Generación automática de resúmenes extractivos genéricos de múltiples documentos basado en Word2Vec

Álvaro José Echeverría Restrepo

Director: PhD. Carlos Alberto Cobos Lozada

Co-Director: PhD. Martha Eliana Mendoza Becerra

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas
Grupo de I+D en Tecnologías de la Información (GTI)
Área de Interés: Gestión de la Información y Sistemas Inteligentes
Popayán, Abril de 2018

Tabla de contenido

LISTA DE FIGURAS

LISTA DE TABLAS

1 INTRODUCCIÓN

1.1 Planteamiento del problema

Cada día más personas se acercan al mundo de la tecnología y las comunicaciones para encontrar información que les permita resolver diversas dudas de la vida diaria y compartir datos e información que creen útiles para los demás. Es por esto, que en la actualidad se encuentra en internet una gran cantidad de información en documentos o textos digitales, que crece exponencialmente día a día. Cuando un usuario desea buscar sobre cierto tema, se encuentra con una gran cantidad de documentos que contienen la información deseada pero difícilmente puede ser leída en su totalidad debido a la sobrecarga de información, gastando así mucho tiempo y esfuerzo para encontrar lo que realmente necesita. Debido a esto, surge la necesidad de acceder a la información en forma resumida y contar con resúmenes que preserven la información más importante de los documentos [1, 2].

El área de investigación de generación automática de resúmenes de textos desde el siglo pasado ha utilizado diferentes técnicas para resolver éste problema [3-18]. Sin embargo, dada la complejidad de generar un resumen de calidad (similar al generado por un humano) se sigue en la búsqueda de técnicas que mejoren los resultados obtenidos hasta el momento por los algoritmos que hoy son el estado del arte [2].

Entre los principales métodos de generación automática de resúmenes extractivos para un documento se encuentran [4]: métodos estadísticos [6-8], basados en técnicas de reducción algebraica [6, 7], basados en técnicas de aprendizaje de máquina [7, 8], basados en conectividad de textos [6], basados en grafos [6-8] y basados en modelos evolutivos. Para generar automáticamente resúmenes de múltiples documentos se encuentran: basados en técnicas de reducción algebraica [6, 7], basados en agrupamiento [9-11], modelos probabilísticos [12, 13], basados en conectividad de textos [6], basados en grafos [6-8], modelos evolutivos, combinaciones de métodos o métodos híbridos [9, 14-18] entre otros. De estos métodos, los basados en grafos, reducción algebraica, agrupamiento, modelos probabilísticos y metaheurísticas, son independientes del lenguaje y no supervisados, es decir que no tienen dependencia del idioma del documento y no necesitan realizar una etapa de entrenamiento previo.

El Grupo de I+D en Tecnologías de la Información (GTI) del Departamento de Sistemas de la Universidad del Cauca, ha experimentado con diversas técnicas para la generación de resúmenes basadas en grafos, en reducción algebraica y metaheurísticas, obteniendo buenos resultados con los algoritmos de LexRank, LSI y MCMR [19-21], pero se observó que la calidad de los resúmenes podría mejorar si se utilizan esquemas de representación de las oraciones que tengan más en cuenta la semántica de los términos, en este sentido, se consideró que usar un esquema de representación basado en una nueva tecnología presentada por Google Inc. llamada Word2Vec [22] (véase sección 2.2) podría brindar mejores resultados.

Word2Vec ha sido utilizado hasta el momento en traducciones de documentos o en la búsqueda de la similitud entre palabras, ya que organiza las palabras según su contexto [22], pero hasta ahora muy poco se ha explorado en el área de investigación de generación automática de resúmenes de textos.

Teniendo en cuenta que la tecnología de Word2Vec ha sido usada en diversos entornos para obtener mejores soluciones informáticas [23-25] y que ya existen unos primeros acercamientos en tareas afines a la generación automática de resúmenes [26, 27], en esta investigación se planteó la siguiente pregunta de investigación: ¿Cómo integrar Word2Vec como mecanismo de representación de oraciones, buscando obtener resúmenes extractivos genéricos de múltiples documentos que tengan mejor calidad usando los algoritmos LexRank, LSI y MCMR?

1.2 Justificación

Uno de los mayores problemas que enfrentan los investigadores en temas de computación inteligente es la representación que deben utilizar en sus algoritmos. Es por esto que en el presente trabajo se buscó una nueva forma de representar los documentos a ser resumidos mediante diferentes algoritmos de generación automática de resúmenes.

En el área de la generación automática de resúmenes se han realizado diversos esfuerzos para obtener cada vez mejores resultados (resúmenes similares a los realizados por humanos) ya que para los usuarios es importante obtener la información precisa cuando acceden a un resumen. La propuesta presentada en este trabajo genera conocimiento teórico a la comunidad académica y científica en el área, además presenta información y resultados de experimentos que pueden ser usados para comparar con los resultados de nuevas propuestas de investigación.

Mediante la integración de Word2Vec como mecanismo de representación de documentos para la generación automática de resúmenes extractivos genéricos de múltiples documentos se expandieron los conocimientos del Grupo de I+D en Tecnologías de la Información (GTI) del Departamento de Sistemas de la Universidad del Cauca. Además, se fortaleció el programa de Ingeniería de Sistemas a través de la retroalimentación obtenida, para ser aplicada a los algoritmos en asignaturas electivas y constituir futuros trabajos de grado.

1.3 Objetivos

A continuación, se presentan el objetivo general y los objetivos específicos del presente trabajo tal como fueron aprobados en el anteproyecto de grado aprobado por parte del Consejo de Facultad de la Facultad de Ingeniería Electrónica y Telecomunicaciones.

1.3.1 Objetivo General

Proponer un mecanismo de representación de oraciones basado en Word2Vec que permita a los algoritmos LexRank, LSI y MCMR obtener mejores resúmenes extractivos genéricos de múltiples documentos según las medidas ROUGE.

1.3.2 Objetivos Específicos

1. Adaptar el modelo propuesto en Word2Vec para representar oraciones de múltiples documentos que serán resumidas de forma extractiva y genérica.
2. Integrar el modelo propuesto de Word2Vec como modelo de representación de oraciones, en tres algoritmos para la generación automática de resúmenes extractivos genéricos de múltiples documentos (LexRank, LSI y MCMR).
3. Evaluar la calidad de los resúmenes generados por los tres algoritmos que integran el modelo de representación Word2Vec, sobre documentos de noticias de la Conferencia de Entendimiento del Documento (DUC2005, DUC2006 y DUC2007) utilizando las medidas de ROUGE-1, ROUGE-2 y ROUGE-SU4, y comparar los resultados obtenidos con los reportados por otros algoritmos del estado del arte.

1.4 Resultados obtenidos

La **Tabla 1** lista los resultados obtenidos en el presente trabajo de grado en relación con la generación de nuevo conocimiento y desarrollo tecnológico.

Tabla 1 Resultados obtenidos.

