la paráfrasis, para así poder aplicarlos en distintas tareas como: la generación de respuestas y traducciones automáticas, evaluación y generación de resúmenes automáticos, búsquedas especializadas de carácter semántico sobre grandes cantidades de datos, asignación de autoría y detección de plagio. Siendo

la detección de plagio una de las principales aplicaciones de la detección de paráfrasis y también motivo de confusión entre el concepto de plagio y la paráfrasis ya que ambos se relacionan directamente.
El plagio se define como la reutilización de ideas, procesos, resultados o palabras sin reconocer explícitamente al autor original y la fuente [37]. En muchas situaciones si un fragmento de un texto es considerado plagio, entonces el segmento es también considerado como paráfrasis. Pero también existen otras situaciones en las que alguien puede parafrasear un texto sin llegar a ser considerado un plagio, por ejemplo, parafrasear a un autor y agregar correctamente la cita o la referencia al texto original. Así entonces, que un texto sea considerado como paráfrasis de otro es una condición necesaria (aunque tal vez no suficiente o determinante) para que el texto sea considerado como plagio. Por lo tanto, la detección de plagio es un aplicación de la detección de paráfrasis (además de todas las aplicaciones adicionales mencionadas anteriormente).
En ese sentido, contar con herramientas que puedan detectar la paráfrasis de forma automática, ayudaría a identificar casos posibles de *plagio* (tanto en campo profesional como académico) con el objetivo principal de promover la *originalidad* de los textos.

CAPÍTULO 2

ANTECEDENTES

En esta parte del documento vamos a hacer una revisión de conceptos y fundamentos que son base para entender el problema que abordaremos en este proyecto de investigación, la detección automática de paráfrasis.

2.1. Procesamiento del lenguaje natural

El estudio del lenguaje natural es sin duda una de las áreas de estudio que pueden ser vistas desde distintos enfoques académicos. El Procesamiento del Lenguaje Natural (PLN) es una línea de investigación de las ciencias de la computación que estudia el lenguaje natural desde una perspectiva computacional y que busca proveer a las computadoras de habilidades referentes al entendimiento, procesamiento y análisis del lenguaje natural. Es decir, el PLN es un puente entre el lenguaje natural y las computadoras en beneficio de la comprensión del lenguaje [9]. Hablando en términos generales, PLN busca crear modelos de IA que sean capaces de procesar y generar lenguaje. Esencialmente se intenta modelar el lenguaje natural desde un enfoque computacional intentando abstraer características propias e inherentes del lenguaje. La complejidad, variedad y ambigüedad son ejemplos de características del lenguaje que cada modelo propuesto busca incluir.
Las áreas de Inteligencia Artificial y Aprendizaje Maquinal (incluido el Aprendizaje Profundo, AP) en combinación con la lingüística, hacen de PLN una rama de investigación multidisciplinaria con muchos áreas de oportunidad para crear tecnologías relacionadas con el lenguaje. En la Figura 2.1 presentamos un diagrama de la relación del PLN y las áreas de investigación ya mencionadas.

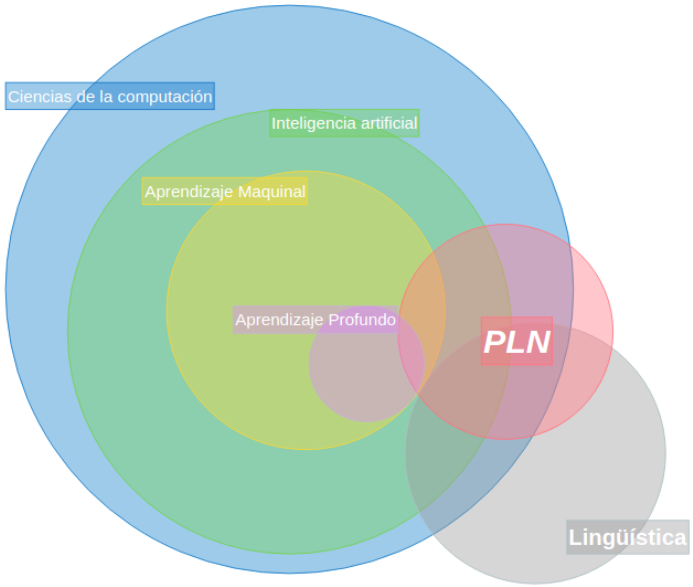


Figura 2.1: *PLN y su relación con otras áreas de investigación.*

2.1.1. Tareas de PLN

Dentro del PLN existen sub-áreas de investigación (o como se conocen en este contexto, tareas del lenguaje) que tratan un problema distinto del lenguaje. De entre las más populares podemos encontrar la generación automática de textos o expresiones discursivas, el etiquetado lingüístico automático, los traductores automáticos, codificadores de voz a texto y viceversa, etc [9]. Particularmente en este trabajo nos enfocamos en una tarea básica del lenguaje dentro del PLN conocida como *clasificación de textos* ya que, como veremos más adelante, el problema de detección de paráfrasis se puede plantear como un problema de clasificación dentro de ciencias de la computación.

2.1.2. Detección de paráfrasis

El problema a resolver tiene que ver con un caso particular de clasificación de textos, a saber: la detección de paráfrasis. Pero antes de todo es necesario saber qué es la paráfrasis. La paráfrasis puede ser definida de muchas formas. Y eso, en sí mismo, ya es paráfrasis. Aquí algunas definiciones de paráfrasis según la Real Academia de la Lengua (RAE):

- Traducción en verso en la cual se imita el original, sin verterlo con escrupulosa exactitud.
- Frase que, imitando en su estructura otra conocida, se formula con palabras diferentes.

En términos generales se dice que un texto es paráfrasis de otro texto si ambos tienen distinta forma pero aproximadamente el mismo significado semántico [3] o si ambos transmiten la misma información pero tienen distintas estructuras léxico-sintácticas [4]. De modo que el problema de la detección de paráfrasis consiste en decidir si dos textos, en principio distintos, podrían comunicar la misma información. En la Figura 2.2 se puede ver un diagrama de flujo del proceso de identificar si dos textos son paráfrasis uno del otro o no lo son.

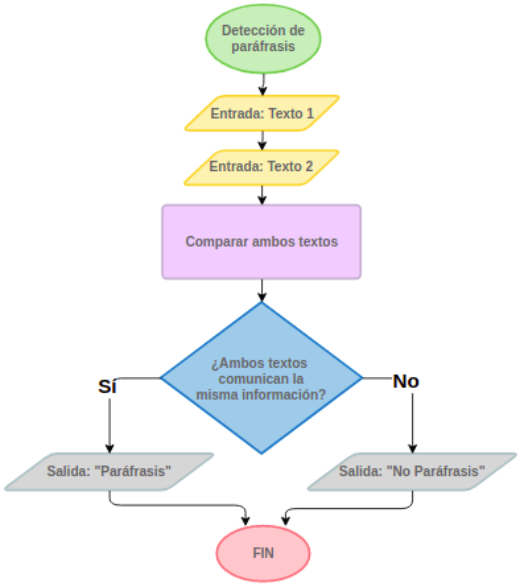


Figura 2.2: *Diagrama de flujo del proceso de identificar si dos textos son paráfrasis.*

La Figura 2.2 representa el planteamiento del problema desde una perspectiva algorítmica donde podemos observar componentes importantes como la entrada del algoritmo (dos textos), una métrica de comparación (el recuadro color violeta de la Figura) y un módulo de decisión. Componentes que atenderemos a lo largo de esta investigación.

2.1.3. Corpus de paráfrasis

Toda técnica o modelo de AM tienen un enfoque de aprendizaje guiado por datos, lo que genera una dependencia de calidad y cantidad de los datos para lograr buenos resultados. Es decir, mucho del desempeño de un modelo de AM depende de la calidad y cantidad de los datos que se utilizan para realizar el procese de entrenamiento, que como veremos más adelante en esta sección es una de las etapas más importantes en el proceso de creación de un modelo de AM. Un corpus en PLN es una recopilación o conjunto de textos, basado en distintos criterios, con el objetivo de realizar análisis lingüístico [4]. De modo que existen distintas formas de estructurar una base de datos y

recolectar datos para crear un corpus, por consiguiente existen distintos tipos de corpus, en este proyecto nos vamos a centrar en uno tipo de corpus en particular, el corpus de paráfrasis. Los corpus de paráfrasis tienen la característica de ser paralelos, es decir, están estructurados por pares y, adicionalmente, se incluye una etiqueta identificando si efectivamente ambos textos son paráfrasis o no lo son. En la Tabla 2.1 se pueden ver algunos ejemplos de textos que se podrían encontrar dentro de un corpus de paráfrasis.

Texto 1	**Texto 2**	**Etiqueta**
En junio de 1997, Kristin conoció a Richard Armstrong.	*En junio de 1997, Armstrong conoció a Kristin Richard.*	**Paráfrasis**
Ella comenzó con la práctica de enseñar y agrupar a estudiantes con el mismo problema de lenguaje.	*Ella comenzó a enseñar a los estudiantes con el mismo problema del habla y agruparlos.*	**Paráfrasis**
Se espera que el mercado de chips de EE. UU. Disminuya un 2.1 % este año y luego crezca un 15.7 % en 2004.	*El mercado americano disminuirá un 2.1 % a $30.6 mil millones en 2003, y luego crecerá un 15.7 % a $35,4 mil millones en 2004.*	**No Paráfrasis**

Tabla 2.1: Ejemplos de un corpus de paráfrasis

Observamos en la Tabla 2.1 que el criterio para crear un corpus de paráfrasis es tener una colección paralela de textos y su etiquetado correspondiente.
Las formas en que se recolectan los datos para crear un corpus de paráfrasis son variadas. Algunos corpus de paráfrasis se realizan por traducción, es decir, se toman traducciones de mismas expresiones lingüísticas y se traducen a un lenguaje común (generalmente al lenguaje de corpus que se quiera crear) debido a que se presupone que un segmento discursivo mantienen el mismo significado al traducirlos a una segunda lengua [5]. Otra forma de crear un corpus de paráfrasis es de forma manual, los corpus creados de esta forma se conocen como de *reescritura*. El proceso que se realiza comienza seleccionando un conjunto de textos los cuales se le dan a un grupo de personas (comúnmente conocidos como anotadores) [4]. Los anotadores deben tomar cada segmento discursivo y reescribirlo o parafrasearlo. Podemos entender que esta forma de crear un corpus de paráfrasis podría tomar mucho tiempo y la cantidad de textos que se pueden parafrasear es relativamente pequeña (dentro de un lapso de tiempo corto) si la comparamos con la cantidad de información que un modelo de AM podría necesitar para devolver buenos resultados.

2.2. Problemas de clasificación

En ciencias de la computación un *problema de clasificación* hace referencia a la tarea de asignar una clase (o etiqueta) a determinadas entidades o ejemplos dentro

del dominio del problema. Una entidad o ejemplo en este contexto es un "objeto" (abstraído del mundo real) representado mediante una serie finita de atributos. A su vez, la clase (o etiqueta) define una categoría a la que pertenece el ejemplo en función a los atributos que lo definen y siempre depende del problema que se resuelve y el contexto que lo envuelve. El problema de determinar si un correo electrónico se debe considerar como *Spam* o *No Spam* es uno de los ejemplos más representativos de un problema de clasificación. En la Figura 2.3 se representa el problema de clasificación mencionado.

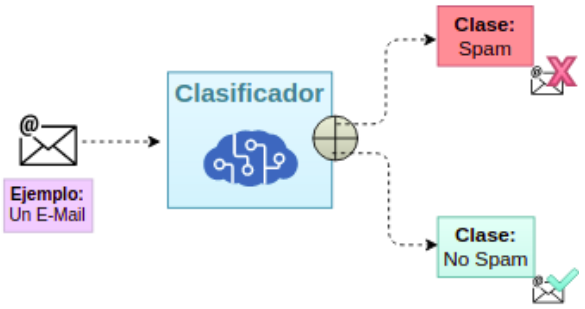


Figura 2.3: *Ejemplo de problema de clasificación: Clasificación de correos electrónicos.*

Los problemas de clasificación pueden ser categorizados en función del número de etiquetas que se desean predecir en un problema dado. Si el número de clases es dos, entonces se dice que es un problema de *clasificación binaria*. En otro caso, si el número de clases es mayor a dos, entonces se dice que es un problema de *clasificación multi-clase*.
En AM un *clasificador* es una entidad funcional que representa a la solución de un problema de clasificación. Existe una gran variedad de clasificadores distintos que pueden ser tipificados de acuerdo con distintos criterios dependiendo de, por ejemplo, el tipo de aprendizaje empleado (supervisado, no supervisado y por refuerzo) [9] o por la base conceptual con la que son definidos; en los que encontramos los bio-inspirados, basados en reglas de decisión, basados en modelos matemáticos o basados en modelos probabilísticos. Más adelante vamos a dar una breve explicación de algunos de los más relevantes en el contexto de nuestro proyecto.

2.3. Modelos de AM

Los recientes avances en el poder computacional y la creación de información digital, cada vez más extensa, han hecho posible la aplicación de técnicas de AM en una gran diversidad de problemas de casi cualquier área de estudio. Las técnicas de AM son cada vez más robustas, complejas y tomadas de una gran diversidad de conceptos teóricos. Recordemos que estas técnicas son las que generan modelos que aprenden a desempeñar

cierta tarea. Ya vimos que, por ejemplo, un clasificador aprende a predecir clases de ciertos ejemplos dentro de un dominio de un problema. Pero la base conceptual de cada modelo puedo variar. Existen modelos basados en teoría de la optimización, otros en teorías de probabilidad y otros inspirados en fenómenos propios de la naturaleza.
Al mismo tiempo, cada modelo de AM puede ser categorizado como *Aprendizaje supervisado* y *Aprendizaje no supervisado*. En el *Aprendizaje supervisado* se desarrollan modelos predictivos basados en datos de entrada previamente procesados y etiquetados. Los modelos de *Clasificación* y *Regresión* son los más representativos de esta categoría. En cambio, en el *Aprendizaje no supervisado* se crean modelos que tratan de encontrar todo tipo de patrones, a priori desconocidos, en datos que no han sido etiquetados. El problema de Agrupamiento o *Clustering* es de los más representativos de esta clase. En la Figura 2.4 se puede ver la división de los modelos de AM en función del tipo de aprendizaje y también, en los recuadros en la parte inferior, algunos de los algoritmos más relevantes dentro de cada categoría.

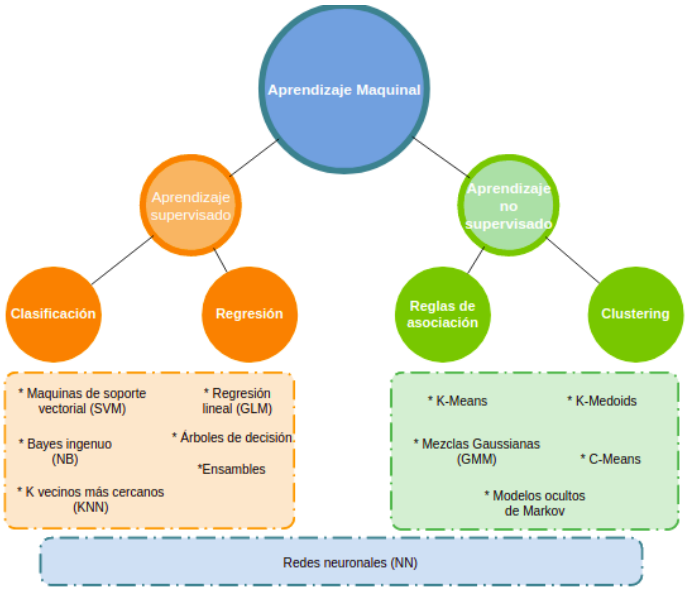


Figura 2.4: *Modelos de AM divididos en función del tipo de aprendizaje.*

La Figura 2.4 es solo representativa ya que muchos algoritmos pueden adaptarse a muchos problemas en general, por ejemplo, una red neuronal puede emplearse tanto a problemas de clasificación como a problemas de agrupamiento.

A continuación vamos a dar un breve bosquejo de las bases conceptuales de algunos de los modelos antes mencionados. Concretamente los modelos que están basados en teoría de la optimización, otros en teorías de probabilidad y otros bio-inspirados, en particular, los inspirados en el funcionamiento del cerebro humano, las redes neuronales.

2.3.1. Optimización

Muchos de estos modelos tratan de formular el problema en términos de una función de coste o aptitud que se busca (o se tiene como objetivo) minimizar o maximizar. La forma más usual para crear este tipo de modelos de AM es definiendo espacios vectoriales (o espacios de características) [9] que intentan representar (de la forma más conveniente posible) a los ejemplos dentro del dominio del problema (más adelante en esta sección vamos a revisar algunas de las técnicas más comunes para realizar la representación vectorial de las palabras y textos completos). Dicha representación sirve de medio para que las computadoras puedan trabajar con los ejemplos del problema y, además, ayude a simplificar desde un enfoque computacional la solución del problema. Una vez definido el espacio de características se define la función de costo o aptitud que generalmente se asocia a una métrica de distancia dentro del espacio vectorial y que, según las características propias dadas por el problema, se busca optimizar.
En este tipo de modelos el concepto de *distancia* toma gran importancia debido a que muchas veces esa es la métrica que se usa para comparar a cada entidad dentro del problema.

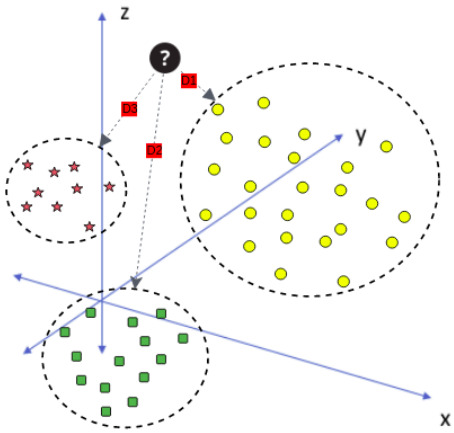


Figura 2.5: *La distancia en un espacio vectorial como un métrica para asignación de clase o grupo.*

En la Figura 2.5 se muestra un ejemplo de como la distancia puede ser un punto de referencia para, por ejemplo, asignar una clase a un ejemplo (marcado con un signo de interrogación en la Figura) si se considera la distancia más corta de entre las tres opciones y sus respectivas distancias: *D1*, *D2* y *D3* marcadas en rojo en la Figura 2.5. Se puede ver claramente que el ejemplo nuevo (señalado con un signo de interrogación en la Figura 2.5) puede tener más "*similitud*" con los ejemplos marcados con círculos amarillos, ya que al menos visualmente la distancia *D1* parece ser la menor. Por lo tanto es natural pensar que un clasificador basado en la optimización usando la métrica de distancia podría clasificar como *circulo amarillo* al ejemplo nuevo.

2.3.2. Probabilísticos

Las matemáticas son la base de la Inteligencia Artificial y las sub-áreas de las matemáticas, como el Álgebra lineal, la probabilidad y la estadística, pueden considerarse partes integrales del Aprendizaje Maquinal. Los modelos de AM basados en conceptos probabilísticos hacen un análisis estadístico cuyo objetivo es el de crear distribuciones de probabilidad que pueden predecir el comportamiento de cierto fenómeno dentro del problema al que se aplican. Una de las principales ventajas de los modelos probabilísticos es que proporcionan una idea sobre la incertidumbre asociada con las predicciones de forma nativa. En otras palabras, podemos tener una idea de la confianza que tiene la predicción del modelo creado gracias a todas la herramientas que la estadística tiene para dar estimaciones.
Muchos modelos dentro de PLN crean distribuciones de probabilidad basadas en el número de ocurrencias de una palabra dentro de un texto o analizan secuencias dentro de un contexto para poder predecir en otro momento una palabra dada una secuencia similar. La Figura 2.6 muestra una secuencia y como un modelo probabilístico seleccionaría la palabra que sigue.

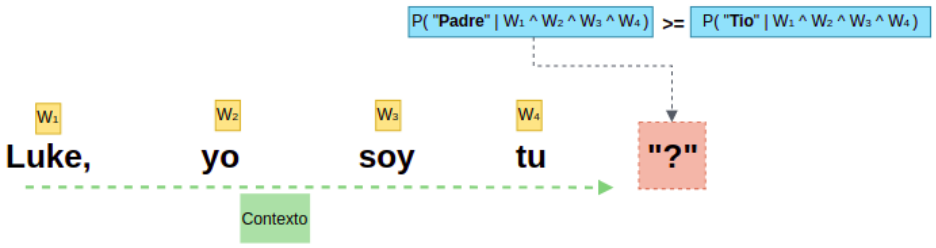


Figura 2.6: *Modelos probabilístico para predecir palabras dentro una secuencia.*

Podemos ver que se tiene la secuencia *"Luke, yo soy tu..."* y el modelo probabilístico intenta determinar cuál de las palabras: *"Padre"* o *"Tio"*, es la que debería seguir dada la secuencia de palabras anterior. Un modelo entrenado previamente podría haber recibido la frase completa *"Luke, yo soy tu padre"* y entonces la distribución de probabilidad generada tendría que resolver que la $P($*"Padre"* $\mid$ *"Luke, yo soy tu"*$)$ sea mayor que, por ejemplo, la $P($*"Tio"* $\mid$ *"Luke, yo soy tu"*$)$.

2.3.3. Redes neuronales

Como ya hemos mencionado las Redes Neuronales (RN) son una sub-área de estudio de AM. Generalmente la acción de aplicar modelos basados en redes neuronales a grandes cantidades de datos, hace referencia al aprendizaje profundo (Deep Learning, DL). El objetivo de hacerlo es principalmente por la necesidad de que la computadora aprenda a realizar una tarea en específico. Dicha tarea puede ser desde un problema de clasificación trivial, hasta un problema de razonamiento complejo. En este apartado

vamos a describir algunos aspectos básicos e importantes de los modelos de redes neuronales más conocidos dentro del área de DL.

2.3.3.1. Perceptrones multicapa

Una red neuronal básica, también conocida como *perceptrón multicapa*, se alimenta de datos a través de capas de *perceptrones* para producir una salida deseada. El *perceptrón* es el componente más básico de las redes neuronales, es un modelo simplificado de las neuronas biológicas de nuestro cerebro. En la Figura 2.7 se puede ver una representación gráfica del modelo del *perceptrón*.

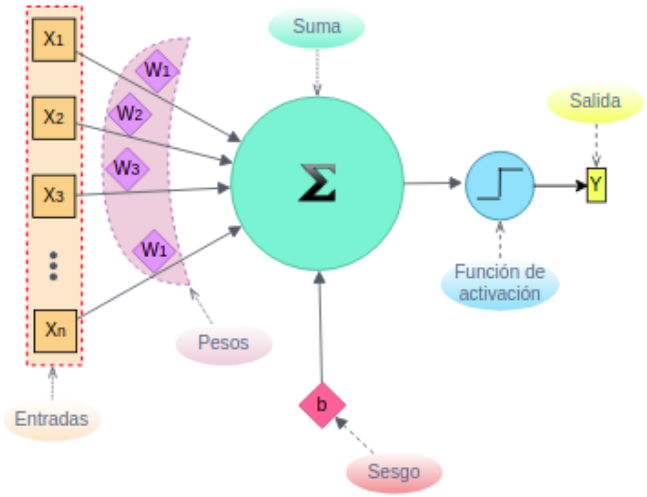


Figura 2.7: *Modelo de Perceptron.*

El *perceptrón* funciona de la siguiente forma. Al inicio se tienen valores de *entrada* (vector $\overrightarrow{X}$) y *pesos* de la red (vector $\overrightarrow{W}$). Luego, los vectores $\overrightarrow{X}$ y $\overrightarrow{W}$ se obtiene la suma ponderada $\sum_{i=1}^{n} w_i x_i$ y se suma también un valor conocido como *sesgo* (generalmente los pesos y sesgos de toda red neuronal son inicializados de forma aleatoria). Después, a la suma ponderada y al *sesgo* se le aplica una función de activación, misma que produce la salida del perceptrón. La *función de activación* desempeña el papel integral de garantizar que la salida que se asigne a los valores de entrada sean propios del problema.
Una red neuronal es una red inter-conectada de *perceptrones* distribuidos en distintas capas o niveles de conexión. Existen tres tipos de capas dentro de una red neuronal estándar, capa de entrada, capa oculta y capa de salida. La capa de entrada recibe los datos iniciales que se introducen a la red para su posterior procesamiento por capas ocultas de la red. La capa oculta es la capa entre la capa de entrada y la de salida, donde los *perceptrones* toman un conjunto de entradas y producen una salida a través

de una función de activación que a su vez es enviada a *perceptrones* posteriores dentro del mismo nivel. Y por último, la capa de salida es la última capa de neuronas que produce salidas esperadas por la definición del problema. La Figura 2.8 muestra la estructura básica de una red neuronal estándar con las capas mencionadas.

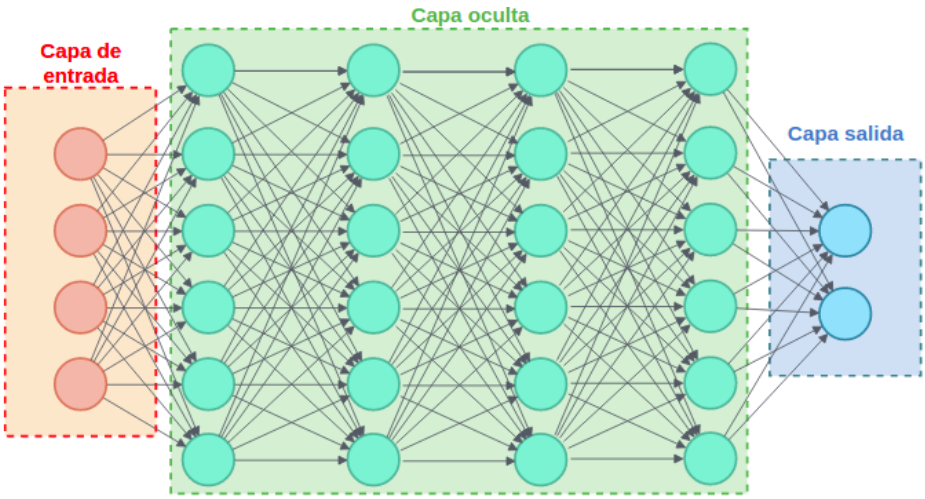


Figura 2.8: *Distribución de perceptrones en capas dentro de una red neuronal.*

Para la detección de paráfrasis se han aplicado distintas variantes de una red neuronal. Por una parte tenemos a las redes neuronales *convolucionales*, que son inspiradas en la corteza visual humana y que su nombre se deriva por la aplicación de la operación matemática *convolución* [9]. Por otra parte, se han creado algunas redes neuronales recurrentes, particularmente las de memoria a corto y largo plazo (Long-Short Term Memory LSTM). Esta clase de redes neuronales agregan el concepto de memoria haciendo que algunas salidas de neuronas vuelvan a algunas capas de la red neuronal de menor nivel [6]. Dichas redes difieren principalmente en la arquitectura y operaciones adicionales internas, pero se basan en un problema de optimización que se resuelve generalmente usando la técnica del *descenso del gradiente*.

Descenso del gradiente

Uno de los puntos de mayor importancia cuando se habla de redes neuronales es la técnica conocida como el *descenso del gradiente*. Básicamente el proceso de optimización de las redes neuronales se basa en el *descenso del gradiente*. Se trata de un algoritmo que intenta encontrar el mínimo de una función f empleando la derivada de dicha función como herramienta principal. El pseudocódigo del algoritmo del descenso del gradiente se puede ver en la Figura 2.9.

Algoritmo 1: Descenso del gradiente.

ta ← *tasa de aprendizaje*;
iter ← *número de iteraciones*;
$W \leftarrow$ *un punto aleatorio en el dominio de* f;
para $i \leftarrow 0$ ***hasta*** *iter* **hacer**
  $Gradiente \leftarrow \nabla(f) \leftarrow \frac{\partial f}{\partial x}$;
  $W \leftarrow W - ta * Gradiente$;
fin

Figura 2.9: Pseudocódigo del algoritmos del descenso del gradiente.

Al iniciar el algoritmo se establecen tres parámetros; una *tasa de aprendizaje*, el número de veces que vamos a iterar el algoritmo y un punto, dentro del dominio de la función, de forma aleatoria. Suponga que W es un punto aleatorio en el dominio de la función, entonces por cada iteración se calcula el gradiente de la función f en el punto W para después multiplicarlo por la *tasa de aprendizaje ta* y se resta a W. Básicamente lo que se hace en cada iteración es provocar una perturbación en el punto W que hace que se mueva hacia una región donde el movimiento de la función es descendente y, eventualmente, se llegará al mínimo de la función. Veamos un ejemplo, en la Figura 2.10 se muestra una función f y queremos encontrar su valor mínimo.

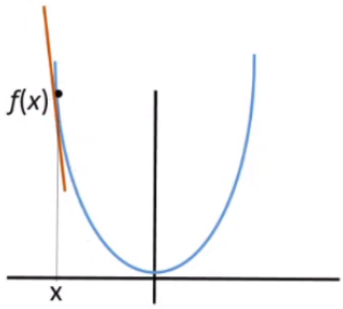


Figura 2.10: Función f

Por ejemplo, sea la función $f(x) = x^2$ y suponga que elegimos el valor inicial $W = -1$. La derivada de la función f entonces sería $f'(x) = 2x$ y por lo tanto el gradiente de la función en el punto inicial W sería $f'(-1) = -2$. Al aplicar la primer iteración del algoritmo tendríamos que actualizar W aplicando $W \leftarrow W - ta * \nabla(f(W))$. Supongamos que tenemos una *tasa de aprendizaje* $ta = 0{,}001$, entonces tendríamos que: $W = (-1) - (0{,}001)(-2) = -1 + 0{,}02 = -0{,}98$. Podemos verificar que $f(-0{,}98) < f(-1)$ y por lo tanto la perturbación que se le ha hecho a W en la primera iteración ha reducido el valor de f. El proceso continua hasta alcanzar el número de iteraciones indicado.

Si la red se necesita ajustar para resolver un problema más complejo, se necesita un mecanismo adicional que regule y auto-ajuste el vector $\overrightarrow{W}$ de cada *perceptrón* dentro de la red. Este mecanismo se conoce como *retropropagación del error*.

Retropropagación

El algoritmo de retropropagación del error o *Backpropagation* permite a una red neuronal auto-ajustar sus parámetros, en particular los pesos y sesgos mostrados en la Figura 2.7 (que recordemos, generalmente son inicializados de forma aleatoria al inicio del entrenamiento de la red neuronal). Este algoritmo hace que las redes neuronales puedan auto-calibrase para que las predicciones (o salidas) vayan mejorando. Para entender mejor cómo funciona el algoritmo de retropropagación vamos a usar un ejemplo, considere la red neuronal mostrada en la Figura 2.11. Podemos ver que para la capa de entrada tenemos dos neuronas, para la capa oculta también dos neuronas y para la capa de salida una sola neurona. La red neuronal recibe una vector X como entrada y, en la capa de salida, devuelve una predicción P. A esa predicción le asociamos un error, el error de la predicción P refleja la diferencia del valor esperado dado el vector X y la predicción.

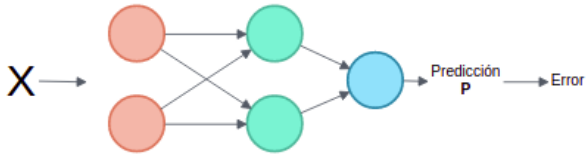


Figura 2.11: Red neuronal de ejemplo.

