UNIVERSITAT DE BARCELONA

Trabajo de final de grado

Grado en ingeniería informática

Facultad de Matemáticas y Informática
Universidad de Barcelona

Generación Automática de Cuestionarios mediante Modelos de Lenguaje Masivos

---

Autor: Carlos Gómez Domínguez

Tutores: Daniel Ortiz Martínez y Eloi Puertas Prats
Realizado: Departamento de Matemáticas y Informática

Barcelona, 10 de junio de 2024

Resumen

El procesamiento de lenguaje natural (PLN, por sus siglas), es el área de la informática especializada en las interacciones entre humanos y ordenadores utilizando el lenguaje. En los últimos años, hemos visto cómo nuestra vida cotidiana se ha visto influenciada por esta rama de la informática. Esta tecnología ha tenido un gran impacto en muchos ámbitos, especialmente en el educativo, ya que ofrece una gran variedad de usos.

En el desarrollo de este trabajo, se mostrarán los pasos seguidos para la implementación de una herramienta de PLN, cuyo objetivo es la generación de preguntas a partir de un documento para la autoevaluación del alumnado.

Resum

El processament del llenguatge natural (PLN, per les seves sigles), és l'àrea de la informàtica especialitzada en les interaccions entre humans i ordinadors utilitzant el llenguatge. En els últims anys, hem vist com la nostra vida quotidiana s'ha vist influïda per aquesta branca de la informàtica. Aquesta tecnologia ha tingut un gran impacte en molts àmbits, especialment en l'educatiu, ja que ofereix una gran varietat d'usos.

En el desenvolupament d'aquest treball, es mostraran els passos seguits per a la implementació d'una eina de PLN, la qual té com a objectiu la generació de preguntes a partir d'un document per a l'autoavaluació de l'alumnat.

Abstract

Natural language processing (NLP) is the area of computer science that specializes in human-computer interactions using language. In recent years, we have seen how our daily lives have been influenced by this branch of computer science. This technology has had a great impact in many fields, especially in education, since it offers a wide variety of uses.

In the development of this work, we will show the steps followed for the implementation of a NLP tool, whose objective is the generation of questions from a document for student self-assessment.

Agradecimientos

Quisiera expresar mi más sincero agradecimiento al Dr. Daniel Ortiz y al Dr. Eloi Puertas por sus contribuciones y constante apoyo durante todo el desarrollo de este TFG. Su guía y consejos han sido una parte fundamental para alcanzar los objetivos planteados al inicio de este trabajo.

Índice

1. Introducción

Para que se comprenda y se sitúe la problemática que se pretende resolver, en primer lugar, se expondrá una contextualización completa del problema, donde se introducirán todos los conceptos teóricos relacionados con él, junto con los artículos e investigaciones más relevantes que se han seguido para su solución. Posteriormente, se indicarán las justificaciones de la investigación realizada y, finalmente, se enumerarán los objetivos del presente trabajo y se expondrán las motivaciones para su realización.

1.1. Contextualización

En la actualidad, el campo del Procesamiento del Lenguaje Natural (NLP por sus siglas en inglés) se destaca como uno de los sectores de la Inteligencia Artificial con mayores perspectivas de crecimiento a futuro, debido a su relevancia en una amplia gama de aplicaciones y su constante evolución. El PLN es una tecnología de machine learning que brinda a los ordenadores la capacidad de interpretar, manipular, gestionar y comprender el lenguaje humano escrito.

El procesamiento del lenguaje natural ha encontrado un lugar destacado en el ámbito académico debido a su capacidad para automatizar procesos que, de otro modo, serían laboriosos y complejos. Esta tecnología se ha convertido en un recurso invaluable para una variedad de aplicaciones, ofreciendo soluciones sofisticadas que abarcan desde la generación y evaluación de textos, hasta la detección de fraudes y la creación de sistemas de chats, entre otras posibilidades.

La adopción del PLN en entornos educativos puede proporcionar una significativa reducción de la carga de trabajo tanto para el profesorado como para el alumnado. La capacidad de automatizar tareas complejas abre nuevas oportunidades para la eficiencia y la productividad en el ámbito académico.

Por tanto, considerar la implementación de herramientas de PLN para la automatización de tareas complejas se presenta como una decisión estratégica que puede beneficiar considerablemente a instituciones educativas y profesionales del sector.

En términos generales el enfoque será el siguiente, a partir de un archivo en formato PDF de cualquier materia, se llevará a cabo una extracción de palabras clave. El propósito será la identificación de los conceptos e ideas principales del documento. Posteriormente, estos conceptos serán agrupados en clústeres según su similitud, y se buscará generar definiciones para cada uno de ellos, utilizando un modelo de lenguaje. El paso final será crear preguntas tipo test sobre ideas específicas del archivo. Las respuestas a estas preguntas serán las definiciones generadas para cada uno de los conceptos que pertenecen al mismo clúster del concepto sobre el cual se hace la pregunta.

1.2. Objetivos

A continuación, es importante mencionar que este trabajo es una continuación de otro TFG realizado en 2023 por Lluís Roca Román, titulado *Self-questionnaire: Automatic generation of questions using artificial intelligence techniques for self-regulated learning*, que tenía objetivos similares.

En este trabajo, sin embargo, se ha puesto un mayor énfasis en el análisis de datos para obtener resultados más satisfactorios.

Los grandes modelos de lenguaje pueden considerarse una caja negra, ya que no se sabe bien qué ocurre por dentro. Únicamente se tiene control sobre la información de entrada, el *prompt*, y en menor medida sobre la información de salida. Por ello, este trabajo busca obtener un mayor control sobre esa información de entrada para obtener respuestas más precisas. Con esto, se pretende generar una serie de preguntas tipo test que permitan tanto al profesorado como al alumnado autoevaluarse en una materia dada.

Sabiendo esto, los objetivos establecidos para este trabajo han sido los siguientes:

- **Implementar una arquitectura eficaz y funcional para la extracción de key phrases.**
- **Etiquetar de manera automatica las keyphrases extraidas.**
- **Analizar los resultados para encontrar una combinación óptima de parámetros que produzca resultados satisfactorios.**
- **Comprender el funcionamiento de los modelos de procesamiento del lenguaje natural y crear un software capaz de generar preguntas de opción múltiple a partir de un documento.**

1.3. Motivación

Este trabajo de fin de grado surge de una motivación personal y académica que ha ido creciendo a lo largo de mi trayectoria universitaria. Durante estos cuatro años, he enfrentado diversos desafíos que, con estudio y dedicación, he ido superando gradualmente. En muchas ocasiones, la complejidad de las materias requería un esfuerzo considerable y horas de dedicación para ser superadas.

Sin embargo, estas dificultades no fueron obstáculos, sino oportunidades para crecer y superar las limitaciones que a veces nos imponemos. Por eso, cuando se presentó la oportunidad de realizar un trabajo de fin de grado relacionado con el campo de la Inteligencia Artificial, específicamente en PLN, no dudé en aprovecharla. Este trabajo representa la síntesis de todo el esfuerzo y conocimientos adquiridos a lo largo de la carrera, fundamentados en el esfuerzo y la dedicación.

Las principales motivaciones para llevar a cabo este trabajo pueden resumirse en los siguientes puntos:

- **Un gran interés en el campo de la inteligencia artificial, especialmente en las arquitecturas y algoritmos necesarios para su implementación.**
- **Posibilidad de desarrollar un software con algoritmos de IA que tenga una funcionalidad realmente práctica y que pueda ser utilizada en el futuro por estudiantes y profesores, facilitando así sus tareas académicas.**
- **Mostrar y sintetizar todos los conocimientos adquiridos a lo largo de la carrera.**

1.4. Contenido del documento

En el siguiente apartado se detallan los contenidos de la memoria del trabajo, con el objetivo de aclarar las principales secciones y ofrecer una breve explicación sobre cada una de ellas.

1. **Introducción:** Este apartado se dedica a presentar el problema que este trabajo aborda, contextualizando dentro del ámbito correspondiente. Se establecen los objetivos que se persiguen en el presente trabajo y se proporciona una reflexión sobre las motivaciones para su realización.
2. **Estado del Arte:** En este apartado se mostraran un estudio sobre los diferentes algoritmos y sus enfoques existentes usados para el desarrollo del trabajo. Además se proporcionaran las referencias a investigaciones y articulos usados para completar el estudio.

3. **Análisis del problema:** Tras finalizar la investigación necesaria, este apartado muestra un análisis exhaustivo del problema en cuestión. Se detalla cómo se ha encarado este desafío, describiendo la arquitectura seleccionada y la estrategia adoptada para abordar los obstáculos que surgieron durante el desarrollo del código.

4. **Implementación:** Tras la decisión de la arquitectura, en esta aprte se examinrá detalladamente el proceso de toma de decisiones y las estrategias implementadas para superar los desafíos encontrados en cada uno de los módulos del código.

5. **Resultados:** En esta sección, examinaremos todas las pruebas y resultados obtenidos mediante el software. Presentaremos todos los experimentos realizados junto con sus conclusiones. Muchas de las justificaciones de los parámetros seleccionados se basan en estos experimentos.

6. **Conclusiones:** En este último apartado se expondrán los problemas que han ido surgiendo a lo largo del desarrollo del trabajo, así como posibles ampliaciones o mejoras de este, para una obtención de resultados más óptimos. Además, se incluirá una conclusión final que resume los objetivos conseguidos y el trabajo realizado.

2. Estado del arte

Para este presente trabajo relacionado con la inteligencia artificial, el estado del arte es fundamental para comprender el panorama actual y las tendencias en este campo en constante evolución. En el contexto actual, es importante tener en mente los avances y estudios previos realizados.

Pero antes de comenzar, es importante proporcionar una definición clara de qué es el PLN. El procesado de lenguaje natural, como dice su nombre, es un campo de la inteligencia artificial que estudia las interacciones entre los ordenadores y el lenguaje humano. Se ocupa de la creación de mecanismos eficaces computacionalmente para la comunicación entre las personas y los ordenadores mediante el lenguaje natural.

La comunicación humana está llena de ambigüedades, tales como metáforas, sarcasmos, excepciones gramaticales, modismos, etc. Esto dificulta enormemente la creación de software convencional para comprender estos matices del lenguaje. Mientras que los humanos suelen requerir años para entender estas irregularidades, los desarrolladores deben instruir a las aplicaciones, para que puedan reconocer y entender con precisión estos patrones de lenguaje.

En esta sección se mostrará el estado del arte relacionado con los diferentes puntos del trabajo realizado.

2.1. KeyWord Extraction

Según Prafull Sharma et al.[1] en el entorno digital actual, el crecimiento exponencial de datos no estructurados, provenientes principalmente de redes sociales, documentos digitales y blogs, es evidente. Para dar sentido a este conjunto de datos, es necesario emplear técnicas eficaces que puedan condensar y representar el texto de manera efectiva.

Por esta razón, la extracción de palabras clave (keywords) y frases clave (keyphrases) surge como un área clave para el estudio en este dominio. Las keywords desempeñan un papel esencial al captar de forma rápida la esencia del texto, facilitando tareas como la optimización de motores de búsqueda, la publicidad y las recomendaciones de usuarios.

Un aspecto a destacar que diferencia las keywords de las keyphrases, es que las primeras proporcionan una descripción breve y concisa mientras que las otras ofrecen más contexto y significado. Teniendo en mente todos estos conocimientos la extracción de palabras y frases clave se pueden clasificar mayoritariamente en los siguientes enfoques: Enfoque estadístico, enfoque basado en grafos, enfoque lingüístico, enfoque de machine learning, y enfoques híbridos.

2.1.1. Enfoque estadístico

En la extracción estadística de palabras clave y frases clave, se utiliza la frecuencia como criterio principal para seleccionar las candidatas más relevantes, basándose en un amplio corpus lingüístico. Este enfoque es aplicable a cualquier idioma, ya que no depende de la lengua específica del texto analizado, siempre y cuando se cuente con un corpus de datos lo suficientemente grande y variado. Además, la identificación de la asociación estadística entre las frases clave candidatas puede proporcionar pistas sobre la coherencia semántica del texto en cuestión. Entre los algoritmos utilizados para esta tarea, destaca RAKE [17], reconocido por su eficacia en la extracción de palabras clave en documentos individuales y su potencial para ser escalado a conjuntos de documentos más extensos. En este algoritmo el texto se divide en oraciones y luego se tokeniza para identificar palabras individuales. Después, se eliminan las stopwords y delimitadores del corpus así como se muestra en la figura1

$$\text{Content\_Word} = \text{Corpus} - \text{Stopwords} - \text{Delimiter}$$

Fórmula del algoritmo RAKE. (1)

La siguiente fase del algoritmo es la creación de la matriz de palabras o *Word degree matrix*. En esta matriz se muestran qué palabras aparecen juntas en el texto y cuánto. Si dos palabras aparecen juntas con frecuencia, podría ser un indicativo que son importantes para la representación del texto.

	feature	extraction	complex	algorithm	available	help	rapid	automatic	keyword
feature	2	2	0	0	0	0	0	0	0
extraction	2	3	0	0	0	0	0	1	1
complex	0	0	1	0	0	0	0	0	0
algorithm	0	0	0	1	1	0	0	0	0
available	0	0	0	1	1	0	0	0	0
help	0	0	0	0	0	1	0	0	0
rapid	0	1	0	0	0	0	1	1	1
automatic	0	1	0	0	0	0	1	1	1
keyword	0	1	0	0	0	0	1	1	1
TOTAL SUM	**4**	**8**	**1**	**2**	**2**	**1**	**3**	**4**	**4**

Candidate Key Phrases :
- ✓ feature extraction
- ✓ Complex
- ✓ help
- ✓ algorithms available
- ✓ rapid automatic keyword extraction

Total **Unique** candidate key phrases count is 5. but total 'Feature' is coming with 2 candidate key phrases . Hence 2

Figura 1: *Word degree matrix utilizada por el algoritmo RAKE*
Fuente: Analytics Vidhya por Subhasis Sanyal

No obstante, es importante tener en cuenta que estos métodos pueden ser susceptibles a la interferencia de ruido presente en el corpus de datos, lo que podría impactar en la precisión de los resultados obtenidos.

2.1.2. Enfoque basado en grafos

El enfoque de Key Extraction basado en grafos [18], se fundamenta en la representación del corpus de donde se quiere extraer las keys como un grafo, donde los nodos son las palabras y las aristas representan la similitud o co-ocurrencia entre ellas.