Producto	Indicador
Nueva representación para algoritmos de generación automática de resúmenes extractivos genéricos de múltiples documentos.	La presente monografía de trabajo de grado.
Algoritmos base: LexRank, LSA y MCMR, para la generación automática de resúmenes extractivos genéricos de múltiples documentos.	Anexo A de la presente monografía.
Artículo con los resultados del algoritmo de generación automática de resúmenes extractivos genéricos de múltiples documentos basado en Word2Vec.	Anexo B de la presente monografía.
Representación de documentos basado en Word2Vec e implementado en tres algoritmos de generación automática de resúmenes.	Herramienta software en VS.NET que implementa los algoritmos con el esquema de representación basado en Word2Vec y permite la evaluación del mismo sobre colecciones de datos DUC usando medidas ROUGE.
Manual de instalación para	Anexo C de la presente monografía.

para la configuración del entorno de desarrollo de Herramienta que implementa los algoritmos con el esquema de representación basado en Word2Vec.	

1.5 Organización del resto del documento

A continuación, se introducen los temas que se tratan en cada uno de los capítulos del resto del documento.

Capítulo 2: Contexto teórico y estado del arte. En este capítulo se presentan los conceptos necesarios para la comprensión de la generación automática de resúmenes extractivos genéricos de múltiples documentos y algunos algoritmos sobresalientes. Se introduce Word2Vec, se describe su funcionamiento y se presentan algunos ejemplos. Además se presentan medidas y modelos de representación utilizados para la generación automática de resúmenes.

Capítulo 3: Se muestra específicamente como se usó Word2Vec en la generación automática de resúmenes de múltiples documentos. Este capítulo muestra las diferentes formas identificadas para la representación de los documentos basados en Word2Vec y como se adaptaron los algoritmos LexRank, LSI y MCMR a dichas representaciones.

Capítulo 4: Experimentación. Este capítulo muestra las métricas utilizadas para evaluar la calidad de los resúmenes generados por los algoritmos llamadas medidas ROUGE y los resultados obtenidos en la etapa experimental sobre los datasets DUC2005, DUC2006 y DUC2007. Además, un análisis de dichos resultados y la comparación con otros algoritmos del estado del arte.

Capítulo 5: Conclusiones, trabajo futuro y recomendaciones. En este capítulo se presentan las conclusiones obtenidas a partir de los resultados de los experimentos. Además el trabajo futuro en la línea de investigación y algunas recomendaciones.

Por último, se presentan enumeradas todas las referencias bibliográficas utilizadas en el desarrollo del presente trabajo de grado.

2 CONTEXTO TEÓRICO Y ESTADO DEL ARTE

2.1 Generación Automática de Resúmenes de Texto

El procesamiento de lenguaje natural (Natural Language Processing, NLP) [28] es la ciencia computacional encargada de estudiar las diferentes interacciones entre el lenguaje humano y las computadoras. La generación automática de resúmenes de textos es un área del NPL que busca resumir el contenido de uno o múltiples documentos sin perder la información más relevante en un texto relativamente corto o con un tamaño definido de palabras o letras. Como se define en [28], la generación automática de resúmenes de textos es una "breve pero exacta representación del contenido de un documento".

La taxonomía de un resumen clasifica el tipo del resumen que va a ser generado. Los resúmenes pueden generarse de acuerdo a varios parámetros como la forma, el público al que va dirigido, la cantidad de documentos a resumir, entre otros [28]. A continuación se presentan varios métodos de clasificación para la generación automática de resúmenes de documentos:

1) Con respecto a la forma como se genera el resumen puede ser extractivo o abstractivo. Los resúmenes extractivos se elaboran a partir del fraccionamiento del texto original y la selección de algunas fracciones (frases o segmentos de texto), esto hace que los resúmenes presenten baja coherencia, sin embargo, se consideran computacionalmente sencillos. Además han presentado resultados satisfactorios especialmente en generación de resúmenes de múltiples documentos [29]. Los resúmenes abstractivos contienen la esencia del texto original pero no necesariamente utilizando las mismas palabras, generando así la necesidad de utilizar herramientas de análisis lingüístico para la elaboración de las oraciones del resumen, "además es más difícil replicar o extender a otros dominios" [2].

2) De acuerdo al público al que va dirigido, los resúmenes se consideran: genéricos, basados en consultas, enfocados en el usuario o enfocados en tópicos. En los resúmenes genéricos no se tiene en cuenta un público específico sino que están destinados al público en general. Los basados en consultas generan el resumen de acuerdo a una consulta específica realizada por el usuario. Los enfocados en el usuario generan resúmenes dependiendo del interés de un usuario en particular y los resúmenes enfocados en tópicos enfatizan los resúmenes de acuerdo a temáticas específicas en los documentos [2].

3) "Dependiendo de la cantidad de documentos que son procesados para la generación del resumen, puede ser para un documento o múltiples documentos" [2].

4) "De acuerdo al idioma del documento, ellos pueden ser monolenguaje o multilenguaje" [2].

5) “Con respecto al género del texto original, pueden ser de tipo: artículos científicos, noticias, blogs, entre otros” [2].

Teniendo en cuenta estas características es importante destacar que para la realización este trabajo de grado se decidió enfocar los esfuerzos en la generación de resúmenes del tipo extractivo, genérico, de múltiples documentos, monolenguaje y enfocados en el género de noticias.

2.1.1 Proceso general de generación de resúmenes de múltiples documentos

2.1.1.1 Pre-procesamiento de los documentos

Los documentos que van a ser resumidos deben primero ser pre-procesados, utilizando técnicas lingüísticas como: eliminación de caracteres especiales y etiquetas, segmentación, transformación del texto a minúsculas, separación del texto en tokens (tokenización), eliminación de palabras vacías (stop words), encontrar una raíz léxica a los tokens (stemming), hallar el lema correspondiente de cada token (lematización) y búsqueda de sinónimos.

1) **Eliminación de caracteres especiales y etiquetas** (por ejemplo etiquetas HTML) [30]. La lectura de la colección de documentos tiene como entrada la lista completa de documentos, que en el caso específico del proyecto fueron archivos de noticias que vienen en formato XML, y como resultado entrega la lista completa de oraciones contenidas en la colección de documentos en un único archivo de texto plano separadas por saltos de línea.

Cada documento (de noticias) primero se debe leer en formato texto y se le remplazan los caracteres especiales que no son permitidos en archivos con formato XML, luego se valida que el documento cuente con la estructura adecuada, se lee y se extrae el texto contenido en la etiqueta <BODY>, finalmente, se almacena, se adiciona el contenido en formato texto plano en un único archivo separando con salto de línea el contenido de cada archivo. En el caso de que los documentos de entrada traigan otro formato, por ejemplo .pdf o .docx, este proceso debe adaptarse usando el conversor de texto adecuado.

2) **Segmentación**, “consiste en fraccionar el texto en unidades significativas (normalmente oraciones)” [30]. Para realizar la **segmentación** de las oraciones, el archivo que contiene el texto plano con todo el texto de las múltiples noticias se pasa por splitta (herramienta de código abierto disponible en https://code.google.com/p/splitta) [31], el cual separa oración por oración del archivo y las entrega en un archivo de salida en texto plano separando las oraciones con saltos de línea. Splitta es un segmentador de oraciones en inglés que usa un enfoque de aprendizaje supervisado soportado en Máquinas de Soporte Vectorial (Support Vector Machines, SVM) que reporta una taza de error aproximada del 0.35%, razón por la cual fue seleccionada.

3) **Conversión a minúsculas** [30]. En este proyecto, ésta y las siguientes tareas de pre-procesamiento se realizaron con una herramienta de código libre que está bajo la licencia de Apache Software llamada Lucene versión 2.9.2.2. Es una tecnología para la

recuperación de información que realiza procesos de indexación y búsqueda; está implementada en Java, C#, Ruby, PHP, y otros lenguajes de programación [32].