Hablando en términos generales, el objetivo del algoritmo de retropropagación es de ajustar los pesos y sesgos de las neuronas anteriores (en la capa oculta) hasta que el error, en la capa de salida, sea el mínimo. Es decir, hasta que la red neuronal completa mejore las predicciónes que realiza.
Suponga que la red neuronal anterior tiene el estado mostrado en la Figura 2.12.

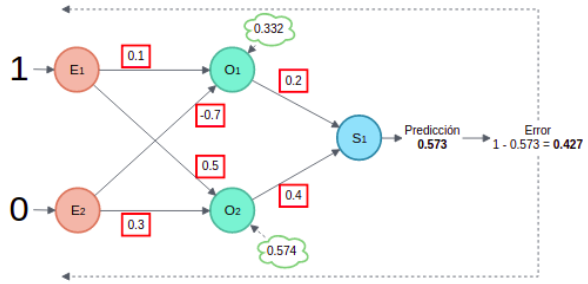


Figura 2.12: Estado de red neuronal ejemplo.

La red neuronal toma como entrada el vector $(1, 0)$ y quiere predecir el valor 1. Los cuadros rojos con un valor dentro representan a los pesos en el estado actual de la red neuronal y los valores dentro de las nubes en las neuronas de la capa oculta O_1 y O_2 representan a las sumas ponderadas. Podemos ver que dado esos valores la predicción de la red neuronal en la capa de salida con la neurona S_1 es de 0,573 lo cual da un error asociado a la predicción de 0,427. El algoritmo de retropropagación buscaría actualizar los valores en los cuadros rojos de tal forma que el error se reduzca. Es decir, que eventualmente la salida de la red sea próxima al valor 1 para cuando recibe como entrada el vector$(1, 0)$ (dado nuestro ejemplo).

2.3.3.2. Redes neuronales convolucionales

Las redes neuronales convolucionales (*Convolutional Neural Network*) también llamadas *CNN's* o *ConvNets* son algoritmos basados en el aprendizaje profundo que han mostrado gran desempeño sobre todo en el procesamiento de imágenes [27]. En términos generales estos modelos reciben un arreglo de dos dimensiones M donde comúnmente este arreglo representa a una imagen y cada entrada $(i, j) \in M$ corresponde al valor de un pixel. El objetivo principal es el de asignar una categoría al contenido de la imagen. Para entender cómo operan estas redes usaremos un ejemplo, suponga que queremos usar una red CNN para determinar si una imagen contiene al operador + ("*suma*") o ÷ ("*división*") como mostramos en la Figura 2.13.

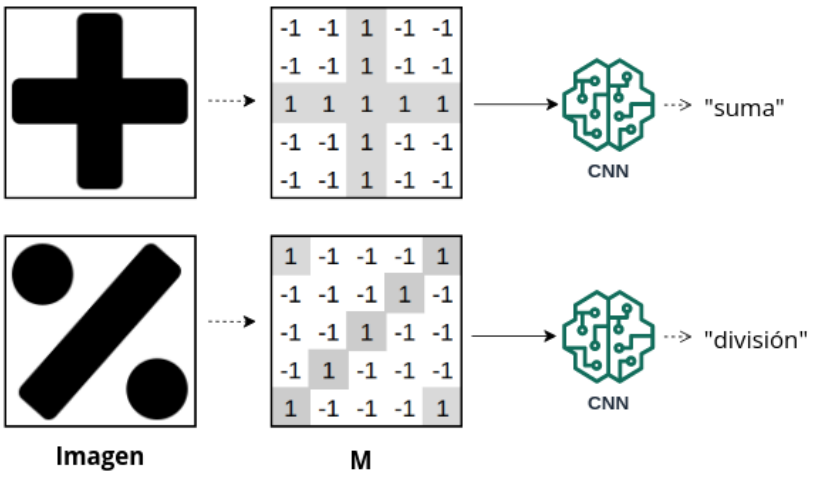


Figura 2.13: Problema ejemplo aplicando una red convolucional.

Como podemos ver, la imagen la convertimos a una matriz $M_{5\times5}$ donde un pixel de color blanco recibe el valor de -1 y uno de color negro el valor de 1. Estas redes buscan patrones en submatrices de la matriz original M que sean útiles para determinar si una nueva matriz M' (con un patrón distinto) es similar a la matriz con el símbolo *suma* (M_{sum}) o *división* (M_{div}). En la Figura 2.14 podemos ver un ejemplo de submatrices

de M y M' de ambos caso (*suma* y *división*) con patrones que la red podría seleccionar y comparar.

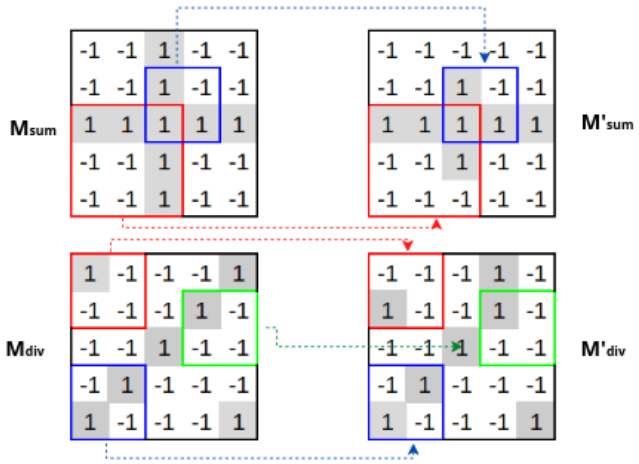


Figura 2.14: Ejemplo de submatrices con patrones a identificar.

Por ejemplo, podemos ver en la Figura 2.14 que entre las matrices M_{div} y M'_{div}, la submatriz marcada en azul presenta un patron similar en la diagonal. Al encontrar este tipo de patrones, la red puede inclinarse a determinar que la matriz M'_{div} contiene el símbolo $\div$ y entonces asignar la clase "*división*". Sin embargo, es necesario ponderar las similitudes sobre todas las posibles combinaciones en donde podemos sobreponer la submatriz azul sobre la matriz M' ya que en principio, la red no sabe exactamente donde se encontraran las coincidencias. Cada combinación de submatriz con un patrón dado, que sirve para comparar las matrices M y M' recibe el nombre de *filtro* y la operación matemática que se aplica entre el filtro y el área de la matriz M' (donde se sobrepone el filtro) se conoce como convolución. En la Figura 2.15 mostramos la aplicación de un filtro sobre 4 posibles regiones de la matriz M'_{div}.

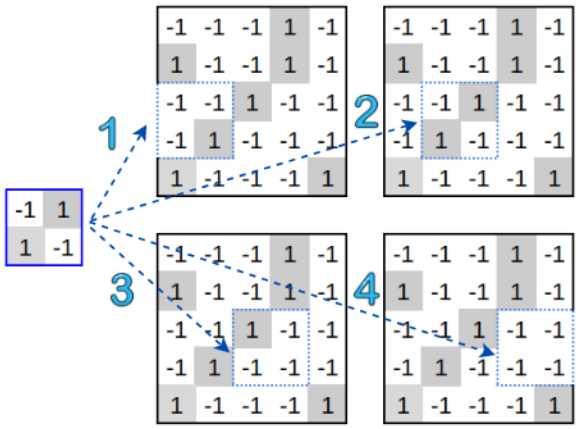


Figura 2.15: Filtro ejemplo y la aplicación a cuatro submatrices de M'.

Para calcular la similitud entre el filtro y la submatriz de M' multiplicamos cada entrada $(i, j) \in filtro$ con el correspondiente $(i, j) \in submatriz \in M'$, lo sumamos todo y lo dividimos entre el número total de entradas (i, j). Observemos que para nuestro ejemplo, si el valor de ambos pixeles en (i, j) es 1, entonces $1 \times 1 = 1$. En otro caso, si el valor es -1, entonces $-1 \times -1 = 1$. Lo que indica que cada coincidencia suma 1 al valor de similitud y, en cambio, cualquier diferencia resta 1 al valor de similitud ya que $1 \times -1 = -1 \times 1 = -1$.
El resultado de comparar un filtro a cada submatriz de M' se almacena en otra matriz M'_F. En la Figura 2.16 mostramos el resultado que obtendríamos si comparamos el filtro de la Figura 2.15 con nuestra matriz ejemplo M'_{div}.

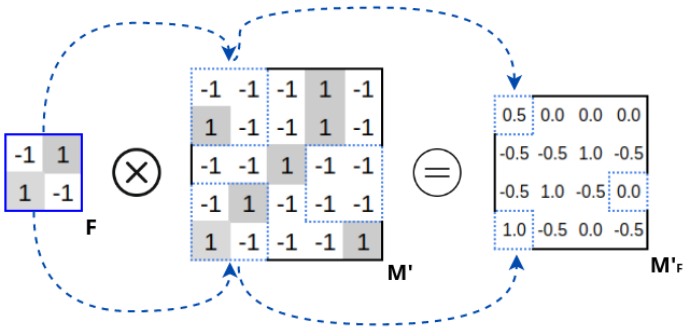


Figura 2.16: Resultado de comparar un filtro F con la matriz M' de ejemplo.

En la matriz resultante M'_F, los valores cercanos a 1 indican una coincidencia perfecta entre el filtro F y la matriz M', valores cercanos a 0 indican que no hay ninguna coincidencia y valores cercanos a -1 indican casos inversos significativos.
Hasta este punto solo hemos descrito un proceso interno de la operación de convolución

completo. Para completar la operación de convolución debemos aplicar cada filtro mostrado en la Figura 2.14 a la matriz M'. Es decir, que el resultado de aplicar la operación de convolución con k filtros es una colección de k matrices con las similitudes de los k filtros. En nuestro ejemplo $k = 3$ (por fines prácticos solo tomamos 3 filtros), por lo tanto el resultado de aplicar una operación de convolución sobre la matriz M' de nuestro ejemplo sería el siguiente (Figura 2.17):

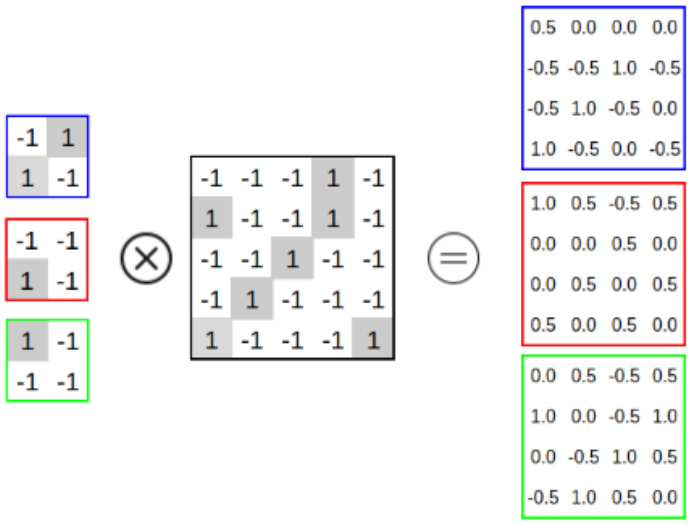


Figura 2.17: Resultado de una convolución completa con 3 filtros.

Selección de filtros

Hemos seleccionado algunos filtros para ejemplificar el proceso convolucional en las redes *CNN*. Sin embargo, la selección de los filtros en cada problema en realidad se lleva a cabo en el proceso de entrenamiento de la misma red. Es decir, la red ajusta los valores de los filtros, usando principalmente la retropropagación del error (2.3.3.1), de tal forma que la red asigne de forma correcta la clase correspondiente a cada muestra en el conjunto de entrenamiento que se le pase a la red. Esto es, cada muestra del conjunto de entrenamiento se pasa por la red y esta genera una salida que indica la clase que la red le ha asignado. El error asociado en dicha salida indica qué tan buenos son los filtros que la red ha ajustado. La red en todo el entrenamiento ajusta los filtros para que, en promedio de todas la muestras que pasan por la red, el error sea el mínimo. Asi las particularidades de una sola imagen se olvidan rápidamente, pero los patrones que ocurren en muchas imágenes se integran en los filtros y la red los utiliza para dar predicciones más precisas.

Pooling

La convolución generalmente va acompañado de otras herramientas para tratar los datos dentro de la red. El *pooling* es una de estas herramientas la cual consiste en tomar matrices grandes y reducirlas de tal forma que se garantice conservar la información más importante en ellas. Esto atendiendo dos aspectos importantes, el primero es la complejidad computacional de la red y el segundo, la reducción de redundancia en información dentro de la matriz. El *pooling* consisten en pasar una pequeña ventana a través de una matriz y tomar el valor máximo (*Max-Pooling*) o promedio (*Average-Pooling*) en el rango de los valores dentro de la región definida por la ventana en la matriz. Retomando nuestro ejemplo, en la Figura 2.18 mostramos la matriz que obtendríamos si aplicamos *Max-Pooling* y *Average-Pooling* a la salida de la convolución mostrada en 2.17 utilizando una ventada de tamaño 2×2.

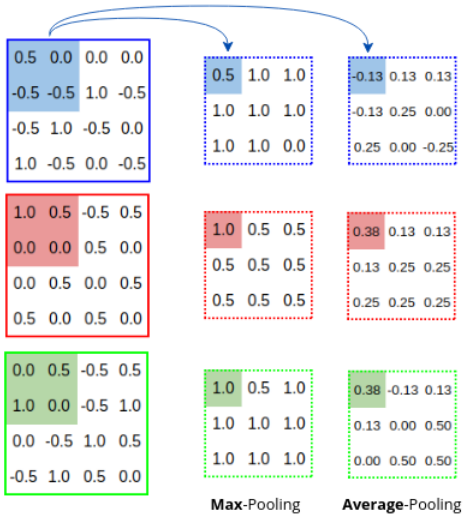


Figura 2.18: *Max-Pooling* y *Average-Pooling* con una ventana de 2×2.

Observemos que el tamaño de la ventana define la reducción en dimensión que se genera, en la práctica una ventana cuadrada de 2, 3 o 4 funcionan bien [27]. Esta operación se puede aplicar a la matriz de entrada de la red (que corresponde a la imagen) o de igual forma se puede aplicar a todas las matrices que se obtienen de la convolución. Por ejemplo, las mostradas en la Figura 2.17.

Salida

En la parte final de una red convolucional se encuentra una perceptrón simple que toma como entrada las matrices generadas por la convolución o el *pooling* pero en lugar de tratar las entradas como una matriz bidimensional, se tratan como una sola lista y todas se tratan de forma idéntica. El perceptrón en función de los valores ingresados, ajusta sus parámetros para dar la predicción final (ver Figura 2.19).

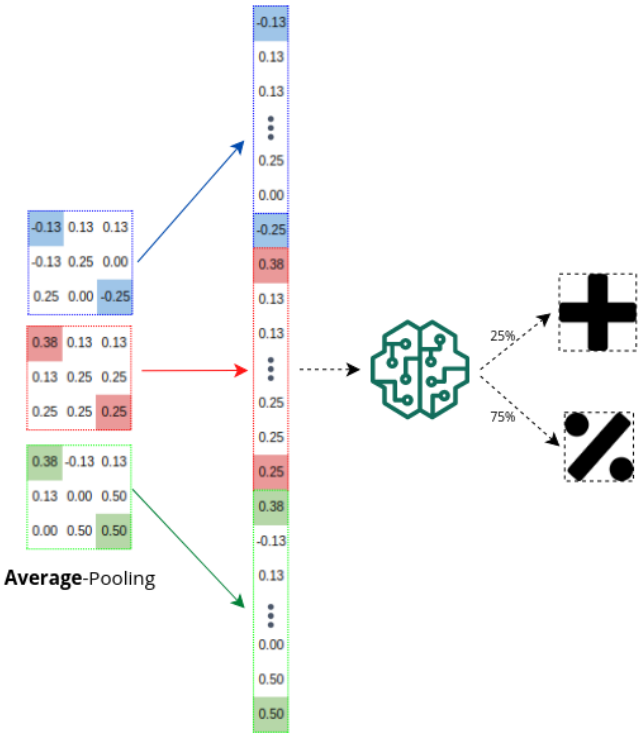


Figura 2.19: Salida de una red convolucional.

2.3.3.3. Redes neuronales recurrentes y *LSTM*

La redes neuronales recurrentes (*RNN's*) también son algoritmos basados en el aprendizaje profundo especialmente orientados a tratar con datos ligados por medio de un tipo de secuencia. Se han aplicado estos algoritmos en tareas como reconocimiento y síntesis de voz, procesamiento de texto, predicciones de índices bursátiles, etc [28]; mostrando buenos resultados sobre cada una de las tarea en las que se han aplicado. Sin embargo, las redes recurrentes sufren de un problema conocido como memoria a corto plazo (*short-term memory*) [29]. El cual se presenta cuando la secuencia de datos es muy larga y a la red le es dificil recordar información importante. Por ejemplo, si la red está procesando texto, la secuencia de datos podría ser la concatenación de palabras dentro de un párrafo. Y si el párrafo es muy largo, las redes recurrentes podrían ignorar información importante contenida al inicio del párrafo.
Las redes neuronales *LSTM* (*Long Short-Term Memory*) fueron creados como una solución al problema de memoria a corto plazo de las redes recurrentes. Para entender cómo operan las redes *LSTM* es necesario comprender la estructura y operación básica de las redes recurrentes. Usaremos el siguiente problema como ejemplo, suponga que queremos predecir el siguiente movimiento (T_4) de un objeto que se mueve en un plano de dos dimensiones como mostramos en la Figura 2.20.

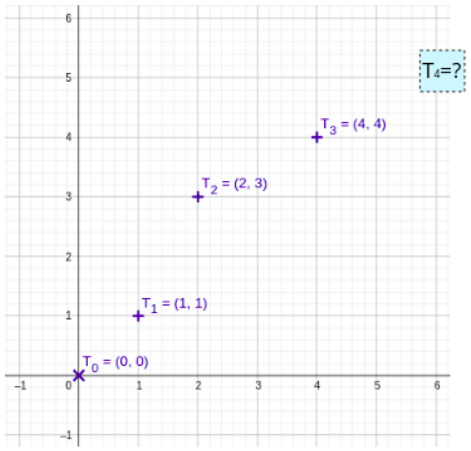


Figura 2.20: Problema ejemplo para la comprensión de las redes recurrentes y *LSTM*.

Evidentemente el movimiento del objeto en el plano genera una secuencia de datos que las redes recurrentes pueden procesar para predecir el siguiente movimiento. En la Figura 2.21 mostramos cómo una red recurrente procesaría esta información.

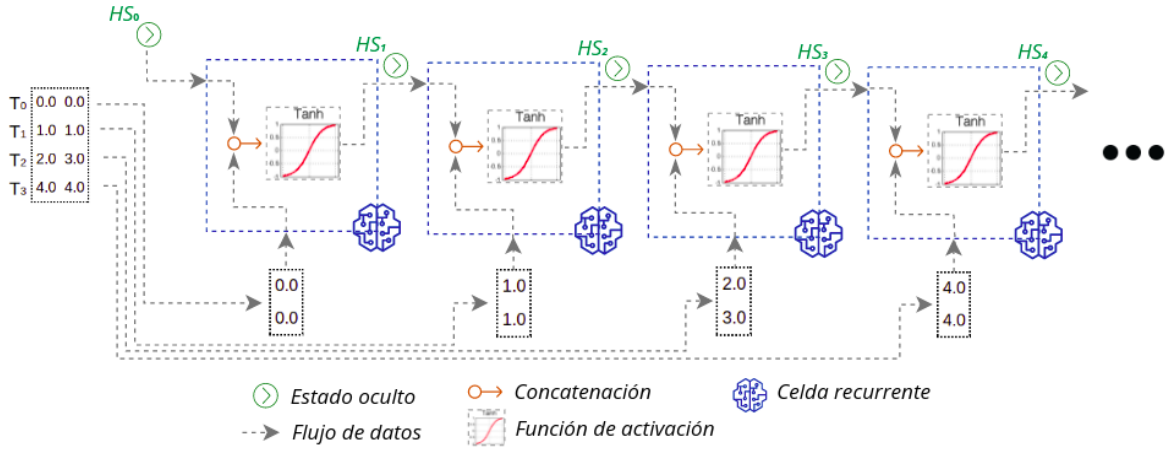


Figura 2.21: Secuencia de procesamiento de una red recurrente.

Las redes recurrentes procesan toda la información en pequeñas unidades conocidas como celdas recurrentes o *time-stamps* que reciben como entrada dos parámetros, el primero es información de la celda anterior inmediata que recibe el nombre de estado oculto (*Hidden State, HS*) y el segundo es la entrada de datos del problema. En nuestro ejemplo la red define cuatro celdas recurrentes y cada una recibe como entrada las coordenadas del objeto a un tiempo T_i dado y el estado oculto correspondiente como mostramos en la Figura 2.21. Cada celda recurrente procesa los datos ingresados y genera como salida el siguiente estado oculto. Para generar el estado oculto, primero el

vector de entrada y el estado oculto anterior se combinan para formar un solo vector, este nuevo vector se pasa por la función de activación y la salida es el nuevo estado oculto (como podemos observar en la Figura 2.21). El estado oculto actúa como la memoria de las redes neuronales ya que contiene información sobre datos anteriores que la red en una celda nueva no conoce.
Generalmente dentro de una red recurrente se utiliza la función de activación *TanH* debido a que el flujo de datos en secuencias largas y las operaciones aritméticas internas puede generar valores muy grandes o muy pequeños [29]. La función de activación *TanH* asegura que los valores permanezcan en un rango de -1 y 1, regulando así el flujo de información. Y si bien, las redes recurrentes tienen muy pocas operaciones internas, su desempeño es bueno solo para secuencias de datos cortas ya que el concepto de memoria solo es aplicado sobre celdas recurrentes inmediatamente conectadas.

Memoria a largo plazo

Como ya mencionamos las redes *LSTM* intentan solucionar el problema de memoria corta de las redes recurrentes. Las redes *LSTM* procesan la información de forma similar como mostramos en la Figura 2.21 pero las operaciones internas en una celda *LSTM* son distintas y más complejas. En la Figura 2.22 mostramos el diagrama interno de una celda *LSTM*.

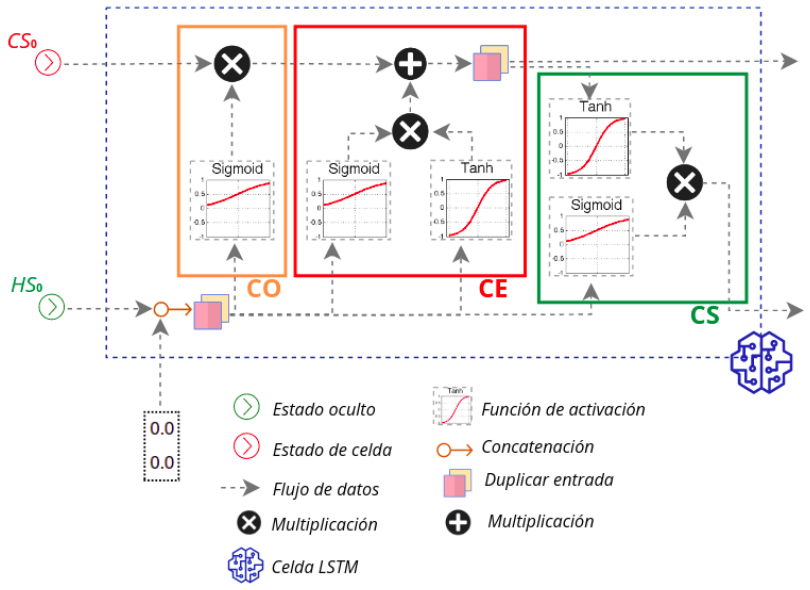


Figura 2.22: Estructura interna de una celda *LSTM*.

El concepto de memoria a largo plazo se define con la ayuda de una entrada de datos adicional a cada celda conocida como estado de celda (*Cel-State, CS*) y con la creación de compuertas de datos dentro de la celda *LSTM* conocidas como: compuerta de olvido

(*Forget Gate*), compuerta de entrada (*Input Gate*) y compuerta de salida (*Output Gate*) señaladas como *CO, CE* y *CS* correspondientemente en la Figura 2.22.
El estado de celda transporta información relevante a lo largo de la secuencia de celdas *LSTM*. Se conoce como la memoria de la red ya que incluso la información en celdas iniciales tiene gran relevancia en toda la secuencia de celdas, por muy larga que esta sea. Lo anterior se logra con el uso de las compuertas debido a que las compuertas son diferentes redes neuronales que deciden qué información se debe mantener y cual rechazar en toda la secuencia de celdas *LSTM*. Es decir, las compuertas aprenden qué información es relevante conservar u olvidar para que la red al final de la secuencia pronostique la salida correcta.

Compuerta de olvido

Esta compuerta recibe como entrada el vector del estado de celda de la celda inmediata anterior CS_{i-1} y la concatenación entre el vector T_i (entrada de la celda actual) y el vector del estado oculto de la celda inmediata anterior HS_{i-1}. La información del estado oculto anterior y la información de la entrada actual se pasan a través de la función de activación *Sigmoid* dado que es útil para actualizar u olvidar datos porque la salida de esta función solo toma valores de entre 0 y 1. Es decir, se interpreta el valor de 0 para "olvidar" y el valor de 1 para "recordar" información que se le pasa ya que la salida de esta función de activación se multiplica por el valor de la celda del estado anterior CS_{i-1}. Ver recuadro *CO* de la Figura 2.22. De esta forma la compuerta de olvido le ha indicado a la celda de estado los valores que debe olvidar o recordar usando la información del estado oculto anterior HS_{i-1} y la nueva información T_i.

Compuerta de entrada

La compuerta de entrada aplica dos funciones de activación a los vectores HS_{i-1} y T_i. La primer función de activación *Sigmoid* tiene el mismo enfoque que la compuerta anterior (olvidar información) y la segunda función de activación *TanH* se emplea para regular los datos de la red (como ya hemos mencionado). Sin embargo, la función principal de esta compuerta es multiplicar la salida de estas dos funciones de activación y sumarla a la información que ya tiene la celda de estado CS_{i-1} después de salir de la compuerta anterior. Lo anterior para generar nuevos valores que la compuerta de entrada considera relevantes y así generar la nueva celda de estado CS_i que recibirá la siguiente celda *LSTM* de la secuencia.

Compuerta de salida

La compuerta de salida genera el vector que debe ser el nuevo estado oculto HS_i. Primero pasamos el estado oculto anterior HS_{i-1} por la función de activación *Sigmoid* y el valor actual de la celda de estado CS_i por la función de activación *TanH*. La salida

de ambas funciones se multiplica para decidir qué información debe llevar el nuevo estado oculto que será enviado a la siguiente celda *LSTM* de la secuencia.

En resumen, la compuerta de olvido decide qué información es necesario olvidar o recordar de la celda *LSTM* anterior, la compuerta de entrada decide qué información actual es relevante agregar a la memoria y por último la compuerta de salida determina la información del siguiente estado oculto.

Secuencia de procesamiento *LSTM*

En la Figura 2.23 mostramos cómo sería la secuencia de procesamiento de datos de la red *LSTM* con el ejemplo propuesto.

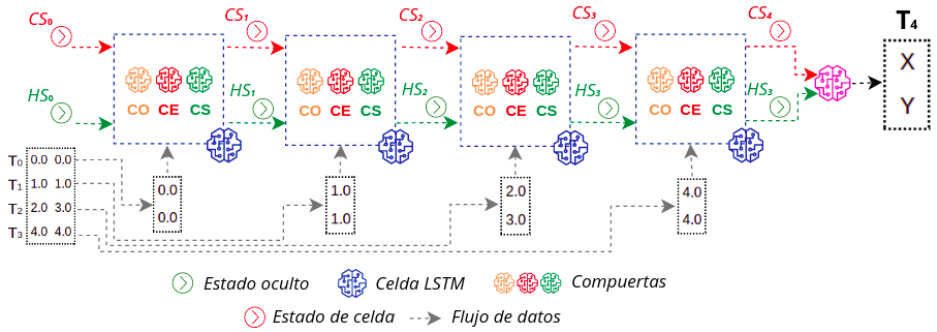


Figura 2.23: Secuencia de procesamiento de la red *LSTM*.

Observemos que la secuencia es similar a la de las redes recurrentes. En esta caso cada celda *LSTM* recibe la posición del objeto a un tiempo dado T_i e internamente ajusta el estado de celda y estado oculto para enviarlo a la siguiente celda. Sin embargo, internamente el proceso de entrenamiento es completamente distinto ya que en este caso la compuertas deben ajustarse para aprender la información que debe olvidar y recordar. Este proceso se realiza con la ayuda del error asociado a la predicción final (T_4 en nuestro ejemplo) y la retropropagación del error.

2.3.4. Ensambles y mezcla de expertos

Los ensambles son técnicas de AM que se refieren a la acción de usar múltiples modelos y combinar los resultados obtenidos de cada modelo para dar una predicción global. El objetivo principal de un ensamble es mejorar el desempeño y predicciones de un solo modelo (reducir el error predictivo). Recordemos que ningún modelo es perfecto para todo tipo de problemas y por lo tanto la combinación de conocimiento y estrategias de múltiples modelos en uno solo puede ser un medio para mejorar el desempeño general).

El error de un modelo de AM puede dividirse en dos partes, el error causado por sesgo (*bias-error*) y el error causado por varianza (*variance-error*). Ambos juegan un papel importante en el desempeño de un modelo. El error por sesgo se refiere a la inhabilidad que presenta un modelo en aprender patrones entre los datos y las clases o salidas deseadas; y el error por varianza hace referencia a la inhabilidad de un modelo de generalizar conocimiento y desempeñarse bien sobre datos nuevo. Un modelo con alto error de sesgo simplifica los patrones en los datos y se dice que está sub-ajustado (*underfitting*). Por otra parte, un modelo que ha aprendido mucho sobre los datos de entrenamiento supone un alto error de varianza y se dice que está sobre-ajustado (*overfitting*).
En la práctica se desea obtener modelos con bajo error de sesgo y varianza pero este escenario es muy dificil de lograr ya que, por ejemplo, incrementando la complejidad de un modelo se reduce el error de sesgo pero incremente el error de varianza, y reduciendo la complejidad de un modelo también se reduce el error de varianza pero se incrementa el error de sesgo. Este fenómeno se conoce como el umbral sesgo-varianza (*bias–variance trade-off*) [30] y los ensambles resultan ser técnicas adecuadas para reducir ambos errores y lograr un equilibro óptimo entre estos.
En este apartado describiremos algunas de las técnicas de ensamble más comunes y el impacto que tienen en la reducción de los errores sesgo-varianza.

2.3.4.1. Bagging

Esta técnica es un tipo de ensamble paralelo y está enfocado principalmente a reducir el error de varianza. Consiste en entrenar k submodelos de forma independiente usando datos de entrenamiento distintos como mostramos en la Figura 2.24.