Este tipo de enfoque se divide en varias partes:

- El primer punto consiste en crear una bolsa de palabras o "bag of words", es decir en crear una lista con todas las palabras que conforman el texto. Seguidamente mediante el coseno de similitud se calcula la co-ocurencia de las palabras entre ellas.
- Seguidamente se representa el documento como un vector N dimensional donde N es el número de palabras y se calcula el coseno de similitud.
- Se construye un grafo donde cada palabra es un nodo y las aristas entre ellos representan la similitud calculada entre las palabras, derivada de su co-ocurrencia en el texto.

- Una vez construido todo el grafo se han de encontrar las palabras clave usando algoritmos de centralidad, como por ejemplo la centralidad de los Eigenvector[6] 2, Pagerank[7] o simplemente el puro grado de centralidad.

$$x_i = \frac{1}{\lambda} \sum_j A_{ij} x_j$$

Fórmula de la centralidad de los Eigenvectors (2)

En PageRank, la importancia de un nodo se determina por los enlaces hacia los nodos vecinos, que actúan como votos de relevancia. La puntuación de cada nodo se calcula recursivamente teniendo en cuenta el peso de estos enlaces y la puntuación de los nodos vecinos. Por otro lado, TextRank un algoritmo parecido a PageRank, se puede aplicar tanto a la resumir textos como a la extracción de palabras clave. TextRank utiliza el concepto de prestigio en la red y el algoritmo PageRank para clasificar los nodos del grafo. Las palabras clave o frases más importantes en el grafo son los nodos con la puntuación más alta. De esta manera, se genera una lista de palabras clave extraídas de la oración.

2.1.3. Enfoque Lingüístico

El enfoque lingüístico se centra en las características sintácticas de las palabras clave, por lo tanto, es completamente dependiente del idioma. Entre los algoritmos más populares que utilizan este enfoque se encuentran los patrones POS (part-of-speech) [19], n-gramas [20] y NP-chunks (fragmentos de sintagmas nominales) [21]. Este enfoque se emplea frecuentemente en corpus específicos de un dominio particular. Es especialmente eficaz para aplicar reglas en la extracción de frases clave, como las combinaciones de Adjetivo + Sustantivo u otras combinaciones que, según el caso, puedan ser útiles.

2.1.4. Enfoque con Machine Learning

Este enfoque es como cualquier otro enfoque donde se usan técnicas de Machine Learning para resolver un problema. Consiste en modelos supervisados que requieren un conocimiento previo para poder ser utilizados, es decir, necesitan una serie de datos para resolver una problemática. Los datos de entrenamiento para la extracción de palabras y frases clave son el corpus y sus correspondientes palabras y frases clave etiquetadas de antemano. Muchos enfoques exitosos, como HMM, Naive Bayes y Support Vector Machine, pertenecen a esta categoría.

Con el avance del Deep Learning, especialmente de los modelos LSTM, se ha demostrado el potencial que tienen para solucionar problemas relacionados con el lenguaje y el texto. Zhang Q et al. [9] presentan un enfoque para la extracción de términos clave en un corpus dado o en plataformas como X (Twitter) o similares. Utilizan una RNN para capturar la información contextual entre las palabras clave y recuperar las keyphrases. Además, emplean un método basado en reglas para crear conjuntos de entrenamiento a partir de los datos de Twitter. Se requiere un conocimiento previo de palabras clave para este enfoque. Por otro lado, Rui Meng et al. [16] proponen un modelo de aprendizaje profundo para la extracción de palabras clave, centrándose en capturar el significado semántico profundo del contenido utilizando un enfoque generativo con un marco codificador-decodificador. Este enfoque se adapta específicamente al ámbito de las publicaciones científicas.

2.1.5. Enfoque Híbrido

Los enfoques híbridos combinan las ventajas de todos los métodos anteriormente mencionados para encontrar un modelo óptimo para cada caso. El enfoque SCKKRS [1] es una variante híbrida que extrae palabras y frases clave de cualquier tipo de texto. Para ello, emplea características contextuales de las palabras, patrones POS, n-gramas y reconocimiento de entidades nombradas (NER) del enfoque lingüístico. Estos datos se utilizan en un modelo de clasificación de palabras con LSTM bidireccional en aprendizaje profundo.

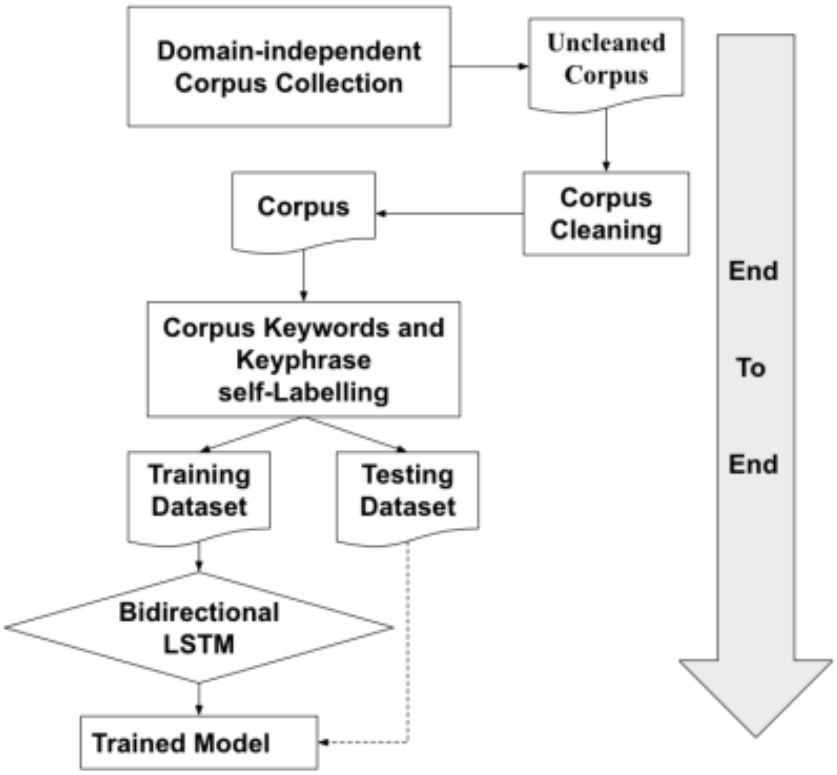


Figura 2: *Flow Chart de SCKKRS*
Fuente: Prafull Sharma, Yingbo Li (2019). Self-supervised Contextual Keyword and Keyphrase Retrieval with Self-Labelling

2.2. Clustering

Al igual que en la sección de keywords, la gran cantidad de datos no ordenados generados en el mundo digital requiere el uso de alguna herramienta que ordene y agrupe estos datos en categorías o etiquetas para su comprensión. Aquí es donde entran los algoritmos de clustering. El clustering es una técnica de aprendizaje automático no supervisado que organiza y clasifica diferentes objetos, puntos de datos u observaciones en grupos o clústeres basados en similitudes o patrones.

Mientras que para los datos etiquetados, la parte más importante es su etiqueta, para los datos no etiquetados, encontrar un objetivo cuantificable que guíe el proceso de construcción del modelo es la cuestión más importante. En las últimas décadas, se han propuesto numerosos métodos de agrupamiento con modelos simples, incluyendo el agrupamiento basado en centroides, el agrupamiento basado en densidad, el agrupamiento basado en distribución y el agrupamiento jerárquico.

Existen varios enfoques para el agrupamiento en clústeres, y cada uno es más adecuado para una distribución de datos en particular. En esta sección, explicaremos algunos de estos enfoques y cómo funcionan.

2.2.1. Enfoque de Particionamiento (k-means)

El algoritmo k-means es un método de agrupamiento que divide un conjunto de datos en k grupos o clusters. Los datos se agrupan de tal manera que los puntos en el mismo clúster sean más similares entre sí que los puntos en otros clusters.

El proceso de k-means inicia seleccionando k puntos iniciales, conocidos como centroides. A continuación cada punto del conjunto de datos es asignado a uno de los k clusters, el que esté más cercano a este. Posteriormente se recalculan los centroides como la media de la distancia de los puntos asignados al cluster. Este proceso de asignación de puntos es repetido hasta que esté converge en una solución óptima, es decir, el cálculo 3 del resultado final a penas varía.

$$\text{WCSS} = \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu_j\|^2$$

Fórmula K-means (3)

2.2.2. Enfoque basado en la densidad

Este tipo de enfoque se basa en agrupar los puntos de datos basándose en la densidad regional de estos. Los clústers se forman donde existe una densidad alta de puntos y se separan donde hay una densidad baja.

Dentro de esta categoría el algoritmo DBSCAN 3, se puede considerar uno de los algoritmos de clustering basado en densidad más conocidos. Este utiliza una distancia especificada para separar los clústeres densos del ruido más disperso. El algoritmo DBSCAN es el método de clustering más rápido, pero solo es apropiado si se puede utilizar una distancia de búsqueda muy clara, y funciona bien con todos los clusters potenciales. Para ello, se requiere que todos los clusters significativos presenten densidades similares. Este método también permite utilizar los parámetros Campo de tiempo e Intervalo de tiempo de búsqueda para buscar clústeres de puntos en espacio y tiempo.

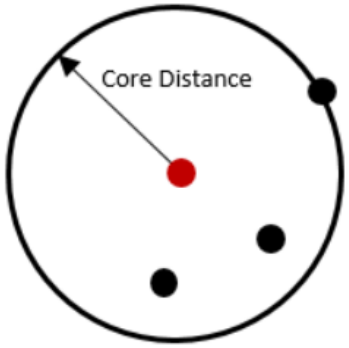


Figura 3: *Esquema clustering densidad*
Fuente: ArcGIS Pro

Sin embargo, su efectividad depende de la selección cuidadosa de parámetros como el radio de búsqueda y el número mínimo de puntos. Además, puede tener dificultades para manejar conjuntos de datos con diferencias significativas en densidad o presencia de puntos ruidosos.

2.2.3. Enfoque basado en Modelos

Este tipo de enfoque consiste en métodos basados en modelos que asumen que los datos se generan a partir de una mezcla de distribuciones subyacentes y tratan de encontrar estas distribuciones. Es decir estos algoritmos se enfocan en construir un modelo probabilístico que explique los datos y luego, una vez construido, asignar los diferentes puntos de datos a diferentes clusters. Dentro de este enfoque encontramos el algoritmo GMM [10], similar a KMeans. Este método es una versión probabilística de KMeans, ya que este tiene un par de similitudes respecto a GMM.

Uno de los puntos débiles de KMeans es que este asume que los clusters tienen una forma esférica regular, y eso no es una suposición válida en el mundo real. Y otro punto es que es un método "duro", ya que los puntos únicamente pertenecen a un solo cluster, a diferencia de GMM que crea un peso para cada dato que indica que probabilidad tiene ese punto de pertenecer al cluster.

GMM funciona de la siguiente manera:

- Inicializar los parámetros del modelos, como los pesos de mezcla que representan la proporción de cada componente gaussiano en mezcla. las medias de las distribuciones gaussianas y las matrices de covarianza de las distribuciones gaussianas.
- (E-step) Calcular la probabilidad de cada uno de los puntos de pertenecer al cluster c 4

$$r_{ic} = \frac{\pi_c N(x_i \mid \mu_c, \Sigma_c)}{\Sigma_{k=1}^{K} \pi_k N(x_i \mid \mu_k, \Sigma_k)}$$

Fórmula probabilidad de i de pertenecer a c (4)

- Maximización, es decir, actualizar los parámetros del paso E-step. Los pesos se actualizan de la siguiente manera 5.

$$\pi_k = \frac{N_k}{N}$$

Fórmula actualización de los pesos de mezcla. (5)

- Actualizar las medias 6

$$\mu_k = \frac{\sum_{i=1}^{N} \gamma(z_{ik}) x_i}{N_k}$$

Fórmula actualización de las medias. (6)

- Actualizar las matrices de covarianza 7

$$\Sigma_k = \frac{\sum_{i=1}^{N} \gamma(z_{ik})(x_i - \mu_k)(x_i - \mu_k)^T}{N_k}$$

Fórmula para actualizar las matrices de covarianza. (7)

- Realizar los pasos anteriores hasta la convergencia.

2.2.4. Enfoque Jerárquico

A diferencia del resto de enfoques, los métodos de clusterizado jerárquicos construyen una jerarquía de clusters. Estos pueden ser aglomerativos (empezando con cada punto como un clúster individual y fusionándose) o divisivos (empezando con todos los puntos en un solo cluster y dividiéndolos).

El método de *Single Linkage*, también conocido como método de la mínima distancia, es un enfoque aglomerativo en la clustering jerárquico. En este método, cada punto de datos comienza como un cluster separado, y en cada iteración, se unen los dos clusters que están más cerca. La cercanía se define como la distancia mínima entre cualquier par de puntos, uno de cada cluster. Tiene la ventaja de detectar grandes clusters o grandes cadenas de puntos pero puede generar clusters no óptimos dependiendo del contexto.

El segundo método que tenemos es *Complete Linkage*, el cual es muy parecido al primer algoritmo. Este es otro enfoque aglomerativo en la clusterización jerárquica. En este método, cada punto de datos también comienza como un cluster separado, pero en cada iteración, se unen los dos clusters cuya distancia máxima entre sus puntos es la más pequeña. Su principal ventaja es que tiende a producir clusters compactos y esféricos, sin embargo puede ser sensible a valores atípicos y puede tender a romper clusters grandes en clusters más pequeños.

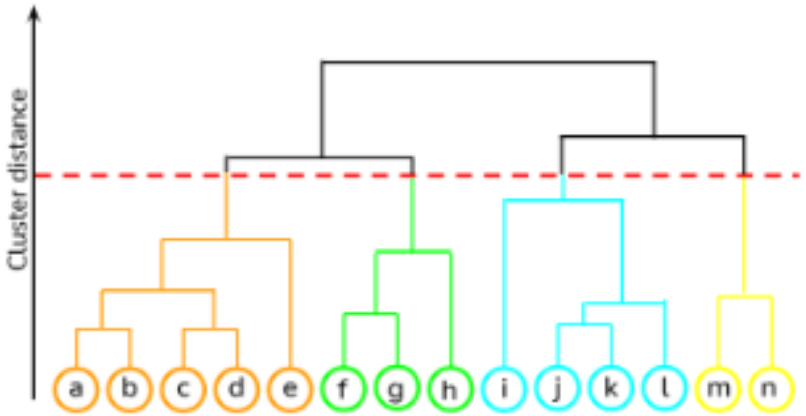


Figura 4: *Esquema del clusterizado jerárquico*
Fuente: Medium por Abdullahabrar

2.3. Deep Learning y el PLN

Como se ha comentado al inicio de esta sección el Procesamiento de Lenguaje Natural, es una rama del aprendizaje automático. Si bien el PLN se estudia desde diversas disciplinas, ya que requiere conocimientos matemáticos y especialmente de estadística, también involucra otras ramas como la lingüística o la informática.