4) **Tokenización**. Consiste en dividir en partes una secuencia de caracteres (oración o frase) y a la vez desechar signos de puntuación. Estas partes son conocidas como tokens y se puede referir a ellos como términos o palabras. Por ejemplo si se tiene la siguiente oración: “Friends, Romans,Countrymen, lend me your ears;” [30], el resultado de la tokenización es la siguiente:

Friends	Romans	Countrymen	Lend	me	Your	Ears

5) **Eliminación de palabras vacías**, son aquellas palabras con bajo contenido semántico, como preposiciones, artículos, pronombres, etc [30]. En las oraciones hay palabras comunes de poco valor en el procesamiento de lenguaje natural y en específico en el proceso de generación automática de resúmenes, las cuales se conocen como palabras vacías (stop words) y debido a su poco aporte al resumen son eliminadas. Al eliminar estos términos, se evita la inclusión de “ruido” en el proceso de generación de los resúmenes. En el idioma ingles (idioma utilizado en los datasets véase sección 2.1.1.3) la lista de palabras vacías es extensa e incluye palabras como: i, a, are, is , the, of, in [30].

6) **Stemming**. Por razones gramaticales en los documentos se encuentran diferentes formas para una palabra, como, organize, organizes y organizing, Además hay palabras derivadas relacionadas con significados similares como democracy, democratic y democratization. El objetivo del stemming es reducir las formas flexivas y las formas derivadas relacionadas de una palabra a la forma base común [30]. Por ejemplo:

- car, cars, car’s => car
- am, are, is => be
- the boy’s cars are different colors => the boy car be differ color

El Stemming es un proceso heurístico que corta el final de las palabras (sufijos) y con frecuencia incluye la eliminación de los afijos derivados. El resultado del proceso de stemming no siempre es una palabra valida en el idioma que se está trabajando, por ejemplo, acknowledg para la palabra acknowledge. La herramienta Lucene versión 2.9.2.2 [32] también contiene las funciones de aplicar stemming a los tokens de los documentos.

7) **Lematización**, consiste en determinar variantes morfológicas de una palabra y reemplazarla por la palabra raíz. Hace uso correcto del análisis del vocabulario y la morfología de las palabras, normalmente con el objetivo de eliminar las terminaciones flexivas y devolver la forma base. Teniendo el token del verbo saw, el stemming devolvería solo s, en cambio la lemantización retorna see.

8) **Sinónimos.** Son palabras o expresiones que tienen significados iguales o muy parecidos, y que pertenecen a la misma categoría gramatical, por lo tanto, se pueden

substituir o intercambiar en un texto sin que esta sufra modificación en su sentido. Se utiliza para reducir el tamaño del vocabulario.

- Automobiles => Automobile (aplicación de lematización)
- Automobile => Car (aplicación de sinónimos)

WordNet [33] es una base de datos léxica del Inglés ampliamente usada por la comunidad de NLP. Nombres, verbos, adjetivos y adverbios se agrupan en conjuntos de sinónimos cognitivos (synsets). Los synsets están vinculados entre sí por medio de las relaciones conceptuales semánticas y léxicas. WordNet también es libre y públicamente disponible para descarga. La estructura de WordNet hace que sea una herramienta útil para la lingüística computacional y procesamiento del lenguaje natural como lo es la lematización y la búsqueda de sinónimos. Para aprovechar los beneficios de WordNet existe la herramienta llamada NLTK 3.0 (Natural Language Toolkit) [34]. Es una herramienta de Python para trabajar con los datos del lenguaje humano. Proporciona interfaces fáciles de usar y recursos léxicos como WordNet, junto con un conjunto de bibliotecas de procesamiento de texto que permite realizar lematización y búsqueda de sinónimos.

Además en esta etapa de pre-procesamiento se determina la representación de los documentos, que tradicionalmente se basa en el modelo espacio vectorial [2].

2.1.1.2 Métodos de generación automática de resúmenes extractivos para múltiples documentos

Entre los principales métodos de generación automática de resúmenes extractivos para múltiples documentos se encuentran los siguientes [4, 8, 14, 35]:

1) **Métodos Basados en Conectividad de Textos [6]:** En [36] proponen la generación de resúmenes de múltiples documentos mediante el uso de cadenas léxicas. Este método consiste en la generación del resumen de cada documento para luego manejar la redundancia de tal forma que se eliminen oraciones con gran similitud, generando así un resumen de los resúmenes de cada documento.
2) **Métodos Basados en Grafos [6, 8]:** Este método utiliza la teoría de grafos para la extracción de resúmenes donde cada porción de texto (oraciones extraídas de los documentos) es representada como un nodo del grafo y la similitud entre las oraciones representan las aristas, para el caso de múltiples documentos, el grafo es formado por todas las porciones de texto de todos los documentos. Entre los métodos basados en grafos se encuentra el algoritmo LexRank propuesto por Erkan y Radev en [19], destacado por sus buenos resultados en la generación automática de resúmenes [19].
3) **Métodos Basados Reducción Algebraica [6, 7]:** El método utiliza una técnica de generación automática de resúmenes basado en LSA (Latent Semantic Analysis), el cual permite comparar los significados de las palabras incluidas en el texto mediante un análisis algebraico-estadístico. El primer paso es crear la matriz de términos que contiene todas las oraciones contenidas en la colección de documentos, a cada oración se le asigna un puntaje para posteriormente ser seleccionadas las mejores oraciones. Sin embargo, antes de añadir una oración al resumen, se realiza un

proceso de selección de la oración en caso de redundancia. En [37] se propone un algoritmo que utiliza LSA, MMR (Maximal Marginal Relevance) y basado en consulta para que la oración a ser incluida en el resumen dependa de la consulta del usuario.