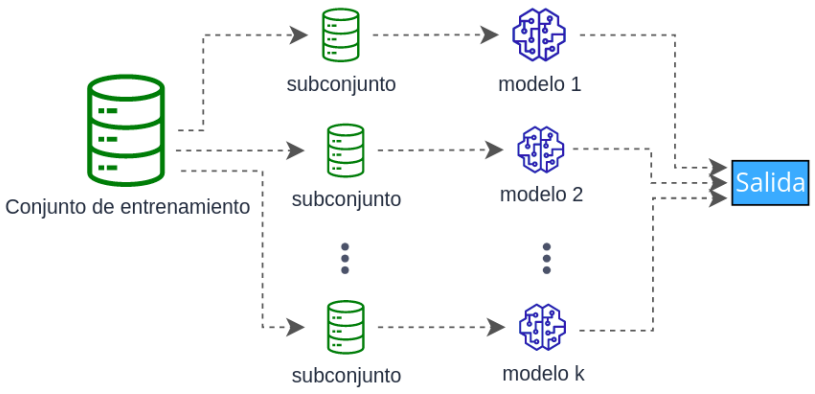


Figura 2.24: *Bagging.*

Este ensamble puede combinar distintos modelos, algoritmos o incluso modelos con hiperparámetros distintos y la salida generarla con un promedio o criterio de voto entre las k predicciones de los submodelos obtenidos.

2.3.4.2. Boosting

Esta técnica de ensamble combina distintos modelos iterando sobre cada modelo y sucesivamente cada modelo es entrenado reforzando el aprendizaje sobre los errores que ha hecho el modelo anterior para al final tener un modelo con mejor desempeño que cualquiera de los modelos anteriores dentro del ensamble. Es por lo anterior que esta técnica de ensamble de utiliza más para reducir el error por sesgo que por varianza. En la Figura 2.25 mostramos el proceso iterativo de *boosting* con 3 submodelos.

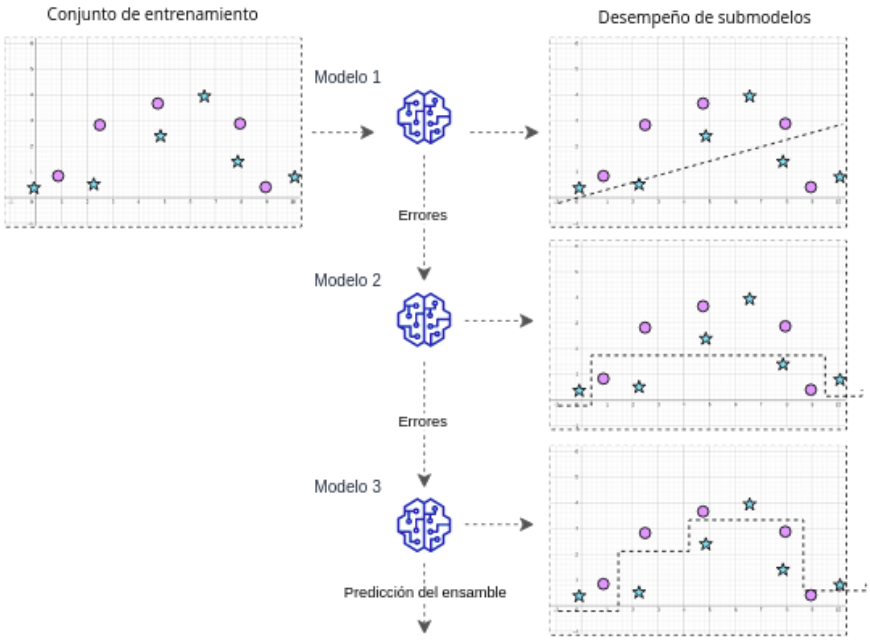


Figura 2.25: *Boosting.*

Podemos ver en la Figura 2.25 que el primer modelo dentro del ensamble tiene un desempeño malo, pero el siguiente mejora debido a que este se enfoca en los errores del anterior. Y en todo el proceso iterativo, las predicciones mejoran en la medida que los errores de un modelo tienden a cero. Por lo tanto, el ensamble resultante se convierte sucesivamente en un modelo más complejo, lo que explica que *boosting* es particularmente bueno reduciendo el error por sesgo.

2.3.4.3. Stacking o mezcla de expertos

Esta última técnica de ensamble puede pensarse como una extensión de *bagging* 2.3.4.1 ya que en lugar de promediar o aplicar un criterio de voto a la salida de los k modelos en *bagging*, *stacking* propone un esquema de pesos a cada modelo, ajustados por un modelo extra, para aprender a combinar las salidas de cada modelo dentro del ensamble y así mejorar el desempeño. Es decir, *stacking* usa las salidas

de los modelos en *bagging* como características para entrenar un *meta-modelo*. En la Figura 2.26 agregamos el modelo extra.

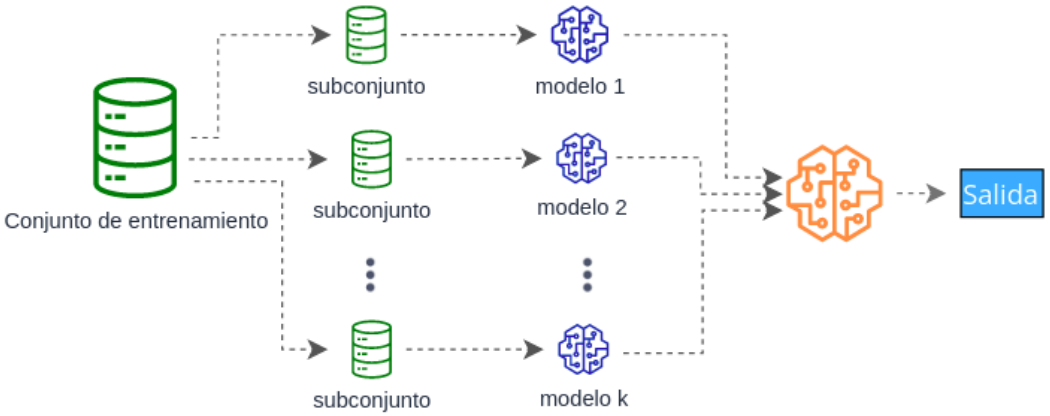


Figura 2.26: *Stacking.*

El agregar un modelo extra a la salida del ensamble proporciona todos los beneficios de reducir el error de varianza que vimos en *bagging*, pero adicionalmente aporta un mecanismo de manejo del error por sesgo entre todos los k modelos entrenados.

2.4. Representación de textos

Muchos de los modelos de AM no reciben como entrada el texto directamente. Hemos visto, muchos requieren de un vector de dimensión fija que represente de alguna forma una palabra, una oración o un segmento discursivo completo. En toda tarea de PLN el proceso de extracción de características para definir una espacio vectorial en el que sean representadas todas las entidades lingüísticas, es un proceso principal y muy importante. Hablando en términos generales, el proceso de representación (o también conocido como de codificación) de textos consiste en generar un correlación entre una expresión lingüística y un vector de dimensión fija. En la Figura 2.27 se puede ver un ejemplo de lo que se esperaría obtener si se representan, por ejemplo, textos de distintos contextos en un espacio vectorial de dimensión igual a tres.

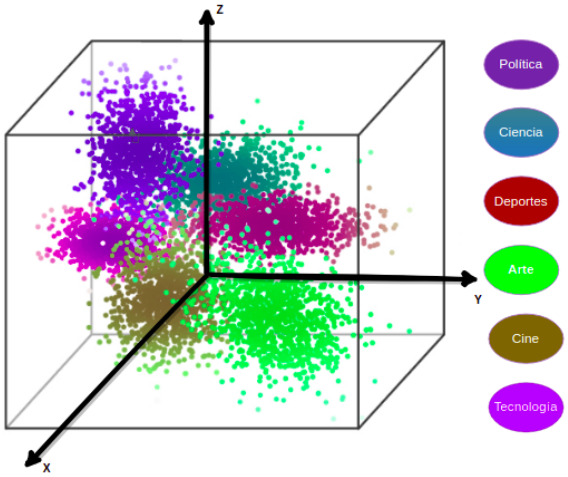


Figura 2.27: *Representación vectorial de textos en distintos contextos.*

El objetivo de la representación vectorial que vemos en la Figura 2.27 y que tomamos como ejemplo, es asignar un vector de dimensión tres a expresiones lingüísticas que están dentro de un mismo contexto. En concreto, definir vectores "similares" a palabras, oraciones o textos que hablen de un mismo tema: *Política, Ciencia, Deportes, Arte, Cine o Tecnología.*
Existen distintas técnicas para realizar el proceso de representación vectorial y algunas presentan unas ventajas sobre otras. Algunas son más sofisticadas que otras, pero el objetivo es el mismo. A continuación vamos a describir algunos procesos de codificación más importantes y relevantes dentro de PLN.

2.4.1. Vectores One-Hot

Esta técnica de codificación es la más sencilla pero no por eso pierde trascendencia. En esta codificación se crean vectores de dimensión igual a la del tamaño del vocabulario. Es decir, se crea una columna para cada palabra distinta dentro del vocabulario dentro del conjunto de textos que estamos codificando y, para cada registro o palabra, se marca con un 1 la columna a la que pertenezca dicha palabra y se dejan las demás columnas con el valor de 0. A continuación un ejemplo: suponga que se quieren codificar las siguientes frases de Juan Rulfo de su obra *"El llano en llamas"*.

- *"El tiempo es más pesado que la más pesada carga que puede soportar el hombre."*
- *"Y es que la alegría cansa. Por eso no me extrañó que aquello terminara."*

Si contamos el número de palabras distintas en los textos observamos que son 24. Entonces tendremos vectores de dimensión 24. En la Figura 2.28 se puede ver la codificación resultante de cada palabra dentro de los textos.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
el	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
tiempo	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
es	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
más	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pesado	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
que	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
la	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pesada	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
carga	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
que	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
puede	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
soportar	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
hombre	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
y	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
alegría	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
cansa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Por	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
eso	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
no	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
me	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
extrañó	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
que	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
aquello	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
terminara	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figura 2.28: *Vectores One-Hot.*

Esta forma de codificar palabras tiene dos desventajas principales, la primer desventaja es que la dimensión del vector que representa a cada palabra resulta ser muy alta si el tamaño del vocabulario es grande (aspecto deseable en toda colección de textos). Segundo, no hay una métrica significativa entre diferentes palabras sobre el espacio de características ya que toda palabra es igualmente distante una de la otra. Por ejemplo, palabras altamente relacionadas (y que por lo tanto esperaríamos estuvieran más cercanas) como *"pesado"* y *"cansa"* no están más cerca el uno del otro (en el espacio de representación anterior) lo que implica una pérdida de semántica entre las palabras.

Bolsa de palabras

Bolsa de palabras (Bag of Words, BoW) es una técnica de codificación de oraciones muy similar a la anterior pero en este enfoque se cuenta el número de veces que cada palabra aparece en el texto. Por lo tanto nos permite codificar oraciones completas. Cada texto se representa con un vector en el que cada entrada indica la ocurrencia de cada palabra y en el caso de no contener la palabra se pone el valor de 0. Es más sencillo verlo con un ejemplo: suponga que se quieren codificar los siguientes textos.

- *"El gato se sentó."*
- *"El gato se sentó en el sombrero."*
- *"El gato con el sombrero."*

Observemos que el vocabulario, el conjunto de todas las palabras en cada texto, es: {el, gato, se, con, sentó, en, sombrero}. El vector de cada oración lo generamos contando el número de veces que aparece cada palabra en la oración. En la Figura 2.29 mostramos el vector generado de cada oración que usamos como ejemplo.

	el	gato	se	con	sentó	en	sombrero
El gato se sentó	1	1	1	0	1	0	0
El gato se sentó en el sombrero	2	1	1	0	1	1	1
El gato con el sombrero	2	1	0	1	0	0	1

Figura 2.29: *Vectores BoW.*

2.4.2. Representaciones distribuidas

Existen otro tipo de técnicas más sofisticadas que usan estrategias de aprendizaje profundo para estimar la mejor forma de definir un vector que represente desde una palabra, una oración y hasta un documento completo. Estas técnicas aprenden relaciones contextuales entre palabras secuencialmente relacionadas de tal forma que el espacio vectorial definido las representa. A diferencia de las técnicas antes mencionadas como *BoW* o vectores *One-Hot*, en el que diferentes palabras tienen representaciones completamente diferentes independientemente del orden del texto, el aprendizaje de una representación distribuida aprovecha el uso de palabras en un mismo contexto para proporcionar representaciones similares a palabras que están íntimamente relacionadas de forma sintáctica.

Para el caso de codificación a nivel palabra se tienen dos esfuerzos principales en la literatura [10] y [11] comúnmente conocidos como modelos *Word2Vec* (palabra a vector). Estos modelos fijan la dimensión del espacio vectorial (que puede ser considerablemente más pequeño que las técnicas anteriores) y entrenan una red neuronal que recibe como entrada vectores *One-Hot* (en primera instancia) y obtienen una representación vectorial eficiente usando el conjunto de datos que se usa como entrenamiento. Típicamente se entrenan a partir de enormes conjuntos de textos con millones de palabras y con miles de palabras en el vocabulario [11].

Para el caso de codificaciones a un nivel mayor que palabras se tiene el proyecto [12] *Doc2Vec* (documento a vector) en el que se aprende a generar representaciones vectoriales distribuidas continuas para fragmentos de textos o textos completos. Es decir, los textos pueden ser de longitud variable, desde oraciones hasta documentos. El entrenamiento se realiza de forma similar a la del modelo anterior (*Word2Vec*) pero en *Doc2Vec* la entrada de la red neuronal es un vector formado por la concatenación de los vectores *One-Hot* de todas las palabras dentro del documento. La ventaja principal que se tiene con esta técnica es que el modelo es capaz de construir representaciones de secuencias textuales de longitud variable en vectores de dimensión fija. En otras palabras, a diferencia de algunos de los enfoques anteriores, esta técnica es general y aplicable a textos de cualquier extensión: oraciones, párrafos y documentos.

Más adelante, en la sección 3.2, vamos a dar un esbozo más detallado de como funcionan los modelos *Word2Vec* y *Doc2Vec* para realizar representaciones distribuidas de palabras o textos.

2.5. Minería de datos

La Minería de Datos (MD) es, en ciencias de la computación, un conjunto de técnicas, métodos y estándares que se aplican sobre grandes y complejas bases de datos con el objetivo principal de descubrir patrones en los datos que aporten conocimiento sobre un fenómeno en específico. Muchos de los esfuerzo de MD se hacen con el objetivo de automatizar el proceso de búsqueda de información predictiva basada en experiencias previas registradas en grandes cantidades de datos. Podemos suponer que la MD es una respuesta natural a la amplia disponibilidad de grandes cantidades de datos y la inminente necesidad de convertir dichos datos en información y, después, en conocimiento útil sobre un área en específico.
La acción de minar datos es un paso dentro de un proceso conocido como Descubrimiento de Conocimiento en Bases de Datos (*Knowledge Discovery in Databases, KDD*) que consta de una serie de pasos iterativos que describen una guía para poder obtener el conocimiento. En la Figura 2.30 se muestra un esquema del proceso que define KDD.

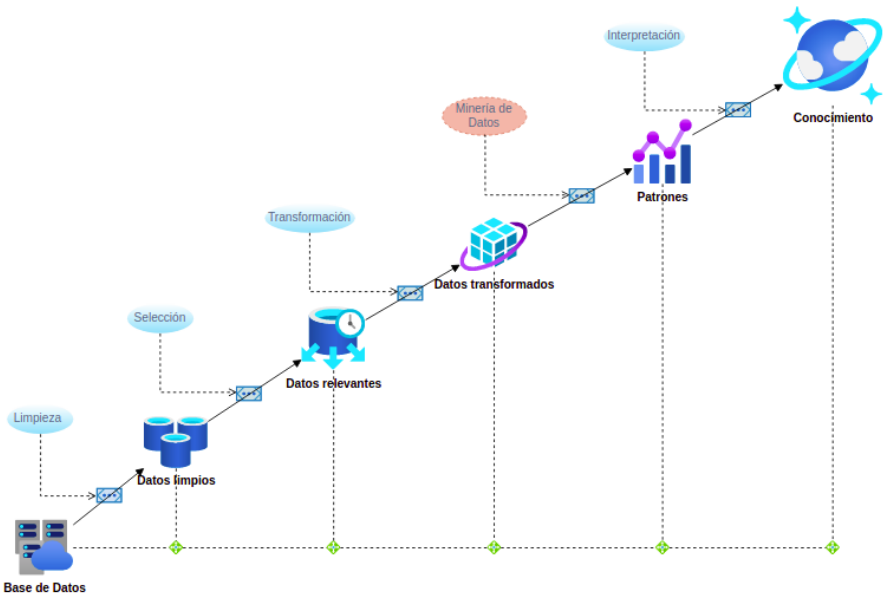


Figura 2.30: *Esquema del proceso KDD.*

Cada paso del proceso definido por KDD consiste en lo siguiente:

* **Limpieza**: Es el primer paso de KDD, recibe como entrada la base de datos y devuelve como salida un conjunto de datos limpio, sin *ruido*.

* **Selección**: En este paso se seleccionan todos los posibles datos que podrían aportar mayor relevancia para el fenómeno que se está analizando. La salida de este paso es un conjunto de datos relevantes para el análisis en siguientes pasos dentro del proceso.

* **Transformación**: Este paso es uno previo al de *Minería de datos* y su posición dentro del proceso justifica su tarea. Muchas de las técnicas de MD para reconocer patrones (como algoritmos de ML o AP) requieren que los datos tengan cierto formato. Algunos otros ciertas estructuras o, como vimos en la sección anterior, requieren de una codificación específica. La tarea de este paso del proceso es justo transformar los datos que recibe del paso anterior y adaptarlos a los requerimientos que dicte el siguiente paso.

* **Minería de Datos**: Es este paso del proceso el más importante y central dentro de KDD ya que es aquí donde se obtiene patrones que pueden ayudar a vislumbrar aspectos importantes del fenómeno que se analiza. De forma general el reconocimiento de patrones hace referencia al *proceso cognitivo* que ocurre en nuestro cerebro cuando hacemos coincidir cierta información que encontramos con datos almacenados en nuestra memoria. De forma análoga, KDD en este paso busca encontrar patrones sobre los datos que recibe mediante técnicas de IA, más específicamente, modelos de AM y AP. La salida es un conjunto de instancias de un objeto que describen el comportamiento o forma de actuar del fenómeno que se analiza, patrones.

* **Interpretación**: Es el paso final del proceso. Particularmente en este paso, el conocimiento se descubre por medio de un análisis e interpretación minuciosa de los patrones obtenidos, generalmente se presentan al usuario por medio de técnicas de visualización. Eso ayuda a los usuarios a comprender e interpretar los resultados de la minería de datos.

En este proyecto de investigación vamos a seguir los pasos que dicta KDD para resolver el problema objetivo.

CAPÍTULO 3

ESTADO DEL ARTE

En esta parte del documento vamos a hacer una revisión del estado del arte sobre los retos que se tienen, de momento, para intentar resolver el problema de la detección automática de paráfrasis.
Un desafío importante en la investigación de la paráfrasis es la falta de corpus de paráfrasis en Español ya que existen diversos esfuerzos por recolectar caso reales de paráfrasis, sin embargo, la mayoría de estos son en idioma Inglés. Nosotros haremos mención de algunos corpus de paráfrasis importantes, tanto en idioma Inglés como en Español.
Como ya hemos visto, todo modelo de AM recibe como entrada valores numéricos que representan a un objeto del mundo. En particular para nuestro proyecto trabajamos con objetos que se construyen de secuencias de palabras que forman un segmento discursivo con cierto relación semántica bajo un contexto dado, por lo que es necesario transformar nuestros objetos en estructuras numéricas que puedan ser tomadas por los modelos de AM y usadas para la detección automática de paráfrasis.
Por otra parte, vamos a revisar los modelos de AM que se han aplicado para la detección de paráfrasis, las implementaciones y esfuerzos anteriores.

3.1. Corpus de paráfrasis

Por parte de la creación de un corpus de paráfrasis en idioma Ingles se tienen las siguientes referencias más representativas. En el año 2003 se creó un corpus de paráfrasis que contiene, por una parte, traducciones paralelas al Inglés de artículos periodísticos sobre el mismo evento realizadas por personas especialistas en traducción; y por otra parte, una colección de textos extraídos por medio de un método automático para extraer paráfrasis que incluye definiciones de un mismo concepto de diferentes enciclopedias, biografías de la misma persona compuestas por diferentes escritores

y diferentes descripciones de una enfermedad en libros de medicina [38]. El corpus obtenido contiene un total de 25,962 pares de traducciones y 1,496,284 pares de oraciones extraídos con el método propuesto. En el año 2004 se presentó el corpus *MSP Paraphrase Corpus* [17] que fue creado a partir de artículos de noticias en Ingles agrupadas usando un modelo de Maquinas de Soporte Vectorial. El corpus obtenido tiene un total de 4,891 de pares de oraciones. Más adelante en el año 2013 se liberó el corpus *Webis-CPC-11* [39], un corpus creado de forma manual por anotadores (con fluidez en el idioma Inglés) contratados por medio de la plataforma *Amazon's Mechanical Turk*[1] (*AMT*). El corpus contienen un total de 4,067 de pares de textos que fueron seleccionados de forma aleatoria dentro de libros obtenidos de una plataforma en línea de libros electrónicos gratuitos[2]. Después, en el año 2017 el equipo de ciencia de datos de la plataforma *Quora*[3], liberó la primer versión del corpus *Quora* [40] que fue creado recopilando preguntas en idioma Inglés realizadas en el mismo sitio web y utilizando el método de aprendizaje maquinal *Random Forest* para identificar preguntas duplicadas. El corpus obtenido tiene un total de 404,348 pares de oraciones identificadas con el método propuesto. En el mismo año (2017) se propuso un nuevo método para recolectar casos de paráfrasis en oraciones contenidas en *tweets* dentro de la plataforma *Twitter*[4] mediante el seguimiento de URL's[41]. Con la primer prueba del método propuesto lograron extraer un total de 51,524 pares de sentencias etiquetadas como paráfrasis y no paráfrasis. Sin embargo, el método fue pensado para recolectar al menos 30,000 pares de sentencias nuevas, cada mes, con una precisión de etiquetado de al menos 70 %. Aunque el proyecto tomó en cuenta solo *tweets* en idioma Inglés, el método es fácilmente escalable a *tweets* de otros idiomas.

Para el caso de esfuerzos por crear un corpus de paráfrasis totalmente en idioma Español tenemos las siguientes referencias. En el año 2011 se desarrolló un nuevo corpus de paráfrasis que contiene 12 textos en Español de tópicos relacionados con el *sushi*, *sexualidad* y *astronomía* y extraídos de Wikipedia[5], revistas científicas y periódicos [15]. La creación de este corpus fue totalmente manual, de modo que le solicitaron a distintos voluntarios que intencionalmente reformularan o parafrasearan dichos textos. Más adelante, en el año 2016 se creó un nuevo corpus de paráfrasis en idioma Español que contiene 36 textos de tópicos relacionados con matemáticas, psicología y sexualidad (12 textos de cada tópico) [4]. Todos lo textos fueron extraídos del corpus *RST Spanish Treebank* [42]. En este caso de le solicitó a tres anotadoras que parafrasearan 4 textos distintos de cada tópico, con la opción de utilizar recursos como diccionarios de antónimos y sinónimos, de la lengua española y terminológicos.

[1] https://www.mturk.com/
[2] https://www.gutenberg.org/
[3] https://www.quora.com/
[4] https://twitter.com/
[5] https://www.wikipedia.org/

3.2. Representación de textos distribuida

Como ya hemos mencionado, en la sección 2.4.2, existen dos modelos que intentan estimar la mejor forma de representar palabras en vectores [11] (*Word2Vec*) o textos completos en vectores [12] (*Doc2Vec*), usando un enfoque distribuido de palabras o textos en un espacio vectorial. Dichos modelos, inspirados en técnicas de AM y AP (particularmente usando modelos de redes neuronales).
Vamos a ver que el modelo *Doc2Vec* es una extensión del primero modelo propuesto *Word2Vec*. Es decir, después de comprender cómo funciona el modelo *Word2Vec*, será más sencillo comprender cómo funciona el modelo *Doc2Vec*.
Además, revisaremos una variante importante del modelo *Word2Vec* conocida como *FastText* [18]. Esta variante toma el hecho de que modelos como *Word2Vec* ignoran aspectos morfológicos de las palabras ya que asignan un vector distinto a cada palabra. En cambio, *FastText* representa a cada palabra como una bolsa de caracteres n-*gramas* y a cada n-*grama* le asocia una representación vectorial. Y luego, las palabras se representan como la suma de estas representaciones.

3.2.1. Palabra a vector

La representación de una palabra en un vector de dimensión fija, usando el modelo *Word2Vec*, se obtiene entrenando una red neuronal simple de tipo *FeedForward* con una sola capa de entrada, una oculta y una de salida; y usando las técnicas *descenso del gradiente* y *retropropagación del error* para actualizar los pesos de cada capa y así estimar el vector que representa a una palabra. Veremos más adelante que estos pesos son en realidad los vectores de cada palabra que estamos tratando de representar.
Los autores en [11] propusieron dos arquitecturas distintas para realizar el entrenamiento de la representación vectorial, concretamente Bolsa de Palabras Continua (*Continuous Bag of Words, CBoW*) y *Continuous Skip-gram*. A continuación describimos cada una.
CBoW realiza el entrenamiento con el objetivo de que la red neuronal pueda predecir el vector de la palabra i-*ésima* (dentro de la secuencia) usando como entrada un vector que se crea sumando los vectores de palabras vecinas dentro de una ventana previamente definida.

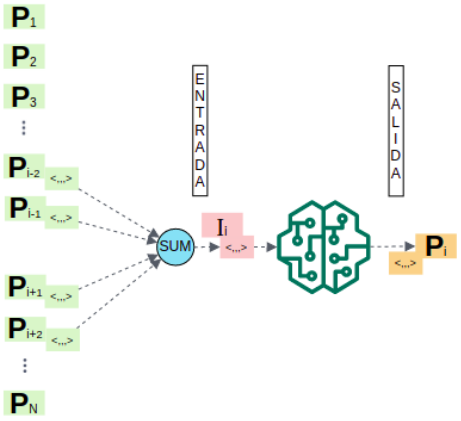


Figura 3.1: Bolsa de Palabras Continua (*Continuous Bag of Words, CBoW*).

En la Figura 3.1 se muestra un esquema general de la forma en que la red neuronal se entrena usando la arquitectura *CBoW*. Como podemos ver, dada una lista de palabras indexadas, por cada palabra p_i se genera un vector sumando los vectores de cada palabra vecina al índice i. En el ejemplo mostrado en la Figura 3.1 se toma una rango de 2 que resulta en considerar dos palabras pasadas y dos palabras futuras dentro de la secuencia de palabras. Es decir, dada la palabra p_i se toman los vectores de las palabras p_{i-2}, p_{i-1}, p_{i+1}, p_{i+2} para formar, mediante la suma, al vector I_i que forma un ejemplo para entrenar la red neuronal. El vector I_i como la entrada y el vector p_i como la predicción deseada.
Continuous Skip-gram funciona de forma un tanto similar a *CBoW* pero en esta arquitectura en lugar de predecir el vector de la palabra p_i usando como entrada el vector I_i formado por los vectores de palabras vecinas, esta intenta predecir una o varias palabras cercanas (dentro de un rango antes definido) dado el vector de la palabra p_i. En la Figura 3.2 se muestra un esquema general de la forma en que la red neuronal se entrena usando la arquitectura *Continuous Skip-gram*.

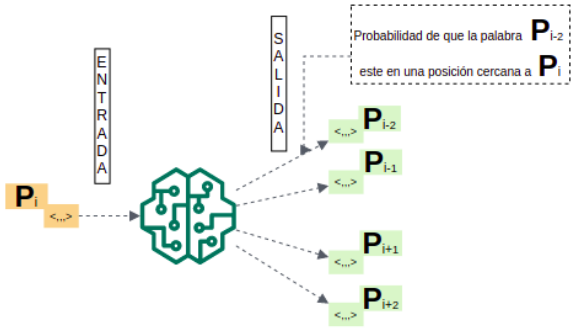


Figura 3.2: *Continuous Skip-gram.*

Cuando decimos palabras cercanas nos referimos al rango de palabras que se van a tomar en cuenta. En el ejemplo de la Figura 3.2 el rango es de 2 que resulta en considerar dos palabras pasadas y dos palabras futuras dentro de la secuencia de palabras.
La red neuronal se entrena de la siguiente forma, dada una palabra p_i en una secuencia de palabras dentro del texto, se seleccionan las palabras más cercanas p_{i-2}, p_{i-1}, p_{i+1}, p_{i+2} para que la red neuronal incremente la probabilidad de que dichas palabras se asocien a la palabra p_i.

En la Figura 3.3 se puede ver cómo sería el muestreo por cada palabra (seleccionada en azul) del texto ejemplo *"El tiempo es más pesado que la más pesada carga que puede soportar el hombre."* y sus correspondientes vecinos tomando un rango de 2 palabras pasadas y futuras.

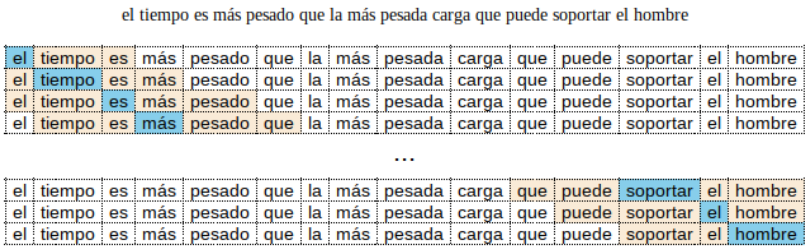


Figura 3.3: Muestreo de palabras vecinas.

De ese modo si se usa la arquitectura *CBoW* entonces, por ejemplo, se usarían los vectores de las palabras *"que", "puede", "el"* y *"hombre"* sumados para predecir el vector de la palabra *"soportar"*. O en el caso de la arquitectura *Continuous Skip-gram*, por ejemplo, se usaría el vector de la palabra *"soportar"* para maximizar la probabilidad de que las palabras *"que", "puede", "el"* y *"hombre"* sean las más cercanas.
En ambas arquitecturas, *CBoW* y *Continuous Skip-gram*, los vectores iniciales de cada palabra dentro del vocabulario en el corpus de entrenamiento son de tipo *One-Hot*. Una vez entrenada de red neuronal, los vectores de cada palabra se generan por los pesos en la capa oculta de la red neuronal. Para ver lo anterior vamos a ejemplificar el proceso usando la arquitectura *Continuous Skip-gram*.

En primer lugar, recordemos que no podemos ingresar palabras (de forma directa como una cadena de caracteres) a una red neuronal, de modo que para entrenar la red es necesario usar una representación, al menos inicial, de las palabras. Para hacer eso, primero se construye el vocabulario de palabras que se encuentran dentro del conjunto de textos que se van a usar para entrenar el modelo. Retomemos el conjunto de textos usado en la sección 2.4.1 para obtener los vectores *One-Hot*. En dicho ejemplo se encontraron un total de 24 palabras distintas dentro de los textos. Los vectores *One-Hot* asociados a cada palabra se encuentran en la Figura 2.28. Podemos ver que cada vector

One-Hot tiene dimensión 24 y que ponemos un 1 en la posición correspondiente a cada palabra y 0 en las demás posiciones.
Adicionalmente, pensemos que queremos construir un espacio de dimensión 4 para representar cada palabra en el vocabulario y que el rango de palabras cercanas sea de 3. Con esta configuración la salida de la red neuronal tiene 6 neuronas en la capa de salida, donde cada neurona predice la probabilidad de las palabras vecinas. En la Figura 3.4 se puede ver la arquitectura de la red neuronal generada con la configuración anterior.