Aunque su desarrollo empezó en la década de los años 60 del siglo pasado, su verdadero auge no se produjo hasta la introducción y el uso de las redes neuronales LSTM en la década del 2010. Durante muchos años hubo ciertos enfoques que se basaban en hacer el uso de modelos estadísticos muy específicos, sin embargo hasta la aparición de las primeras redes neuronales no se consiguieron resultados realmente satisfactorios.

2.3.1. Modelos basados en FFNN

Como se ha mencionado previamente, el procesamiento del lenguaje natural es un campo de estudio con varias décadas de investigación. Sin embargo, no fue hasta que Yoshua Bengio et al. [11] presentaron *A Neural Probabilistic Language Model* que se produjo un avance significativo.

Al modelar el lenguaje, la "maldición de la dimensionalidad" complicaba la tarea de modelar la probabilidad conjunta de secuencias de palabras debido al vasto número de combinaciones posibles. Bengio.Y ponía el ejemplo de que para modelar la distribución conjunta de 10 palabras consecutivas en un vocabulario de 100.000 palabras implicaría potencialmente manejar $10^{50} - 1$ parámetros libres, lo cual no es algo viable.

En el trabajo[11] se consiguieron demostrar los siguientes puntos:

1. **Asignación de Vectores:**
   A cada palabra del vocabulario se le asocia o asigna un vector de características en un espacio continuo. Esta representación permite que el modelo generalice de una manera mucho más eficiente las secuencias de palabras no vistas durante el entrenamiento

2. **Aprendizaje mediante Redes Neuronales:**
   El modelo aprende mediante el uso de redes neuronales FFNN 5 las representaciones de las palabras y la distribución de probabilidad de las secuencias de palabras. Esto supone que una secuencia de palabras nunca antes vista puede tener una alta probabilidad de ser creada si previamente se ha visto una semánticamente similar.

3. **Entrenamiento a gran escala:**
   Se demostró que es posible entrenar modelos a gran escala con millones de parámetros de manera eficiente

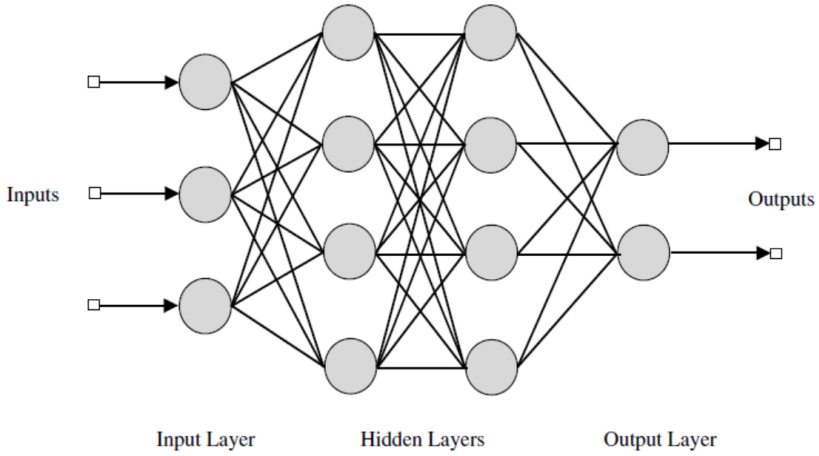


Figura 5: *Esquema red neuronal Feedforward 2 capas.*
Fuente: Medium por Rishabh Mall

Las redes Feedforward se basan en un conjunto de nodos conectados llamados neuronas artificiales, en las que cada una de estas transmiten una señal a las otras neuronas de manera unidireccional.

Se pueden diferenciar tres partes:

- La capa de entrada o input, que se encarga de procesar los datos de entrada.
- La capa o capas oculta, formada por neuronas encargadas de aplicar la función de activación a las señales recibidas.
- La capa de salida encargada de representar el output de la red.

Este modelo de entrenamiento para el PLN supuso una gran mejora, ya que era mejor que cualquier modelo estadístico previamente diseñado para esta disciplina. Sin embargo, el inconveniente de este tipo de redes neuronales es que antes del entrenamiento se tenía que fijar la longitud fija del contexto, ya que normalmente solo tenía en cuenta entre las 5 o 10 últimas palabras para predecir la siguiente.

2.3.2. Recurrent Neural Networks (RNN)

Con el tiempo la implementación de las redes neuronales recurrentes o RNN, pretendían arreglar el problema presentado en las redes FFNN. La idea presentada por Mikolov et al. [12], y basada en los estudios de Bengio, consistía en el uso de redes neuronales recurrentes que se retroalimentan en diferentes momentos.

En el trabajo se plantea que hasta el momento de la realización de este, era difícil medir el éxito de los modelos de lenguaje, debido a que todos los estudios realizados hasta el momento solo funcionaban bien con cantidades limitadas de datos, y únicamente presentaban pequeñas mejoras sobre líneas básicas de texto. Sin embargo el uso de redes neuronales recurrentes permitía no utilizar un tamaño limitado de contexto.

Al usar las conexiones recurrentes, la información sobre entradas pasadas puede ser retenida y utilizada en un futuro para tomar decisiones internas. De esta manera se mantiene la información de los datos que van entrando en su orden. Además no se limita en ningún momento el tamaño del contexto. Gracias a esto, el resultado obtenido en el estudio de Mikolov et al. [12] obtuvo buenas predicciones, más fiables y concisas.

El funcionamiento de la RNN presentada en el trabajo de Mikolov et al. se puede considerar simple. El procesamiento de la información dentro de la red se lleva a cabo mediante funciones de activación que transforman las entradas que reciben. La red tiene una capa de input x, una capa o capas ocultas s, y una capa output llamada y . El vector de input se puede expresar como se ve en la siguiente formula 8.

Además la función de activación es una función sigmoide 11 y una función softmax 12 para la distribución de probabilidades

$$x(t) = w(t) + s(t-1)$$

Fórmula del vector de input. (8)

$$s_j(t) = f\left(\sum_i x_i(t) u_{ji}\right)$$

Fórmula del estado de la capa oculta (9)

$$y_k(t) = g\left(\sum_j s_j(t) v_{kj}\right)$$

Fórmula output layer (10)

$$f(z) = \frac{1}{1 + e^{-z}}$$

Figura 6: *Fórmula función activación (11)*

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

Figura 7: *Fórmula soft max (12)*

El entrenamiento de la red se realiza mediante el algoritmo de retropropagación del error con descenso del gradiente estocástico. Inicialmente se utiliza una tasa de aprendizaje fija que se va ajustando a lo largo de varios Epoch de entrenamiento. La RNN propuesta se entrena de manera secuencial, utilizando a los datos del corpus de entrenamiento, y se valida después de cada Epoch 13. En caso de que no observar ninguna mejora en los datos de validación, la tasa de aprendizaje se reduce a la mitad y el proceso de entrenamiento continúa hasta alcanzar la convergencia .

$$\text{error}(t) = \text{desired}(t) - y(t)$$

Fórmula error (13)

El último paso propuesto en el artículo para mejorar el rendimiento de los resultados, es la mezcla de todas las palabras que aparecen menos de un cierto número de veces, según un umbral establecido. Esto reduce el tamaño del vocabulario que la red debe de manejar y permite que este generalice mejor, en vez de asignar probabilidades muy pequeñas a cada palabra el modelo puede concentrar más capacidad en las palabras comunes y representativas del corpus.

$$P(w_i(t+1)|w(t), s(t-1)) = \begin{cases} \frac{y_{rare}(t)}{C_{rare}} & \text{if } w_i(t+1) \text{ is rare} \\ y_i(t) & \text{otherwise} \end{cases}$$

Fórmula optimización para los tokens (14)

Aunque el uso de las redes neuronales recurrentes fue un avance importante, estas presentaban una serie de problemas. El primero era el llamado desvanecimiento y explosión del gradiente. Este es un problema que aparece durante el entrenamiento, las RNN pueden sufrir de gradientes que se vuelven muy pequeños (desvanecimiento) o muy grandes (explosión). Esto dificulta la propagación de información a lo largo de secuencias largas de texto.

2.3.3. Encoder Decoder

Más adelante aparecieron los modelos encoder-decoder como una posible solución a las limitaciones de las redes neuronales recurrentes en las tareas de predicción de texto, especialmente en el campo de la traducción automática.

La primera vez que este enfoque dio su aparición fue en el trabajo de Bahdanau et al. [13]. El objetivo del trabajo fue mejorar el rendimiento de los modelos de traducción automática mediante la introducción de un mecanismo de atención en los modelos de encoder-decoder. En su artículo *Neural Machine Translation by Jointly Learning to Align and Translate (2015)* propusieron un enfoque innovador para abordar las limitaciones de los modelos de traducción automática que había hasta el momento.

Desde una perspectiva probabilística, la traducción se ve como una búsqueda de la manera en la que una oración objetivo y maximice la probabilidad condicional arg maxy $p(y|x)$dada una oración fuente x.

Este modelo se basa en la concatenación de dos modelos RNN diferentes, en encoder y el decoder.

Encoder: El codificador lee la oración de entrada, una secuencia de vectores $x = (x1, ...xi)$ y la convierte en un vector c. El enfoque más común es usar una RNN donde el estado oculto ht en el tiempo t se calcula con la fórmula 15. El vector c se genera a partir de la secuencia de estados ocultos así como se muestra en la siguiente fórmula 16.

$$c = q\left(\{h_1, \cdots, h_{T_x}\}\right)$$

Fórmula del estado oculto h en el periodo t (15)

$$c = q\left(\{h_1, \cdots, h_{T_x}\}\right)$$

Fórmula vector c de la secuencia de estados ocultos (16)

Decoder: El decodificador se entrena para predecir la siguiente palabra *yt* dado el vector de contexto *c* y todas las palabras predichas previamente, es decir, esta parte según Bahdanau, define una probabilidad sobre la traducción y, descomprimiendo la probabilidad conjunta de condicionales ordenados .

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c)$$

Fórmula de la probabilidad de traducción (17)

Como se ha comentado previamente se introduce una extensión al modelo, un mecanismo de atención que aborda las limitaciones del vector de contexto de longitud fija. Es decir, en lugar de intentar comprimir toda la información de la oración en un solo vector fijo, el mecanismo de atención permite que el decodificador enfoque diferentes partes de la oración fuente en cada iteración de la red neuronal. De esta manera el modelo puede "prestar atención" a las partes relevantes de la oración fuente cuando predice cada palabra de la oración objetivo.

En este modelo, el codificador es una red neuronal recurrente bidireccional, lo que quiere decir que la secuencia de entrada se procesa tanto de izquierda a derecha como de derecha a izquierda. Para cada posición t en la secuencia de entrada, la red genera un estado oculto llamado ht, que es la concatenación de los estados ocultos previamente. Esto permite capturar información contextual en ambas direcciones, proporcionando una representación más rica y completa.

El decodificador es una RNN que genera la traducción palabra por palabra. En cada paso de tiempo t, el decodificador genera una distribución de probabilidad sobre la siguiente palabra en la oración objetivo, condicionada en todas las palabras generadas anteriormente

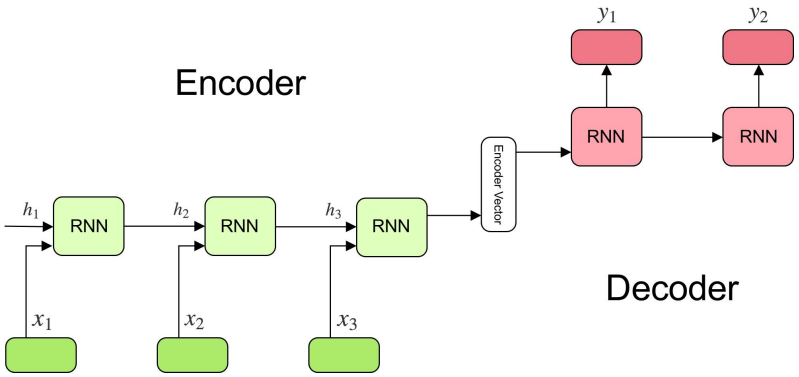


Figura 8: *Esquema de la topología de un sistema formado por un encoder-decoder Fuente: Medium by Simeon Kostadinov*

El trabajo de Bahdanau et al.[13] presentó una revolución en la traducción automática al introducir mecanismo de atención que mejoran la capacidad de manejar secuencias largas y proporcionar soluciones más precisas entre las oraciones de origen y destino. Sin embargo el modelo aún se enfrentaba al problema de su gran complejidad computacional y a problemas de precisión para oraciones que fueran muy largas.

2.3.4. Transformer

El siguiente paso fue la introducción del concepto de transformer, un trabajo realizado por Vaswani et al., 2017 [14]. El documento .Attention Is All You Need"presenta un nuevo modelo de red neuronal con arquitectura encoder-decoder llamado transformer, el cual se basa en los mismos mecanismos de atención prescindiendo de las recurrencias y convoluciones.

En el documento se menciona que los modelos dominantes hasta la fecha estaban basados en redes neuronales como RNN y CNN, las cuales tienen limitaciones significativas en términos de paralelización y eficiencia computacional.

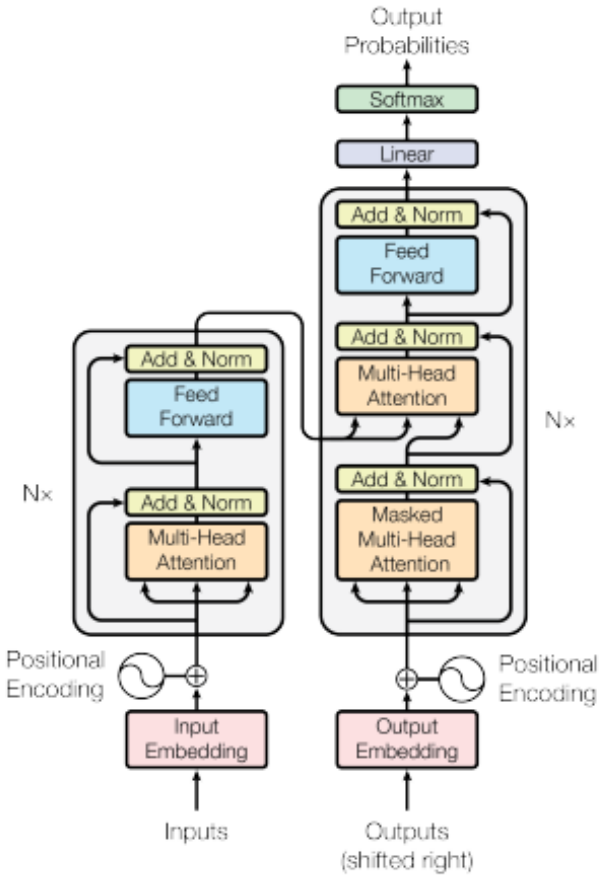


Figura 9: *Arquitectura de la red neuronal Transformer.t*

El algoritmo Transformer sigue una estructura general de codificador-decodificador

Encoder: El codificador está compuesto por 6 capas idénticas, cada capa tiene dos subcapas. La primera es una un mecanismo automatizado de atención en paralelo o "multi-head" y la segunda es una red neuronal FFNN. El output de la capa será la normalización de la salida aplicando una conexión residual.