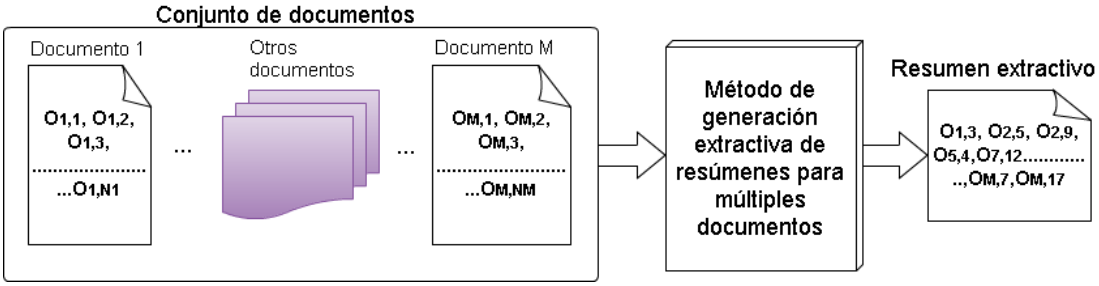
4) **Métodos Basados en Agrupamiento [4]:** Los métodos actuales de agrupamiento se suelen representar los documentos como una matriz términos por documento y aplican los algoritmos de agrupación en ésta. En 2004 [9, 10], se presenta un generador de resúmenes llamado MEAD, el cual produce resúmenes utilizando centroides de grupo producidos por un sistema de detección y seguimiento de tópicos. Luego en 2008 [11], se propone un modelo llamado lenguaje de factorización que simultáneamente hace agrupamiento y generación de resúmenes, por medio de la matriz de términos por documento y la matriz de términos por oración. Después en 2009 [38], se propone un algoritmo de agrupamiento orientado a consulta, donde la consulta se involucra en el conjunto de documentos, los grupos se mezclan en un solo grupo y se utiliza una modificación de la Relevancia Marginal Máxima (MMR) para seleccionar las oraciones pertenecientes al resumen. También en este año [39], se proponen dos técnicas, la primera consiste en la adición de la característica de similitud con la primer oración en el método MEAD llamada CPSL, y la segunda es una combinación de CPSL y LEAD (seleccionar primera y última oración del párrafo seleccionado) llamada LESM.
5) **Métodos Basados Modelos Probabilísticos [7]:** En 2009 [12], se presenta un método orientado a consulta basado en análisis semántico latente probabilístico en el cual utilizan un mecanismo de representación de oraciones y consultas como distribuciones de probabilidad sobre tópicos latentes. También en este mismo año [13], se presenta un modelo probabilístico generativo de tópicos bayesiano basado en oraciones (Bayesian Sentence-based Topic Models), usando la matriz de términos por documento y la matriz de términos por oración. Finalmente, en 2011 [40], se presenta un método probabilístico que agrupa y ordena las oraciones paralelamente, diferenciándose de otros métodos que lo hacen de forma secuencial.
6) **Modelos Evolutivos [4]:** En el 2011 [21], presentan un modelo basado en máxima cobertura y mínima redundancia (MCMR), abordando la generación de resúmenes como un problema de programación lineal entera. Para generar un buen resumen se deben optimizar tres propiedades: Relevancia, Redundancia y Longitud. También en este año, proponen un algoritmo basado en evolución diferencial cuyos parámetros para el cruce y la mutación son adaptativos, y su función objetivo es la división entre cobertura y redundancia [41]. En 2012 se presenta el modelo orientado a restricción, con dos enfoques: orientado cobertura y orientado a la diversidad; el problema se propone como programación entera cuadrática y se resuelve con PSO [42]. En este mismo año mediante la aplicación de evolución diferencial y la utilización del problema de la p-mediana se crea un método para generar resúmenes de múltiples documentos que tiene en cuenta la relevancia, cobertura del contenido, diversidad y longitud del resumen [43].
7) **Combinaciones de Métodos o Métodos Híbridos:** En el año 2010 [18], presentan un modelo híbrido compuesto de dos etapas: un modelo generativo para descubrir patrones y un modelo de regresión para la inferencia, de esta forma evitan la

necesidad de construir un modelo generativo para los nuevos grupos de documentos. Además en este año, se presenta un método de generación automática de resúmenes para múltiples documentos que utiliza técnicas de agrupamiento y un algoritmo de optimización de enjambres de partículas discretas (PSO) combinando de esta forma modelos evolutivos con agrupación [44].

8) **Otros Métodos**: En el 2004 [9], presentan un método basado en la entropía como factor para ordenar las oraciones de acuerdo a la relevancia, mediante el conocimiento específico de determinado tópico. Luego en el 2008 [45], presentan un método de generación automática de resúmenes de múltiples documentos basado en el descubrimiento de la característica de la palabra, si la palabra está contenida en varios párrafos de un documento, es una palabra de evento y si está involucrada en todos los documentos, es una palabra de tópico. Después en el 2010 [17], presentan un método con un enfoque de abajo hacia arriba para ordenar las oraciones de los documentos. El algoritmo propuesto iterativamente concatena segmentos de texto hasta que un segmento es ordenado con todas las oraciones. Para poder definir el orden y la asociación de dos segmentos de texto definen cuatro criterios: Cronología, Cercanía de Temas, Precedencia y Sucesión.

A continuación, en la **Figura 1** se presenta un esquema general para la generación automática de resúmenes extractivos de múltiples documentos. Siendo $O_{i,j}$ la j-ésima oración del documento *i*. Donde $i = 1,2,\ldots M$ y $j = 1,2,\ldots N$ (*M* es el número de documentos y *N* es el número de oraciones que varía para cada documento).

Figura 1. Generación de resúmenes extractivos para múltiples documentos (Adaptada de [2])



2.1.1.3 Evaluación de la calidad del resumen

En relación con la evaluación de la generación automática de resúmenes, existen dos tipos de evaluación importantes [46]:

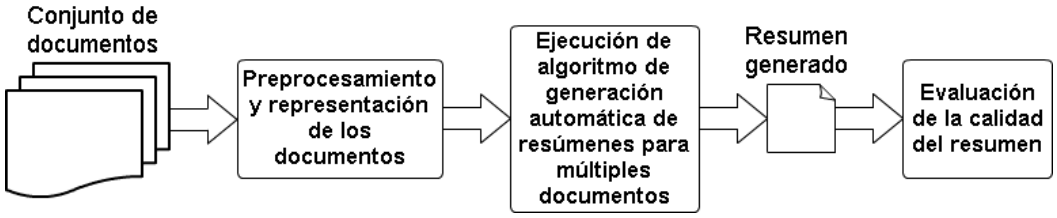
1) **La evaluación intrínseca** que mide la calidad del resumen mediante un punto de referencia que puede ser un conjunto de datos existente o un resumen "ideal" creado por un humano. El problema con ese tipo de evaluación es la variabilidad que puede presentar el ser humano.
2) **La evaluación extrínseca** que parte de una tarea donde se puede hacer uso de la generación automática de resúmenes y medir el efecto de reemplazar el texto original por resúmenes automáticos. Los problemas que surgen para este tipo de evaluación son la selección de la tarea y los indicadores para la medición del efecto.

En el 2004 [47], se presentó una herramienta para la evaluación intrínseca de los resúmenes generados en forma automática llamada ROUGE (Recall-Oriented Understudy for Gisting Evaluation). Esta herramienta incluye medidas para determinar automáticamente la calidad de un resumen generado comparándolo con un resumen ideal, midiendo la similitud entre resúmenes por medio de N-gramas. Entre las principales medidas están: ROUGE-1, ROUGE-2 y ROUGE-SU4.

En la Conferencia de Entendimiento del Documento (Document Understanding Conference, DUC) se han realizado evaluaciones de diferentes métodos de generación automática de resúmenes desde el año 2001. Para hacer esto definen un conjunto de documentos a resumir junto con los resúmenes "ideales" creados por varios humanos expertos (básicamente se puede definir como una colección cerrada de prueba). Para evaluar la calidad de los resúmenes generados utilizan los juicios humanos por medio de medidas ROUGE. Para los años 2002 y 2003, los resúmenes de un solo documento son de 100 palabras, para resúmenes de un solo documento muy cortos son 10 palabras y para resúmenes de múltiples documentos se utilizan diferentes tamaños (10, 50, 100, 200, 400 palabras). Como resultado encontraron que ROUGE-1, ROUGE-2, ROUGE-SU4, entre otros, funcionaron razonablemente bien para múltiples documentos [47].

La **Figura 2** muestra el proceso general de elaboración y evaluación de resúmenes para múltiples documentos (explicado anteriormente).