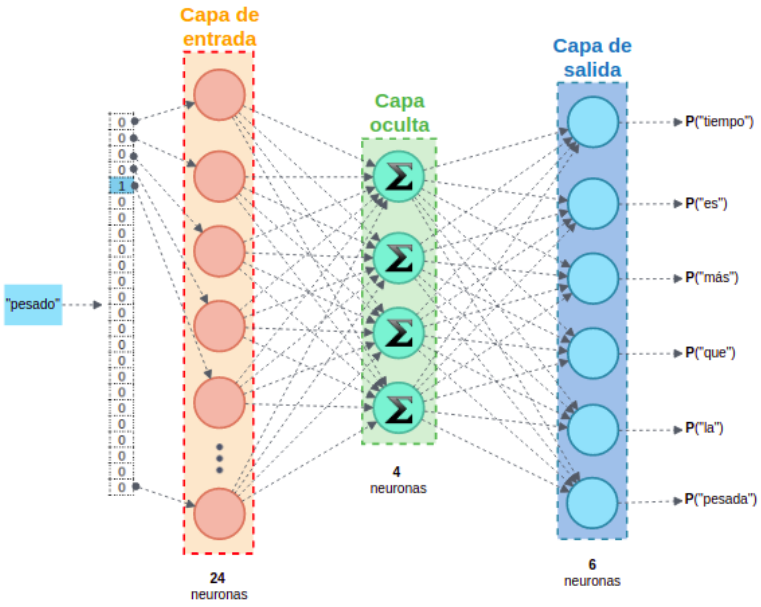


Figura 3.4: Arquitectura de la red neuronal del ejemplo usando *Continuous Skip-gram*.

De modo que, con el ejemplo anterior, estamos aprendiendo a representar palabras en vectores de dimensión 4. Por lo tanto, la capa oculta va a ser representada por una matriz de pesos que va a tener 24 filas (una por cada palabra) y 4 columnas (una por cada característica). Van a ser los pesos en la capa oculta los que van a darnos la representación de cada palabra que estamos buscando. De modo que el objetivo principal es solamente entrenar esta matriz de pesos en la capa oculta, la capa de salida al final del entrenamiento se desecha. En la Figura 3.5 se muestra la conversión de matriz de pesos en la capa oculta a vectores de características aprendidas.

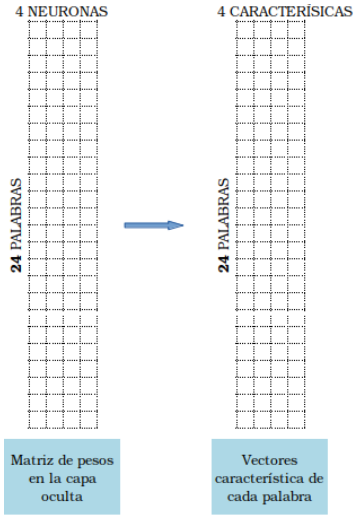


Figura 3.5: Definición de los vectores característica con los peso en la capa oculta.

Los vectores *One-Hot* iniciales son solo una forma de seleccionar la fila de la matriz de pesos (o características) correspondiente a la posición donde se encuentra el valor 1 mediante el producto de matrices. En la Figura 3.6 se muestra un ejemplo para obtener el vector asociado a la palabra *"pesado"* usando el vector *One-Hot* asociado y la matriz de pesos en la capa oculta que la red neuronal con el ejemplo anterior podría generar.

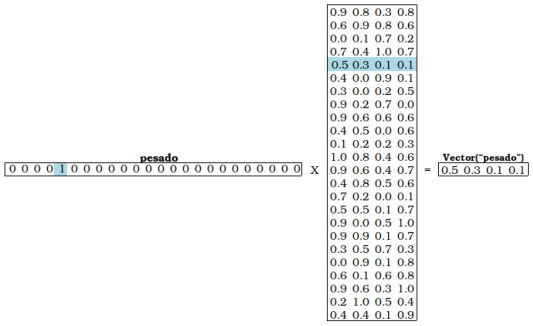


Figura 3.6: Vector de características usando el vector *One-Hot*.

Por otro lado, en la capa de salida lo que se tiene es un clasificador de regresión *SofMax*, es decir, que por cada neurona en la capa de salida (una por cada palabra cercana dentro del rango) se produce un valor entre 0 y 1 tal que la suma de todos esos valores es 1. Recordemos que cada neurona en la capa de salida tiene asociada una palabra cercana que queremos clasificar correctamente, entonces el valor de la regresión se asocia a la probabilidad de que la palabra en cada neurona sea la más cercana. De forma más específica, cada neurona en la capa de salida tiene un vector asociado a la palabra que se quiere clasificar correctamente, ese vector se multiplica con el vector que viene de

la capa oculta asociado a la palabra de entrada. Al valor resultante x se le aplica la función exponencial e^x y finalmente se divide por la suma de todos los resultados en la capa de salida. En la Figura 3.7 se muestra cómo se calcularía la probabilidad de que al seleccionar de forma aleatoria una palabra cercana a *"pesado"* esta sea *"que"* tomando los valores del ejemplo anterior.

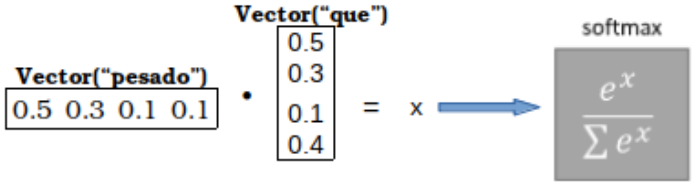


Figura 3.7: Calculando la salida de la neurona de la palabra *"que"* en el ejemplo.

Recordemos que el entrenamiento se hace con el objetivo de maximizar la probabilidad de que las palabras seleccionadas sean las más cercanas. Los pesos en la capa oculta se actualizan con la *retropropagación del error*.

Los autores de esta técnica reportan que de manera sorpresiva se encontró que la similitud entre los vectores de palabras, usando esta técnica de codificación, se puede evaluar usando simples operaciones algebraicas entre vectores. Por ejemplo, se mostró que (con el conjunto de datos que entrenaron y probaron el modelo) el vector $\overrightarrow{V}_{rey} - \overrightarrow{V}_{hombre} + \overrightarrow{V}_{mujer}$ resultaba en un vector que estaba en una vecindad al $\overrightarrow{V}_{reina}$ [11].

3.2.2. Documento a vector

El objetivo de los modelos *Doc2Vec* es crear una representación numérica de un documento, independientemente de su longitud. Pero a diferencia de las palabras, los documentos no vienen en estructuras lógicas (o secuencias) como palabras en oraciones, por lo que era necesario encontrar otro método. En [12] propusieron un cambio sencillo al modelo anterior *Word2Vec* [11] para poder representar textos de longitud variable en vectores. Propusieron agregar otro vector en el proceso aprendizaje. Para el caso de la arquitectura *CBoW* además de usar a las palabras para entrenar la red neuronal, también agregan un vector que representa al documento completo. En la Figura 3.8 se puede ver la estructura de entrenamiento propuesta en [12], se puede ver que es muy similar a la mostrada en la Figura 3.1.

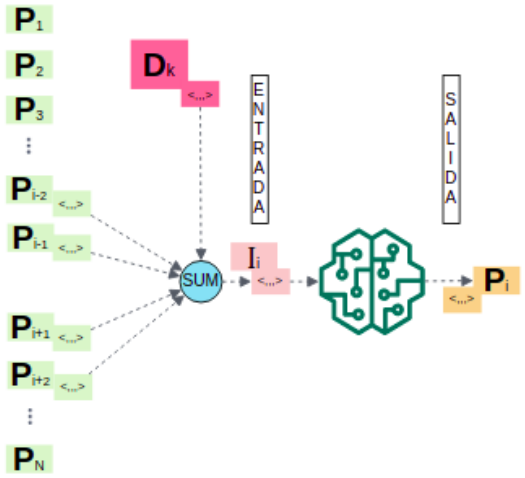


Figura 3.8: Arquitectura *CBoW* para representar documentos.

El vector que se agrega se ve marcado con rojo en la Figura 3.8 y corresponde al documento k-*ésimo* dentro del corpus de entrenamiento. Es decir, por cada documento D_k, se extraen las palabras P_i que contiene todo el documento y se crea un vector adicional al de los vectores de las palabra. De ese modo, al entrenar cada palabra P_i (usando los vectores de las palabras vecinas para predecir el vector de la palabra P_i) también el vector del documento D_k se ajusta y, al final del entrenamiento, en la matriz de pesos dentro de la capa oculta se tiene (en un columna adicional) la representación vectorial del documento completo, que recordemos, es de una dimensión fija.

Es importante mencionar que el vector de cada palabra P_i es único dentro de el conjunto de documentos. Es decir, si por ejemplo la palabra *"guerra"* se encuentra en dos documentos distintos D_k y D_j, el vector de la palabra *"guerra"* va a ser único y por lo tanto compartido entre los dos documentos al momento del entrenamiento.

Para el caso del entrenamiento del modelo usando la arquitectura *Continuous Skip-gram* proponen predecir un conjunto de palabras aleatoriamente elegidas dentro de un documento. En la Figura 3.9 se muestra la arquitectura resultante, que como se puede ver, es similar a la anterior Figura 3.2.

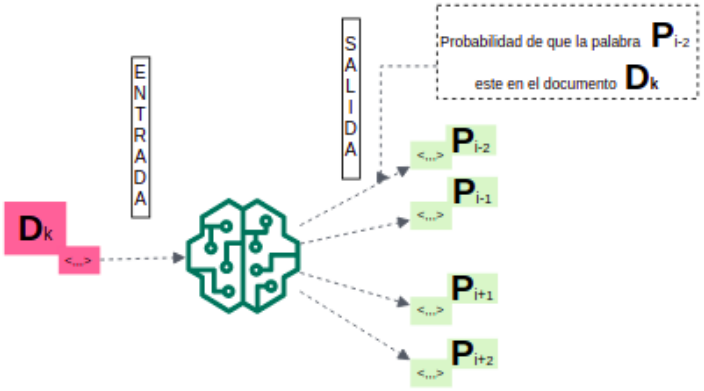


Figura 3.9: Arquitectura *Continuous Skip-gram* para representar documentos.

Por cada documento D_k dentro del corpus de entrenamiento, se toma una conjunto de palabras y la red neuronal se entrena para maximizar la probabilidad de que los vectores de salida de la red neuronal sean los de las palabras que se seleccionaron. Este método es menos costoso, hablando en términos de recursos computacionales, debido a que no es necesario almacenar los vectores de todas las palabras dentro del documento en el proceso de entrenamiento.
Ambas arquitecturas también son entrenadas, de la misma forma que el modelo de palabras a vectores *Word2Vec*, usando la *retropropagación del error* y en la capa de salida lo que se tiene es un clasificador de regresión *SofMax*.

3.2.3. FastText

En [18] se presentó una técnica de representación continua conocida como *FastText* que difiere principalmente en la forma en que se definen los vectores de cada palabra. Vimos que los modelos *Word2Vec* y *Doc2Vec* definen el vector de cada palabra usando la matriz de pesos obtenida durante el entrenamiento del modelo. Esta forma de representación ignora de alguna forma la estructura interna de las palabras ya que generalmente la formación de palabras siguen ciertas reglas morfológicas [18]. De modo que *FastText* agrega una capa de aprendizaje previa para aprender representaciones de n-*gramas* de caracteres y asi crear la representación vectorial de cada palabra con la suma de los vectores de n-*gramas*.

Del mismo modo que *Word2Vec* y *Doc2Vec*, *FastText* utiliza las mismas arquitecturas de entrenamiento *CBoW* y *Skip-gram*. En la Figura 3.10 mostramos un diagrama de la arquitectura *CBoW* con la capa extra de los vectores de n-*gramas*.

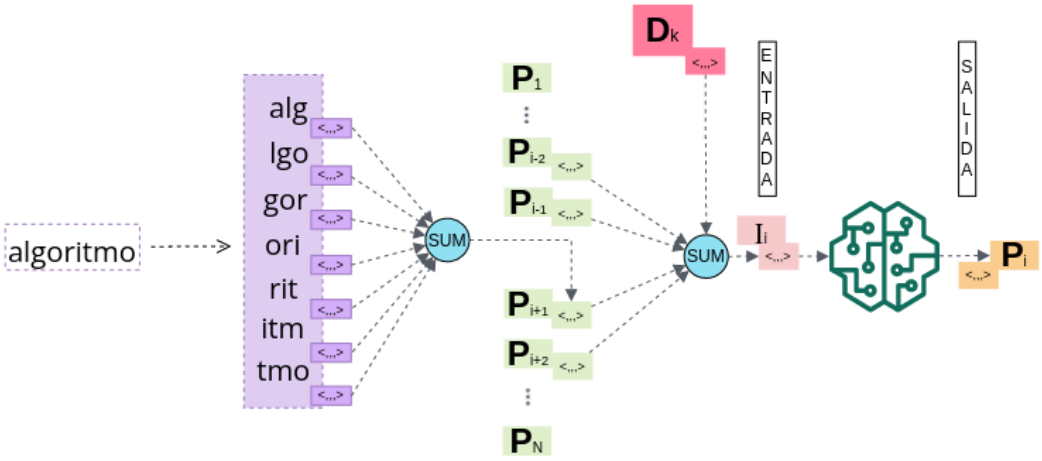


Figura 3.10: Arquitectura *CBoW* para representar documentos usando *FastText*.

Observemos que la palabra (P_{i+1}, en la Figura 3.10) es "*algoritmo*". Y descomponemos en pequeños segmentos de 3-gramas que *FastText* aprenderá y asignará un vector a cada uno de los segmentos. La suma de cada uno de los vectores es la que define el vector de la palabra P_{i+1}.

3.3. Detección de paráfrasis

Generalmente el problema asociado a la detección de paráfrasis automática se plantea como un problema de clasificación en ciencias de la computación [6] en donde un par de textos, en principio distintos, se evalúan y etiquetan (o clasifican) como paráfrasis o no paráfrasis en función de distintos parámetros a considerar para determinar si son parecidos.
En el año *2006* se creó el sistema *ParaEval* [5], en el que desarrollaron un modelo probabilístico combinado con técnicas de búsqueda local en el que miden las semejanzas léxico-sintácticas por pares de textos y realizan una búsqueda local con relaciones entre los pares, a nivel sintáctico, para determinar si ambos textos tiene palabras similares (incluyendo la revisión de sinónimos) para de esa forma determinar si un texto es paráfrasis de otro o no lo es. El algoritmo que proponen usa una tabla de referencia entre palabras u oraciones que tienen relación semántica y que sirve para calcular un score de similitud tomando como referencia la frecuencia de coincidencias en los registros de la tabla. En la Figura 3.11 mostramos un ejemplo mostrado en [5] con el tipo de relaciones semánticas entre palabras u oraciones que fueron consideradas.

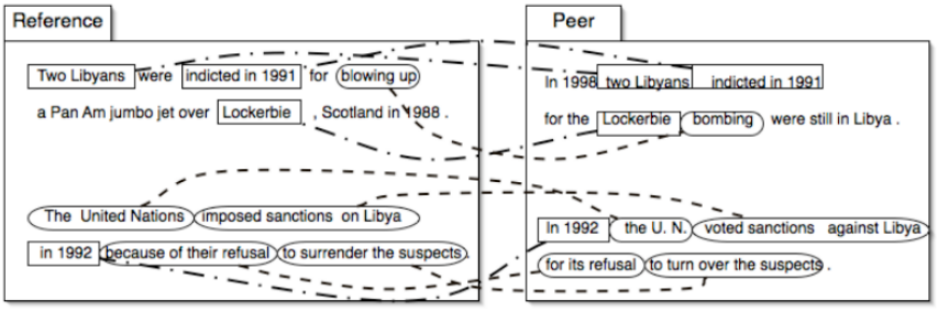


Figura 3.11: Comparación entre pares de textos del sistema *ParaEval* [5].

El desempeño del modelo mostró resultados prometedores. Sin embargo, la escalabilidad y desempeño del modelo dependen directamente de la cantidad y calidad de información que tiene la tabla de referencia que se utliza.

Más adelante en el año *2014* se publicaron los resultados de un algoritmo llamado *SimTex* [3] que reportó resultados prometedores debido a que agregaron un analizador de estructuras discursivas (*Rhetorical Structure Theory*, RST) que ayudó en una primera etapa del proceso a identificar secciones o segmentos del texto semejantes en términos discursivos y luego, de los pares de segmentos con relevantes similitudes discursivas el algoritmo analiza similitudes semánticas entre palabras usando relaciones conocidas como *synsets* o *conjuntos de términos sinónimos* incluidas en la base de

datos *EuroWordNet*[6]. Al final el algoritmo pondera lo puntajes de similitud discursiva y semántica entre el par de textos y calcula un resultado de similitud semántica normalizado. Otra de las contribuciones importantes de [3] fue la implementación de un nuevo corpus de paráfrasis en Español creado de forma manual para realizar sus experimentos.

Después, pasando a métodos que modelan ambos textos como una bolsa de palabras y son representados por medio de vectores, en *2015* se presentó el proyecto *ASOBEK* [7] en donde plantearon una representación vectorial de los textos aplicando técnicas de *tokenización* y superposición de características para poder aplicar el modelo de clasificación Maquina de Soporte vectorial (MSV).
Demostraron empíricamente que para este tipo de problema, los *kernel* basados en funciones radiales (*RBF*) presentan mejores resultados. Además, mostraron que la concatenación de características en el espacio vectorial a nivel palabra y caracter, puede dar un rendimiento comparable a modelos y método del *PLN* más sofisticados.
Más adelante, en el año *2017* se propuso un modelo de red neuronal [8] que combinaba dos tipos de redes, por una parte una red convolucional (*convolutional neural network, CNN*) que se utiliza para procesar y extraer características de similitud entre representaciones vectoriales pre-entrenadas de cada texto. Y por otra parte, una red recurrente (*Long short-term memory, LSTM*) que recibe como entrada las características codificadas de la red CNN para crear dependencias semánticas de la representación del par de textos. El modelo propuesto busca obtener un buen desempeño detectando paráfrasis entre pares de textos cortos e informales como *Tweets* de la plataforma *Twitter*[7] ya que pueden representar un reto adicional dado que puede haber problemas adicionales como faltas de ortografía, acrónimos de la web y estilos de escritura distintos.
En el año *2020* se publicaron los resultados de un proyecto similar al anterior excepto que configuraron una cascada de redes neuronales que estima la mejor forma de representar a los vectores de características asociados a los textos que se comparaban [6]. Se usaron tres tipos de redes neuronales en cascada: CNN, LSTM y LSTM con atención propia. Al final de la configuración que propusieron colocaron una red neuronal clásica con dos capas ocultas que realizaba la clasificación final.

En general observamos que todo esfuerzo anterior por desarrollar sistemas para la detección de paráfrasis han dado aportes en el proceso de mejora continua para lograr sistemas cada vez más robustos y eficientes para la tarea. La Tabla 3.1 resume los enfoques y esfuerzos discutidos en esta sección.

[6]http://www.illc.uva.nl/EuroWordNet
[7]https://twitter.com/

Trabajo	**Nombre**	**Tipo**	**Concepto/Modelo**
[5]	Paraeval: Using paraphrases to evaluate summaries automatically	Probabilístico y de Optimización	Búsqueda local
[3]	Simtex: An approach for detecting and measuring textual similarity based on discourse and semantics	Analizador léxico-sintáctico y Probabilístico	Morfología del texto
[7]	ASOBEK: Twitter Paraphrase Identification with Simple Overlap Features and SVMs	Optimización	MSV
[8]	A deep network model for paraphrase detection in short text messages	Red neuronal	CNN, LSTM.
[6]	A multi-cascaded model with data augmentation for enhanced paraphrase detection in short texts	Red neuronal	CNN, LSTM y LSTM con atención propia.

Tabla 3.1: Trabajos previos sobre detección de paráfrasis automática

CAPÍTULO 4

METODOLOGÍA

En esta sección presentamos la forma en que abordamos el problema de detectar paráfrasis de forma automática empleando técnicas de aprendizaje maquinal. En términos generales nuestro método consta de 3 etapas:

* Selección y extracción de textos, desde una base de datos minable, para crear un nuevo corpus de paráfrasis en Español.
* Codificación de textos empleando técnicas de aprendizaje profundo.
* Aplicación de técnicas de aprendizaje supervisado para la clasificación de un par de textos.

4.1. Corpus de paráfrasis

Para la adquisición de paráfrasis, ha sido esencial encontrar y desarrollar técnicas simples y efectivas para crear pares de oraciones candidatos a paráfrasis y no paráfrasis. Como vimos en la sección 3.1 se han realizado distintos esfuerzos para recolectar muestras representativas del fenómeno de la paráfrasis. Muchos de estos en idioma Inglés y otros especialmente en idioma Español. Algunos creados de forma manual con anotadores parafraseando oraciones y textos, otros empleando traducciones y otros generados artificialmente. Pero todos atendiendo al hecho de que la eficacia y desempeño de modelos inspirado en AM y AI depende, en gran medida, de la calidad y cantidad de los datos que se utilizan. Por lo tanto, la creación de nuevos conjuntos de datos es de gran importancia.
En este apartado mostraremos la forma en que extrajimos segmentos discursivos de textos académicos y cómo creamos paráfrasis por medio de traducción que automatiza la generación de un nuevo corpus de paráfrasis y recopila grandes cantidades de

muestras. Y, como complemento de nuestro trabajo, usaremos tres corpus de paráfrasis importantes en la literatura para evaluar nuestro método tomando en cuenta como punto de referencia los esfuerzos previos en cada uno de los corpus.

4.1.1. Corpus *CBI-M-00-21*

Para la creación de nuestro corpus de paráfrasis usamos como fuente de datos minable el repositorio *TESIUAMI*[1], una colección de tesis electrónicas que administra la Coordinación de Servicios Escolares Documentales de la Universidad Autónoma Metropolitana unidad Iztapalapa. En ella se encuentran las tesis de alumnos egresados. Las tesis las podemos encontrar organizadas por: división (CBI, CBS y CSH), nivel (Licenciatura, Maestría y Doctorado), área de estudio y año de realización. Para nuestros experimentos hemos decidido enfocarnos unicamente en las tesis de la división de *CBI*, de nivel *Maestría*, de los años 2000 a 2021 y de las áreas *Matemáticas, Química, Ciencias y Tecnologías de la Información, Física, Biomédica y Energía*. El número total de tesis que obtuvimos bajo dichos criterios es de 817. En la Figura 4.1 se puede ver la distribución, por área, de todas las tesis obtenidas.

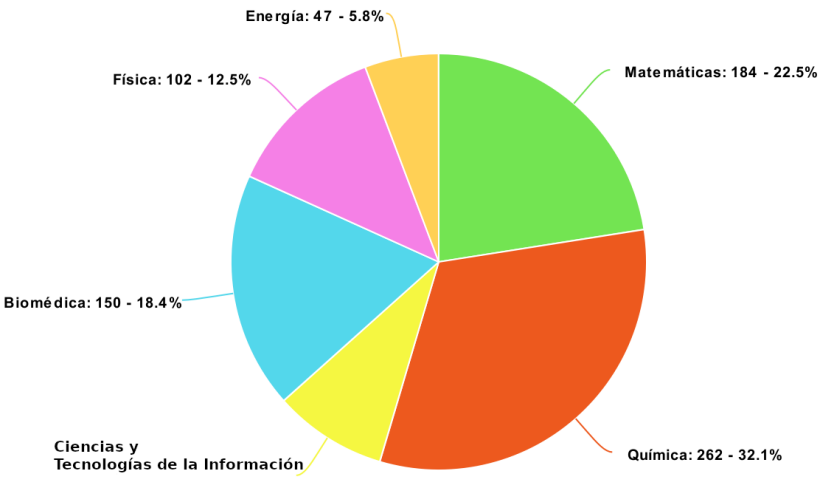


Figura 4.1: Distribución de número de tesis por área.

4.1.1.1. Extracción

Extrajimos el texto que contenía cada una de las tesis seleccionadas, definiendo como unidad mínima a un segmento discursivo y aplicando expresiones regulares para identificarlos. Consideramos a los segmento discusivos como todas las oración del texto delimitadas por un punto y seguido o punto y aparte. En la Figura 4.2 se muestra un ejemplo de lo que tipificamos como un segmento discursivo.

[1]http://tesiuami.izt.uam.mx/uam/default.php En un futuro: http://bindani.izt.uam.mx/

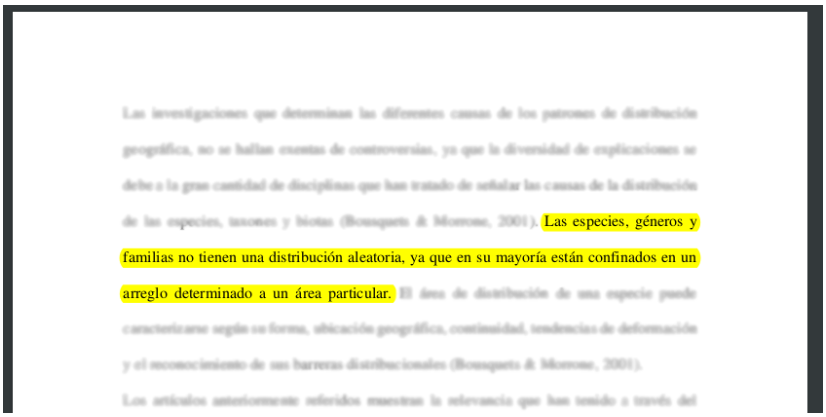


Figura 4.2: Ejemplo de selección de un segmento discursivo.

El orden jerárquico de la información obtenida se puede ver en la Figura 4.3. Cada documento (identificado por su clave única *UAMI**) tiene un número N de páginas y a su vez cada página tiene un número M de segmentos.

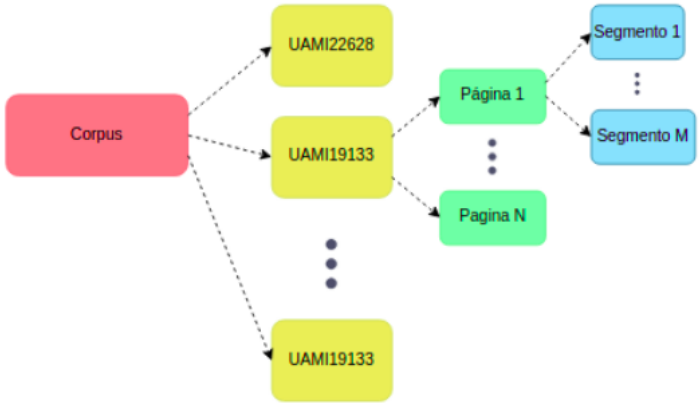


Figura 4.3: Jerarquia de información del corpus *CBI-M-00-21*.

4.1.1.2. Generación de paráfrasis

A cada segmento extraído de los documentos descargados le asociamos otro texto que representa a la paráfrasis que hemos generado por medio de traducción [4], asegurándonos que la traducción sea similar en términos semánticos, pero con estructuras lexico-sintácticas distintas. Para nuestros experimentos nos inclinamos por usar *MariaMT*[2], un modelo de traducción de textos pre-entrenado con datos de *OPUS*[3] desarrollado por el equipo de traducción de Microsoft (*Microsoft Translator Team*). Cada segmento que hemos extraido de cada página dentro de cada tesis lo hemos traducido a varios idiomas, en secuencia, hasta llegar de vuelta al idioma original. En

[2]https://huggingface.co/transformers/model_doc/marian.html
[3]https://opus.nlpl.eu/

la Figura 4.4 se puede ver un esquema de la secuencia de traducciones por las que cada segmento ha pasado y un ejemplo de la secuencia.

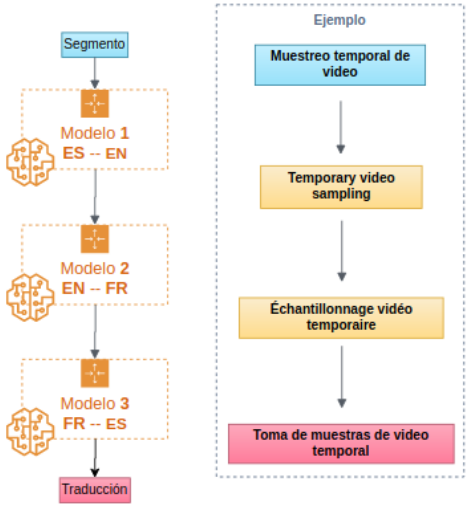


Figura 4.4: Secuencia de traducción de cada segmento.

4.1.1.3. Evaluación de paráfrasis generada

Para evaluar las traducciones obtenidas usamos *BLEU score* (*Bilingual Evaluation Understudy*)[13] una puntuación para comparar una traducción de texto candidata con una o más traducciones de referencia. Una coincidencia perfecta da como resultado una puntuación de 1.0 lo cual indica que los textos comparados son exactamente identicos, mientras que una discrepancia perfecta da como resultado una puntuación de 0.0 indicando que los textos comparados no tienen ninguna coincidencia a nivel de {1,2,3,4}-*gramas*, donde un *n-grama* es una secuencia de n palabras dentro de una oración. En la Figura 4.5 se muestra un ejemplo de una coincidencia de *3-gramas* entre un par de oraciones.

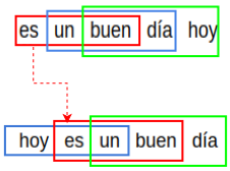


Figura 4.5: Ejemplo de una coincidencia en 3-gramas.

4.1.1.4. Generación de no paráfrasis

Hasta ahora hemos asignado a cada segmento otro texto paráfrasis por medio de traducción. Para cubrir el caso contrario, es decir, asociar a cada segmento otro texto que represente un forma de no paráfrasis, decidimos extraer segmentos adicionales

de tesis dentro del repositorio *TESIUAMI*, del mismo rango de años (*2000-2021*) y la misma división y área pero ahora de distinto grado (*Licenciatura*). Y sobre este conjunto de segmentos nuevo buscar, por cada segmento del corpus *CBI-M-00-21*, un segmento cuya intersección de palabras sea máxima. De esa forma asociaremos a cada segmento un texto que sea de forma léxico-sintáctico similar por la intersección de palabras pero, por la aleatoriedad con que se elige y la fuente de datos distinta, semánticamente diferente. En la Figura 4.6 se muestra un ejemplo de la selección de candidatos a no paráfrasis de cada segmento. Es importante mencionar que para contar las palabras de intersección entre un par de segmentos solo hemos considerado aquellas palabras que no son *StopWords*. Las palabras *StopWords* o palabras vacías son las palabras más comunes en cualquier lenguaje natural que pueden no agregar mucho valor al significado de una oración. En la Figura 4.7 mostramos las palabras que hemos considerado como *StopWords*.

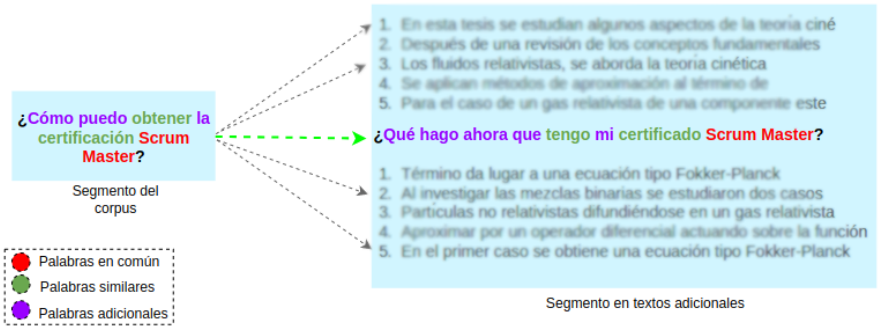


Figura 4.6: Intersección de palabras entre un par de segmentos.