Decoder: También está formado por 6 capas idénticas, muy parecido al encoder. Además de tener 2 subcapas igual que en el codificador, aquí se le añade una subcapa extra, que realiza mecanismos de atención en paralelo de la salida del encoder. Al igual que en la capa anterior se emplea una conexión residual alrededor de cada subcapa seguido de otra capa de normalización.

El Transformer utiliza la atención de producto punto escalado (Scaled Dot-Product Attention) que calcula pesos de atención como una función de compatibilidad entre la consulta y las claves. Estos pesos se usan para ponderar los valores y producir la salida.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Arquitectura del Transformer (18)

Además, emplea mecanismos de atención en paralelo o *Multi-Head Attention*, donde múltiples mecanismos de atención operan en paralelo, permitiendo al modelo atender a diferentes partes de la secuencia simultáneamente y desde diferentes subespacios de representación. Esto se logra proyectando linealmente las consultas, claves y valores en diferentes subespacios antes de aplicar la atención.

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O \\ \text{where head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$

Fórmula Multi-Head Attention (19)

2.3.5. BERT GPT

Hoy en día, los modelos de lenguaje más importantes incluyen a BERT como codificador y a los modelos más recientes como GPT-3 y GPT-4 como decodificadores.

BERT, introducido por Google en 2019 [15], es un modelo de codificación basado en la arquitectura Transformer, visto en el apartado anterior, para gestionar múltiples tareas de PLN.

Su entrenamiento se divide en dos fases, el pre-entrenamiento con datos no etiquetados y el *Fine-Tunning* con datos supervisados para mejorar la precisión de los resultados. Durante el pre-entrenamiento, *BERT* tiene dos puntos clave, el Modelado de Lenguaje enmascarado (Masqued Language Modeling), que predice palabras ocultas en una oración, y la Predicción de la Siguiente Palabra (Next Sentence Prediction), que decide si una oración sigue a otra. Como codificador bidireccional, *BERT* ofrece una gran capacidad para captar las representaciones contextuales de las palabras de manera satisfactoria.

GPT, desarrollado por *OpenAI*, es un modelo de decodificación también basado en la estructura Transformer. Los modelos más recientes como *GPT-3* y *GPT-4* han mejorado significativamente respecto a *GPT-2*. Estos modelos siguen un proceso de dos fases al igual que en *BERT*, primero, un pre-entrenamiento con un modelo de lenguaje sobre un extenso conjunto de datos, y luego un *Fine-Tunning* para obtener resultados de texto más específicos.

Los modelos de *GPT* destacan por su capacidad de generar texto coherente de manera autónoma, aunque eso sí, presentan ciertas limitaciones y no siempre dan resultados precisos.

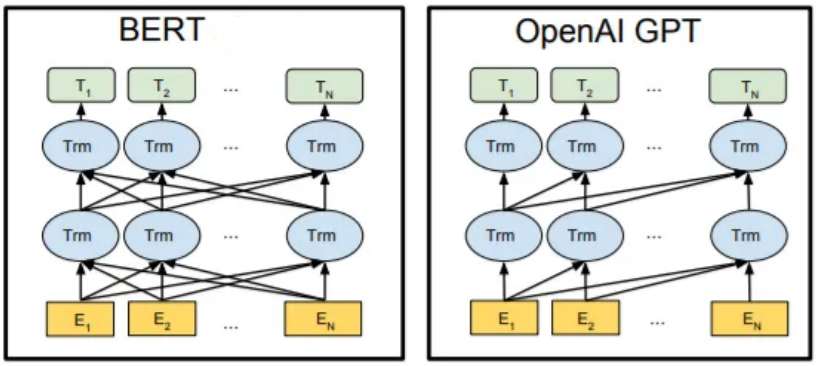


Figura 10: *Esquema de BERT y GPT.*
Fuente: Zhen Huang et al. (2020)

3. Análisis del problema

A lo largo de esta sección se explicará con más detalle cuál es el problema que se quiere abordar y solucionar en este trabajo. En secciones previas se ha explicado de manera más general y abstracta cuál era la problemática a solucionar y cuáles eran las motivaciones y objetivos. Sin embargo, es necesario hacer una explicación más en profundidad de cómo es la arquitectura que busca solucionar el problema planteado.

Actualmente, la empresa *OpenAI* ya da la posibilidad a sus usuarios de analizar grandes cantidades de información. El nuevo modelo de *ChatGPT-4o* permite analizar en breves instantes documentos de cualquier temática. Incluso es posible realizar análisis de datos complejos, basados en información proporcionada por el usuario.

Sin embargo, el problema que tiene *GPT* y cualquier *LLM* es su falta real de contexto. Estos modelos son capaces de dar una respuesta que sea considerada estadísticamente correcta, es decir, pueden proporcionar una respuesta que en términos generales puede ser correcta, pero que quizás para una problemática muy específica y concreta no sea suficiente.

Aunque ya existen herramientas en el mercado que permiten la generación de texto personalizado sobre un tema específico a partir de información proporcionada (archivos, PDF, libros, etc.), este TFG plantea una posible mejora. El objetivo principal es obtener el contexto o las ideas principales de los archivos PDF proporcionados por el usuario y, a partir de estas ideas, lograr un mayor control sobre la generación de preguntas. Esto es crucial, ya que los modelos *GPT* son cajas negras y la única manera de ejercer control es a través de las preguntas o *prompts* formulados.

Al concentrar todo el texto en ideas muy concretas y precisas, es más sencillo generar preguntas que resulten en definiciones más específicas y concretas.

3.1. Arquitectura propuesta

La arquitectura propuesta se basa en la combinación de una serie de algoritmos que tiene como objetivo obtener y etiquetar las keyphrases de un documento dado por el usuario. Esta solución se divide en varias partes que se han de ejecutar en cadena, es decir, cada sección de la arquitectura dependerá de un cálculo previo.

El primer punto será la conversión de los archivos PDF que suministre el usuario a un formato que sea más maleable por el código. Seguidamente se le realizará una extracción de términos clave, para así obtener las ideas principales del texto. El tercer punto será la clusterización de las términos clave extraídas, esto generará una agrupación de *keyphrases* por temarios, lo que supondrá que sea más fácil la generación de texto. Finalmente, mediante el uso de *prompts*, se generan las definiciones de cada uno de los clusters generados en el paso anterior.

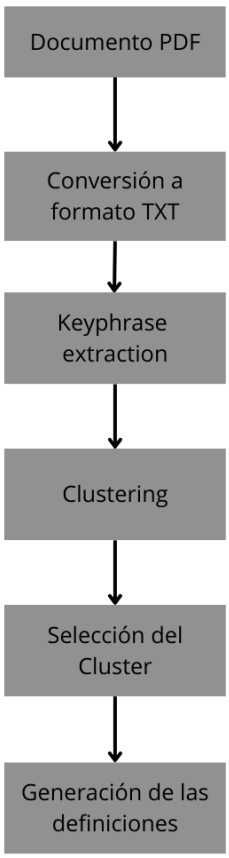


Figura 11: *Arquitectura de la solución propuesta*

3.1.1. Conversor de PDF a TXT

El primer paso consiste en la conversión de los documentos dados por los usuarios de los cuales se quiere extraer las información para realizar las preguntas a un formato manejable con el código. Como la mayoría de material ofrecido en el aula siempre son pdf, debido a su ligereza y seguridad, se ha tomado como una opción viable únicamente procesar archivos con format PDF.

Este punto está dividido en tres partes:

1. La primera parte consiste en transformar cada una de las paginas en un conjunto de imagenes.
2. El segundo paso consiste en transformar esta imagen al formato txt. Este formato permite transformar el texto en variables de tipo string, para así poder manejar este con el código.
3. El último paso consiste en limpiar y corregir los posibles errores del archivo TXT. Esto es debido a que la conversión puede generar una serie de errores o malas conversiones que deben ser corregidas.

3.1.2. Embeddings

Los embeddings son una técnica de PLN que convierte datos no numéricos en vectores numéricos. Estos vectores se utilizan para representar el significado subyacente de palabras, que de otra manera no podrían ser entendidos por los ordenadores.

En otras palabras, los embeddings permiten que las palabras sean tratadas como datos y puedan ser manipuladas. Es una técnica ampliamente utilizada en inteligencia artificial. Hay diversos métodos para generar embeddings, como *Word2Vec*, *GloVe* y *FastText*. Cada uno presenta sus propios puntos fuertes y débiles, es por ello que en la seccion 5.1, se analizaran con mas detallo los resultados obtenidos con algunos modelos encontrados en Hugging Face.

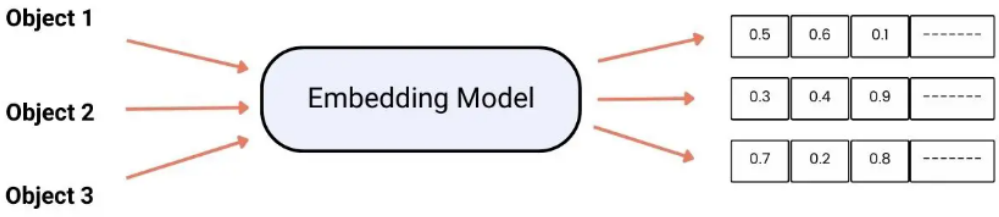


Figura 12: *Esquema del funcionamiento del embedding*

3.1.3. Keyphrase extraction

Este puede ser considerado el punto vital de todo el trabajo, pues la calidad de las preguntas está directamente relacionada con la calidad de las claves o frases extraídas, según el contexto de los documentos.

En esta etapa, se emplean modelos avanzados de procesamiento de lenguaje natural para identificar y extraer las palabras o frases más relevantes de un documento. Este proceso implica la capturar de las ideas y conceptos más representativos del archivo.

El procedimiento comienza con la tokenización del texto, donde el documento se divide en unidades manejables que pueden ser procesadas por los algoritmos 3.1.2.

Además al finalizar esta etapa, es necesario la implementeción de algun tipo de lógica que permita al usuario decidir cuales de las keyphrases extraidas son relevantes y cuales no. Esto ofrece un mayor control al usuario de la información que se va a introducir en los modelos de lenguaje en secciones posteriores.

3.1.4. Clustering

Este apartado tiene como objetivo analizar, etiquetar y agrupar las diferentes keyphrases extraídas en el apartado anterior 3.1.3. Para poder realizar las preguntas de tipo test con multi respuesta, lo que se busca es presentar una pregunta sobre un concepto del documento. En las opciones de respuesta, se mostrará la definición correcta del concepto junto con definiciones de otros conceptos agrupados en el mismo clúster. Esto asegura que las respuestas incorrectas sean similares pero incorrectas, lo que ayuda a generar preguntas de opción múltiple de calidad. Con este enfoque, se consigue que las respuestas incorrectas sean coherentes y contextualmente relevantes con el tema de la pregunta.

Debido a la particularidad de este clusterizado, ya que lo que se busca es que haya el mínimo de clusters vacíos, se decidió coger dos enfoques del algoritmo. El primer enfoque es el *KMeans* usando el método de Elbow para encontrar el número óptimo de clusters. Y el segundo enfoque un *constrained KMeans* el cual permite establecer los tamaños mínimos y máximos que se desea que tengan los clusters.

3.1.5. Modelo LLM

En esta última sección, se generarán todas las definiciones y preguntas basadas en el documento proporcionado.

Primero, se seleccionará el clúster del cual se desean generar las definiciones. Con el uso de *prompts*, se solicitaran estas definiciones a un modelo *GPT*. Una vez hecho esto, se guardarán todos los conceptos con sus definiciones en un archivo TXT. De manera aleatoria, se seleccionará una de las frases clave para formular la pregunta, y el resto de las respuestas se escogerán también aleatoriamente del mismo clúster.

Con ello, se habrá conseguido obtener el objetivo del presente trabajo, la generación de una pregunta para la autoevaluación a partir de un documento dado.

4. Implementación

Este apartado pretende mostrar cómo ha sido todo el proceso paso a paso de construcción del código, cuáles han sido los problemas que han surgido y cuales han sido las soluciones encontradas para estos problemas.

Cada sección de este apartado abordará los diferentes módulos del sistema, las librerías y los materiales utilizados, explicando su implementación y los desafíos encontrados. Además, se presentarán los experimentos y análisis realizados en las secciones de extracción de claves y clustering, que han permitido encontrar una solución potencialmente satisfactoria.

4.1. Librerías y Materiales

Para las características del proyecto realizado, se ha escogido el lenguaje Python. Los motivos para esta elección son variados. Python es ampliamente utilizado en el campo de la inteligencia artificial por varias razones que lo hacen especialmente adecuado para este tipo de trabajos.

Entre los principales factores están:

- **Extensa Biblioteca y ecosistema de paquetes:** Python cuenta con una gran cantidad de bibliotecas y frameworks diseñados específicamente para realizar tareas con inteligencia artificial y machine learning, algunas de las más populares incluyen, TensorFlow, Pytorch, scikit-learn etc.
- **Facilidad de uso:** Este lenguaje es conocido por su sintaxis clara y legible, lo que facilita tanto su uso como su aprendizaje. Esta claridad permite generar de una manera más rápida prototipos que con otros lenguajes se tardaría más. Esta característica es especialmente valiosa en el campo de la IA, donde la experimentación y las iteraciones rápidas son esenciales.
- **Soporte y comunidad:** La comunidad de Python es una de las más grandes y activas del mundo de la programación. Esto supone una gran cantidad de recursos disponibles, ya sean tutoriales, documentación, foros, grupos de discusión o libros, que permiten obtener y compartir conocimientos.
- **Despliegue y escalabilidad:** Con herramientas como TensorFlow Serving, Flask y Docker, los algoritmos de inteligencia artificial desarrollados en este lenguaje son fáciles de desplegar. Esto permite una gran escalabilidad para proyectos de IA desarrollados con este lenguaje

La combinación de una gran cantidad de herramientas internas, junto con su facilidad de uso y su fuerte soporte, han sido claves a la hora de escoger una herramienta con la que programar este proyecto. Todos estos factores se resumen en que permite a los desarrolladores centrarse en resolver problemas complejos de inteligencia artificial, sin tener que preocuparse por detalles de implementación del lenguaje.