Figura 2. Proceso general de generación de resúmenes multi-documento (Adaptada de [2])



2.2 Word2Vec

En muchos sistemas actuales de Procesamiento Natural del Lenguaje y técnicas de tratamiento de palabras como unidades atómicas, no existe la noción de similitud entre estas, ya que son representadas como índices en un vocabulario. Con el progreso de las técnicas de aprendizaje automático en los últimos años, se ha hecho posible entrenar modelos más complejos basado en un conjunto amplio de datos, y por lo general estos nuevos modelos superan a los modelos simples. Word2Vec propone técnicas que se pueden utilizar para el entrenamiento de vectores de palabras de alta calidad a partir de grandes conjuntos de datos con miles de millones de palabras, y con millones de palabras en el vocabulario [22].

Word2Vec es un grupo de modelos relacionados que se utilizan para producir representaciones de palabras en un modelo espacio vectorial. Estos modelos son de poca profundidad, generalmente usan redes neuronales con una única capa oculta que está

capacitada para reconstruir los contextos lingüísticos de las palabras. Word2Vec toma como entrada un gran corpus de texto y produce un modelo de espacio vectorial, típicamente de varios cientos de dimensiones, en donde cada palabra única en el corpus queda relacionada con un vector correspondiente en dicho espacio. Los vectores de palabras están colocados en el espacio vectorial de modo que las palabras que comparten contextos comunes en el corpus se encuentran en estrecha proximidad entre sí en el espacio [22]. Word2Vec utiliza el gradiente descendiente y la propagación hacia atrás (backpropagation) y puede utilizar cualquiera de los siguientes dos modelos de arquitectura para producir una representación distribuida de las palabras: Continuous Bag-of-Words Model o Continuous Skip-gram Model. Esta sección del documento contiene una traducción y adaptación de los artículos [48] y [49], además se agregaron algunos detalles.

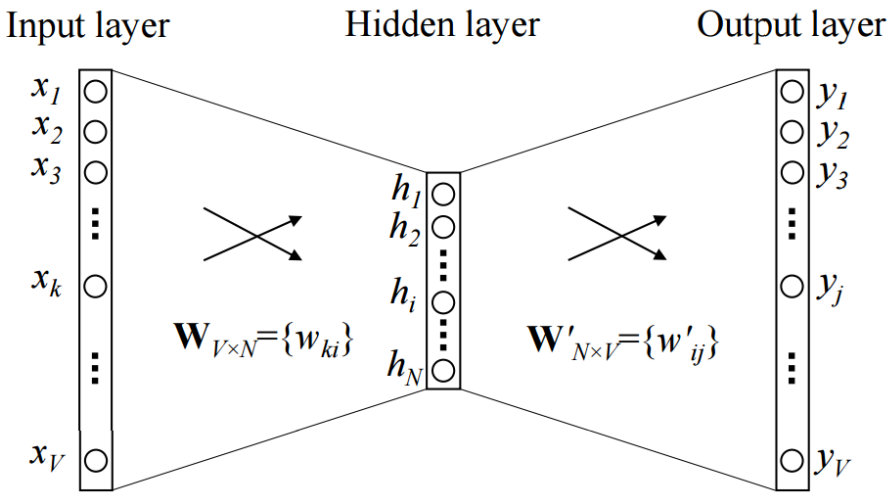
2.2.1 Continuous Bag-of-Words Model (CBOW)

2.2.1.1 Una palabra de contexto (One-word context)

Es la versión más simple del modelo continuo de bolsa de palabras (CBOW) introducido en [22]. Bajo la suposición que sólo hay una palabra considerada por el contexto, el modelo permite predecir una palabra objetivo dado un contexto de palabras.

La **Figura 3** muestra el modelo de red bajo la definición del contexto simplificado. Se considera que el tamaño del vocabulario es V y el tamaño de la capa oculta es N. Las unidades de las capas adyacentes están totalmente conectadas. La entrada es un vector codificado en caliente (one-hot encoded vector, en inglés), lo que significa que, para una palabra de contexto de entrada dada, sólo una de las unidades de V, $\{x_1, \ldots, x_V\}$, será 1 y todas las demás unidades son 0.

Figura 3 Un modelo CBOW simple con una sola palabra en el contexto (Tomada de [49])



Las ponderaciones entre la capa de entrada y la capa de salida son representadas por una matriz **W** de tamaño V × N. Cada fila de **W** corresponde a la representación del vector N-dimencional v_w de la palabra asociada de la capa de entrada. Formalmente, la fila i de

W es v_w^T. Dado un contexto (una palabra), asumiendo $x_k = 1$ y $x_{k'} = 0$ para $k' \neq k$ se tiene que:

$$h = W^T x = W_{(k,.)}^T := v_{w_i}{}^T \qquad (2\text{-}1)$$

Que esencialmente copia la k-ésima fila de **W** a h. v_{Wi} es la representación vectorial de la palabra de entrada w_i. Esto implica que la función de enlace (activación) de las unidades de la capa oculta es simplemente lineal (es decir, la suma ponderada de las entradas es pasada directamente a la siguiente capa). A continuación se presenta la ecuación (2-1) con mayor detalle.

Se tiene

$$W_{V\times N} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{V1} & w_{V2} & \cdots & w_{VN} \end{bmatrix} \qquad (2\text{-}1.1)$$

Y

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_V \end{bmatrix} \qquad (2\text{-}1.2)$$

Entonces

$$h = x^T W = [x_1 \quad x_2 \quad \ldots \quad x_k \quad \ldots \quad x_V] \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k1} & w_{k2} & \cdots & w_{kN} \\ \vdots & \vdots & \ddots & \vdots \\ w_{V1} & w_{V2} & \cdots & w_{VN} \end{bmatrix} \qquad (2\text{-}1.3)$$

$$= [x_k w_{k1} \quad x_k w_{k2} \quad \cdots \quad x_k w_{kN}] \text{ \# para } x_k = 1 \text{ y } x_{k'} = 0 \text{ para } k' \neq k \qquad (2\text{-}1.4)$$

$$= [w_{k1} \quad w_{k2} \quad \cdots \quad w_{kN}] \text{ \# K-ésima fila de W} \qquad (2\text{-}1.5)$$

Por lo tanto

$$= W_{(k,.)}^T := v_{w_i}{}^T \qquad (2\text{-}1.6)$$

De la capa oculta a la capa de salida, hay una matriz de peso diferente $W' = \{w'_{ij}\}$, de tamaño N × V. Usando estos pesos, se puede calcular el valor de u_j para cada palabra en el vocabulario.

$$u_j = {v'_{wj}}^T h \tag{2-2}$$

Donde v'_{wj} es la j-ésima columna de la matriz W'. Entonces se puede usar softmax (sección 2.2.2.1 de este documento), un modelo de clasificación log-lineal, para obtener la distribución posterior de las palabras, que es una distribución multinomial [49].

$$p(w_j|w_i) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})} \tag{2-3}$$

Donde y_j es la salida de la j-ésima unidad en la capa de salida. Sustituyendo (2-1) y (2-2) en (2-3), obtenemos:

$$p(w_j|w_i) = y_j = \frac{\exp({v'_{wj}}^T v_{wi})}{\sum_{j'=1}^{V} \exp({v'_{wj'}}^T v_{wi})} \tag{2-4}$$

Se debe tener en cuenta que v_w y v'_w son dos representaciones de la palabra w. v_w viene de las filas de W, y es llamado vector de entrada (input vector) de la palabra w, y v'_w viene de las columnas de W', y usualmente se le llama vector de salida (output vector) de la palabra w [49].