Figura 4.7: *StopWords*, palabras vacias.

El objetivo principal de esta creación de pares de textos (segmento - no paráfrasis) es crear un tipo de ruido para los modelos de AM. En específico lograr similitudes en palabras principales, pero una variación importante en las formas sintácticas de la no paráfrasis con respecto al segmento original que, además, aporte palabras ajenas al contexto.

4.1.2. Corpus *Sushi*, *Mrpc* y *Quora*

Como complemento de nuestro estudio usamos tres conjuntos de datos del mundo real en nuestras evaluaciones experimentales. Cada unos de estos conjuntos de datos contiene pares de oraciones previamente clasificadas en las clases *paráfrasis* y *no paráfrasis*.

1) **Sushi** [14][15]: un corpus en Español basado en un artículo de un blog relacionado con el *sushi* en donde se solicitó a varios voluntarios que reformularan o parafrasearan intencionalmente el mismo artículo.

2) **Quora** [16]: este corpus en idioma Inglés fue recopilado a partir de preguntas realizadas en el sitio web de preguntas y respuestas *Quora*[4].

3) **Microsoft Research Paraphrase Corpus (*MRPC*)** [17]: un corpus en idioma Inglés que contiene pares de oraciones extraídas de frases en artículos de noticias en la web.

Observemos que los corpus *Quora* y *Mrpc* están en idioma Inglés, de modo que fue necesario traducirlos al idioma Español para mantener persistencia con los corpus *Sushi* y *CBI-M-00-21*. La traducción la realizamos aplicando un modelo *MariaMT* especializado en traducciones Inglés-Español. En la Tabla 4.1 se muestran dos ejemplos de pares de oraciones traducidas por cada corpus.

Corpus	(Inglés)		(Español)		Clase
	Texto uno	Texto dos	Texto uno	Texto dos	
Mrpc	The songs are on offer for 99 cents each, or $9.99 for an album.	The company will offer songs for 99 cents and albums for $9.95.	Las canciones están en oferta por 99 centavos cada una, o $9.99 para un álbum.	La compañía ofrecerá canciones por 99 centavos y álbumes por $9.95.	Paráfrasis
	Police official S.K. Tonapi told Reuters at least 40 people had been killed and more than 100 wounded.	State interior ministry spokesman Vasant Pitke told Reuters that at least 42 people had been killed and 112 injured.	El oficial de policía S.K. Tonapi dijo a Reuters que al menos 40 personas habían muerto y más de 100 habían resultado heridas.	Vasant Pitke, portavoz del Ministerio del Interior del Estado, dijo a Reuters que al menos 42 personas habían muerto y 112 habían resultado heridas.	No paráfrasis
Quora	Can I learn martial arts on my own ?	How can I learn martial arts at home through internet ?	¿Puedo aprender artes marciales por mi cuenta?	¿Cómo puedo aprender artes marciales en casa a través de Internet?	Paráfrasis
	Which are the manga related episodes in "Naruto Shippuden.[a]nime ?	How many episodes does Naruto have ?	¿Cuáles son los episodios relacionados con el manga en el anime Naruto Shippuden?	¿Cuántos episodios tiene Naruto?	No paráfrasis

Tabla 4.1: Ejemplos de pares de oraciones traducidas al Español de los corpus *Mrpc* y *Quora*.

[4]https://www.quora.com

4.1.3. Conjuntos de datos

Para entrenar y evaluar cada modelo de AM utilizado debemos tener nuestros conjuntos de datos o corpus dividido en tres distintos conjuntos de datos; *entrenamiento*, *validación* y *prueba*. El conjunto de entrenamiento lo usamos para ajustar el modelo (pesos y sesgos en el caso de una red neuronal) es decir, el modelo aprende de estos datos. El conjunto de validación lo usamos para evaluar un modelo de manera frecuente (generalmente durante el proceso de entrenamiento). El modelo ocasionalmente ve estos datos, pero nunca "aprende" de ellos. Y por último el conjunto de prueba es un conjunto de datos separado que utilizamos para evaluar el desempeño del modelo después de completar el entrenamiento. En la Figura 4.8 mostramos la proporción de división que fijamos en cada corpus.

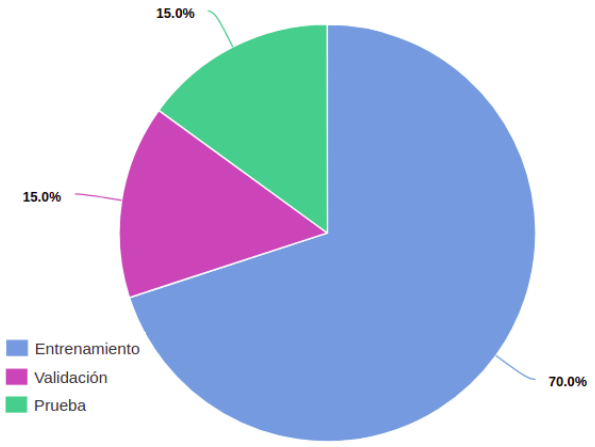


Figura 4.8: Proporción de división de los conjuntos de datos entrenamiento, validación y prueba.

Las proporciones que tomamos son las más comunes en esta área de estudio, y además han mostrado ser las óptimas en trabajos previos [35][36].

4.2. Codificación de textos

Para codificar cada par de oraciones dentro de cada corpus decidimos usar y entrenar dos tipos de *embeddings*. Usamos *Doc2Vec* por la capacidad que tiene para crear vectores de longitud fija sobre textos de longitud variable[12]. Y también usamos *FastText* porque cada palabra se representa como una bolsa de n-gramas de caracteres de tal forma que una representación vectorial de cada palabra está asociada a cada n-grama de caracteres donde las palabras se representan como la suma de estas representaciones[18]. El objetivo principal de seleccionar dos técnicas distintas es determinar qué *embedding* se ajusta mejor a nuestras necesidades. Creemos que un enfoque de documento a vector (*Doc2Vec*) podría representar una ventaja al proyecto ya que tratamos con segmentos de longitud variable; pero una aproximación distinta

utilizando el enfoque de n-gramas de caracteres de *FastText* podría agregar información relevante en los casos en donde se encuentren palabras fuera del vocabulario ya que estas palabras obtienen un vector en una vecindad, a nivel caracter, de una palabra similar[18].

4.2.1. Selección de modelo embedding

Para determinar qué modelo embedding presenta mejor desempeño en definir vecindades en el espacio vectorial con relaciones de similitud semántica, planteamos el siguiente experimento. Tomamos el conjunto de palabras de cada segmento de nuestro corpus (*CBI-M-00-21*) y lo unimos con el conjunto de palabras del texto paráfrasis (que creamos con la traducción). Al unir el par de oraciones logramos tener un conjunto más grande de palabras que están contextualmente relacionadas entre sí. En la Figura 4.9 se muestra un ejemplo de la unión de oraciones para crear una muestra de entrenamiento. La idea detras de esta configuración se debe al hecho de que ambos *embeddings* (tanto *Doc2Vec* como *FasText*) crean vecindades de palabras que están contextualmente relacionadas. De modo que agrupando palabras contextualmente similares, creamos relaciones numéricas más fuertes entre las palabras por todo el espacio vectorial definido por el *embedding*.

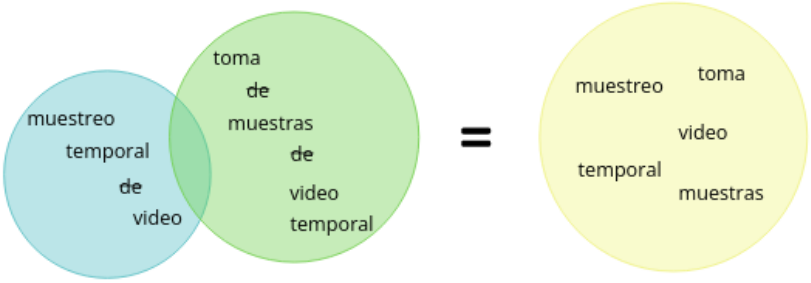


Figura 4.9: Unión los conjuntos de palabras del par de oraciones para crear una muestra de entrenamiento del *embedding*.

Usamos cada conjunto de palabras unido como una muestra de entrenamiento del *embedding*. La configuración nos permite proponer la siguiente comparación.

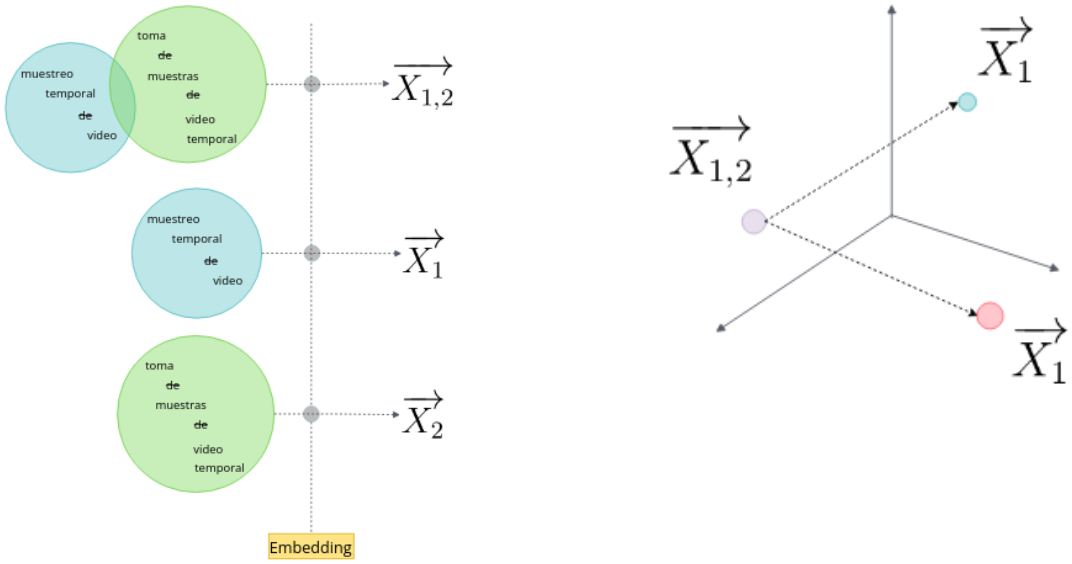
Evaluación y comparación

Consideramos crear por cada oración dentro de cada corpus, tres vectores generados por el propio *embedding* creado. El primer vector $\overrightarrow{X_1}$, es creado tomando el conjunto de palabras de una oración. El segundo vector $\overrightarrow{X_2}$, es creado tomando la segunda oración. Y por último, el tercer vector $\overrightarrow{X_{1,2}}$, es creado tomando el conjunto de palabras de la unión de las dos oraciones del par; como mostramos en la Figura 4.10(a).
Para evaluar cada *embedding* planteamos comparar el vector $\overrightarrow{X_{1,2}}$ con los vectores $\overrightarrow{X_1}$

y $\overrightarrow{X_2}$ generados (ver Figura 4.10(b)), ya que en principio se trata de vectores distintos porque ambos, tanto $\overrightarrow{X_1}$ como $\overrightarrow{X_2}$, no contienen palabras que $\overrightarrow{X_{1,2}}$ sí contiene. Como métrica de comparación usamos la similitud coseno definida como:

$$\mathbf{S}im(\overrightarrow{A}, \overrightarrow{B}) = \frac{\overrightarrow{A} \bullet \overrightarrow{B}}{\|\overrightarrow{A}\| \times \|\overrightarrow{B}\|} \tag{4.1}$$

esperando obtener, en promedio, un valor próximo a 1.0 ya que eso indicaría que, incluso sin las palabras que $\overrightarrow{X_{1,2}}$ contiene, el *embedding* está definiendo una similitud con los vectores $\overrightarrow{X_1}$ y $\overrightarrow{X_2}$.



(a) Definición de vectores para evaluación del embedding.

(b) Representación de la evaluación.

Figura 4.10: Definición y representación de vectores para evaluación de embedding.

4.2.2. Ajuste de hiperparámetros

Un número de hiperparámetros son requeridos para entrenar todos los *embeddings*. Calibramos y seleccionamos estos hiperparámetros en cada *embedding* tabulando cada combinación y reportando el promedio de la función de pérdida durante el entrenamiento. Para el tipo de entrenamiento usamos las técnicas *skipgram* y *cbow*[11].

Las funciones de pérdida usadas fueron *NS* (*Negative Sampling*), *HS* (*Hierarchical softmax*) y *Softmax*. Las dimensiones de los vectores creados fueron de 200, 300 y 400. Y por último, una tasa de aprendizaje fija de 0.05.

4.3. Pre-procesamiento

4.3.1. Contexto

El contexto, la variación de significado de una palabra en función del entorno lingüístico, es sin duda una característica de lenguaje que todo modelo de AM debe modelar para lograr buenos resultados. En nuestro método decidimos entrenar un modelo embedding distinto por cada corpus (*CBI-M-00-21*, *Sushi*, *Quora* y *Mrpc*) usando solo las oraciones dentro del conjunto de entrenamiento como mostramos en la Figura 4.11.

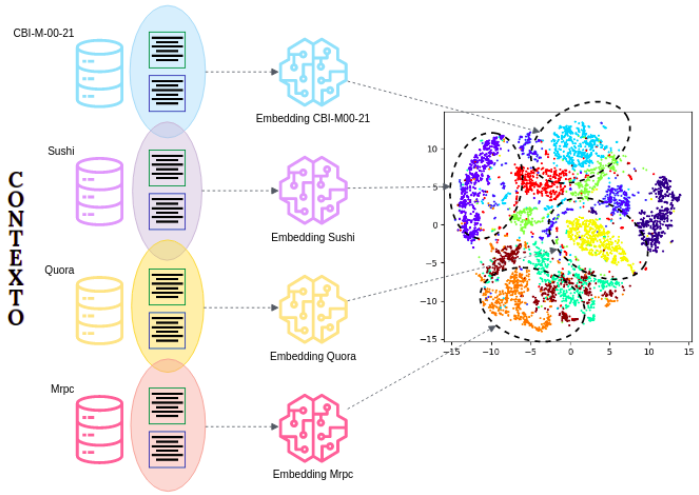


Figura 4.11: Esquema de entrenamiento de 4 embeddings.

Cada corpus (de los cuatro que utilizamos) contiene distintos textos con un contexto distinto. Lo que implica que las palabras en cada oración tienen un entorno distinto por cada corpus. Nosotros aprovechamos esta característica para modelar variantes contextuales simuladas por variaciones numéricas (por palabra) contenidas en cada modelo *embedding* definido.

4.3.2. Transformación

Ya con los modelos embedding entrenados podemos definir el esquema de transformación de los datos para tratarlos con técnicas de AM. Este esquema de transformación quedara estático en todo proceso de entrenamiento y predicción de

los modelos AM que utilizaremos. En la Figura 4.12 se muestra el esquema de transformación de los datos propuesto.

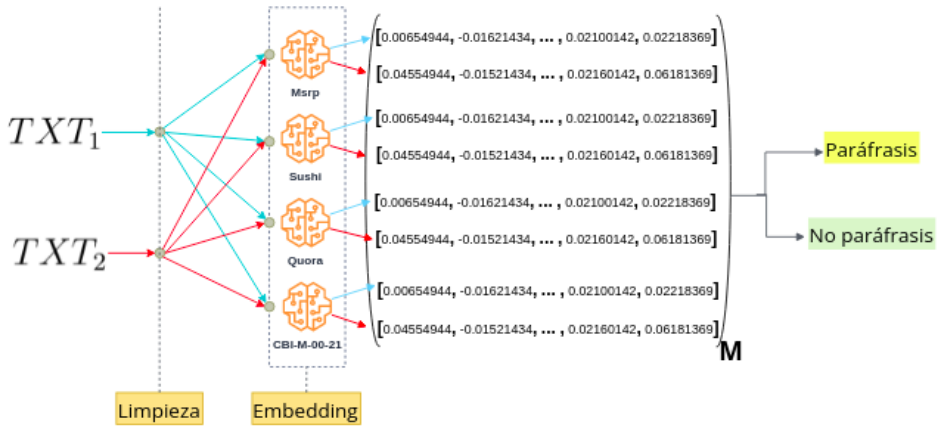


Figura 4.12: Esquema de transformación de datos

Cada par de textos (TXT_1, TXT_2) pasa por una capa de limpieza en donde eliminamos caracteres especiales, pasamos todo a minúsculas y removemos *Stop Words*. La salida de la capa de limpieza pasa a la capa de embedding donde por cada modelo embedding entrenado se estima un vector correspondiente a cada texto TXT_1, TXT_2 limpio. La salida de la capa embedding es una matriz M que corresponde a una muestra para entrenamiento validación y prueba etiquetada con la clase *paráfrasis o no paráfrasis* correspondiente al par (TXT_1, TXT_2).

4.3.3. Balanceo de clases

Los modelos de AM en muchas ocasiones aprenden mejor cuando los datos que se emplean para entrenarse están completamente balanceados, es decir, cuando el conjunto de entrenamiento tiene el mismo número de muestras por cada clase o etiqueta. Una de las característica favorables de nuestro corpus *CBI-M-00-21* es que el número de muestras está totalmente balanceado con respecto a cada clase (*Paráfrasis* y *No Paráfrasis*) por la forma en que fue construido. Sin embargo, los corpus *Sushi, Quora* y *Mrpc* presentan un desbalance en el número de muestras por clase. Para tratar este problema usamos una técnica conocida como *up-sampling* (sobre-muestreo) en la que se replican y generan muestras sintéticas adicionales de la clase minoritaria para lograr un balance entre clases. Particularmente usamos SMOTE (*Synthetic Minority Over-Sampling Technique*) [19] un algoritmo que construye ejemplos sintéticos de las clases minoritarias analizando el espacio vectorial definido por el embedding y utilizando un enfoque de vecinos más cercanos.

4.4. Modelado del problema y selección de técnicas de AM

Como ya hemos mencionado, el problema de la detección de *paráfrasis* automática ha sido abordado usando una gran variedad de técnicas y modelos que, de forma empírica, han mostrado obtener buenos resultados sobre una conjunto de textos que cada proyecto elige como base experimental para la construcción y evaluación del modelo. Como primera fase de desarrollo planteamos modelar el problema de la detección de *paráfrasis* como un problema de clasificación binaría, y usar cada uno de los corpus descritos como base experimental.

4.4.1. Técnicas de AM

Los problemas de clasificación binaría pueden ser resueltos por una gran variedad de técnicas de *AM*. En nuestros experimentos decidimos intentar con los siguientes algoritmos:

- Bayes Ingenuo (*Naive Bayes*)
- Regresión Logística (*Logistic Regression*)
- Maquina de Soporte Vectorial (*Support Vector Machine*)
- Perceptron multicapa (*Feedforward Neural Networks*)
- Redes neuronales convolucionales (*Convolutional Neural Networks, CNN*)
- Redes neuronales recurrentes, particularmente LSTM (*Long short-term memory, LSTM*)
- Ensamble de tipo mezcla de expertos utilizando como sub modelos redes LSTM.

En la implementacion los algoritmos *Bayes Ingenuo, Regresión logística y Maquina de soporte vectorial* decidimos utilizar una serie de hiperparámetros para la fase de entrenamiento. En el caso de *Regresión logística* utilizamos los optimizadores *sag* (Stochastic Average Gradient descent) [20] y *saga* [21]. Para Maquina de Soporte Vectorial experimentamos con los kernels: *lineal, polinomial, rbf y sigmoide*. La búsqueda de los mejores hiperparámetros la realizamos usando la técnica *GridSearch* con validación cruzada de 10 *folds*.
En el caso de las redes neuronales utilizamos las siguientes arquitecturas.

4.4.1.1. Perceptrón multicapa

En la Figura 4.13 se muestra la arquitectura del perceptrón multi-capa que proponemos.

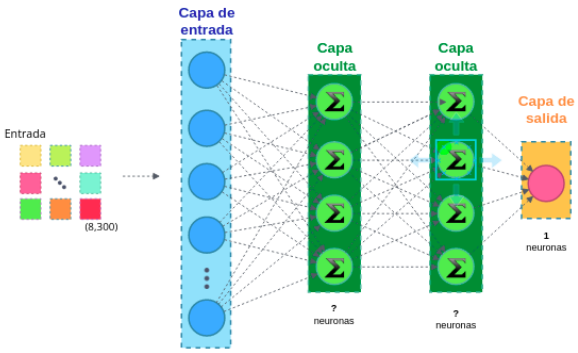


Figura 4.13: Arquitectura de perceptrón multi-capa.

Consideramos un *perceptrón* multi-capa estándar con 2,400 neuronas en la capa de entrada (igual a la dimensión de un vector cuando aplanamos la matriz de características). Dos capas ocultas, cada una con un número variable de neuronas con función de activación *ReLU*. Y al final, en la capa de salida 1 neurona a la que le aplicamos una función de activación *Sigmode* que usamos para predecir las clase *paráfrasis* y *no paráfrasis*. El proceso de entrenamiento fue realizado aplicando la función de pérdida *BinaryCrossentropy*[5] y optimizándola usando el algoritmo de *retropropagación* del error y *descenso del gradiente*.

4.4.1.2. Red neuronal convolucional

En la Figura 4.14 presentamos la arquitectura de la red neuronal convolucional que proponemos para la tarea de detección de *Paráfrasis* dado un par de segmentos.

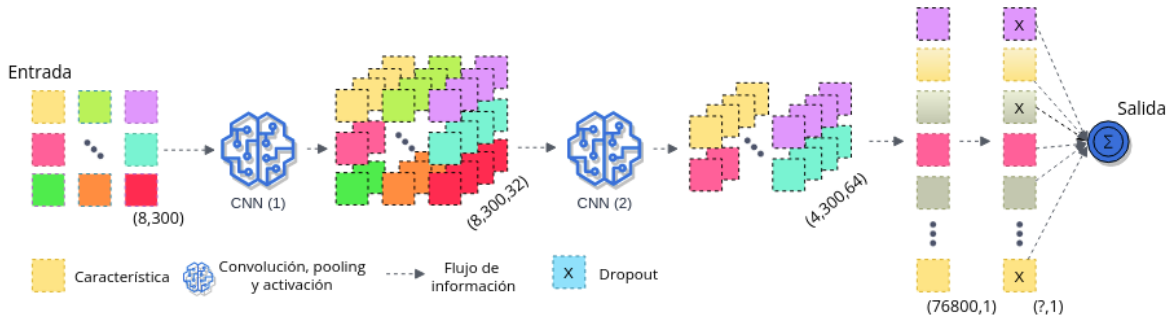


Figura 4.14: Arquitectura de red neuronal convolucional propuesta.

A la matriz generada, descrita en el apartado anterior, le aplicamos dos operaciones de convolución, la primera aplicando 32 filtros y la segunda 64 filtros. Sobre la convolución de 64 filtros aplicamos un aplanamiento que posteriormente se pasa por una capa de

[5]https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy

Dropout[6] para generar un vector de características nuevo que va a alimentar a una capa de activación (*Dense*) final.

Ajuste de hiperparámetros

Seleccionamos una serie de hiperparámetros que consideramos examinar para el entrenamiento de nuestro modelo. La búsqueda de los mejores hiperparámetros la realizamos usando *GridSearch* [22] con validación cruzada de 10 *folds*. Para seleccionar un optimizador intentamos con *nAdam*, *Adam*, *Adadelta* y *SGD*. Para la capa de *Dropout* consideramos tasas de 0.1, 0.2, 0.3, 0.4 y 0.5. Para la operación de *Pooling* consideramos *Max* y *Average Pooling*. Y examinamos las funciones de activación *relu, sigmoid, tanh, selu, elu* y *exponential* tanto para la capa de convolución como para la de discriminación final.

4.4.1.3. Red neuronal recurrente *LSTM*

En la Figura 4.15 se muestra la arquitectura de la red neuronal *LSTM* que proponemos para tratar el problema.

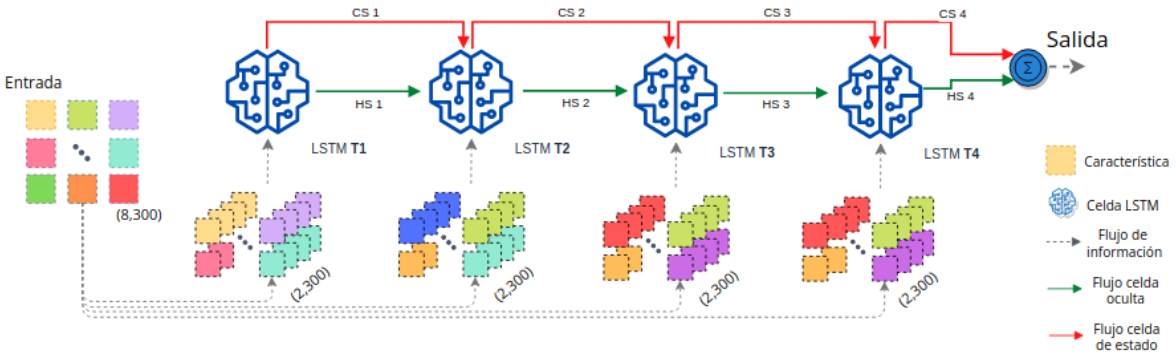


Figura 4.15: Arquitectura de red neuronal recurrente *LSTM*.

La matriz generada por cada embedding la descomponemos en cuatro partes de tal forma que a cada *time-stamp* o *celda LSTM* de la red (T_1, T_2, T_3 y T_4 en la Figura 4.15) le asignamos una secuencia de la matriz de entrada. El primer *time-stamp* (T_1) recibe la primer secuencia de características y conforme al proceso descrito en la sección 2.3.3.3, genera dos salidas (CS_1 y HS_1), que después se combinan con la entrada del segundo *time-stamp* (T_2) para generar la salidas (CS_2 y HS_2) que se definen en el proceso de recurrencia de una red neuronal *LSTM*. El proceso continua hasta generar las salidas (CS_4 y HS_4) que usamos como entrada de una red neuronal simple que predice si la entrada corresponde a un caso de *paráfrasis* o *no paráfrasis*.

[6]https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout

Ajuste de hiperparámetros

Para el caso de las redes neuronales recurrentes *LSTM* decidimos mantener la configuración estándar de cada celda. Es decir, mantuvimos la función de activación *sigmoid* para la compuerta de olvido (*forget gate*) y la función de activación *tahn* para las compuertas de actualización y salida. Sin embargo, experimentamos con distinto número de neuronas dentro de cada celda *LSTM* (32, 64, 128, 256 y 512) valores estándar en la literatura.

4.4.1.4. Ensamble de tipo mezcla de expertos

Aplicamos un tipo de ensable paralelo con el objetivo de minimizar el error debido a la varianza en los modelos de AM. Es decir, reducir la incapacidad para generalizar sobre datos nuevos. En el ensamble que proponemos cada submodelo usa un conjunto de entranamiento y validación distinto. Más especificamente, si usamo k submodelos, entonces definimos k conjuntos de datos separados que usamos para entrenar cada submodelo del ensamble. En la Figura 4.16 mostramos un diagrama con la definición de cada conjunto de datos para cada submodelo del ensamble.

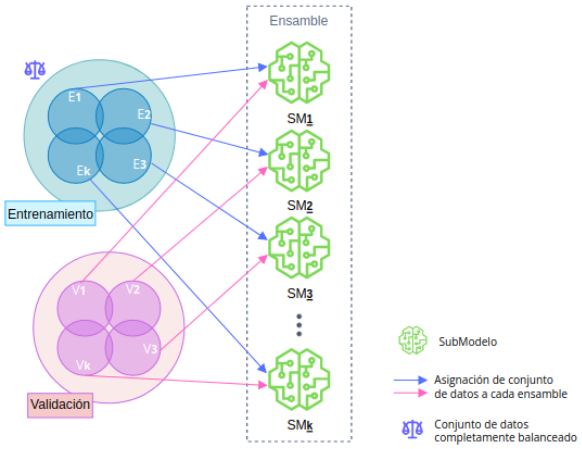


Figura 4.16: Asignación de cada conjunto de datos (E_i, V_i) a cada submodelo SM_i dentro del ensable.

Cada conjunto de datos E_i y V_i (de la Figura 4.16) lo construimos aplicando una técnica de muestreo aletorio (con remplazo) conocida como *Stratified ShuffleSplit* [7] en la que cada subconjunto de datos (E_i , V_i) se construye conservando el porcentaje de muestras de cada clase, es decir, mantenemos un balance de clases igual al conjunto de datos completo, y además existe una alta probabilidad de que a alguno de los conjuntos de datos (E_i , V_i) le falten algunas muestras del conjunto de datos original, pero también cualquier conjunto de datos (E_i , V_i) probablemente tendrá muestras repetidas con respecto a otro subconjunto de datos.

[7] https://scikitlearn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html

Submodelos y salida del ensamble

Existen distintas formas de formar un ensamble con submodelos de distintas clases, a veces usando diferentes modelos, distintos algoritmos o incluso funciones objetivo diferentes. El esquema de ensamble que proponemos lo construimos definiendo cada submodelo como una red neuronal recurrente *LSTM* con la misma arquitectura propuesta en 4.15.
La salida de cada submodelo dentro del ensamble la utilizamos para dar la predicción final. Esta predicción generalmente se realiza calculando un promedio entre la salida de cada submodelo o un esquema de tipo mayoría de votos. En particular para nuestro ensamble la salida de los k submodelos del ensamble la ingresamos a un perceptrón multicapa simple con dos capas ocultas. En la Figura 4.17 mostramos la arquitectura del ensamble completo.

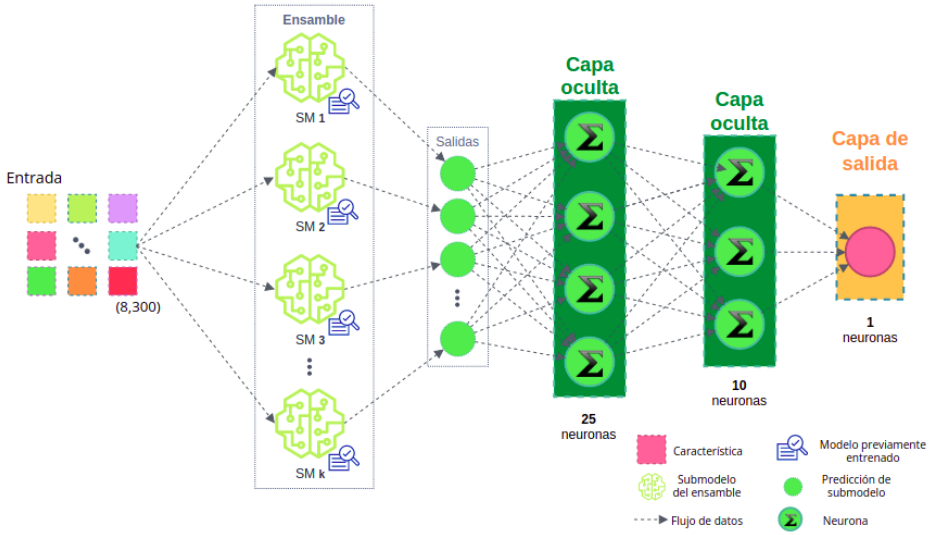


Figura 4.17: Arquitectura del ensamble propuesto.