4.1.1. Librerías

En este apartado se listan todas las librerías externas que se han usado en este trabajo y cuál ha sido su uso.

1. **Transformers:** Esta librería proporciona herramientas y modelos preentrenados de lenguaje, como GPT-2, facilitando el uso de modelos *LLM*. En el presente trabajo, es utilizada para descargar y trabajar con modelos de lenguaje desde Hugging Face.
   `https://huggingface.co/transformers/`

2. **Numpy:** Librería para operaciones matemáticas y manejo de arrays.
   `https://numpy.org/`

3. **Sentence-transformers:** Proporciona modelos para generar embeddings de frases, permitiendo representaciones vectoriales de texto. En el proyecto, es utilizado para transformar frases en embeddings.
   `https://www.sbert.net/`

4. **K-means-constrained:** Extensión del algoritmo K-means que permite aplicar restricciones.
   `https://github.com/joshlk/k-means-constrained`

5. **Sklearn (scikit-learn):** Ofrece herramientas para minería de datos y análisis de datos, incluyendo algoritmos de clustering. En el proyecto, es usado principalmente realizar un clustering con KMeans.
   `https://scikit-learn.org/`

6. **Langchain:** Facilita la manipulación y transformación de texto en diversas formas. En el proyecto, es usado para diferentes operaciones de procesamiento de texto.
   `https://github.com/hwchase17/langchain`

7. **KeyBERT:** Herramienta para la extracción de palabras clave basadas en BERT embeddings. En el proyecto, se utiliza para extraer palabras clave relevantes de los textos.
   `https://github.com/MaartenGr/KeyBERT`

8. **Easygui:** Proporciona una forma sencilla de crear interfaces gráficas. En el proyecto, se utiliza para desarrollar interfaces gráficas simples.
   `https://easygui.readthedocs.io/en/latest/`

9. **Spacy:** Librería de PLN que incluye herramientas para el reconocimiento de palabras. En este presente trabajo, es usado para el reconocimiento de palabras.
   `https://spacy.io/`

10. **pdf2image:** Convierte archivos PDF en imágenes.
    `https://github.com/Belval/pdf2image`

11. **Pytesseract:** Interfaz de Python para Tesseract OCR, una herramienta para el reconocimiento óptico de caracteres.
    `https://github.com/madmaze/pytesseract`

12. **Torch:** Librería para el cálculo numérico con soporte para aceleración en GPU, utilizada principalmente para el entrenamiento y fine-tuning de modelos. En el proyecto, es usado para aprovechar la GPU en el fine-tuning de modelos de lenguaje de GPT-2.
    `https://pytorch.org/`

13. **Openai:** Proporciona acceso a las API de OpenAI, incluyendo GPT-3.5 y otros modelos avanzados de procesamiento del lenguaje natural. En el proyecto, se utiliza para interactuar con estos modelos a través de la API de OpenAI.
    `https://beta.openai.com/docs/`

4.2. PDF to TXT

Este primer módulo, como se mencionó en la sección 3.1.1, consta de tres partes o puntos clave.

Para obtener la ruta del archivo PDF de una manera cómoda, se implementó la librería *easygui*. Esta librería permite crear ventanas con entorno gráfico para realizar diversos trabajos. De esta manera, se evita tener que introducir a mano la ruta absoluta del archivo por el terminal o tener que colocar el archivo en un lugar específico con un nombre predefinido. Este enfoque facilita el paso de la ruta del archivo PDF a la librería encargada de convertir cada una de las páginas en imágenes.

El segundo punto consiste en el uso de *pytesseract* para la extracción de todos los textos de las páginas de los PDF convertidas en imágenes. Procesando cada una de las imágenes, se extrae todo el texto para ser almacenado en una variable string. Este proceso puede tomar entre 14 y 16 minutos debido a la intensa demanda de recursos.

Finalmente, el último punto consiste en la reconstrucción de palabras y la limpieza del texto. La conversión de imagenes a texto plano, genera una serie de errores que deben ser corregidos. Las palabras pueden aparecer cortadas por saltos de línea, aunque esto pueda parecer trivial, es vital mantener la consistencia del texto. Es por ello que, mediante el uso de una función llamada "process-dataset", se revisan cada uno de los caracteres del texto, y cuando se encuentra un guión, esta función se encarga de reconstruir la palabra, verificando si existe en el diccionario inglés. En caso de que la palabra no exista, simplemente se descarta. Para la verificación de las palabras, se utilizó la librería *spacy*.

Una vez reconstruido el texto, se procede a su limpieza. Basándonos en el estudio de Prafull Sharma et al. [1], se recomienda eliminar todos los caracteres que no sean letras o signos de puntuación, incluyendo números, de la misma manera que se muestra en la figura 13.

Raw Data	Cleaned Data
Descent Pass (77°51′S 163°5′E) is a pass leading from Blue Glacier to Ferrar Glacier, in Victoria Land, Antarctica. 77 ° 51.	Descent Pass is a pass leading from Blue Glacier to Ferrar Glacier, in Victoria Land, Antarctica.

Figura 13: *Ejemplo de como limpiar el texto antes de la extracción de términos clave.*

4.3. Key Extraction

El módulo de extracción de las términos clave es considerado el más importante dentro de este trabajo, pues la calidad de las preguntas está directamente relacionada con la calidad de las keyphrases/keywords extraídas en este punto.

Para abordar esta problematica, se encontró una librería sencilla de usar y muy eficaz, que además está basada en el estudio de Prafull Sharma et al.[1], llamada *KeyBERT*. La librería *KeyBERT* es una herramienta en Python diseñada para la extracción de palabras clave de textos. *KeyBERT* utiliza la técnica de embeddings de palabras basada en BERT para identificar y extraer las palabras clave más relevantes de un documento.

Debido a su simplicidad de su uso, la personalización a la hora de ajustar parámetros y la integración de varios modelos de embeddings, incluyendo modelos multilingües, se decidió finalmente utilizar esta librería .

Como se ha mencionado previamente, el ajuste de los parámetros era ideal para este trabajo, ya que permitía realizar una serie de experimentos. Estos experimentos, explicados en la sección 5.1, permitieron evaluar la calidad de los resultados en función de los parámetros establecidos.

Para procesar y comprender todo el corpus, se decidió utilizar la librería *sentence-transformers*. Esta librería convierte el texto del cual se desean extraer las palabras clave en vectores numéricos que los algoritmos de *KeyBERT* pueden interpretar.

Para la obtención de una mejor calidad de keyphrases, se estableció un enfoque particular con todo el conjunto del corpus. Durante las primeras fases de desarrollo de este trabajo, se observó que introducir todo el corpus directamente en la librería de *KeyBERT* no daba resultados satisfactorios. Las claves extraídas eran una serie de palabras demasiado generales que, si bien daban una idea del contexto del PDF, no eran lo suficientemente precisas para generar definiciones relevantes.

Dado que el enfoque híbrido en el que se basa la librería *KeyBERT* incluye un enfoque estadístico, se observó que la importancia de las keyphrases relevantes se diluía a lo largo de todo el corpus. Para solucionar esto, se pensó en dividir el texto en partes iguales. Para llevar a cabo esta tarea, se usó la librería *langchain*, que dispone de un splitter inteligente capaz de cortar fragmentos de texto sin dejar palabras entrecortadas y respetando los párrafos, y símbolos de puntuación. De esta manera, se consiguió obtener una serie de buenos resultados a la hora de extraer *keyphrases*.

Sin embargo, surgió un pequeño problema, dado que el temario de las diferentes secciones del PDF de ejemplo es muy irregular, esto provocaba que el algoritmo extrajera varias veces las mismas *keyphrases*. Para solucionar este inconveniente, se decidió usar un set de Python. Este tipo de estructura de datos permite guardar todas las keys sin que estas se repitan, solucionando el problema sin más complicación.

Finalmente, el conjunto o set que contiene todas las palabras clave, se decidio convertir en una lista para hacer más manejable su uso por el resto de módulos del código.

4.4. Clustering

En esta sección se detalla el proceso de clustering de keyphrases, una etapa crucial para organizar y categorizar las claves extraídas previamente. Durante el desarrollo, se pensó en las particularidades que debía de tener este clusterizado, pues lo que se buscaba es que se generará el máximo número de clusters que tuviesen más de una palabra.

Es por ese motivo que finalmente se decidió incorporar dos enfoques de clusterizado, *KMeans* y *Constrained Kmeans*. Debido a que ambos enfoques dan resultados bastantes parecidos se pensó en incorporar los dos. Con *KMeans*, utilizando el método de elbow strength, se puede crear el número óptimo de clusters, mientras que para crear clusters con un número mínimo de keys es mejor usar Constrained KMeans.

Una vez se tienen las keys y se selecciona la opción de clustering, el programa pasa a preguntar al usuario qué algoritmo de clusterizado desea usar. Si se selecciona KMeans, el proceso de clusterizado comenzará, si se elige el método de constrained, el programa preguntará al usuario cuál quiere que sea el tamaño mínimo de los clusters.

4.4.1. KMeans y Elbow Method

El algoritmo KMeans tiene la particularidad de que se le debe indicar el número k de clústeres que se desea, pero con el método mencionado, ese valor se determina automáticamente mediante una serie de cálculos.

Este método funciona de la siguiente manera:

- Se realiza el clusterizado por cada k posible del conjunto de datos.
- Por cada iteración se calcula la suma de cuadrados de cada centro con la fórmula 20 y se guardan los resultados en una lista para su posterior recuperación.
- El inconveniente que tiene este método es que es visual, es decir se ha de plasmar el resultado en una gráfica y ver donde esta el codo de la función y

$$\text{WCSS} = \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2$$

Fórmula WCSS (20)

encontrar el número óptimo de k. Para abordar esta limitación, se implemento una solución automatizada utilizando la función *np.diff* de *NumPy*. Este método permite buscar la posición en la gráfica donde la diferencia entre cada punto y su valor anterior es mayor. En otras palabras, busca el punto donde la curva presenta el cambio más significativo, lo que indica el número óptimo de clústeres.

- Finalmente recuperamos los clusters con la k óptima y se devuelve el resultado en formato lista.

4.4.2. Constrained KMeans

Para la implementación de *constrained KMeans* hacemos uso de la librería *k-means-constrained*, la cual permite definir el tamaño máximo y mínimo de los clusters. En de la problemática planteada, interesa establecer un mínimo que se preguntará a través del terminal al usuario.

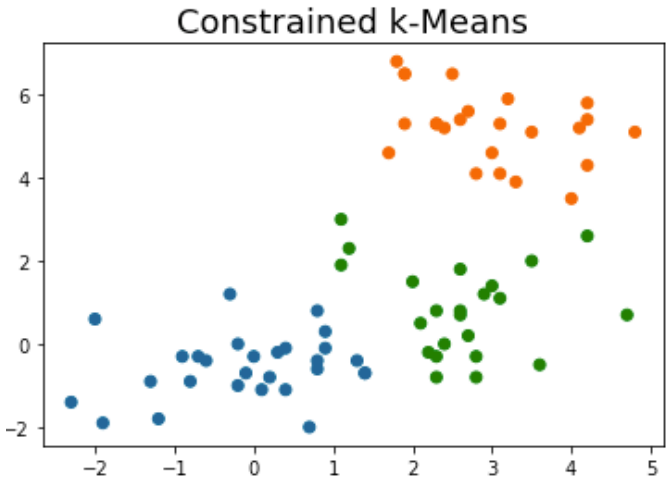


Figura 14: *KMeans constrained*

4.5. Generación de texto

Para este punto, finalmente se han tomado dos enfoques diferentes a la hora de buscar definiciones para las keyphrases obtenidas en los pasos anteriores.

El primer enfoque, que ya ha sido comentado, consiste en, mediante el uso de fine-tuning, entrenar un modelo de *GPT-2* y, usando *prompts*, generar definiciones para todos los conceptos de un clúster seleccionado por el usuario. Este punto requiere dos cosas principales. Primero, que se haya realizado la clusterización de los términos clave extraídos, ya que necesitamos que estén agrupados. Y segundo, necesita toda la información del archivo PDF convertida a formato TXT, para que pueda ser procesado por este modelo. Gracias a estos dos puntos, se busca la generación de definiciones de conceptos usando los mismos datos del documento.

El segundo enfoque consiste en el uso de la API de *OpenAI* para la generación de las definiciones. El problema de *GPT-2* es que sus definiciones se pueden considerar muy simples y, a veces, sin sentido, lo que hace que no sea el modelo idóneo para la resolución del problema planteado en el presenta trabajo. Simplemente preparando la clave de acceso a la API y el *prompt* de la misma manera que con *GPT-2*, es posible generar preguntas de una gran calidad.

En este apartado, mediante el uso del terminal, se pregunta al usuario que introduzca el número del clúster del cual desea generar las preguntas. Internamente, el programa guarda todo el clúster en una lista de listas. Seguidamente, se recupera el clúster y se procede a generar el *prompt* que deberá responder el modelo. Mediante una iteración, se van creando todos los *prompts* y guardando las preguntas y respuestas en una variable.

Para construir las preguntas, se itera sobre cada una de las *keyphrases* del cluster seleccionado. Dentro de esta iteración, se seleccionan aleatoriamente otras tres definiciones de manera aleatoria, excluyendo la definición de la keyphrase sobre la que se formulará la pregunta. Luego, se mezclan las definiciones y se guardan en una variable.

Es importante destacar que, para aumentar la dificultad y la utilidad de las preguntas para la autoevaluación, se sigue un proceso adicional. Aunque se indique a los modelos LLM que no incluyan el nombre de las *keyphrase* en las definiciones, a veces esta instrucción es ignorada. Para solucionar esto, se realiza un proceso de sustitución, se buscan las *keyphrases* en las definiciones y se reemplazan por la keyphrase de la pregunta.

De esta manera, conseguimos formular una pregunta con varias definiciones, donde estas definiciones no dejan claro a qué otros conceptos se refieren. Esto permite crear una pregunta tipo test en la que, si el usuario no tiene claro el concepto por el cual se le pregunta, responder no resulta tan sencillo.

5. Resultados

Durante el presente trabajo se han ido mostrando todos los conocimientos teóricos necesarios previos para su realización, la arquitectura pensada para la implementación de la solución y, finalmente, se ha explicado la implementación paso a paso de toda la solución, teniendo en cuenta todos los conocimientos previos adquiridos y requeridos.

Sin embargo, no es hasta este punto donde se van a mostrar todos los resultados y los análisis realizados para encontrar una combinación de parámetros que ofrezcan un mejor rendimiento de los algoritmos implementados.

A lo largo de este quinto punto se van a mostrar todos los análisis, junto con sus reflexiones y explicaciones de las decisiones tomadas, más los resultados que han dado estos. Mostrando así que, efectivamente, el planteamiento de este trabajo ofrece una solución que, si bien no es perfecta ni pretende serlo, puede estar encaminada a convertirse en una solución factible y satisfactoria.