Actualización de pesos: de la capa oculta a la capa de salida

El objetivo del entrenamiento (para una muestra de entrenamiento) es maximizar la Ecuación (2-4), la probabilidad condicional de observar la palabra de salida real w_O (denotar su índice en la capa de salida como j^*) dada la palabra de contexto de entrada w_I (y con respecto a los pesos). Es decir:

$$\max p(w_O|w_I) = \max y_{j^*} \tag{2-5}$$

$$= \max \log y_{j^*} \tag{2-6}$$

$$= u_{j^*} - \log \sum_{j'=1}^{V} \exp(u_{j'}) := -E \tag{2-7}$$

Donde $E = -\log p(w_O|w_I)$ es la función de pérdida (lo que se quiere minimizar), y j^* es el índice de la palabra de salida real (en la capa de salida) [49].

El siguiente paso es derivar la ecuación de actualización de los pesos entre capas ocultas y de salida. Se toma la derivada de E con respecto a la j-ésima unidad de la capa de entrada u_j, se obtiene:

$$\frac{\partial \mathrm{E}}{\partial u_j} = \mathrm{y}_j - \mathrm{t}_j := \mathrm{e}_j \tag{2-8}$$

Donde $t_j = 1(j = j^*)$, es la función indicatriz, es decir, t_j será 1 cuando la j-ésima unidad es la palabra de salida real, de lo contrario $t_j = 0$. En este caso, esta derivada coincide con el error de predicción e_j de la capa de salida [49].

El siguiente paso es tomar la derivada en w'_{ij} para obtener el gradiente en los pesos de la capa oculta a la de salida que (por la regla de la cadena) es:

$$\frac{\partial \mathrm{E}}{\partial \mathrm{w}'_{ij}} = \frac{\partial \mathrm{E}}{\partial \mathrm{u}_j} \cdot \frac{\partial u_j}{\partial \mathrm{w}'_{ij}} = \mathrm{e}_j \cdot \mathrm{h}_i \qquad (2\text{-}9)$$

Ya que

$$\frac{\partial \mathrm{E}}{\partial u_j} = \frac{\partial (u_{j^*} - \log \sum_{j'=1}^{V} \exp(u_{j'}))}{\partial u_{j^*}} \qquad (2\text{-}10)$$

$$= \frac{\partial (\log \sum_{j'=1}^{V} \exp(u_{j'}))}{\partial u_j} - \frac{\partial u_{j^*}}{\partial u_j} \qquad (2\text{-}11)$$

$$= \frac{\exp(u_j)}{\log \sum_{j'=1}^{V} \exp(u_{j'})} - t_j \qquad (2\text{-}12)$$

Por la ecuación (2-3) se obtiene

$$= \mathrm{y}_j - t_j \qquad (2\text{-}13)$$

Ahora, usando el descenso de gradiente estocástico, se obtiene la ecuación de actualización de peso para los pesos de la capa oculta a la de salida:

$$w_{ij}'^{(new)} = w_{ij}'^{(old)} - \eta \cdot e_j \cdot h_i \qquad (2\text{-}14)$$

O

$$v_{wj}'^{(new)} = v_{wj}'^{(old)} - \eta \cdot e_j \cdot \mathrm{h} \qquad (2\text{-}15)$$

Donde $\eta > 0$ es la tasa de aprendizaje (estándar SGD), $\mathrm{e}_j = t_j - \mathrm{y}_j$ (Ecuación (2-8)), h_i es la i-ésima unidad en la capa oculta y v'_{wj} es el vector de salida para la palabra w_j [49].

Actualización de pesos: de la capa de entrada a la capa de oculta

Habiendo obtenido las ecuaciones de actualización para W', se puede pasar a W. Se toma la derivada de E en la salida de la capa oculta, obteniendo

$$\frac{\partial \mathrm{E}}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial \mathrm{E}}{\partial \mathrm{u}_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} \mathrm{e}_j \cdot w'_{ij} := EH_i \tag{2-16}$$

Aquí h_i es la salida de la i-ésima unidad en la capa oculta, u_j se define en la Ecuación (2-2) (la entrada de la unidad j-ésima a la capa de salida), y $\mathrm{e}_j = t_j - \mathrm{y}_j$ es el error de predicción de la j-ésima palabra en la capa de salida. EH es un vector N-dimensional, es la suma de los vectores de salida de todas las palabras del vocabulario, ponderadas por su predicción de error e_j.

El siguiente trabajo es tomar la derivada de E con respecto a W. Recordando primero que la capa oculta realiza un cálculo lineal sobre los valores de la capa de entrada, específicamente

$$h_i = \sum_{k=1}^{V} x_k \cdot w_{ki} \tag{2-17}$$

Ahora tomando la derivada de E con respecto a cada elemento de W, se obtiene

$$\frac{\partial \mathrm{E}}{\partial w_{ki}} = \frac{\partial \mathrm{E}}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = EH_i \cdot x_k \tag{2-18}$$

Esto es equivalente al producto tensorial[1] de x y EH, es decir,

$$\frac{\partial \mathrm{E}}{\partial \mathrm{W}} = x \otimes \mathrm{EH} = \mathrm{xE}H^T \tag{2-19}$$

A partir de la cual se obtiene una matriz V x N. Dado que sólo un componente de x es diferente de cero, sólo una fila de $\frac{\partial \mathrm{E}}{\partial \mathrm{W}}$ no es cero, y el valor de esa fila es $\mathrm{E}H^T$, un vector N-dimencional. Se logra la ecuación de actualización de W como:

$$v_{wI}^{(new)} = v_{wI}^{(old)} - \eta \mathrm{E}H^T \tag{2-20}$$

Donde v_{wI} es una fila de W, el "vector de entrada" de la única palabra de contexto, y es la única fila de W cuya derivada es distinta de cero. Todas las demás filas de W permanecerán sin cambios después de esta iteración, porque sus derivadas son cero [49].

Intuitivamente, dado que el vector EH es la suma de vectores de salida de todas las palabras en vocabulario ponderadas por su error de predicción $\mathrm{e}_j = \mathrm{y}_j - t_j$, se puede entender (2-20) como la adición de una porción de cada vector del vocabulario de salida en el vector de entrada de la palabra de contexto. Si en la capa de salida se sobreestima la probabilidad de que una palabra w_j sea la palabra de salida ($\mathrm{y}_j > t_j$), entonces el vector

[1] "Un caso representativo de producto tensorial es el producto de Kronecker de dos matrices, por ejemplo:" [50] c. d. Wikipedia. (2017, Producto tensorial. Available: https://es.wikipedia.org/w/index.php?title=Producto_tensorial&oldid=94254938

$$\begin{bmatrix} a1 \\ a2 \\ a3 \end{bmatrix} \otimes \begin{bmatrix} b1 & b2 & b3 & b4 \end{bmatrix} = \begin{bmatrix} a1b1 & a1b2 & a1b3 & a1b4 \\ a2b1 & a2b2 & a2b3 & a2b4 \\ a3b1 & a3b2 & a2b3 & a2b4 \end{bmatrix}$$

de entrada de la palabra de contexto w_I tenderá a alejarse del vector de salida de w_j; inversamente, si la probabilidad de que w_j sea la palabra de salida sea subestimada ($y_j > t_j$), entonces el vector de entrada w_I tiende a acercarse al vector de salida de w_j; Si se predice con exactitud la probabilidad de w_j, entonces tendrá poco efecto sobre el movimiento del vector de entrada de w_I. El movimiento del vector de entrada de w_I está determinado por el error de predicción de todos los vectores en el vocabulario [49].