La forma en que el ensamble predice la clase es la siguiente: la muestra a clasificar se ingresa a los k submodelos del ensamble ($SM_1, SM_2, SM_3, ..., SM_k$ en 4.17). Cada uno de los submodelos predice un valos entre 0 y 1 indicando el grado de similitud. Un valor cercano a 0 se interpreta como un caso de *no paráfrasis* y próximo a 1 se interpreta como un caso de *paráfrasis*. Las k salidas del ensamble las ingresamos al perceptrón multicapa para que aprenda patrones en las k salidas y, con los patrones aprendidos, generar la clasificación final.

Entrenamiento del ensamble

El entrenamiento de todo el ensamble lo dividimos en dos fases. En la primer fase entrenamos cada submodelo del ensamble tomando el correspondiente conjunto de entrenamiento E_i y conjunto de validación V_i. También consideramos calibrar

hiperparámetros distintos por cada submodelo de forma similar a la que describimos en (4.4.1.3). El resultado de la primer fase de entrenamiento son los k submodelos entrenados con la información que se les ha asignado.
En la segunda fase de entrenamiento del ensamble ajustamos los pesos y sesgos del perceptrón multicapa que da la predicción final. En esta segunda fase de entrenamiento usamos los conjuntos de entrenamiento y validación completos pasándolos por los k submodelos y después por el perceptrón en la parte final. Es importante mencionar que los submodelos ya no son entrenados en esta segunda fase, sino solo son usados como generación de las entradas del perceptrón final que se ajusta en esta fase.

4.5. Experimentos

El esquema de experimentación que aplicamos entre cada *corpus* y todos los modelos de AM seleccionados lo mostramos en la Figura 4.18.

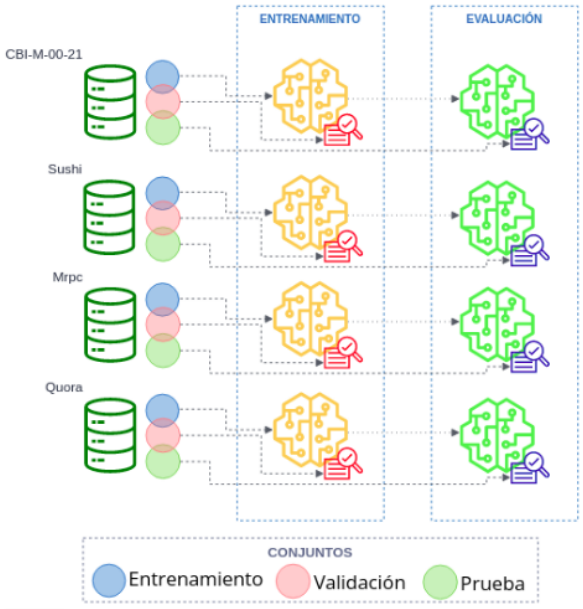


Figura 4.18: Esquema de experimentos con los modelos de AM seleccionados.

Por cada uno de los *corpus* tomamos el conjunto de entrenamieto para entrenar cada uno de los modelos de AM. En el caso de los modelos inspirados en el aprendizaje profundo usamos el conjunto de validación para monitorear el correcto entrenamiento del modelo, evitando llegar a un sobre ajuste con respecto a los datos de entrenamiento. Una vez el entrenamiento concluye, usamos el conjunto de prueba para validar el modelo obtenido y resportar las métricas de evaluación correspondientes.

4.6. Evaluación

Al ser un problema de clasificación binaría el que hemos modelado, podemos usar métricas de evaluación estándar de clasificación para evaluar nuestro modelo [23]. En nuestros experimentos evaluamos el desempeño de nuestros modelos utilizando las medidas de evaluación exactitud (*accuracy*, 4.2), precisión (*precision*, 4.3), exhaustividad (*recall*, 4.4) y valor-F ($F_1 score$, 4.5).

$$exactitud = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2}$$

$$precision = \frac{TP}{TP + FP} \tag{4.3}$$

$$exhaustividad = \frac{TP}{TP + FN} \tag{4.4}$$

$$F_1 = 2 \times \frac{precision \times exhaustividad}{precision + exhaustividad} \tag{4.5}$$

Donde *TP, FP, FN* y *TN* se obtienen de la matriz de confusión 4.2 y tienen la siguiente interpretación:
En un problema de clasificación binaría en general, un clasificador clasifica las muestras en las clases *positivo* y *negativo*.

- **TP** (*True Positives*): Muestras correctamente clasificadas como positivas.
- **FP** (*False Positives*): Muestras negativas clasificadas incorrectamente como positivas.
- **FN** (*False Negatives*): Muestras positivas incorrectamente clasificadas como negativas.
- **TN** (*True Negatives*): Muestras correctamente clasificadas como negativas.

		Valor real	
		Positivo	Negativo
Predicción	Positivo	TP	FP
	Negativo	FN	TN

Tabla 4.2: Matriz de confusión.

Adicionalmente, para visualizar el desempeño de nuestro modelo, realizamos un análisis ROC (*Receiver Operating Characteristics Curve*) [24], una métrica que se utiliza para medir el rendimiento de un modelo de clasificación. La curva ROC representa la tasa de verdaderos positivos (TVP, 4.6, *sensibilidad*) con respecto a la tasa de falsos positivos (TFP, 4.7, *especificidad*), lo que resalta la sensibilidad del modelo de clasificación.

$$TVP = \frac{TP}{TP + FN} \tag{4.6}$$

$$TFP = \frac{TN}{TN + TP} \tag{4.7}$$

De forma visual, el modelo con mejor desempeño predictivo es aquel cuya curva *ROC* se acerca más a la esquina superior izquierda. En la Figura 4.19 se muestra un ejemplo con dos curvas *ROC* representando el desempeño de dos modelos de demostración.

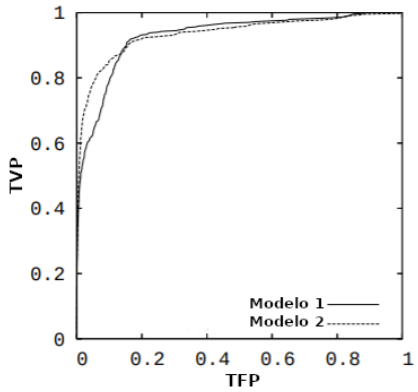


Figura 4.19: Curvas *ROC* de dos modelos de demostración.

Por otro lado, un posible uso práctico y otra forma de evaluación de nuestro modelo que consideramos en nuestros experimentos, es la realización de mapas de calor (*HeatMaps*) entre pares de documentos completos dentro del corpus *CBI-M-00-21* comparando cada combinación de página y reportando la similitud contextual que obtiene el modelo para obtener una forma visual de la distribución de similitud entre los documentos y la forma en que el modelo asigna la similitud.

4.7. Hardware y software utilizado

Usamos *Python*[8] como lenguaje principal en toda implementación en todos nuestros experimentos. Para implementar la red neuronal utilizamos *Keras*[9] que a su vez utiliza *TensorFlow*[10] como *core* funcional principal. Para evaluar nuestro modelo usamos la librería *Scikit-lear*[11].
Toda ejecución se realizó en un PC con dos procesadores *Intel(R) Xeon(R) E5645 @ 2.40GHz* y 64 Gb de memoria ram.

[8]https://www.python.org/
[9]https://keras.io/
[10]https://www.tensorflow.org/
[11]https://scikit-learn.org/

CAPÍTULO 5

RESULTADOS

5.1. Recolección de datos

5.1.1. Corpus *CBI-M-00-21*

En la Tabla 5.1 mostramos el número total de documentos y páginas (por cada área) que hemos procesado y el número total de segmentos u oraciones extraídos.

Área	Segmentos	Páginas	Documentos
Matemáticas	152,841	15,193	184
Química	179,829	21,386	262
Ciencias y Tecnologías de la Información	55,525	5,438	72
Biomédica	101,807	10,606	150
Física	62,483	6,911	102
Energía	49,497	5,122	47
Total:	**601,982**	**64,656**	**817**

Tabla 5.1: Segmentos, páginas y documentos procesados, extraídos y traducidos.

Descargamos un total de 817 tesis, en las que procesamos un total del 64,656 páginas que incluían texto y logramos extraer un total de 601,982 segmentos discursivos. Cada segmento extraído le asociamos otro texto paráfrasis y no paráfrasis. De modo que el tamaño total del corpus (número de pares de textos) es de 1,203,964.

5.1.1.1. Evaluación de paráfrasis

En la Figura 5.1 se puede ver un diagrama de cajas con el *BLEU score* de cada segmento extraído de TESIUAMI comparado con el tipo de paráfrasis que le hemos asociado por medio de traducción aplicando la cascada de modelos de traducción *MariaMT*.

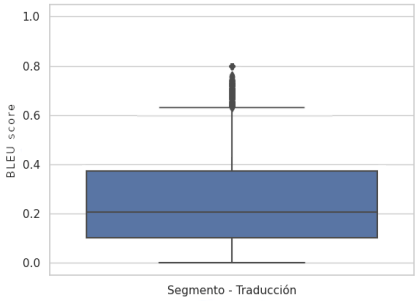
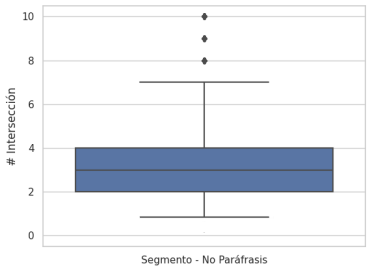


Figura 5.1: BLEU score obtenido al comparar cada segmento con su traducción.

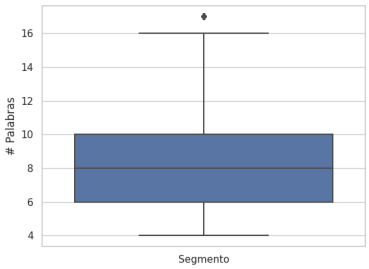
Como podemos ver obtenemos, en promedio, un *BLEU score* de 0.2 al comparar el segmento con la traducción generada. Como punto de referencia sobre este valor obtenido, en [13] se experimentó sobre un corpus con 500 oraciones con buenas traducciones hechas por humanos profesionales se obtuvo un *BLEU score* de 0.2571, cuanto más se aproxime una traducción automática a una traducción humana profesional, mejor.

5.1.1.2. Evaluación de no paráfrasis

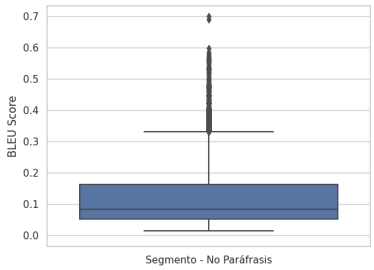
Los resultados de generar un tipo de *no paráfrasis* a cada segmento los mostramos en la Figura 5.2. En 5.2(a) se puede ver un diagrama de cajas con el número de palabras que tienen en común cada segmento de nuestro corpus y su texto asociado como *no paráfrasis* (recordemos que solo consideramos las palabras de cada texto que no son *stopwords* 4.7). Podemos ver que en general obtenemos pares de segmentos cuyo número de palabras en común es 3; lo que representa un porcentaje de intersección del 35.7 % si consideramos que el número promedio de palabras de cada segmento es 8, como podemos ver en la Figura 5.2(b). Y por otra parte, si calculamos el *BLEU score* de cada par *(segmento - no paráfrasis)* obtenemos valores por debajo de 0.20, ver Figura 5.2(c).



(a) Intersección de palabras entre cada segmento y *no paráfrasis*.



(b) Número de palabras (sin *stopwords*) por cada segmento.



(c) BLEU score entre cada segmento y *no paráfrasis*.

Figura 5.2: Evaluación de la No Paráfrasis generada.

Recordemos que el objetivo principal de esta creación de pares de textos es crear un tipo de ruido para los modelos de AM. En específico, lograr similitudes en palabras principales pero una variación importante en las formas sintácticas de la *no paráfrasis* con respecto al segmento original que además aporte palabras ajenas al contexto.

Conjuntos de datos

En la Tabla 5.2 mostramos la división de todo el conjunto de datos en subconjuntos disjuntos de *entrenamiento, validación* y *prueba* que utilizaremos para entrenar nuestros modelo de AM, además el total de pares para el caso de muestras etiquetadas como *Paráfrasis* y *No Paráfrasis*.

CBI-M-00-21	Número total de pares	Pares Paráfrasis	Pares No paráfrasis
Entrenamiento	869,866	434,933	434,933
Validación	153,504	76,752	76,752
Prueba	180,594	90,297	90,297

Tabla 5.2: División en conjuntos del corpus *CBI-M-00-21*.

5.1.2. Corpus *Sushi*

Este corpus cuenta con un total de 6,896 pares de oraciones de las que 314 tienen asignada la clase *paráfrasis* y 6,582 tienen asignada la clase *no paráfrasis*.

Conjuntos de datos

La Tabla 5.3 muestra la división de todo el conjunto de datos en subconjuntos disjuntos de *entrenamiento, validación* y *prueba* que utilizaremos para entrenar nuestro modelo de AM, además el total de pares para el caso de muestras etiquetadas como *Paráfrasis* y *No Paráfrasis* resultante.

Sushi	Número total de pares	Pares Paráfrasis	Pares No paráfrasis
Entrenamiento	4,692	215	4,477
Validación	826	37	789
Prueba	1,378	62	1,316

Tabla 5.3: División en conjuntos del corpus *Sushi*.

5.1.3. Corpus *Mrpc*

El corpus *Mrpc* tiene un total de 4,891 pares de oraciones de los que 3,900 son de la clase *paráfrasis* y 1,901 de la clase *no paráfrasis*.

Conjuntos de datos

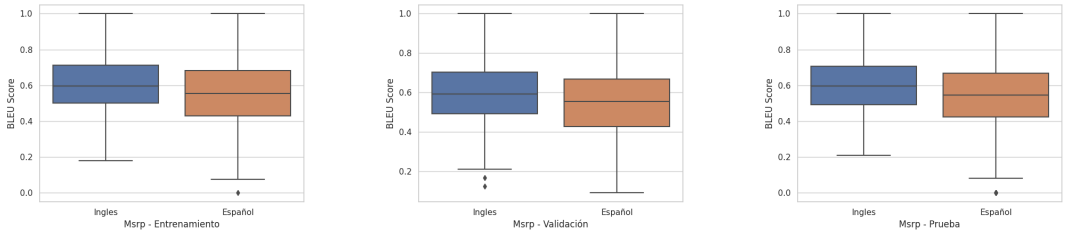
El número de pares de oraciones por conjunto de datos se puede ver en la Tabla 5.4.

Mrpc	Número total de pares	Pares Paráfrasis	Pares No paráfrasis
Entrenamiento	3,261	2,202	1,059
Validación	815	551	264
Prueba	1,725	1,147	578

Tabla 5.4: División en conjuntos del corpus *Mrpc*.

5.1.3.1. Evaluación traducción

Para evaluar la traducción al idioma Español que hemos generado a cada par de oraciones dentro del corpus, calculamos el *BLEU score* de cada par en Inglés y de cada par en Español. En La Figura 5.3 se pueden ver unos diagramas de cajas con los valores obtenidos por cada conjunto de datos.



(a) Entrenamiento (b) Validación (c) Prueba

Figura 5.3: *BLEU score* obtenido por cada par de oración en Inglés y Español del corpus *Mrpc* por conjunto de datos.

Se puede ver que en promedio se obtiene un *BLEU score* menor en los pares de texto en Español que los de Inglés en todo conjunto de datos (5.3(a), 5.3(b) y 5.3(c)) lo que nos confirma que el idioma Español presenta variaciones *léxico-sintácticas* más relevantes; una característica importante del idioma Español que los modelos de AM pueden tomar para presentar un mejor desempeño (en comparación con el idioma Inglés). También aplicamos una prueba de *Wilcoxon* [25] en la que se obtuvo un *p-value* de 0.0, por lo que se concluye que ambos poblaciones provienen de medias distintas o que tienen una distribución diferente.

5.1.4. Corpus *Quora*

El corpus *Quora* tiene un total de 404,348 pares de oraciones de los que 149,306 son de la clase *paráfrasis* y 255,042 de la clase *no paráfrasis*.

Conjuntos de datos

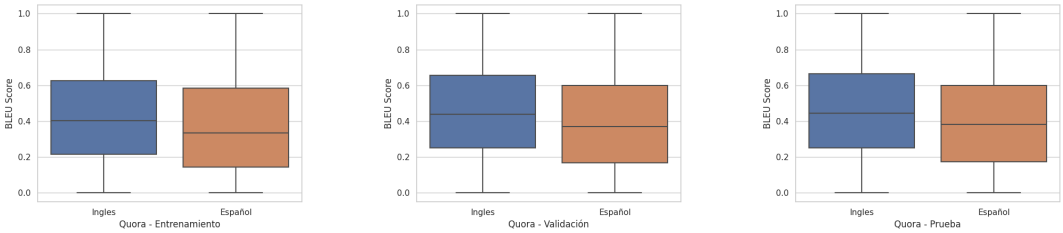
El número de pares de oraciones por conjunto de datos se puede ver en la Tabla 5.5.

Quora	Número total de pares	Pares Paráfrasis	Pares No paráfrasis
Entrenamiento	384,348	139,306	245,042
Validación	10,000	5,000	5,000
Prueba	10,000	5,000	5,000

Tabla 5.5: División en conjuntos del corpus *Quora*.

5.1.4.1. Evaluación traducción

El resultado de calcular el *BLEU score* de cada par en Inglés y en Español se muestra en Figura 5.4.



(a) Entrenamiento (b) Validación (c) Prueba

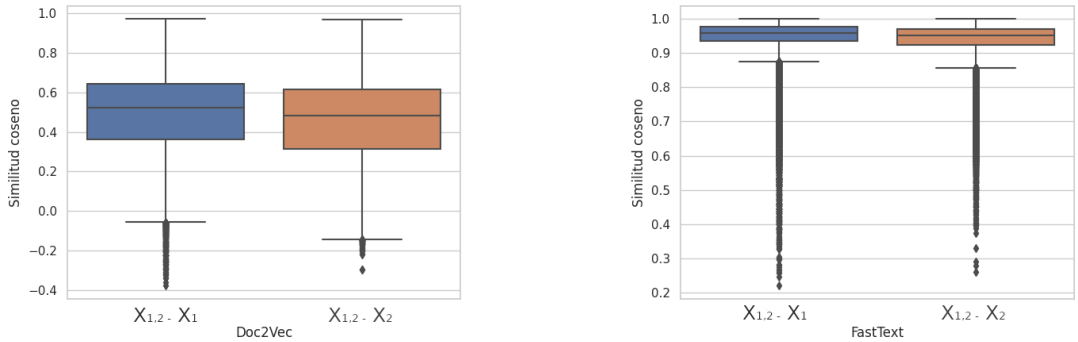
Figura 5.4: *BLEU score* obtenido por cada par de oración en Inglés y Español del corpus *Quora* por conjunto de datos.

De forma similar al corpus anterior, se puede ver que en promedio se obtiene un *BLEU score* menor en los pares de texto en Español que los de Inglés en todo conjunto de datos (5.4(a), 5.4(b) y 5.4(c)). Al hacer la prueba de *Wilcoxon* [25] se obtuvo un *p-value* de 0.0, por lo tanto se concluye igualmente que ambas poblaciones provienen de medias distintas o que tienen una distribución diferente.

5.2. Embeddings

5.2.1. Evaluación

Los resultados del experimento propuesto en 4.2.1 se pueden ver en la Figura 5.5, un diagrama de cajas con los resultados obtenidos de calcular la similitud coseno entre los vectores generados por los embeddings *Doc2Vec* y *FastText* entrenados.

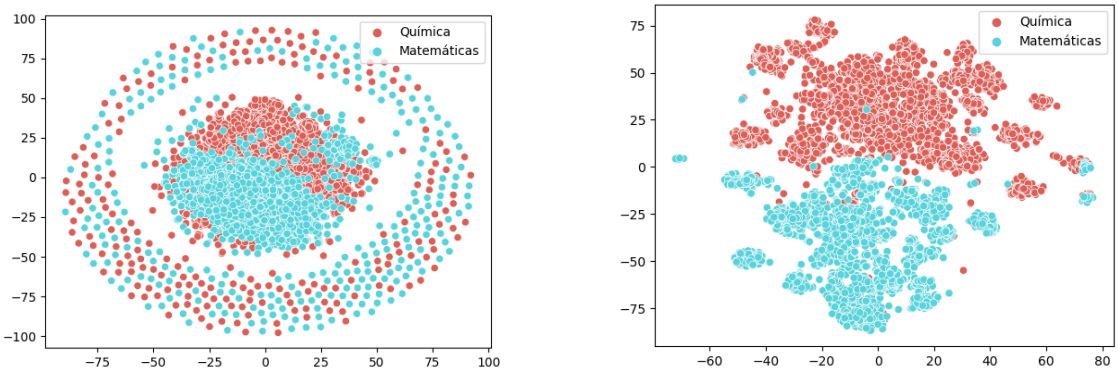


(a) Similitud coseno entre la combinación de vectores usando el embedding *Doc2Vec* previamente entrenado.

(b) Similitud coseno entre la combinación de vectores usando el embedding *FastText* previamente entrenado.

Figura 5.5: Evaluación de los *embeddings* generados.

Podemos ver que *FastText* 5.5(b) muestra un mejor desempeño sobre *Doc2Vec* 5.5(a) ya que obtiene, en promedio, una similitud arriba de 0.9 mientras que *Doc2Vec* en promedio obtiene una similitud por debajo de 0.6. Lo que significa que el embedding *FastText* está asignando a cada vector $\overrightarrow{X_1}$, $\overrightarrow{X_2}$ en una vecindad más cercana al vector $\overrightarrow{X_{1,2}}$ (aún sin las palabras eliminadas en $\overrightarrow{X_1}$ y $\overrightarrow{X_2}$).
Otra forma de ver y evaluar como ambos Embeddigs definen el espacio vectorial es aplicando una técnica de reducción de dimensionalidad *t-SNE* [26] y etiquetar cada oración del corpus *CBI-M-00-21* con su correspondiente área (*Matemáticas, Matemáticas, Ciencias y Tecnologías de la Información, Biomédica, Física y Energía*). En la Figura 5.6 se muestra un ejemplo aplicando *t-SNE* a los vectores definidos por los embeddings *Doc2Vec* y *FastText* con las oraciones de las áreas *Matemáticas* y *Química*.



(a) Representación con *Doc2Vec*. (b) Representación con *FastText*.

Figura 5.6: Visualización con *T-distributed Stochastic Neighbor Embedding* (t-SNE) de las oraciones del corpus *CBI-M-00-21* de las área de *Matemáticas* y *Química*.

Se puede ver que al reducir la dimensión de cada vector, el embedding *FastText* 5.6(b) define vectores por área más agrupados, mientras que *Doc2Vec* 5.6(a) define vectores por área más dispersos y difíciles de agrupar; lo que nos indica que el embedding *FastText* aprendió características contextuales por área de estudio que el embedding *Doc2Vec* no logró aprender.

5.2.2. Revisión de contexto

Usamos los cuatro embeddings (que generamos por cada corpus) para verificar qué tan diferentes eran los vectores que cada embedding define de una palabra cualquiera. Por ejemplo, en la Figura 5.7 se muestra la representación del vector generado usando la palabra "algoritmo" en cada uno de los embedding entrenado por cada uno de los corpus obtenido y aplicando *t-SNE* para representarlo.

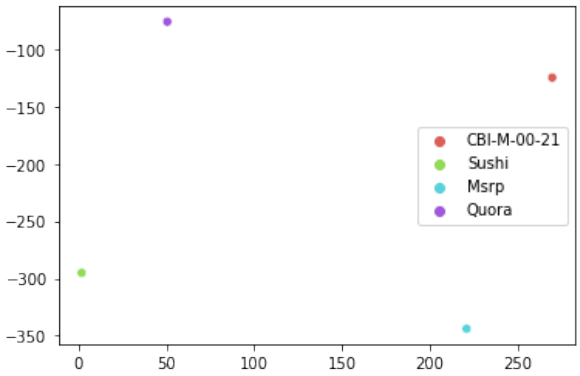
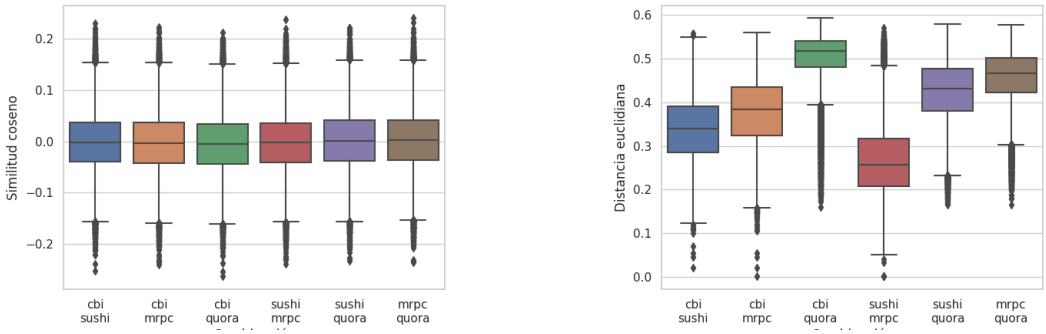


Figura 5.7: Representación vectorial con *t-SNE* de la palabra "algoritmo" usando cada embedding entrenado.

Podemos ver que cada embedding obtenido crea una representación vectorial distinta de la palabra "algoritmo" debido a que cada corpus contiene textos de distintos contextos y que, por lo tanto, la palabra de nuestro ejemplo ("algoritmo") se representa en otra región del espacio vectorial definido por cada embedding.
En la Figura 5.8 podemos ver diagramas de caja con la similitud coseno y distancia euclidiana entre los vectores que genera cada par de embeddings (la combinación de cada par) de cada palabra del vocabulario de los cuatro corpus.



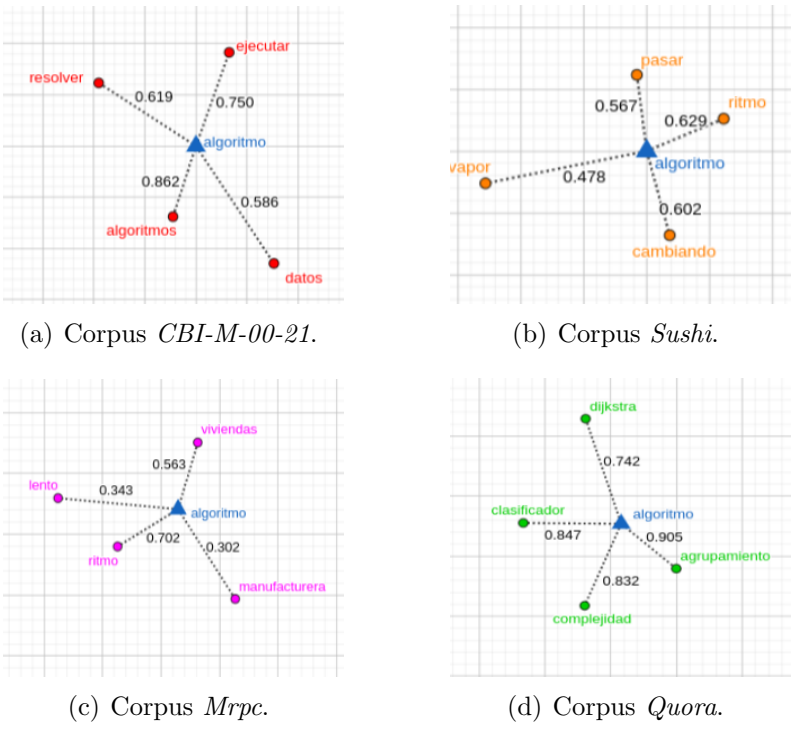
(a) Similitud coseno.

(b) Distancia euclidiana.

Figura 5.8: Comparación vectores de cada palabra entre cada par de embeddings.

Observamos que el promedio de la similitud coseno 5.8(a) entre los vectores que genera cada par de embedding (de cada palabra) es 0.0, lo que implica que cada par de vectores son linealmente independientes (o vectores ortogonales) por definición. Incluso vemos que existen valores de similitud negativa, es decir, vectores completamente opuestos.
También podemos ver que la distancia euclidiana 5.8(b) nos da información extra. Vemos, por ejemplo, que los embeddings creados con los corpus *CBI-M-00-21* y *Quora* son lo que definen vectores de palabras más dispersos.

Las distintas representaciones vectoriales de una palabra en función del contexto, pueden implicar una ventaja para los modelos de AM en términos del desempeño predictivo y capacidad de generalización. Más aún, las relaciones contextuales en vecindades del espacio vectorial (por palabra) pueden brindar información adicional y cambiar radicalmente la representación vectorial de una oración completa. Para evaluar las distintas relaciones contextuales entre todos los embeddings, examinamos (por cada palabra) las palabras más cercanas. Por ejemplo, en la Figura 5.9 mostramos los vectores de las palabras más cercanas al vector de la palabra "algoritmo", y el valor de similitud coseno obtenido entre los vectores que representan a cada palabra.



(a) Corpus *CBI-M-00-21*. (b) Corpus *Sushi*.

(c) Corpus *Mrpc*. (d) Corpus *Quora*.

Figura 5.9: Vectores en la vecindad del vector "algoritmo".

Podemos comprobar que cada embedding asocia distintas palabras a la palabra "algoritmo". Por ejemplo, para el caso del embedding entrenado con los textos del corpus *Sushi*, las palabras más cercanas a la palabra"algoritmo" son *ritmo, cambiando, pasar* y *vapor* que tienen más relación en un contexto de cocina o de recetas. En cambio, en el caso del embedding entrenado con el corpus *CBI-M-00-21*, las palabras más cercanas a la palabra "algoritmo" son *algoritmos, ejecutar, resolver* y *datos* que tienen más relación en un contexto científico.
Si ahora contamos (por cada palabra en el vocabulario de los cuatro corpus) el número de palabras en común dentro de las 10 más cercanas en la vecindad por cada par combinación de embedding, obtenemos lo siguiente (ver Figura 5.10):

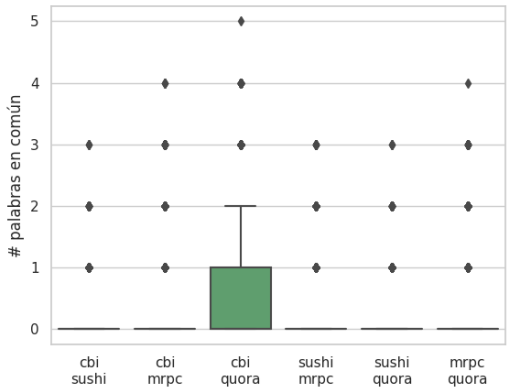


Figura 5.10: Número de palabras en común en vecindades de cada par de embedding.