5.1. Analisis

5.1.1. Embeddings

A la hora de extraer los términos clave, la primera duda que surgió fue cuál de todos los modelos de lenguaje ofrecidos por la página Hugging Face podría funcionar mejor. Si bien desde la propia página se indicaba que el modelo que ofrecía un mejor rendimiento es *all-mpnet-base-v2*, era necesario comprobar cuál de todos estos funcionaba mejor en la solución propuesta.

Para ello se decidió analizar los 7 mejores modelos según Huggin Face:

- *all-mpnet-base-v2*
- *all-MiniLM-L12-v2*
- *all-MiniLM-L6-v2*
- *all-distilroberta-v1*
- *multi-qa-distilbert-cos-v1*
- *multi-qa-mpnet-base-dot-v1*
- *paraphrase-MiniLM-L6-v2*

Model Name	Performance Sentence Embeddings (14 Datasets)	Performance Semantic Search (6 Datasets)	Avg. Performance	Speed	Model Size
all-mpnet-base-v2	69.57	57.02	63.30	2800	420 MB
all-distilroberta-v1	68.73	50.94	59.84	4000	290 MB
all-MiniLM-L12-v2	68.70	50.82	59.76	7500	120 MB
all-MiniLM-L6-v2	68.06	49.54	58.80	14200	80 MB
multi-qa-mpnet-base-dot-v1	66.76	57.60	62.18	2800	420 MB
multi-qa-distilbert-cos-v1	65.98	52.83	59.41	4000	250 MB
paraphrase-multilingual-mpnet-base-v2	65.83	41.68	53.75	2500	970 MB
paraphrase-albert-small-v2	64.46	40.04	52.25	5000	43 MB
multi-qa-MiniLM-L6-cos-v1	64.33	51.83	58.08	14200	80 MB
paraphrase-multilingual-MiniLM-L12-v2	64.25	39.19	51.72	7500	420 MB
paraphrase-MiniLM-L3-v2	62.29	39.19	50.74	19000	61 MB
distiluse-base-multilingual-cased-v1	61.30	29.87	45.59	4000	480 MB
distiluse-base-multilingual-cased-v2	60.18	27.35	43.77	4000	480 MB

Figura 15: *Embeddings ordenados por su rendimiento Hugging Face.*

El primer experimento consistió en extraer keyphrases con los mismos parámetros pero solo modificando el embedding con el cual se transforma el texto de entrada.

Los parámetros establecidos para el primer experimento fueron:

- **División del texto en trozos de 2500 caracteres con un margen de 20 caracteres para evitar cortar el texto.**
- **Obtención de n-gramas o keyphrases de 1 a 2 palabras.**
- **Obtener las top 3 keyphrases más descriptivas de cada chunk.**
- **Eliminación de las stopwords que existen en el diccionario inglés.**

Una vez realizada esta extracción, se revisaba una a una cuáles de las keys extraídas se podían considerar relevantes y cuáles se podían desechar, además de contar la cantidad total de keys extraídas por cada uno de los embeddings. El primer resultado del experimento se puede ver en la imagen 16. La primera columna guarda el nombre del embedding, la segunda guarda todas las keys obtenidas con ese embedding, la tercera muestra el total de keys obtenidas, independientemente de si son relevantes o no, la cuarta columna muestra el número final de keys que se pueden considerar relevantes o de las cuales se puede generar una definición, y finalmente en la última columna se muestra cuántas de esas keys obtenidas son irrelevantes y se han desechado.

	Embedding	Keywords	Unique_keywords	Relevant_words	Irrelevant_words
0	all-mpnet-base-v2	successful satisfiability, boolean satisfiabil...	378	325	53
1	all-MiniLM-L12-v2	recursion subproblems, test primality, rsa bob...	439	246	193
2	all-MiniLM-L6-v2	successful satisfiability, nodes tree, edges d...	398	248	150
3	all-distilroberta-v1	called memoization, deriving conclusions, algo...	379	305	74
4	multi-qa-distilbert-cos-v1	knight graph, rsa bob, subtree, expand recurre...	419	267	152
5	multi-qa-mpnet-base-dot-v1	boolean satisfiability, edges depthfirst, recu...	394	275	119
6	paraphrase-MiniLM-L6-v2	zoe variable, nodes nlog, backtracking reveals...	465	241	224

Figura 16: *Data Frame con los resultados del primer experimento.*

A primera vista, resalta que el embedding mostrado por Hugging Face como el que ofrece un mejor rendimiento, es el embedding con un mayor número de keys que pueden ser consideradas relevantes. Con un total de 325 *keys* relevantes para el contexto del documento, se posicionó como el mejor embedding, seguido por all-distilroberta-v1 con un total de 305 *keys*.

Como los resultados fueron bastante parecidos y ante la incapacidad de recordar el conjunto de todas las keyphrases obtenidas por los embeddings, se usó una matriz de coincidencia de palabras 17 para ver cuántas de ellas coinciden entre los diferentes modelos.

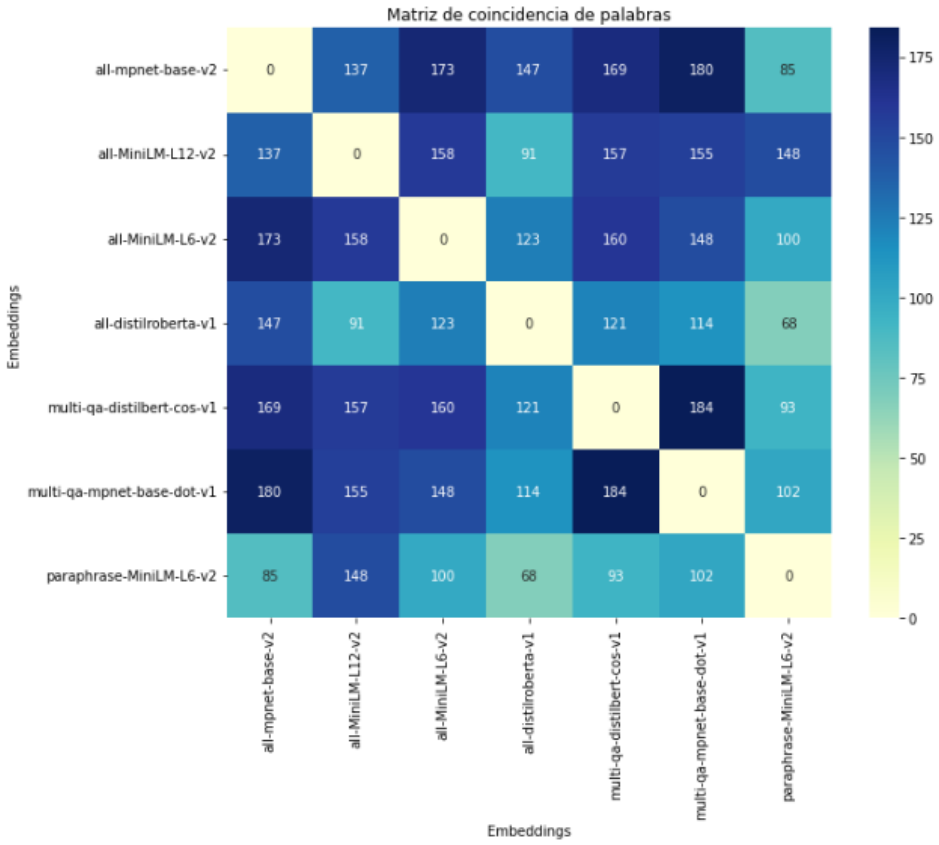


Figura 17: *Matriz de coincidencia de palabras.*

El siguiente paso a analizar fue cómo funcionaba el clusterizado con cada uno de estos embeddings, independientemente del resultado anterior. Para ello, usando el algoritmo de KMeans en combinación con el método del codo (*Elbow method*), se observó que el número óptimo de clusters en la mayoría de los embeddings ronda entre 80 y 100 clusters, a excepción de *paraphrase-MiniLM-L6-v2*, que ronda entre 50 y 75 clusters, ya que este extrajo in número menor de keyphrases.

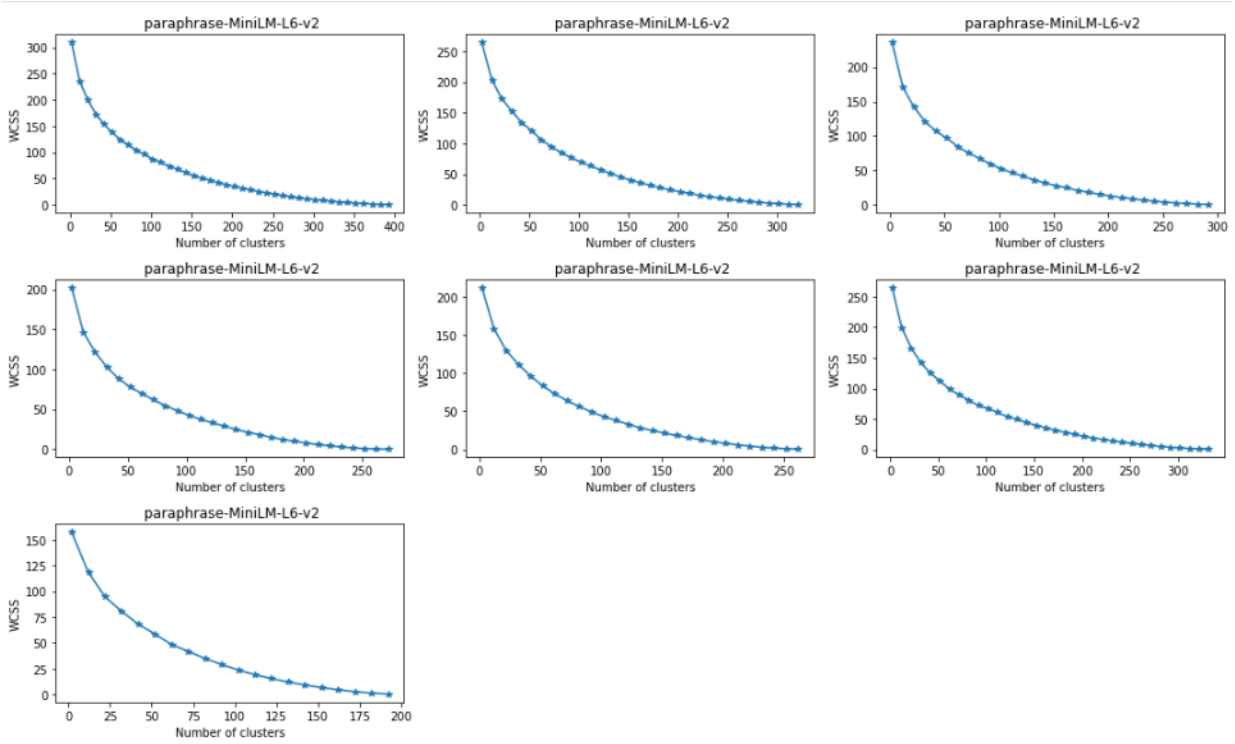


Figura 18: *Gráfico obtenido con elbow method con cada uno de los embeddings.*

Para este análisis se examinó la distribución del tamaño de cada uno de los clusters según el embedding utilizado. Este análisis fue realmente revelador, pues permitió decidir cuál de los embeddings era mejor entre todos los analizados. Aunque se sabe que se pueden generar clusters con un tamaño mínimo con KMeans constrained, es de vital importancia conocer cuántos clusters de una palabra genera cada uno de los embeddings. Que el algoritmo clasifique muchas keyphrases en un cluster de un solo concepto puede ser indicativo de que esa palabra sea un outlier, que no describe demasiado el contexto ni la relevancia del documento. Es por ello que es de un interés prioritario los embeddings que generen el mínimo número de clusters de una sola palabra, utilizando *KMeans* junto con el método de *elbow* y *elbow strength*.

En la imagen 19 se observa que, por el momento, los dos *embeddings* considerados mejores por *Hugging Face* obtienen buenos resultados. El modelo *all-mpnet-base-v2*, aunque tiene una gran cantidad de *clusters* de tan solo dos palabras, no tiene ni un solo *cluster* de una sola palabra. Mientras tanto, *all-distilroberta-v1* muestra

que tiene diez *clusters* de una sola palabra, pero demuestra una mejor distribución, es decir, tiene *clusters* de tamaño variado. Otro *embedding* a mencionar es *multi-qa-mpnet-base-dot-v1*, el cual también presenta una buena distribución a pesar de tener casi veinticinco *clusters* de una sola palabra.

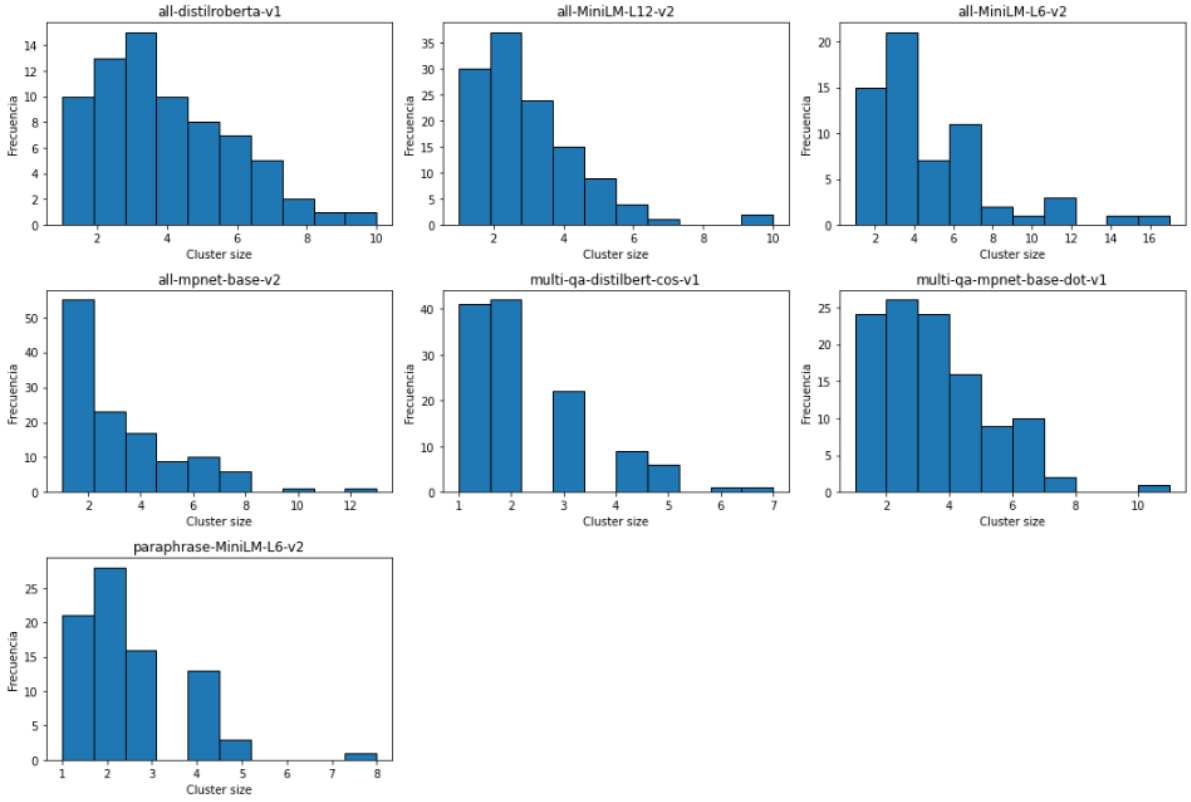


Figura 19: *Distribución de los tamaños de los clusters por cada uno de los embeddings.*

Teniendo en cuenta los datos obtenidos, la cantidad de *keys* relevantes extraídas por cada uno de los *embeddings* y los tamaños de los *clusters* generados, se puede decir que tanto *all-mpnet-base-v2* como *all-distilroberta-v1* son buenas opciones para el problema planteado en este trabajo. Sin embargo, en caso de tener que escoger solo uno, quizás la mejor opción sería *all-mpnet-base-v2*, ya que en esta problemática se busca principalmente generar el mínimo número de *clusters* con tan solo una palabra. Aunque *all-distilroberta-v1* genera buenos resultados, el primer *embedding* no genera ningún *cluster* con solo una palabra. Por ello, para la solución de la problemática presentada en este trabajo, se decidió utilizar *all-mpnet-base-v2* como *embedding* para la extracción de *keys*.