Ahora, para ilustrar de una mejor forma el proceso realizado en la generación de vectores de palabras, se presenta un corto ejemplo realizado con CBOW de una palabra de contexto. Para el ejemplo se tiene en cuenta el siguiente corpus de entrada:

“El perro vio al gato
El perro persiguió al gato
El gato subió al árbol”.

Lo primero que se hace es determinar el vocabulario. En este caso el vocabulario está compuesto por ocho palabras. A continuación, el vocabulario debe ser ordenado alfabéticamente y así referenciar cada palabra por su índice, quedando así: {“al”, “árbol”, “el”, “gato”, “perro”, “persiguió”, “subió”, “vio”}. Para este ejemplo, la red neuronal tendrá ocho neuronas de entrada y ocho neuronas de salida. Suponiendo que se desean desarrollar vectores de tres dimensiones para las palabras, se deben utilizar tres neuronas en la capa oculta. Esto significa que W será una matriz de 8 × 3 y W' una matriz 3 × 8. Antes empezar con el entrenamiento, se deben inicializar estas matrices con pequeños valores aleatorios como es habitual en la formación de redes neuronales. Para este ejemplo, supongamos W y W' ser inicializado con los siguientes valores:

$$W = \begin{bmatrix} -0.094491 & -0.443977 & 0.313917 \\ -0.490796 & -0.229903 & 0.065460 \\ 0.072921 & 0.172246 & -0.357751 \\ 0.104514 & -0.463000 & 0.079367 \\ -0.226080 & -0.154659 & -0.038422 \\ 0.406115 & -0.192794 & -0.441992 \\ 0.181755 & 0.088268 & 0.277574 \\ -0.055334 & 0.491792 & 0.263102 \end{bmatrix}$$

$$W' = \begin{bmatrix} 0.023074 & 0.479901 & 0.432148 & 0.375480 & -0.364732 & -0.119840 & 0.266070 & -0.351000 \\ -0.368008 & 0.424778 & -0.257104 & -0.148817 & 0.033922 & 0.353874 & -0.144942 & 0.130904 \\ 0.422434 & 0.364503 & 0.467865 & -0.020302 & -0.423890 & -0.438777 & 0.268529 & -0.446787 \end{bmatrix}$$

Para el ejemplo se desea entrenar la red con la relación entre las palabras "gato" y "subió". La palabra "gato" se conoce como la palabra de contexto y la palabra "subió" se conoce como la palabra objetivo. Para ello, la red debe mostrar una probabilidad alta para "subió" cuando "gato" se introduce en la red.

En este caso, el vector de entrada será $X = [0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]^T$. Sólo el cuarto componente del vector es 1. Esto se debe a que la palabra de entrada "gato", está en la cuarta posición del vocabulario. Teniendo en cuenta que la palabra objetivo es "subió", el vector objetivo sería $[0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0]^T$

Con el vector de entrada que representa a "gato", la de salida a las neuronas de la capa oculta se puede calcular mediante la ecuación (2-1) o la ecuación (2-1.3) así:

$$h = [0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]^T \begin{bmatrix} -0.094491 & -0.443977 & 0.313917 \\ -0.490796 & -0.229903 & 0.065460 \\ 0.072921 & 0.172246 & -0.357751 \\ 0.104514 & -0.463000 & 0.079367 \\ -0.226080 & -0.154659 & -0.038422 \\ 0.406115 & -0.192794 & -0.441992 \\ 0.181755 & 0.088268 & 0.277574 \\ -0.055334 & 0.491792 & 0.263102 \end{bmatrix}$$

$$h = [0.104514 \quad -0.463000 \quad 0.079367]$$

Hasta aquí se han realizado los cálculos respectivos entre la capa de entrada y la capa oculta. Ahora para la capa oculta a la de salida se realizan operaciones similares, el vector de activación de las neuronas de la capa de salida se puede escribir como se indica en la ecuación (2-2):

$$u_j = W'h$$
$$= [0.206326 \quad -0.117586 \quad 0.201337 \quad 0.106533 \quad -0.087468 \quad -0.211193 \quad 0.116228 \quad -0.132753]$$

En este punto se producen las probabilidades de obtener una palabra dada su palabra de contexto y dichas probabilidades están en la capa de salida, $p(w_j|w_{contexto})$ para j $= 1, \dots, V$, aquí es donde se debe aplicar alguno de los métodos de optimización de la eficiencia computacional (o algoritmos de entrenamiento, véase secciones 2.2.2 y 2.2.4). Para este ejemplo como es para una palabra de contexto, se utiliza la ecuación (2-3).

$$y = [0.150291 \quad 0.108707 \quad 0.149543 \quad 0.136017 \quad 0.112031 \quad 0.098993 \quad \mathbf{0.137342} \quad 0.107071]$$

La probabilidad en negrita corresponde a la palabra objetivo elegida "subió". Dado el vector objetivo $[0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0]^T$, el vector de error de la capa de salida se calcula restando el vector objetivo t con el vector de probabilidad y según la ecuación (2-8).

$$y_j - t_j := e_j = \mathbf{-0,862657}$$

Dado que:

$$[0.150291 \quad 0.108707 \quad 0.149543 \quad 0.136017 \quad 0.112031 \quad 0.098993 \quad \mathbf{0.137342} \quad 0.107071]$$
$$- [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \mathbf{1} \quad 0]$$

$$= [0.150291 \quad 0.108707 \quad 0.149543 \quad 0.136017 \quad 0.112031 \quad 0.098993 \quad \mathbf{-0,862657} \quad 0.107071]$$

Una vez que se conoce el error, los pesos en las matrices W y W' se pueden actualizar mediante propagación hacia atrás descrita en las ecuaciones (2-14), (2-15), (2-20). Para

el ejemplo se tiene en cuenta una tasa se aprendizaje $\eta = 0.025$. Por lo tanto la nueva matriz de pesos de la capa oculta a la capa de salida sería:

$$w'^{(new)}_{0,6} = w'^{(old)}_{0,6} - \eta \cdot e_6 \cdot h_0 = 0{,}266070 - 0{,}025 * -0{,}862657 * 0{,}104514 = \mathbf{0{,}268323994}$$

$$w'^{(new)}_{1,6} = w'^{(old)}_{1,6} - \eta \cdot e_6 \cdot h_1 = -0{,}144942 - 0{,}025 * -0{,}862657 * -0{,}463 = \mathbf{-0.15492726}$$

$$w'^{(new)}_{2,6} = w'^{(old)}_{2,6} - \eta \cdot e_6 \cdot h_2 = 0.268529 - 0{,}025 * -0{,}862657 * 0.079367 = \mathbf{0.270240663}$$

$$W' = \begin{bmatrix} 0.023074 & 0.479901 & 0.432148 & 0.375480 & -0.364732 & -0.119840 & \mathbf{0.268323994} & -0.351000 \\ -0.368008 & 0.424778 & -0.257104 & -0.148817 & 0.033922 & 0.353874 & \mathbf{-0.15492726} & 0.130904 \\ 0.422434 & 0.364503 & 0.467865 & -0.020302 & -0.423890 & -0.438777 & \mathbf{0.270240663} & -0.446787 \end{bmatrix}$$

Teniendo en cuenta que columnas de W', representan los vectores de salida (output vector), para esta iteración el vector que representa la palabra "subió" sería $vec(subió) = [0.268323 \quad -0.154927 \quad 0.270240]^{\mathrm{T}}$. Así sucesivamente hasta realizar todas las iteraciones necesarias para entrenar el corpus de entrada.