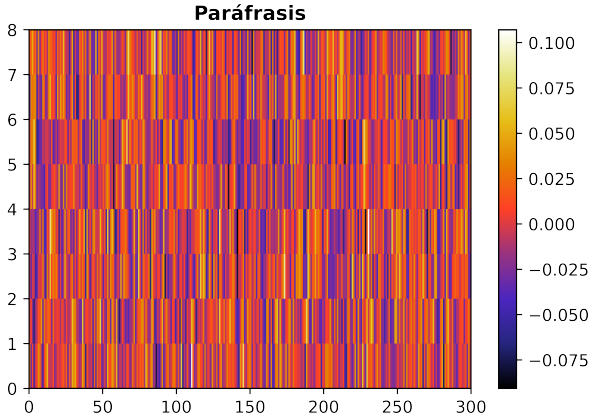
Podemos observar por los datos en la Figura 5.10 que en la mayoría de los casos cada embedding le asocia a cada palabra distintas palabras en la vecindad. Vemos que el número de palabras en común en promedio es 0. Lo que demuestra que cada embedding ha aprendido de forma distinta la representación vectorial de cada palabra y las relaciones contextuales entre ellas, en función del contexto que contienen todos los textos de cada corpus con los que los hemos entrenado.

Codificación o transformación

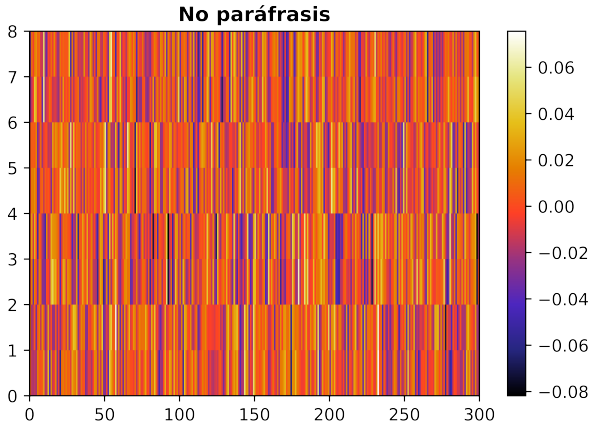
En la Figura 5.11 mostramos el resultado de codificar dos oraciones del corpus *Mrpc* que mostramos en la Tabla 5.6 usando el esquema propuesto en la Figura 4.12.

corpus	TXT 1	TXT 2	Clase
Mrpc	Las canciones están en oferta por 99 centavos cada una, o $9.99 para un álbum.	La compañía ofrecerá canciones por 99 centavos y álbumes por $9.95.	Paráfrasis
	El oficial de policía S.K. Tonapi dijo a Reuters que al menos 40 personas habían muerto y más de 100 habían resultado heridas.	Vasant Pitke, portavoz del Ministerio del Interior del Estado, dijo a Reuters que al menos 42 personas habían muerto y 112 habían resultado heridas.	No paráfrasis

Tabla 5.6: Ejemplos de pares de oraciones en Español del corpus *Mrpc* para visualización de contexto.



(a) Matriz de características de la muestra etiquetada como paráfrasis.



(b) Matriz de características de la muestra etiquetada como no paráfrasis.

Figura 5.11: Ejemplos de matrices de características de pares de oraciones.

Recordemos que cada par de renglones de la matriz de características que generamos corresponden a dos vectores de dimensión 300 que generamos por cada oración (TXT_1 y TXT_2 en la Tabla 5.6) usando los 4 embeddings. Podemos ver que la gama de colores difiere por cada par de renglones en ambos casos (5.11(a) y 5.11(b)) lo que nos muestra cómo cada embedding define un vector distinto de la misma oración.
Las Figuras en 5.11 representan un ejemplo de la codificación (o forma de transformación) del par de textos en una estructura numérica que tenga información relevante del problema que tratamos y que los modelos de AM puedan procesar para extraer patrones y, después. conocimiento.

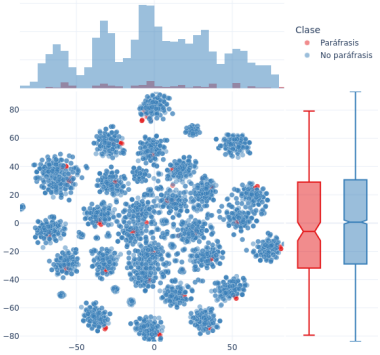
5.3. Balanceo de clases

En la Tabla 5.7 mostramos el resultado de crear muestras sintéticas de las clases minoritarias sobre el conjunto de entrenamiento de los corpus *Sushi, Mrpc* y *Quora*.

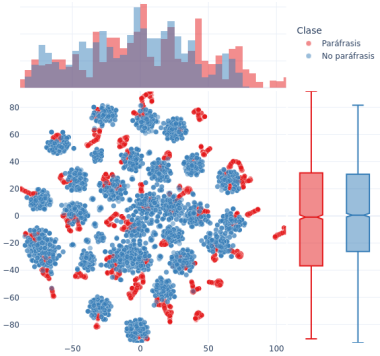
		Original		Sobre-muestreo	
Corpus	Conjunto	Paráfrasis	No paráfrasis	Paráfrasis	No paráfrasis
Sushi	Entrenamiento	215	4,215	**4,215**	4,215
Mrpc		2,202	1,059	2,202	**2,202**
Quora		139,306	245,042	**245,042**	245,042

Tabla 5.7: Conjuntos de entrenamiento balanceados de los corpus *Sushi, Mrpc* y *Quora*.

Como podemos ver ajustamos el número de muestras por cada clase de tal forma que logramos un balance, creando muestras sintéticas de la clase que tenía menor número de muestras. Por ejemplo, para el corpus *Sushi* el número de muestras de la clase minoritaria (paráfrasis) era 215 y, después de crear muestras sintéticas, se ajustó el número de muestras de esa misma clase a 4,215.
Por otro lado, podemos visualizar la distribución de muestras de los conjuntos de entrenamiento de cada corpus que hemos balanceado, aplicando la reducción de dimensionalidad con *t-SNE* [26]. En la Figura 5.12 mostramos los resultados que obtuvimos.

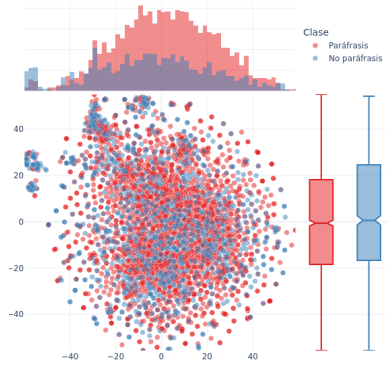


(a) Muestras del conjunto de entrenamiento original del corpus *Sushi*.

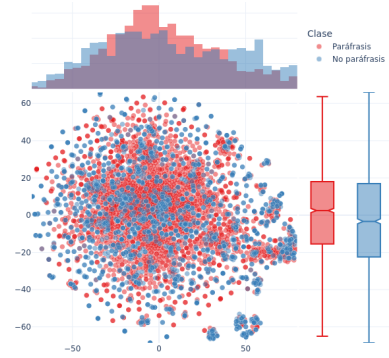


(b) Muestras del conjunto de entrenamiento con sobre muestreo del corpus *Sushi*.

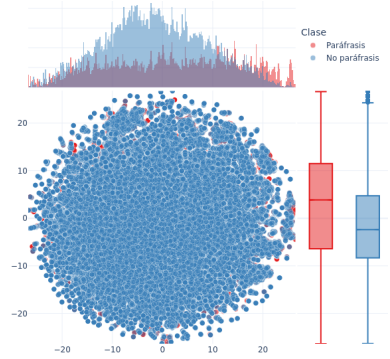
Figura 5.12: Distribución de muestras del conjunto de entrenamiento original y con sobre muestreo de los corpus *Sushi, Mrpc* y *Quora*.



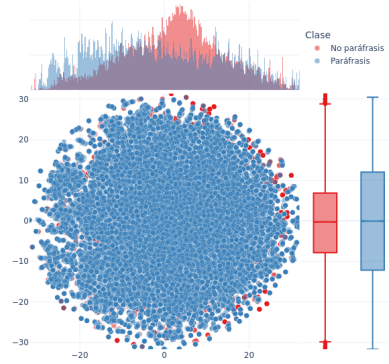
(a) Muestras del conjunto de entrenamiento original del corpus *Mrpc*.



(b) Muestras del conjunto de entrenamiento con sobre muestreo del corpus *Mrpc*.



(c) Muestras del conjunto de entrenamiento original del corpus *Quora*.



(d) Muestras del conjunto de entrenamiento con sobre muestreo del corpus *Quora*.

Figura 5.12: Distribución de muestras del conjunto de entrenamiento original y con sobre muestreo de los corpus *Sushi, Mrpc* y *Quora*.

En general podemos ver que la presencia de muestras de la clase minoritaria en cada corpus se representa de forma equilibrada con respecto a la clase mayoritaría. Retomando el ejemplo del corpus *Sushi*, podemos ver que la clase minoritaria es *paráfrasis* y se representa con puntos de color rojo en las Figura 5.12(a) y 5.12(b). Sin embargo, en la Figura 5.12(b) la presencia de puntos rojos es más significativa ya que ésta corresponde al resultados de crear muestras sintéticas. De la misma forma ocurre con las Figuras 5.13(a) y 5.13(b) en el caso del corpus *Mrpc* y 5.13(c) y 5.13(d) para el caso del corpus *Quora*. También podemos ver que las muestras sintéticas se mantienen en una vecindad a las originales y que mantienen cierto patrón en la distribución con respecto al conjunto original.

5.4. Modelos de AM seleccionados

5.4.1. Desempeño

En esta apartado presentamos el desempeño de los modelos y técnicas de AM en términos de rendimiento predictivo de detección de paráfrasis, sobre cada unos de los corpus que hemos seleccionado como fuentes de datos principal. Todos los resultados que mostramos se obtuvieron usando el conjunto de *prueba* de cada corpus mostrado en las Tablas 5.2, 5.3, 5.4 y 5.5 que corresponden a los corpus *CBI-M-00-21, Sushi, Mrpc* y *Quora*, respectivamente.
Primero presentamos los resultados de cada conjunto de datos reportando las métricas de evaluación especificadas en 4.6 usando los modelos entrenados con los conjuntos de *entrenamiento* y *validación* correspondientes a cada corpus. Después mostraremos un panorama general del desempeño de todos los modelos seleccionados.

5.4.1.1. Corpus *Sushi*

En la Tabla 5.8 podemos ver el desempeño de todos los modelos entrenados y evaluados sobre el corpus *Sushi* usando el esquema de experimentos 4.5.

Modelo	Accuracy	Precision	Recall	F_1
Bayes Ingenuo	0.681	0.513	0.564	0.457
Regresión Logística	0.712	0.514	0.565	0.470
MSV	**0.986**	**0.934**	0.901	**0.917**
Perceptron	0.884	0.558	0.624	0.573
CNN	0.969	0.783	**0.936**	0.841
LSTM	0.931	0.659	0.756	0.694
Ensamble	0.973	0.855	0.817	0.835

Tabla 5.8: Desempeño de los modelos de AM sobre el corpus *Sushi*.

Observamos que el modelo que obtuvo el mejor desempeño fue la Maquina de Soporte Vectorial (MSV) con un score $F_1 = 0.917$. En este modelo usamos un parámetro de regularización $C = 0.90$, el *kernel rbf (Radial Basis Function)* con un valor $gamma = 0.05$ y una Tolerancia para el criterio de paro de 0.001.
Para validar que el modelo de MSV mostró un mejor desempeño en la Figura 5.13 mostramos la curva ROC con el desempeño de todos los modelos.

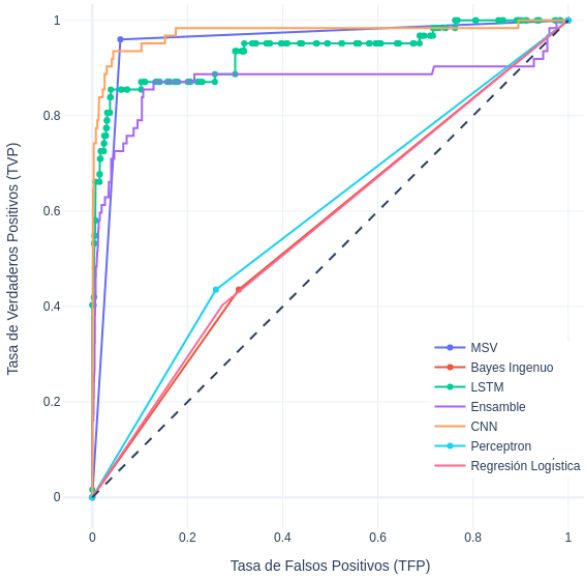


Figura 5.13: Curva *ROC* con el desempeño de los modelos de AM sobre el corpus *Sushi*.

Podemos ver que el modelo que se acerca más a la esquina superior izquierda es el modelo de *MSV*, lo que confirma que es el modelo que mejor desempeño tiene en rendimiento predictivo de detección de paráfrasis. La matriz de confusión asociada a las predicciones del mejor modelo se puede ver en la Tabla 5.9.

		Valor real	
		Paráfrasis	No paráfrasis
Predicción	Paráfrasis	50	12
	No paráfrasis	7	1309

Tabla 5.9: Matriz de confusión obtenida para el conjunto de *prueba* del corpus *Sushi* usando el modelo *MSV*.

Podemos notar por la información en la Tabla 5.9 que el modelo de *MSV* se desempeña mejor clasificando las muestras de la clase *no paráfrasis*, que en perspectiva es la clase con más muestras, lo que incrementó el valor de exactitud *Accuracy* de la Tabla 5.8. Sin embargo, el valor F_1 de 0.917 (por definición) nos indica de forma más precisa el desempeño del modelo clasificando las clases *paráfrasis* y *no paráfrasis*.
Algunas de las muestras correctamente e incorrectamente clasificadas de las matriz de confusión 5.9 las podemos ver en la Tabla 5.10.

Texto 1	Texto 2	Clase original	Predicción
Los cultivadores de arroz a lo largo del río Mekong descubrieron la forma de conservar durante largos períodos de tiempo el pescado en arroz cocido, donde la fermentación proporciona diversos compuestos químicos que aromatizan y proporcionan sabor: ácido láctico y ácido acético (vinagre).	El método para conservar sazonado el pescado solía ser factible de realizarse en los pueblos que existían en el río Mekong, actualmente se le conoce como shiokara en Japón, es una tipo de salsa de pescado.	No paráfrasis	Paráfrasis
A pesar de todo el sabor del pescado es agradable para el consumo humano (con toques ácidos).	Después de la fermentación el pescado se sacaba del arroz, se limpiaba muy bien, se cortaba en pedazos y estaba listo para servirse teniendo un gusto ligeramente ácido y bastante apetitoso.	Paráfrasis	No paráfrasis
En Japón este plato proliferó bajo el concepto de una nueva forma de conservar el pescado.	El método para conservar sazonado el pescado solía ser factible de realizarse en los pueblos que existían en el río Mekong, actualmente se le conoce como shiokara en Japón, es una tipo de salsa de pescado.	No paráfrasis	No paráfrasis
En Japón se emplea una variedad de arroz denominada koshihikari su capacidad para compactarse se encuentra en la tipología de los arroces glutinosos.	En Japón usan un tipo de arroz que se llama koshihikari, cuya mayor característica es que se compacta; es decir que es arroz glutinoso.	Paráfrasis	Paráfrasis

Tabla 5.10: Muestras del corpus *Sushi* que fueron correctamente e incorrectamente clasificadas.

5.4.1.2. Corpus *Mrpc*

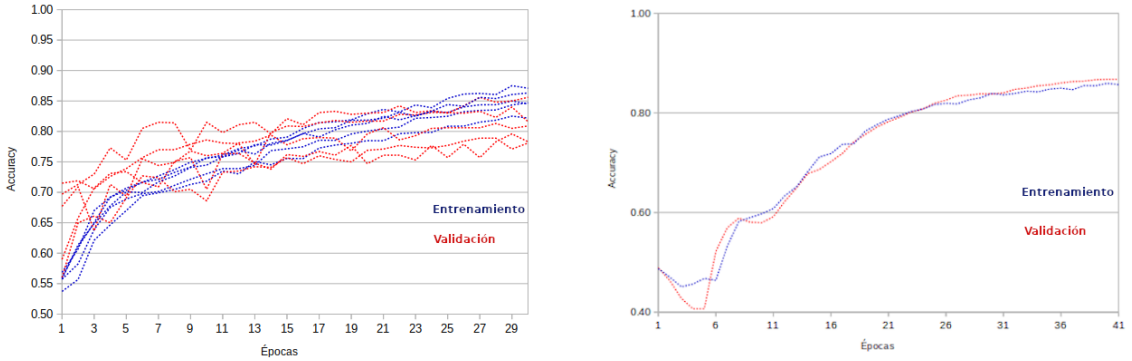
En la Tabla 5.11 podemos ver el desempeño de todos los modelos entrenados y evaluados sobre el corpus *Mrpc* usando el esquema de experimentos 4.5.

Modelo	Accuracy	Precision	Recall	F1
Bayes Ingenuo	0.514	0.489	0.488	0.484
Regresión Logística	0.621	0.576	0.576	0.576
MSV	0.655	0.589	0.566	0.565
Perceptron	0.634	0.569	0.573	0.571
CNN	0.626	0.559	0.550	0.550
LSTM	0.644	0.585	0.580	0.582
Ensamble	**0.866**	**0.847**	**0.858**	**0.852**

Tabla 5.11: Desempeño de los modelos de AM sobre el corpus *Mrpc*.

En este caso podemos ver que el modelo que obtuvo el mejor desempeño fue el Ensamble propuesto obteniendo un score $F_1 = 0.852$. El número de sub modelos del ensamble

fue de $k = 5$. El desempeño del entrenamiento de cada sub modelo en términos de la exactitud (*accuracy*) y con respecto al conjunto de entrenamiento y validación se puede ver en la Figura 5.14.



(a) Desempeño del entrenamiento de cada sub modelo.

(b) Desempeño del entrenamiento de la red final del ensamble

Figura 5.14: Desempeño del entrenamiento del ensamble sobre el corpus *Mrpc*.

Podemos ver en la Figura 5.14(a) que cada sub modelo del ensamble tiene un desempeño de entrenamiento distinto dado que cada uno trata con muestras distintas del conjunto de entrenamiento. Con respecto al conjunto de validación notamos que existen variaciones más notables por cada sub modelo. En el desempeño de entrenamiento de la red neuronal final que usa como características las predicciones de cada sub modelo (Figura 5.14(b)), observamos que el desempeño es complicado al inicio pero la red se logra ajustar a las predicciones de todos los modelos con respecto a todos los datos del conjunto de entrenamiento, lo que nos indica que se ha aumentado la capacidad de generalizar sobre todos los datos.

Para validar que el modelo Ensamble propuesto mostró un mejor desempeño, mostramos en la Figura 5.15 la curva *ROC* con el desempeño de todos los modelos.

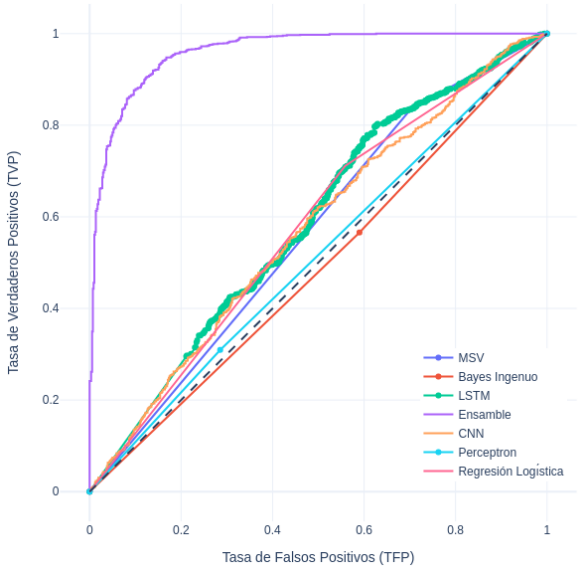


Figura 5.15: Curva *ROC* con el desempeño de los modelos de AM sobre el corpus *Mrpc*.

Observemos que el modelo que se acerca más a la esquina superior izquierda es el modelo de Ensamble que hemos propuesto, lo que confirma que es el modelo que mejor desempeño tiene en rendimiento predictivo de detección de paráfrasis. La matriz de confusión asociada a las predicciones del mejor modelo se puede ver en la Tabla 5.12.

		Valor real	
		Paráfrasis	No paráfrasis
Predicción	Paráfrasis	1111	36
	No paráfrasis	140	438

Tabla 5.12: Matriz de confusión obtenida para el conjunto de *prueba* del corpus *Mrpc* usando el modelo Ensamble.

En este caso podemos notar por la información en la matriz de confusión 5.12 que el modelo de Ensamble se desempeña mejor clasificando las muestras de la clase *paráfrasis*. En el caso de las muestras de la clase *no paráfrasis* observamos que son en la mayoría clasificadas correctamente, pero también una buena proporción del total de muestras están siendo clasificadas incorrectamente. Lo que también explica el valor F_1 de 0.852 de la Tabla 5.11.
Algunas de las muestras correctamente e incorrectamente clasificadas de las matriz de confusión 5.12 las podemos ver en la Tabla 5.13. Agregamos también el texto en Inglés para comprobar la traducción que realizamos.

Texto 1	Texto 2	Clase original	Predicción
Garner dijo que el autoproclamado alcalde de Bagdad, Mohammed Mohsen al-Zubaidi, fue liberado después de dos días bajo custodia de la coalición. –INGLÉS– *Garner said the self-proclaimed mayor of Baghdad, Mohammed Mohsen al-Zubaidi, was released after two days in coalition custody.*	Garner dijo que el autoproclamado alcalde de Bagdad, Mohammed Mohsen Zubaidi, fue liberado 48 horas después de su detención a finales de abril. –INGLÉS– *Garner said self-proclaimed Baghdad mayor Mohammed Mohsen Zubaidi was released 48 hours after his detention in late April.*	No paráfrasis	Paráfrasis
La entrevista llegó un día después de un informe en la prensa británica de que había sido detenido. –INGLÉS– *The interview came a day after a report in the British press that he had been taken into custody.*	La entrevista del jueves llegó un día después de que el Daily Mirror de Gran Bretaña informara que Sahhaf había sido detenido. –INGLÉS– *The interview Thursday came a day after Britain's Daily Mirror reported Sahhaf had been taken into custody.*	Paráfrasis	No paráfrasis
Durante un partido de gritos en 1999, Carolyn le dijo a John que todavía estaba durmiendo con Bergin. –INGLÉS– *During a screaming match in 1999, Carolyn told John she was still sleeping with Bergin.*	Ella, a su vez, ocasionalmente le dijo a John que todavía estaba durmiendo con un ex-novio, Baywatch Hunk Michael Bergin. –INGLÉS– *She, in turn, occasionally told John that she was still sleeping with an ex-boyfriend, Baywatch hunk Michael Bergin.*	No paráfrasis	No paráfrasis
La mujer había aceptado testificar después de recibir inmunidad protegiéndola de castigo por mentir y otras violaciones del código de honor de la academia. –INGLÉS– *The woman had agreed to testify after receiving immunity protecting her from punishment for lying and other violations of the academy's honor code.*	La defensa dijo que la mujer testificó bajo un acuerdo de inmunidad protegiéndola de castigo por mentir y otras violaciones del código de honor de la academia. –INGLÉS– *The defense said the woman testified under an immunity deal protecting her from punishment for lying and other violations of the academy's honor code.*	Paráfrasis	Paráfrasis

Tabla 5.13: Muestras del corpus *Mrpc* que fueron correctamente e incorrectamente clasificadas.

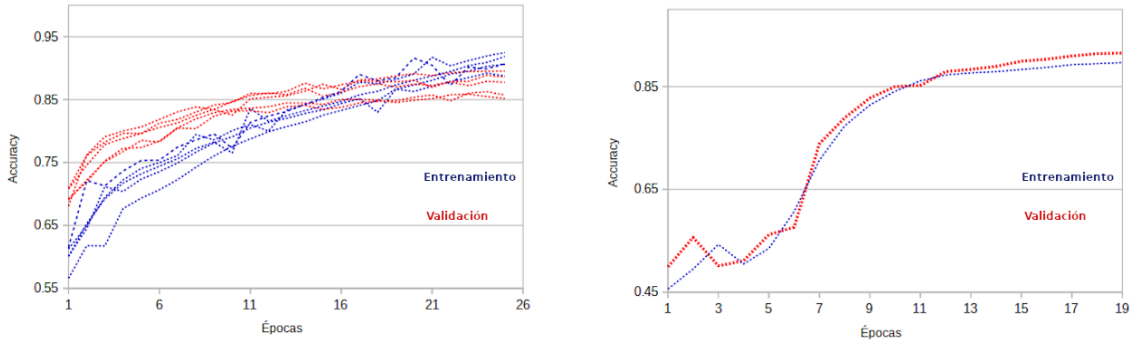
5.4.1.3. Corpus *Quora*

En la Tabla 5.14 podemos ver el desempeño de todos los modelos entrenados y evaluados sobre el corpus *Quora* usando el esquema de experimentos 4.5.

Modelo	Accuracy	Precision	Recall	F1
Bayes Ingenuo	0.639	0.640	0.639	0.638
Regresión Logística	0.706	0.706	0.706	0.706
MSV	0.744	0.748	0.744	0.743
Perceptron	0.642	0.638	0.640	0.641
CNN	0.766	0.767	0.766	0.766
LSTM	0.785	0.785	0.785	0.785
Ensamble	**0.946**	**0.945**	**0.946**	**0.946**

Tabla 5.14: Desempeño de los modelos de AM sobre el corpus *Quora*.

Observamos que en el caso del corpus *Quora*, el modelo que obtuvo el mejor desempeño fue el Ensamble que hemos propuesto. El número de sub modelos del ensamble fue de $k = 5$. El desempeño del entrenamiento de cada sub modelo en términos de la exactitud (*accuracy*) y con respecto al conjunto de entrenamiento y validación se puede ver en la Figura 5.16.



(a) Desempeño del entrenamiento de cada sub modelo.

(b) Desempeño del entrenamiento de la red final del ensamble

Figura 5.16: Desempeño del entrenamiento del ensamble sobre el corpus *Quora*.

Podemos ver por la Figura 5.16(b) que la red neuronal a la salida del ensamble se ajusta bien a las predicciones de los sub modelos dentro del ensamble.
Para validar que el modelo de Ensamble es el que se desempeña de mejor forma podemos ver en la Figura 5.17 la curva *ROC* con el desempeño de todos los modelos.

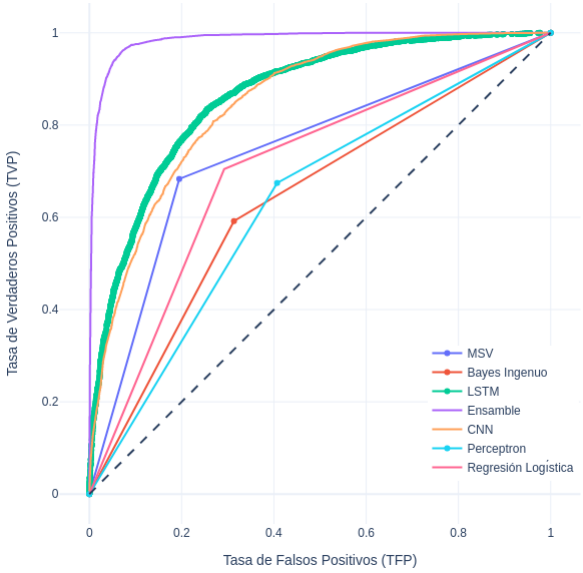


Figura 5.17: Curva *ROC* con el desempeño de los modelos de AM sobre el corpus *Quora*.

Confirmamos que el modelo ensamble propuesto se desempeña mejor ya que es el que más se aproxima a la esquina superior izquierda en la Figura 5.17. La matriz de confusión asociada a las predicciones del mejor modelo se puede ver en la Tabla 5.15.

		Valor real	
		Paráfrasis	No paráfrasis
Predicción	Paráfrasis	4,748	252
	No paráfrasis	312	4,688

Tabla 5.15: Matriz de confusión obtenida para el conjunto de *prueba* del corpus *Quora* usando el modelo Ensamble.

Para el caso del corpus *Quora* podemos advertir por la información en la matriz de confusión 5.15 que el modelo se desempeña proporcionalmente de forma correcta sobre cada clase (*paráfrasis* y *no paráfrasis*).
Algunas de las muestras correctamente e incorrectamente clasificadas de las matriz de confusión 5.15 las podemos ver en la Tabla 5.13. Agregamos también el texto en Inglés para comprobar la traducción que realizamos.

Texto 1	Texto 2	Clase original	Predicción
¿Cuáles son las mejores estrategias de marketing offline para la promoción de aplicaciones? –INGLÉS– *What are the best offline marketing strategies for app promotion ?*	¿Cuáles son las mejores estrategias de marketing de aplicaciones? –INGLÉS– *What are the best app marketing strategies ?*	No paráfrasis	Paráfrasis
¿Cuál es su nueva resolución de año 2017 para mejorar su rutina de la vida diaria? –INGLÉS– *What 's your new year 2017 resolution to improve your daily life routine ?*	¿Cuál puede ser mi resolución de año nuevo para 2017? –INGLÉS– *What can be my new year resolution for 2017?*	Paráfrasis	No paráfrasis
¿Cómo puedo escribir una carta al banco para cerrar una cuenta bancaria? –INGLÉS– *How do I write a letter to the bank to close bank account?*	¿Cómo puedo escribir una carta para cambiar el candidato de una cuenta bancaria conjunta? –INGLÉS– *How do I write a letter to change the nominee of a joint bank account ?*	No paráfrasis	No paráfrasis
¿Quora gana dinero, si sí, entonces cómo? –INGLÉS– *Does Quora make money , if yes then how ?*	¿Cómo hace Quora para ganar dinero como negocio sin publicidad? –INGLÉS– *How does Quora make any money as a business without any advertising ?*	Paráfrasis	Paráfrasis

Tabla 5.16: Muestras del corpus *Quora* que fueron correctamente e incorrectamente clasificadas.

5.4.1.4. Corpus *CBI-M-00-21*

En la Tabla 5.17 podemos ver el desempeño de todos los modelos entrenados y evaluados sobre el corpus *CBI-M-00-21* usando el esquema de experimentos 4.5.

Modelo	Accuracy	Precision	Recall	F1
Bayes Ingenuo	0.553	0.558	0.553	0.542
Regresión Logística	0.748	0.750	0.748	0.748
MSV	0.822	0.830	0.822	0.820
Perceptron	0.672	0.674	0.673	0.672
CNN	0.961	0.962	0.962	0.962
LSTM	**0.970**	0.966	**0.975**	**0.970**
Ensamble	**0.970**	**0.970**	0.970	**0.970**

Tabla 5.17: Desempeño de los modelos de AM sobre el corpus *CBI-M-00-21*.

Observamos que en el caso del corpus *CBI-M-00-21*, el modelo que obtuvo el mejor desempeño fue el Ensamble que hemos propuesto. Sin embargo, la red neuronal

recurrente *LSTM* muestra un desempeño similar en métricas como la exactitud (*accuracy*) y valor F_1. En la Figura 5.18 podemos ver la curva *ROC* con el desempeño de todos los modelos.

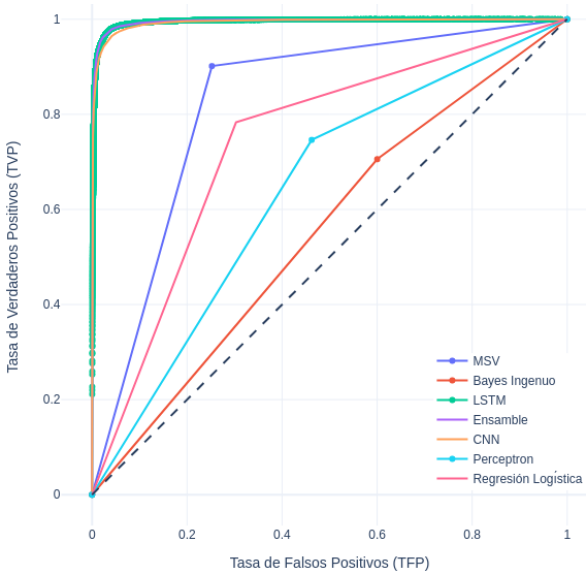


Figura 5.18: Curva *ROC* con el desempeño de los modelos de AM sobre el corpus *CBI-M-00-21*.