5.1.2. Top n palabras

Una de las características que presenta la librería *KeyBERT* es que te permite seleccionar el top n de palabras representativas del texto facilitado.

Si bien es cierto que el objetivo es extraer el máximo número de frases clave de un documento, este es un parámetro que se debe refinar, ya que puede resultar contraproducente generar más claves que no aporten nada. Si no se gestiona con cuidado, lo único que se puede conseguir es generar ruido.

Los parámetros establecidos para el primer experimento fueron:

- **División del texto en trozos de 3500 caracteres con un margen de 20 caracteres para evitar cortar texto.**
- **Obtención de n-gramas o keyphrases de 1 a 2 palabras.**
- **Embedding all-mpnet-base-v2.**
- **Eliminación de las stopwords que tiene el diccionario inglés.**

El resultado de este experimento se muestra en la siguiente imagen 21. La columna top El resultado de este experimento se muestra en la imagen 21. La columna *top* indica el número de palabras que se han extraído por cada trozo de texto, la segunda columna muestra el número total de claves únicas; la tercera, el porcentaje de claves eliminadas, la cuarta, el número total de clústeres generados, la quinta y sexta, el total de clústeres modificados, es decir, el número de clústeres donde se ha eliminado al menos una palabra y el porcentaje que representan, y las dos últimas columnas representan los clústeres completamente eliminados por tener claves irrelevantes y el porcentaje que representan del total.

	top	nº total keys	nº de keys eliminadas	% de keys eliminadas	nº total de Clusters	nº de Cluster modificados	% Cluster modificados	nº de Cluster eliminados	% Cluster eliminados
0	top 2	279	74	26.5	106	27	25.5	25	23.6
1	top 3	411	138	33.6	82	48	58.5	8	9.8
2	top 4	543	188	34.6	186	58	31.2	44	23.7

Figura 20: *Resultado del experimento con el parametro top words de la libreria keyBERT.*

En esta tabla de resultados se puede observar que si se extraen 3 o 4 claves de cada uno de los fragmentos del texto, se pueden desechar más de un tercio de las claves. Sin embargo, se observa un efecto curioso y es que aparentemente el porcentaje de clústeres eliminados suele rondar el 23 %, excepto cuando extraemos las top 3 claves.

Este análisis nos permite ver que, aparentemente, cuando extraemos 3 claves por fragmento de texto, el programa parece distribuir el ruido de estas claves irrelevantes, lo que resulta en la modificación del 58 % de los clústeres al eliminar algunas

de sus claves para el experimento. Mientras que si escogemos extraer 2 o 4 palabras, el algoritmo distribuye esas palabras irrelevantes en clústeres, lo que resulta en la eliminación del 23 % de los clústeres. Debido a que se busca generar el mayor número de claves relevantes e intervenir lo mínimo posible a la hora de revisar y comprobar la calidad de las claves extraídas, una opción viable es obtener las 3 palabras más descriptivas de cada texto. Esto se debe a que un punto medio puede ser más beneficioso, se extraen más claves aunque se genere un poco más de ruido.

5.1.3. Tamaño de las keys extraídas

Otro de los parámetros que permite modificar *KeyBERT* es el tamaño que deben tener las claves extraídas. Es por ello que se considera conveniente realizar los mismos experimentos pero con *keyphrases* de hasta tres palabras.

Los parámetros establecidos para el primer experimento fueron:

- **División del texto en chunks de 2500 caracteres con un margen de 20 caracteres para evitar cortar texto.**
- **Obtención de n-gramas o keyphrases de 1 a 3 palabras.**
- **Obtener las top 3 keyphrases más descriptivas de cada chunk.**
- **Eliminación de las stopwords que tiene el diccionario inglés.**

El resultado es el que se ve en la tabla 21. La primera columna guarda el nombre del embedding, la segunda guarda todas las keys obtenidas con ese embedding, la tercera muestra el total de keys obtenidas, independientemente de si son relevantes o no, la cuarta columna muestra el número final de keys que se pueden considerar relevantes o de las cuales se puede generar una definición, y finalmente en la última columna se muestra cuántas de esas keys obtenidas son irrelevantes y se han desechado.

	Embedding	Keywords	Unique_keywords	Relevant_words	Irrelevant_words
0	all-distilroberta-v1	efficient algorithm, maxflow algorithms, great...	900	273	627
1	all-MiniLM-L12-v2	shortest path, binary algorithm, simulated ann...	933	332	601
2	all-MiniLM-L6-v2	efficient algorithm, shortest path, composite ...	923	302	621
3	all-mpnet-base-v2	maxflow algorithms, coloring graph basic, bina...	903	401	502
4	multi-qa-distilbert-cos-v1	shortest path, heap node tree, color graph, un...	930	270	660
5	multi-qa-mpnet-base-dot-v1	shortest path, biconnected components graph, s...	911	342	569
6	paraphrase-MiniLM-L6-v2	shortest path, color graph, addresses ips hash...	960	200	760

Figura 21: *Tabla con los resultados del primer experimento.*

Como se puede comprobar, la cantidad de claves únicas generadas por todos los embeddings se incrementa en gran medida en comparación cuando solo tienen un

tamaño máximo de 2 palabras por clave. Sin embargo, se evidenció un problema que llevó a no continuar con el experimento, la cantidad de claves irrelevantes que no aportaban nada. Al intentar extraer tanta información de los fragmentos de texto, el resultado que se puede interpretar es que las claves son demasiado grandes y llega un momento en que dejan de ser relevantes.

Es por este motivo que finalmente se decidió dejar el tamaño de las claves entre 1 y 2 palabras.

5.2. Clusters y keyphrases

En este apartado se mostrarán algunos de los clústeres generados por el código implementado. Todos los clústeres que se muestran a continuación han sido obtenidos con el modelo *all-mpnet-base-v2*, usando fragmentos de texto de 3500 caracteres y extrayendo tres palabras por cada uno de ellos.

Como se comentó en la sección de análisis, se puede observar que en los clústeres existen algunas *keyphrases* que no tienen demasiada relevancia. Esto se debe a que en la fase de extracción de claves siempre se generan algunas claves que no son demasiado relevantes. Estas claves que quizás no son satisfactorias simplemente se pueden eliminar justo antes de comenzar el clústerizado. Se han dejado para mostrar los resultados modificándolos lo menos posible.

A continuación, se presentan algunos de los clústeres formados con Constrained KMeans para que tengan un tamaño mínimo de 4 *keyphrases*.

Clusters y sus Temas Representativos

1. **Seguridad y Criptografía RSA:** rsa protocol, security rsa, algorithm rsa, rsa scheme, rsa cryptosystem.
2. **Problema de la Mochila:** knapsack approximation, algorithm knapsack, knapsack subset, general knapsack, knapsack repetition, knapsack problem.
3. **Cortes en Grafos:** minimum cut, smallest cut, min cut, balanced cuts, multiway cut, minimum cuts, maximum cut, cut algorithm, cut maximizing.
4. **Búsqueda en Grafos:** depth search, crawling web, search breadth, breadth search, binary search.
5. **Árboles en la Teoría de Grafos:** connected tree, graph tree, binary tree, spanning trees, spanning tree, tree binary
6. **Grafos Dirigidos:** directed graphs, graphs eulerian, acyclic graph, graph theory
7. **Criptografía:** cryptographic methods, cryptography theory, encryption, cryptographic scheme

8. **Codificación Huffman:** huffman encoding, length encoding, shortest encoding, encoding alphabet, encoding tree

El análisis de los clústeres generados con el *embedding all-mpnet-base-v2* y el método de Constrained KMeans ha demostrado ser efectivo para agrupar *keyphrases* relevantes en el contexto de nuestro estudio. Los resultados obtenidos muestran una buena distribución de las *keyphrases*, aunque algunos clústeres contienen palabras de menor relevancia.

La implementación de un tamaño mínimo para los clústeres ha ayudado a asegurar que cada grupo contenga información más sustancial, reduciendo la presencia de clústeres de una sola palabra que pueden ser considerados *outliers*.En general, los clústers generados, porporcionan una buena base para la generación de definiciones y la posterior construcción de preguntas.

5.3. Definiciones generadas

En esta parte se mostrarán las definiciones generadas por el modelo de *GPT-2* y por la *API de GPT-3.5 turbo*. Como era de esperar, las definiciones generadas por *GPT-3.5* son mucho mejores que las generadas por su antecesor *GPT-2*.

De los clústeres anteriores se ha seleccionado el clúster sobre árboles en la Teoría de Grafos para la generación de las definiciones, debido a que sus *keyphrases* están bastante bien formadas. Se ha seleccionado solo uno de los clústeres para no sobresaturar el trabajo con demasiados ejemplos.

5.3.1. GPT-2

What is the definition of connected tree? Well, at the time of writing, we know that it is a term we used when we started developing connected tree, so that we could learn how to build or use your own connected tree. Before that article I wrote about doing your own connected tree, but here's what I think makes connected tree important: It's like taking a bunch of wires and going, "Hey, my wire will come out of the wiring". It's like

What is the definition of graph tree? The term graph is used in conjunction with graphs to express the visual representation of a process, such as a diagram. Each graph represents a process or a set of processes. A process is a system by which a process moves into and out of a location. The term graph describes a system which describes how a process or processes move into and out of a location. Each graph represents a process in a system. A process is

What is the definition of binary tree? In the previous post I mentioned an example using the binary tree. But how do you define binary tree using "binary tree¿ What is the value of the binary tree? The definition of binary tree is about an .ºbject"where everything happens as an integral tree. In this case it makes sense to define "binary tree.ªs the "binary tree.ªnd the binary tree as a binary tree. In the previous post I wrote

What is the definition of spanning trees? The tree, if you will, is a type of a tree whose root points to a single vertex, or a set of vertex types, and its direction (in direction lines) is a branch line, or the tree with its vertices, if there is no branch line, or the set of vertices, if there is no branch line, or a set of vertices (each of which holds the same value of the given branch line), and any

What is the definition of tree binary? In a nutshell, what is a tree binary? The concept is: the node in the tree is a constant that gives the shortest possible path to its root and returns the shortest possible point at that point on the tree. Tree binary is often used in the creation of an algorithm, by making the nodes of a function call different and vice versa; hence, to be able to do many kinds of programming (such as manipulating the result of an

5.3.2. GPT-3.5

What is the definition of connected tree? A connected tree is a data structure comprising of nodes linked together in a hierarchical manner, where each node has at most one parent but can have multiple children. It is characterized by having a unique starting node known as the root, from which all other nodes branch out. This structure promotes organization and efficient storage of information in a systematic manner, facilitating easy traversal and manipulation of data.

What is the definition of graph tree? A graph tree is a special type of graph that consists of nodes connected by edges. It is a connected undirected graph with no cycles, meaning there is only one unique path between any two nodes. The node without any incoming edges is called the root, and all other nodes have exactly one incoming edge. Graph trees are commonly used in computer science and mathematics to represent hierarchical structures and relationships.

What is the definition of binary tree? A spanning tree of a connected, undirected graph is a subgraph that is a tree and includes all vertices of the original graph. It contains the minimum number of edges required to connect all vertices without creating any cycles. Spanning trees are used in network design, optimization algorithms, and graph theory to analyze the relationships and connectivity within a graph.

What is the definition of spanning trees? A spanning tree is a subgraph of a connected, undirected graph that includes all the vertices of the original graph with the minimum possible number of edges. In a spanning tree, all vertices are connected with no cycles, making it a tree structure. Spanning trees are used in various algorithms and applications, such as network design and optimization problems.

What is the definition of tree binary? A tree binary is a type of data structure in computer science that consists of nodes linked together in a hierarchical manner. Each node has at most two children nodes, referred to as the left and right child. This structure resembles a tree with parent nodes branching out into child nodes. Tree binaries are commonly used to represent hierarchical relationships and perform operations such as insertion, deletion, and traversal efficiently.

Como se puede comprobar, los textos generados por GPT-2 no son definiciones como tal, más bien, parece que el modelo LLM ha extraído fragmentos de texto donde se mencionan los conceptos sin procesarlos adecuadamente. Y aunque los textos generados tienen relación con la pregunta, pueden no ser aptos para funcionar como respuestas en una pregunta de tipo test.

En cambio, el modelo GPT-3.5 genera definiciones que pueden ser utilizadas como respuestas en una pregunta de tipo test. Aquí se demuestra el mejor rendimiento que tiene el modelo de GPT-3.5 respecto a su predecesor.

5.4. Preguntas Generadas

En esta última sección se mostrarán una serie de preguntas generadas a partir de los clusters previamente mostrados. Las siguientes preguntas han sido realizadas con el cluster sobre la Codificación Huffman, Árboles en la Tería de Grafos y Búsqueda en Grafos.

En las preguntas de opción múltiple, a veces las respuestas incorrectas contienen definiciones de conceptos relacionados al mismo grupo del concepto que se está preguntando. Sin embargo, estas respuestas pueden ser fácilmente descartadas si contienen los nombres de esos conceptos relacionados. Para evitar esto, se sustituyen esos nombres por el concepto por el cual se está realizando la pregunta. Por ejemplo, si se pregunta sobre la definición de "binary search", y una respuesta incorrecta contiene la definición de *depth search*, se sustituiría *depth search* por *binary search* en esa respuesta para que no sea tan obvia que es incorrecta. De esta manera, se consigue un desafio autentico para el usuario que esta respondiendo.

Codificación Huffman

QUESTION: What is the definition of huffman encoding?

1. Huffman encoding is a method used for lossless data compression where frequently occurring symbols are assigned shorter binary codes, while less frequent symbols are assigned longer binary codes. **(Correct)**

2. An huffman encoding is a set of characters or symbols used to represent information in a way that allows for efficient storage and transmission, typically by assigning unique codes to each character. **(encoding alphabet)**

3. An huffman encoding is a hierarchical structure used to represent a set of data elements where each element is assigned a unique sequence of bits based on its position within the tree. **(encodign tree)**

4. The process of representing data using the minimum number of bits or symbols possible. **(shortest encoding)**

Árboles en la Tería de Grafos

QUESTION: What is the definition of graph tree?

1. A graph tree of a connected graph is a subgraph that includes all the vertices of the original graph and forms a tree, ensuring that there is exactly one path between any two vertices. **(Correct)**

2. A graph tree is a data structure that consists of nodes connected by edges, where each node has exactly one parent node except for one special node called the root which has no parent. Additionally, there are no cycles or loops in a graph tree, meaning you can only traverse from parent nodes to child nodes in a hierarchical manner. **(spannig tree)**

3. A data structure that consists of nodes connected by edges in a branching pattern with no loops or cycles. **(connected tree)**

4. A graph tree is a subset of a connected, undirected graph that includes all the vertices of the graph with the minimum possible number of edges such that the graph becomes a tree. **(binary tree)**

Búsqueda en Grafos

QUESTION: What is the definition of binary search?

1. Binary search is an algorithm for traversing or searching tree or graph structures. It starts at the root and explores as far as possible along each branch before backtracking. **(depth search)**

2. Binary search is an algorithm for finding a target value within a sorted array by repeatedly dividing the search interval in half. **(Correct)**

3. Binary search is an algorithmic strategy used in computer science to systematically explore and analyze a graph or tree data structure. It involves systematically visiting and processing all nodes at a particular level of the structure before moving on to the next level. **(breadth search)**

4. Binary search refers to the automated process in which a software program systematically browses the internet to discover and index web pages. **(crawling web)**

6. Conclusiones

Después de realizar este trabajo para solucionar la problemática planteada, hemos obtenido una serie de conclusiones. Las investigaciones previas, la búsqueda de trabajos similares y el análisis de los resultados utilizando diferentes parámetros han permitido llegar a varias conclusiones y consideraciones sobre este proyecto que deben ser tomadas en cuenta para posibles mejoras.

Es importante subrayar que, si bien la necesidad de supervisión está dentro de la sección de problemas encontrados, esto no es algo completamente negativo. De hecho, en el módulo de extracción de términos clave, una ventaja de la técnica propuesta es que proporciona control al usuario, a diferencia del uso directo de ChatGPT, que actúa como una caja negra. La supervisión del usuario permite gestionar y eliminar aquellas claves que no aportan valor, mejorando así la precisión y relevancia del contenido generado.

6.1. Problemas encontrados

A lo largo del trabajo, en cada uno de los módulos han surgido una serie de problemas que serán explicados en esta sección.

6.1.1. Necesidad de supervisión

Aunque el objetivo final de este TFG es la automatización de un proceso complejo para agilizar el trabajo tanto del profesorado como del alumnado, toda generación de contenido por parte de algoritmos de inteligencia artificial siempre requiere cierta supervisión. Estos algoritmos tienen la posibilidad de generar errores o información incorrecta.

A lo largo de todas las fases, siempre se han generado pequeños errores que se han intentado solucionar. Pero aun teniendo medidas para corregirlos, los grandes modelos de lenguajes, siempre pueden generar definiciones que no son correctas o no se acaban de adaptar al contexto del todo. Es por ello que se requiere una supervisión de las preguntas por parte del usuario para ver que realmente sean vàlidas

6.1.2. Capacidad computacional

Para desplegar *LLM* y realizar lo que se llama *prompt engineering*, se necesita mucha potencia de cálculo. Durante el desarrollo de este trabajo, faltó acceso a una maquinaria de altas prestaciones.

Aunque existen alternativas incorporadas a este trabajo, como el uso de la API de pago de *OpenAI*, hay muchos modelos tan potentes como *GPT-3.5* que son

completamente gratuitos, pero que, lamentablemente, requieren un ordenador muy potente para ser desplegados localmente.

Modelos como *Mistral-7B* o *Llama2* son *LLM* completamente gratuitos que serían perfectos para la generación de definiciones de conceptos y *fine-tuning*, pero necesitan entornos de desarrollo con mucha potencia. Aunque disponen de versiones cuantizadas, las respuestas obtenidas no son tan satisfactorias.

6.2. Posibles Mejoras

6.2.1. Eliminación de Keyphrases no relevantes

Para la eliminación de *keyphrases* no relevantes existen dos posibles mejoras. La primera consiste en eliminar las claves que queden solas dentro de un cluster usando el método de *elbow*. Esto se debe a que si una palabra no queda debidamente clasificada dentro de un cluster, podría ser un indicio de que esa clave es un *outlier* y no es relevante para el contexto del documento.

La segunda solución estaría basada en un enfoque lingüístico. Es decir, analizar a qué categoría sintáctica pertenecen cada una de las palabras que conforman cada una de las *keyphrases* y eliminar aquellas en las que aparezca un verbo, ya que normalmente un verbo no suele contener información valiosa de un documento.

6.2.2. Implemetación de una interfaz de Frontend

Otra posible solución para ofrecer una mejor selección de keyphrases sería la implementación de una interfaz gráfica donde se mostraran todos los clusters obtenidos por el programa. De tal manera, que estos pudieran ser modificados, ya sea eliminando keyphrases o moviéndolos entre clusters.

Esta interfaz permitiría a los usuarios tener un mayor control sobre el proceso de extracción y organización de keyphrases, facilitando la identificación y corrección de errores que el algoritmo pueda generar. Además, proporcionaría una manera más intuitiva y eficiente de interactuar con los datos, mejorando la experiencia del usuario y la calidad de los resultados finales.

La implementación de esta interfaz gráfica podría realizarse utilizando tecnologías web modernas como React, Angular, o Vue.js para el frontend, y una API backend para gestionar las operaciones de modificación y actualización de los clusters.

Referencias

[1] Prafull Sharma, Yingbo Li (2019). *Self-supervised Contextual Keyword and Keyphrase Retrieval with Self-Labelling*, https://www.preprints.org/manuscript/201908.0073/download/final_file

[2] Hasan, Kazi Saidul, and Vincent Ng. *Automatic keyphrase extraction: A survey of the state of the art."Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* , (Volume 1: Long Papers). Vol. 1. 2014.

[3] Rose S, Engel D, Cramer N, et al. *Automatic keyword extraction from individual documents[J].*, Text mining: applications and theory, 2010: 1-20.

[4] Sheng Zhou, Hongjia Xu, Zhuonan Zheng, Jiawei Chen, Zhao li, Jiajun (2022). *A Comprehensive Survey on Deep Clustering: Taxonomy, Challenges, and Future Directions*, https://ar5iv.labs.arxiv.org/html/2206.07579

[5] Beliga S, Meštrović A, Martinčić-Ipšić S. *An overview of graph-based keyword extraction methods and approaches[J]. Journal of information and organizational sciences*, 2015, 39(1): 1-20.

[6] Lahiri S, Choudhury S R, Caragea C. *Keyword and keyphrase extraction using centrality measures on collocation networks[J].*,

[7] Page L, Brin S, Motwani R, et al. *The PageRank citation ranking: Bringing order to the web[R]*, Stanford InfoLab, 1999.

[8] Tim Schopf , Simon Klimek, Florian Matthes (2022) *PatternRank: Leveraging Pretrained Language Models and Part of Speech for Unsupervised Keyphrase Extraction*,

[9] Zhang Q, Wang Y, Gong Y, et al. *Keyphrase extraction using deep recurrent neural networks on Twitter*, Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. 2016: 836-845

[10] Murphy, K. P. (2012) *Machine Learning: A Probabilistic Perspective.*, MIT Press. https://vdoc.pub/download/machine-learning-a-probabilistic-perspective-5nh9osgl8qq0

[11] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Jauvin (2003) *A Neural Probabilistic Language Model*, https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

[12] Toma´s Mikolov, Martin Karafiat, Luka´s Burget , Jan "Honza" Cernock, Sanjeev Khudanpur (2010). *Recurrent neural network based language model*, https://www.fit.vutbr.cz/research/groups/speech/publi/2010/mikolov_interspeech2010_IS100722.pdf

[13] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio (2015) *Neural Machine Translation by Jointly Learning to Align and Translate*, https://arxiv.org/abs/1409.0473

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan, N. Gomez, Łukasz Kaiser (2017) *Attention Is All You Need*, https://arxiv.org/pdf/1706.03762.pdf

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, https://arxiv.org/abs/1810.04805 Meng R, Zhao S, Han S, et al. Deep keyphrase generation[J]. arXiv preprint arXiv:1704.06879, 2017

[16] Meng R, Zhao S, Han S, et al. . *Deep keyphrase generation[J].*, preprint arXiv:1704.06879, 2017

[17] Rose S, Engel D, Cramer N, et al. (2010). *Automatic keyword extraction from individual documents[J]. Text mining: applications and theory* https://www.researchgate.net/publication/227988510_Automatic_Keyword_Extraction_from_Individual_Documents

[18] Beliga S, Meštrović A, Martinčić-Ipšić S.(2015). *An overview of graph-based keyword extraction methods and approaches[J]. Journal of information and organizational sciences* 39(1): 1-20

[19] R. Barzilay, M. Elhadad (1999). *Using lexical chains for text summarization," Advances in automatic text summarization* pp. 111-121

[20] A. Hulth (2003). *"Improved automatic keyword extraction given more linguistic knowledge* n: Proceedings of the 2003 conference on Empirical methods in natural language processing, ACL, 2003, pp. 216-223.

[21] G. Salton, A. Singhal, M. Mitra, C. Buckley (1997). *"Automatic text structuring and summarization," Information Processing & Management,* vol. 33 (2), 1997, pp. 193-207

Anexo 1

MANUAL DE USUARIO

Para el uso del código se necesitará la instalación previa de Tesseract - OCR. En el siguiente enlace se muestra cómo se debe instalar según la distribución de tu sistema operativo. Es necesario instalar Tesseract para poder usar la librería pytesseract. `https://github.com/tesseract-ocr/tessdoc`

El siguiente paso será la creación del entorno virtual.

```
python -m venv venv
```

Luego, se activa el entorno virtual:

```
. ./venv/Scripts/activate
```

Seguidamente, se deberán instalar todas las librerías que contiene el archivo `requirements.txt`.

```
pip install -r requirements.txt
```

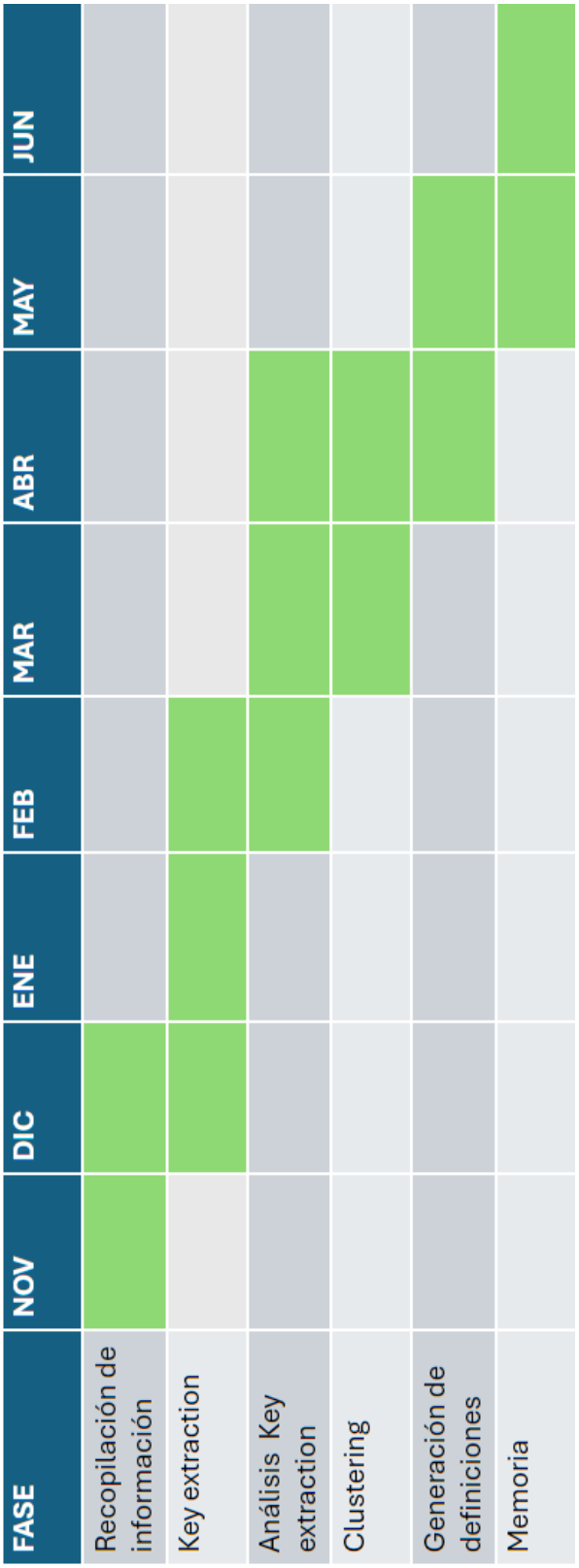
Finalmente, se deberá instalar el modelo encargado de reconocer palabras cuando estas se reconstruyen en el módulo `pdf_to_txt`.

```
python -m spacy download en_core_web_
```

Se debe mencionar que para poder usar la API de OpenAI con *GPT-3.5 Turbo* es necesario registrarse y obtener una clave de uso de la API. Existe una versión de prueba gratuita con un número limitado de usos en el siguiente link.

`https://platform.openai.com/docs/overview`

Anexo 2


FASE
NOV
DIC
ENE
FEB
MAR
ABR
MAY
JUN
Recopilación de información
Key extraction
Análisis Key extraction
Clustering
Generación de definiciones
Memoria