Observamos que el modelo de red neuronal *LSTM* ciertamente presenta un desempeño similar al del ensamble que hemos propuesto ya que ambos son los que más se aproximan a la esquina superior izquierda de la gráfica. Sin embargo, si consideramos la complejidad de ambos modelos, el modelo de red neuronal *LSTM* puede ser el mejor por complejidad.
La matriz de confusión asociada a las predicciones del modelo de red neuronal *LSTM* se puede ver en la Tabla 5.18.

		Valor real	
		Paráfrasis	No paráfrasis
Predicción	Paráfrasis	87,617	2,680
	No paráfrasis	2,751	87,546

Tabla 5.18: Matriz de confusión obtenida para el conjunto de *prueba* del corpus *CBI-M-00-21* usando el modelo de red neuronal *LSTM*.

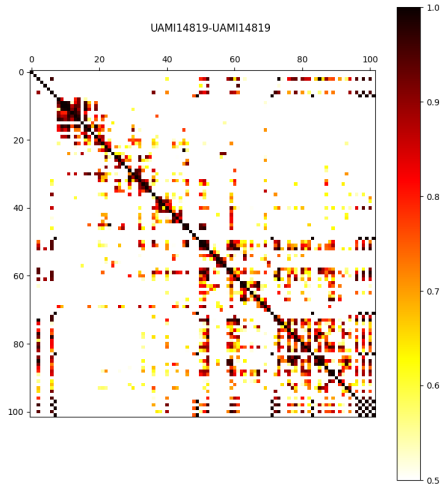
Algunas de las muestras correctamente e incorrectamente clasificadas de las matriz de confusión 5.18 las podemos ver en la Tabla 5.19.

Texto 1	Texto 2	Clase original	Predicción
Estos presentan menor actividad que los catalizadores del primer grupo, sin embargo la presencia incrementa la estabilidad del catalizador	La actividad se incrementa y la reducción de la temperatura de los catalizadores observándose se estabiliza a un máximo dentro del catalizador	No paráfrasis	Paráfrasis
Se muestra la magnetización en función campo magnético aplicado	El imán se muestra en el campo magnético aplicado	Paráfrasis	No paráfrasis
Los catalizadores de germanio sobre alúmina no presentaron actividad de hidrogenación	Los cuales presentaron una menor actividad que los materiales de la serie preparada	No paráfrasis	No paráfrasis
Se describió la metodología para la determinación de la frontera entre las regiones de solución analítica y numérica	La metodología para la identificación de la frontera entre las regiones de solución analítica y cuantitativa se ha descrito	Paráfrasis	Paráfrasis

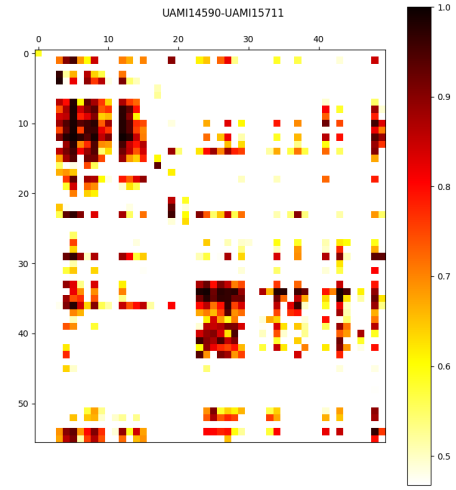
Tabla 5.19: Muestras del corpus *CBI-M-00-21* que fueron correctamente e incorrectamente clasificadas.

Mapas de calor

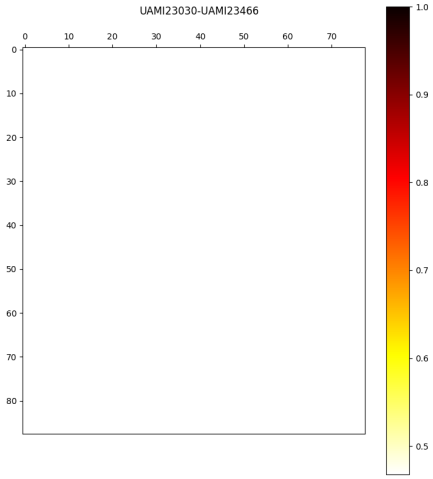
Como mencionamos anteriormente, hemos usado el modelo que presentó el mejor desempeño en términos de rendimiento predictivo de detección de paráfrasis para generar mapas de calor entre pares de documentos y así ver cómo el modelo se desempeña comparando y asignando un valor de similitud contextual a cada combinación de páginas entre los documentos. En la Figura 5.19 se pueden ver distintas comparaciones entre pares de documentos. En la Figura 5.19(a) comparamos dos documentos iguales, podemos ver que en la diagonal el modelo obtiene los valores máximos (de color negro) debido a que en la diagonal se comparan la misma página. Sin embargo, podemos encontrar otras regiones fuera de la diagonal con valores de similitud contextual altas, indicando que el autor ha escrito ideas similares a lo largo de todo el documento.



(a) Mapa de calor para par de textos iguales.



(b) Mapa de calor para par de textos distintos pero que tratan temas similares.



(c) Mapa de calor para par de textos completamente distintos.

Figura 5.19: Mapas de calor entre pares de documentos del repositorio *TESIUAMI* generados con el modelo de red neuronal *LSTM*.

En la Figura 5.19(b) comparamos dos documentos distintos pero ambos tienen la característica de ser de la misma área de estudio y que tratan temas similares, se puede ver que el modelo muestra regiones en la que hay una similitud contextual relevante. Y por último en la Figura 5.19(c) comparamos dos documentos de distinta área de estudios y que tratan temas completamente distintos. Podemos ver que en este último caso el modelo no ha detectado ningún caso de similitud contextual importante.

CAPÍTULO 6

DISCUSIÓN

Planteamos el problema de detección de paráfrasis como un problema de clasificación binaria y aplicamos técnicas de Aprendizaje Maquinal (AM) para atacar el problema desde una perspectiva computacional. En esta sección discutiremos el significado, la importancia y relevancia de todos los resultados descritos en la sección anterior.

6.1. Creación de un nuevo corpus de paráfrasis

De acuerdo con las hipótesis de lograr crear un corpus de paráfrasis sobre los textos que se encuentran en el repositorio *TESIUAMI* y automatizar la generación de paráfrasis en el lenguaje Español, presentamos al corpus *CBI-M-00-21*. Un corpus de paráfrasis creado de forma automática por medio de traducción de segmentos extraídos de documentos en formato PDF dentro del repositorio *TESIUAMI*. Resaltando que solo para las tesis de nivel maestría, de los años 2000 a 2021 y de las áreas de estudio *Matemáticas, Química, Ciencias y Tecnologías de la Información, Física, Biomédica* y *Energía*, obtuvimos un total de 601,982 (Tabla 5.1) segmentos a los que, a cada uno, le asignamos otro texto que representa la paráfrasis y no paráfrasis.
Los resultados de evaluar la paráfrasis generada por medio *BLEU score* (Figura 5.1) sugieren que la paráfrasis que generamos contiene cambios léxico-sintácticos relevantes en cada par (segmento-paráfrasis) debido al valor promedio de 0.2 del *BLEU score*, similar al *BLEU score* de 0.25 obtenido en [13] con pares de traducciones realizadas por personas profesionales.
El experimento proporciona una nueva perspectiva de la relación entre la creación de paráfrasis de forma automática y la de personas expertas en el tema, ya que, la primera, agrega la capacidad de parafrasear grandes cantidades de textos en un periodo relativamente corto y con una calidad similar. Sin embargo, existen algunas implicaciones prácticas en la forma de crear paráfrasis de forma automática.

Principalmente que esta generación de paráfrasis es costosa en términos de complejidad computacional debido a que un solo modelo *MariaMT* requiere de un hardware especifico que incluya *GPU's* para acelerar el tiempo de generación de paráfrasis. El hardware que usamos en nuestros experimentos (Sección 4.7) resultó ser funcional pero limitado, de modo que en estudios futuros se debería considerar usar un hardware más potente para el uso de estos modelos.

6.2. Representación vectorial de textos

La selección de dos modelos embedding y su respectivo entrenamiento, confirmaron la hipótesis en relación a lograr abstraer estructuras semánticas de segmentos discursivos desde una perspectiva vectorial. Particularmente usando el corpus *CBI-M-00-21*, los resultados muestran que el esquema de entrenamiento de los modelos embedding propuesto en 4.2.1 aportan relaciones, en el espacio vectorial definido, de palabras que están contextualmente relacionadas. Este análisis apoya la teoría de que entrenar modelos embedding usando secuencias ordenadas de palabras, aprendiendo patrones en función de relaciones entre vecindades es efectiva y practica para algunos modelos de AM. Sin embargo, los resultados no encajan para todo modelo embedding. Encontramos que el modelo embedding *FastText* define relaciones de distancia (en el espacio vectorial definido) más cercanas entre palabras contextualmente relacionadas (Figura 5.5). También encontramos que, en un nivel de contexto, *FastText* logra definir regiones separables entre oraciones dentro de una misma área de estudio (Figura 5.6). Este experimento muestra que el enfoque de n-*gramas* de caracteres de palabras, y el análisis morfológico de las palabras, representa una mejora para un embedding en términos del desempeño en crear relaciones semánticas de los textos en un espacio vectorial. Por otro lado, la generalización de los resultados está limitada por los corpus que hemos usado como base de datos experimental. Los estudios futuros en este rubro deben incluir el entrenamiento de embeddings sobre conjuntos de datos más grandes (en idioma Español) y de contexto general.

6.2.1. Corpus adicionales y función del contexto

Por el repositorio *TESIUAMI* en que hemos extraído texto y la naturaleza misma de los textos obtenidos, consideramos al corpus *CBI-M-00-21* como un corpus de contexto *científico/académico*. En cambio, los corpus que hemos estudiado de forma complementaria en nuestra investigación tienen un contexto distinto. Los atributos de cada corpus, en términos contextuales, nos dieron la oportunidad de crear estructuras de características numéricas con variaciones de contexto. Los resultados de analizar la definición vectorial creada por cada embedding y comparar las palabras en común de cada palabra en el vocabulario sugieren que dichas variaciones de contexto son importantes y se ven reflejadas en el espacio vectorial definido (ver Figura 5.7). Especialmente, el experimento proporciona una nueva perspectiva de la relación entre

la importancia del contexto en los corpus y la definición de vecindades de palabras en un espacio vectorial creado por un modelo embedding. Sin embargo, los estudios futuros deben considerar el impacto en la viabilidad de un sistema informático con una capa embedding de 4 o más modelos distintos.

6.3. Desempeño de técnicas de AM

En el marco de la hipótesis sobre configurar y entrenar modelos inspirados en técnicas de AM que sean capaces de detectar la paráfrasis, los resultados indican que las técnicas de AM que hemos propuesto obtienen un desempeño destacado sobre cada uno de los corpus empleados como base experimental. Encontramos que el enfoque de aprendizaje profundo con los modelos de redes neuronales convolucionales (CNN) y recurrentes ($LSTM$) obtienen un buen desempeño, en general, sobre otras técnicas estándar del aprendizaje maquinal. Sin embargo, como vimos en el caso del corpus *Sushi* en el que el modelo con mejor desempeño fue el de Maquina de Soporte Vectorial, un modelo no puede ser la mejor opción en todo problema. Los datos obtenidos sobre el buen desempeño del modelo de MSV sobre el corpus *Sushi* contribuyen a una comprensión más clara de la relación que existe entre la elección de un algoritmo inspirado en técnicas de AM y la distribución espacio-vectorial de las muestras a procesar, debido al claro patrón mostrado en las Figuras 5.12(a) y 5.12(b) de las muestras por clase del corpus *Sushi*.
Sin embargo, los resultados también muestran que el desempeño de técnicas inspiradas en el aprendizaje profundo (redes neuronales) superan, en general, a las técnicas básicas de AM (ver Tablas 5.8, 5.11, 5.14 y 5.17). Especialmente el ensamble propuesto en 4.4.1.4, mostró un desempeño sobresaliente sobre los corpus *Mrpc* y *Quora* sugiriendo que el enfoque de mezcla de expertos para reducir la inhabilidad del modelo de generalizar sobre nuevos datos es relevante y futuros estudios en esta línea de investigación deberían incluir el análisis de otros tipos de ensamble, mezclando técnicas de distinta naturaleza y distintas arquitecturas de ensamblaje.

CAPÍTULO 7

CONCLUSIONES

Esta investigación tuvo como objetivo obtener un sistema capaz de detectar casos relevantes de similitud semántica entre pares de textos o documentos completos de forma eficiente, empleando un enfoque computacional y aplicando técnicas inspiradas en *inteligencia artificial* y un corpus de paráfrasis de contexto científico creado de forma automatizada usando el repositorio *TESIUAMI* como base de datos minable. En lo general, los resultados obtenidos indican que el enfoque computacional, la aplicación de técnicas de *inteligencia artificial* y la metodología empleada en este proyecto de investigación son capaces de generar sistemas para detectar paráfrasis de forma eficaz y eficiente sobre textos académicos y de contexto científico. En lo particular y bajo el marco de las hipótesis planteadas en esta investigación hacemos las siguientes anotaciones:

Creación de un corpus de paráfrasis de forma automática

Dada la problemática de no contar con un corpus de paráfrasis en idioma Español lo suficientemente extenso para aplicar técnicas de minería de datos, nosotros detallamos el proceso que diseñamos para crear un corpus de paráfrasis de forma automatizada usando una cascada de modelos de traducción. Con base en un análisis cuantitativo de las diferencias *léxico-sintácticas* entre los pares *texto-traducción* que generamos, se puede concluir que la calidad de paráfrasis asociada a cada segmento discursivo extraído del repositorio *TESIUAMI* es buena, y por lo tanto el corpus creado contiene muestras que pueden representar una forma del fenómeno lingüístico de paráfrasis. Por otra parte, el repositorio *TESIUAMI* contiene más documentos que podemos procesar y agregar al corpus que creamos para extenderlo. El alcance del corpus en esta investigación alcanzó a cubrir las áreas principales dentro de la división de CBI (*Ciencias Básica e Ingeniería*) logrando un total de 1,203,964 de pares de textos en idioma Español clasificados con la clase *paráfrasis* y *no paráfrasis*. Sin embargo, podemos extenderlo usando nuestra técnica a áreas biológicas y de la salud o a áreas

sociales y de humanidades. Creemos que las diversas variantes por área de estudio y grado académico de los textos pueden crear un corpus más robusto y general que sirva para futuras investigaciones relacionadas con el *procesamiento del lenguaje natural*.

Representaciones vectoriales de textos y variantes contextuales

Evaluamos dos modelos distintos para representar palabras, oraciones y textos completos y después de analizar los resultados concluimos que el modelo que agrega un análisis morfológico a la palabras generando el vector de una palabra como la suma de vectores de n-gramas de caracteres, es el que mejor desempeño tiene al agrupar palabras con contexto similar. Los resultados obtenidos indican que el espacio vectorial definido asignan una similitud *coseno* de al menos 90 % entre pares de vectores generados usando palabras distintas pero semánticamente relacionadas. Además los resultados, que obtuvimos al agrupar por área de estudio los vectores generados del corpus que hemos creado y aplicando una técnica de reducción de dimensionalidad, indican que la distribución de las muestras en el espacio vectorial definido crean conjuntos o grupos de datos linealmente separables. Lo que sugiere que el modelo que entrenamos para codificar textos de nuestro corpus, representó también relaciones semánticas entre textos de la misma área de estudio.
Los modelos que entrenamos usando nuestro corpus y otros seleccionados, nos dieron la oportunidad de crear estructuras numéricas con variantes contextuales por palabra que usamos para atender la problemática de representar los textos en algo que la computadora y los algoritmos de AM sean capaces de procesar. Esta investigación demuestra que las distintas representaciones vectoriales de un mismo texto pero con variantes de contexto crean patrones en las estructuras numéricas que las técnicas de AM pueden aprender para lograr un mejor desempeño.

Modelos de AM

Los resultados que obtuvimos al aplicar distintos algoritmos de AM indican que las técnicas inspiradas en *aprendizaje profundo* son las que mejor desempeño tienen en términos de rendimiento predictivo de detección de paráfrasis. Pero también demuestran la importancia de distribuir correctamente la información en un espacio vectorial ya que incluso *Maquina de Soporte Vectorial* puede obtener un mejor desempeño que técnicas más complejas.
Adicionalmente propusimos un modelo ensamble construido con modelos de *aprendizaje profundo* para incrementar la habilidad de generalizar sobre nuevos datos de un solo modelo y los resultados obtenidos confirman que los ensambles *stacking* son un medio útil para reducir la inhabilidad de un modelo en generalizar conocimiento y a su vez aumentar la capacidad de aprender patrones valiosos en los datos que se usan para entrenar.

7.1. Trabajo futuro

El trabajo realizado en esta investigación abre distintas líneas de estudio en aras de perfeccionar la técnica de detección de paráfrasis desde un enfoque computacional e integrar información y nuevas técnicas de AM al proceso. En lo particular señalamos las siguientes:

1. Aumentar el número de segmentos del corpus de paráfrasis. El esquema de extracción propuesto puede ser usando para procesar y extraer segmentos del resto de documentos disponibles en el repositorio *TESIUAMI* para enriquecerlo en términos de vocabulario y variantes *léxico-sintácticas* y *semánticas* integradas en todos los documentos y áreas de estudio asociadas.

2. Codificadores de texto especializados por área de estudio. El esquema de transformación *texto-vector* presentado, contiene variantes contextuales extraídas de los corpus seleccionados en esta investigación. Sin embargo, la cantidad de cambios *léxico-sintácticos* y *semánticos* representados en vectores puede enriquecerse si se crean modelos *embedding* especializados por área de estudio para crear estructuras numéricas con abundantes patrones en los datos que le sirvan a los algoritmos de *AM*.

3. *Embeddings* de propósito general. En nuestro método usamos dos modelos *embedding* distintos pero dado que la transformación de texto a vector es un módulo independiente de todo algoritmo de AM, se pueden experimentar con algoritmos *embedding* distintos y evaluar el impacto que tienen sobre el rendimiento predictivo de detección de paráfrasis con respecto a los *embeddings* usados en esta investigación.

4. Arquitectura de entrenamiento paralelo. Se ha demostrado que incrementando el número de muestras de entrenamiento en modelos inspirados en *aprendizaje profundo* incrementa drásticamente el desempeño pero también incrementan los recursos de memoria y capacidad de procesamiento de las computadoras haciendo que el tiempo de entrenamiento aumente proporcionalmente. Una manera de acelerar el proceso de entrenamiento es aplicando estrategias de distribución de datos y entrenamiento *síncrono* y *asíncrono* [31] sobre multiples recursos de *hardware* y distribuir así el esfuerzo computacional con la intención de reducir el tiempo de entrenamiento.

5. Esquema de evaluación continua. Podemos pensar que el ciclo de vida de un modelo de AM termina cuando ya ha sido entrenado. Pero existen muchos factores dinámicos inherentes del lenguaje natural que pueden hacer que las predicciones de un modelo se degraden a un punto tal que sean inservibles. Lo que provoca que el modelo no sea confiable en términos predictivos. La forma más directa de detectar que nuestro modelo pierde credibilidad es constantemente evaluar

el desempeño sobre nuevos datos (que en nuestro contexto hacemos referencia a la continua creación de textos en el repositorio *TESIUAMI*) y crear métricas para determinar si es necesario que el modelo vuelva a entrenar para aumentar la confiabilidad y, consecuentemente, su uso.

6. Explicación de predicciones. Los modelos para detectar paráfrasis de forma automática que obtuvimos en esta investigación lograron un buen desempeño en términos de métricas como *exactitud* o *precisión*, pero en muchos escenarios sensibles y prácticos de nuestro estudio, los usuarios pueden dudar en aceptar la predicción de un modelo que no da mucha información de cómo genera las predicciones que devuelve. Sobre todo las técnicas inspiradas en el *aprendizaje profundo* que por su naturaleza y definición son complejas de explicar [32] [33] [34]. De modo que aumentar la confianza de los usuarios en los sistemas al proporcionarles una perspectiva de los factores de los datos que influyeron en las predicciones realizadas puede facilitar la aceptación de los usuario en escenarios prácticos como *detección de plagio* o *asignación de autoría* de nuestro estudio. Incluso la explicación de predicciones puede ayudar a detectar sesgos o patrones en los datos que sirvan para aumentar el desempeño de los modelos. Futuros esfuerzos bajo este rubro implican muchas ventajas en nuestro estudio.

Adicionalmente, los modelos obtenidos en esta investigación y los datos que hemos logrado recolectar nos dan la oportunidad de plantear posibles aplicaciones como:

1. **Similitud contextual entre pares de documentos**
   La información contenida en los mapas de calor, obtenidos al comparar dos documentos completos, sin duda se trata de información relevante que podemos emplear para analizar pares de documentos completos y determinar el grado de similitud general entre cada par. Además de encontrar secciones de un documento que se incluyen en otro y determinar su grado de similitud.

2. **Búsquedas especializadas**
   El espacio vectorial definido con los modelos embedding obtenidos nos dan la posibilidad de plantear un esquema de búsqueda por grado se similitud semántica (en el espacio vectorial) de un texto o documento completo contra todo el banco de tesis que hemos recolectado hasta el momento.

REFERENCIAS

[1] K. P. Augustín, "El plagio en las literaturas hispánicas: Historia, Teoría y Práctica". Ph.D. dissertation, Université Paris-Sorbonne, Paris IV, 2010.

[2] Gusenbauer, M., 2018. "Google Scholar to overshadow them all? Comparing the sizes of 12 academic search engines and bibliographic databases". Scientometrics, 118(1), pp.177-214.

[3] Cunha, I. d., Vivaldi, J., Torres-Moreno, J.-M., and Sierra, G. (2014). "Simtex: An approach for detecting and measuring textual similarity based on discourse and semantics". Computación y Sistemas, 18(3):505–516.

[4] Mota Montoya, M. A., Da Cunha, I., and López-Escobedo, F. (2016). "Un corpus de paráfrasis en español: Metodología, elaboración y análisis". RLA. Revista de lingüística teórica y aplicada, 54(2):85–112.

[5] Zhou, L., Lin, C.-Y., Munteanu, D. S., and Hovy, E. (2006). "Paraeval: Using paraphrases to evaluate summaries automatically". In Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, pages 447–454. Association for Computa-tional Linguistics.

[6] Muhammad Haroon Shakeel, Asim Karim, Imdadullah Khan, "A multi-cascaded model with data augmentation for enhanced paraphrase detection in short texts", Information Processing & Management, Volume 57, Issue 3.

[7] A. Eyecioglu and B. Keller, .ªSOBEK: Twitter Paraphrase Identification with Simple Overlap Features and SVMs", 2021.

[8] B. Agarwal, H. Ramampiaro, H. Langseth and M. Ruocco, "A deep network model for paraphrase detection in short text messages", Information

Processing & Management, vol. 54, no. 6, pp. 922-937, 2018. Available: 10.1016-j.ipm.2018.06.005 [Accessed 17 April 2021].

[9] Torfi, Amirsina & Shirvani, Rouzbeh & Keneshloo, Yaser & Tavvaf, Nader & Fox, Edward. "Natural Language Processing Advancements By Deep Learning: A Survey". 2020.

[10] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in Advances in neural information processing systems, pp. 3111–3119, 2013.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," arXiv preprint arXiv:1301.3781, 2013.

[12] Le, Q., Mikolov, T.: "Distributed Representations of Sentences and Documents," In: Proceedings of International Conference on Machine Learning. pp. 1188–1196. Beijing, China (2014)

[13] K. Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. "BLEU: a method for automatic evaluation of machine translation." In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL '02). Association for Computational Linguistics, USA, 311–318.

[14] Gómez-Adorno H., Bel-Enguix G., Sierra G., Torres-Moreno JM., Martinez R., Serrano P. (2020) "Evaluation of Similarity Measures in a Benchmark for Spanish Paraphrasing Detection," In: Martínez-Villaseñor L., Herrera-Alcántara O., Ponce H., Castro-Espinoza F.A. (eds) Advances in Computational Intelligence. MICAI 2020. Lecture Notes in Computer Science, vol 12469. Springer, Cham.

[15] Castro, B., Sierra, G., Torres-Moreno, J.M., Da Cunha, I.: "El discurso y la semántica como recursos para la detección de similitud textual," In: Proceedings of the III RST Meeting (8th Brazilian Symposium in Information and Human Language Technology, STIL 2011). Brazilian Computer Society, Cuiabá (2011).

[16] Z. Wang, W. Hamza, R. Florian, "Bilateral multi-perspective matching for natural language sentences," in: Proceedings of the International Joint Conference on Artificial Intelligence, (IJCAI), 2017, pp. 4144–4150.

[17] B. Dolan, C. Quirk, C. Brockett, "Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources," in: Proceedings of the International Conference on Computational Linguistics, (COLING), 2004, pp. 350–357.

[18] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, “FastText: Enriching Word Vectors with Subword Information”, 2016.

[19] N. Chawla, K. Bowyer, L. Hall and W. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” Journal of Artificial Intelligence Research, vol. 16, pp. 321-357, 2002.

[20] M. Schmidt, N. Le Roux, F. Bach. “Minimizing Finite Sums with the Stochastic Average Gradient.” Mathematical Programming, Springer Verlag, 2017, 162 (1-2), pp.83-112.

[21] A. Defazio, F. Bach, S. Lacoste-Julien, “SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives, ” Advances in Neural Information Processing Systems, Curran Associates, vol. 27, 2014.

[22] B. H. Shekar and G. Dagnew, “Grid Search-Based Hyperparameter Tuning and Classification of Microarray Cancer Data,” 2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP), 2019, pp. 1-8.

[23] S. Raschka, “An Overview of General Performance Metrics of Binary Classifier Systems,” CoRR, abs/1410.5330, 2014.

[24] A. Bradley, ”The use of the area under the ROC curve in the evaluation of machine learning algorithms”, Pattern Recognition, vol. 30, no. 7, pp. 1145-1159, 1997.

[25] D. Rey and M. Neuhäuser, “Wilcoxon-Signed-Rank Test,” International Encyclopedia of Statistical Science, pp. 1658-1659, 2011.

[26] L.J.P. van der Maaten and G.E. Hinton. “Visualizing High-Dimensional Data Using t-SNE.” Journal of Machine Learning Research 9(Nov):2579-2605, 2008.

[27] J. Gu et al., “Recent advances in convolutional neural networks”, Pattern Recognition, vol. 77, pp. 354-377, 2018. Available: 10.1016/j.patcog.2017.10.013 [Accessed 14 January 2022].

[28] A. Sherstinsky, “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory ($LSTM$) network”, Physica D: Nonlinear Phenomena, vol. 404, p. 132306, 2020.

[29] S. Hochreiter, “The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions”, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 06, no. 02, pp. 107-116, 1998.

[30] M. Belkin, D. Hsu, S. Ma and S. Mandal, "Reconciling modern machine-learning practice and the classical bias–variance trade-off", Proceedings of the National Academy of Sciences, vol. 116, no. 32, pp. 15849-15854, 2019.

[31] M. Li, "Scaling Distributed Machine Learning with the Parameter Server," Proceedings of the 2014 International Conference on Big Data Science and Computing - BigDataScience '14, 2014.

[32] A. Ghorbani, A. Abid and J. Zou, "Interpretation of Neural Networks Is Fragile," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 3681-3688, 2019.

[33] S. Lundberg et al., "Explainable machine-learning predictions for the prevention of hypoxaemia during surgery," Nature Biomedical Engineering, vol. 2, no. 10, pp. 749-760, 2018.

[34] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," CoRR, vol. abs/1705.07874, 2017.

[35] Q. Nguyen et al., "Influence of Data Splitting on Performance of Machine Learning Models in Prediction of Shear Strength of Soil," Mathematical Problems in Engineering, vol. 2021, pp. 1-15, 2021.

[36] Guyon, Isabelle. "A Scaling Law for the Validation-Set Training-Set Size Ratio." (1997).

[37] IEEE. "An FAQ on Intellectual Property Rights for IEEE Authors." [online] https://www.ieee.org/content/dam/ieeeorg/ieee/web/org/pubs/author_faq.pdf, [Accessed 2 Feb 2022].

[38] Barzilay, Regina. "Information Fusion for Multidocument Summarization: Paraphrasing and Generation." Tesis de doctorado en filosofía. New York: Universidad de Columbia (2003).

[39] Burrows, Steven, Potthast, Martin y Stein, Benno. "Paraphrase Acquisition via Crowdsourcing and Machine Learning." ACM Transactions on Intelligent Systems and Technology (TIST), (2013).

[40] S. Iyer, N. Dandekar and K. Csernai, "First Quora Dataset Release: Question Pairs," Quora, (2017). [Online]. Available: https://quoradata.quora.com/FirstQuoraDatasetReleaseQuestionPairs. [Accessed: 02 Feb 2022].

[41] W. Lan, S. Qiu, H. He and W. Xu, "A Continuously Growing Dataset of Sentential Paraphrases," Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 1224–1234 Copenhagen, Denmark, (2017).

[42] da Cunha, Iria; Torres-Moreno, Juan-Manuel; Sierra, Gerardo. "On the Development of the RST Spanish Treebank." En Proceedings of the 5th Linguistic Annotation Workshop. 49th Annual Meeting of the Association for Computational Linguistics (ACL). Portland, Oregon, USA, (2011).

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE EXAMEN DE GRADO

No. 00097
Matrícula: 2202800362

Detección de paráfrasis utilizando algoritmos de aprendizaje maquinal.

En la Ciudad de México, se presentaron a las 12:00 horas del dia 4 del mes de agosto del año 2022 en la Unidad Iztapalapa de la Universidad Autónoma Metropolitana, los suscritos miembros del jurado:

DRA. GEMMA BEL ENGUIX
ING. LUIS FERNANDO CASTRO CAREAGA
DR. RICARDO MARCELIN JIMENEZ

Bajo la Presidencia de la primera y con carácter de Secretario el último, se reunieron para proceder al Examen de Grado cuya denominación aparece al margen, para la obtención del grado de:

MAESTRO EN CIENCIAS (CIENCIAS Y TECNOLOGIAS DE LA INFORMACION)

DE: DAVID LUNA LUNA

y de acuerdo con el artículo 78 fracción III del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

APROBAR

Acto continuo, la presidenta del jurado comunicó al interesado el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

DAVID LUNA LUNA
ALUMNO

REVISÓ

MTRA. ROSALIA SERRANO DE LA PAZ
DIRECTORA DE SISTEMAS ESCOLARES

DIRECTOR DE LA DIVISIÓN DE CBI

Román Linares Romero

DR. ROMAN LINARES ROMERO

PRESIDENTA

DRA. GEMMA BEL ENGUIX

VOCAL

ING. LUIS FERNANDO CASTRO CAREAGA

SECRETARIO

DR. RICARDO MARCELIN JIMENEZ