2.2.1.2 Varias palabras de contexto (Multi-word context)

La **Figura 4** muestra el modelo CBOW con una configuración de contexto de varias palabras. Cuando se calcula la salida de la capa oculta, en lugar de copiar directamente el vector de entrada de la palabra de contexto de entrada, el modelo CBOW toma el promedio de los vectores de las palabras de contexto de entrada y utiliza el producto de la matriz de pesos entre las capas de entrada y oculta y el vector promedio como la salida.

$$h = \frac{1}{C} W^T (x_1 + x_2 + \cdots + x_C) \qquad (2\text{-}21)$$

$$= \frac{1}{C} (v_{w1} + v_{w2} + \cdots + v_{wC})^T \qquad (2\text{-}22)$$

Donde C es el número de palabras en el contexto, $w_1, w_2, \ldots, w_C$ son las palabras en el contexto, y v_w es el vector de entrada de una palabra w [49]. La función de pérdida es:

$$\mathrm{E} = -\log \mathrm{p}(\mathrm{w}_O | \mathrm{w}_{I,1}, \ldots, \mathrm{w}_{I,C}) \qquad (2\text{-}23)$$

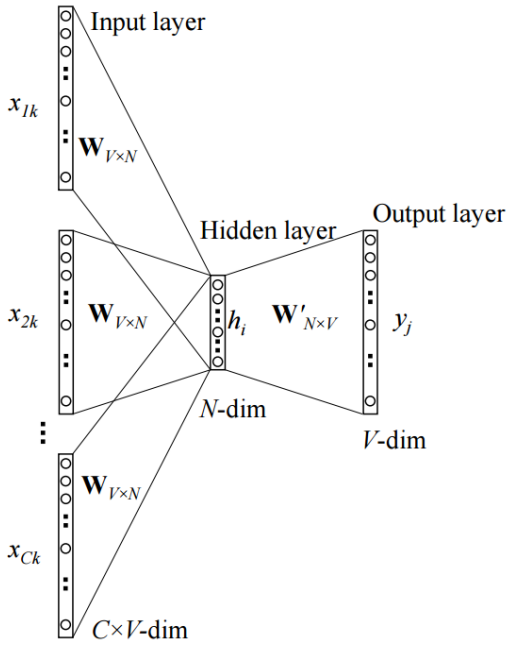
$$= u_{j^*} + \log \sum_{j'=1}^{V} \exp(u_{j'}) \qquad (2\text{-}24)$$

$$= -{v'_{wO}}^T \cdot \mathrm{h} + \log \sum_{j'=1}^{V} \exp({v'_{wj}}^T \cdot \mathrm{h}) \qquad (2\text{-}25)$$

Que es lo mismo que (2-7), el objetivo del modelo de una palabra de contexto, la diferencia es como se define en (2-22) en lugar de (2-1).

La ecuación de actualización para los pesos

Figura 4 Modelo continuo de CBOW con varias palabras de contexto. Tomado de [49].



de la capa de salida a la oculta permanece igual que para el modelo de contexto de una palabra (2-15). Se copia aquí:

$$v'^{(new)}_{wj} = v'^{(old)}_{wj} - \eta \cdot e_j \cdot \mathrm{h} \quad \text{para j} = 1,2,\dots,\mathrm{V}. \tag{2-26}$$

Tiendo en cuenta que se necesita aplicar esto a cada elemento de la matriz de peso de la capa oculta a la salida para cada instancia de entrenamiento, La ecuación de actualización para las ponderaciones de la capa de entrada a la oculta es similar a (2-20), excepto que ahora se necesita aplicar la siguiente ecuación para cada palabra $\mathrm{w}_{I,C}$ en el contexto:

$$v^{(new)}_{W_{I,c}} = v^{(old)}_{W_{I,c}} - \frac{1}{C} \cdot \eta \cdot \mathrm{E}H^T \; para \; c = 1,2,\dots,\mathrm{C}. \tag{2-27}$$

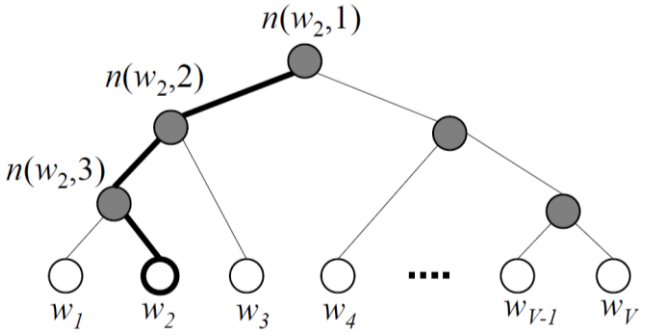
Donde $v_{w_{I,c}}$ es el vector de entrada de la c-ésima palabra en el contexto de entrada; η es la tasa de aprendizaje positiva; y $EH = \frac{\partial \mathrm{E}}{\partial h_i}$ está dada por (2-16). La comprensión intuitiva de esta ecuación de actualización es la misma que para (2-20) [49].

2.2.2 Optimización de la eficiencia computacional I

2.2.2.1 Softmax Jerárquico (Hierarchical Softmax)

Softmax jerárquico es una forma eficiente de calcular softmax introducido en [51] en 2005. El modelo utiliza un árbol binario llamado árbol de Huffman para representar todas las palabras del vocabulario. Las V palabras deben ser unidades de hojas del árbol (nodos) y existen V - 1 unidades internas. Para cada hoja existe un único camino desde la raíz hasta ella; y esta trayectoria se utiliza para estimar la probabilidad de la palabra representada por la unidad de la hoja. A continuación, en la **Figura 5** se muestra un árbol binario como ejemplo de softmax jerárquico. Donde las unidades blancas son palabras en el vocabulario, y las unidades oscuras son unidades internas. Se resalta una ruta de ejemplo de la raíz a w_2 donde la longitud de la trayectoria $L(w_2) = 4$ y n(w; j) es la j-ésima unidad en el camino desde la raíz a la palabra w.

Figura 5 Ejemplo de árbol de Huffman para softmax jerárquico. Tomado de [49].



En el modelo de softmax jerárquico, no hay representación vectorial de salida para las palabras. En su lugar, cada una de las unidades internas V - 1 tiene un vector de salida $v'_{n(w,j)}$. Y la probabilidad de que una palabra sea la palabra de salida se define como:

$$p(w = w_O) = \prod_{j=1}^{L(w)-1} \sigma([\![n(w,j+1) = ch(n(w,j))]\!] \cdot {v'_{n(w,j)}}^T h) \tag{2-28}$$

Donde $\sigma(x) = \frac{1}{1+e^{-x}}$; $L(w)$ es el tamaño de la ruta entre la raíz y la palabra; $ch(n(w,j))$ es el hijo izquierdo de la unidad n; $v'_{n(w,j)}$ es la representación vectorial (vector de salida) de la unidad interior $n(w,j)$; h es el valor de salida de la capa oculta, en el modelo Skip-gram $h = v_{wI}$; y en CBOW $h = \frac{1}{C}\sum_{C=1}^{C} V_{WC}$) y $[\![X]\!]$ es 1 si x es verdadera y -1 en caso contrario [49, 52].

Se define esta probabilidad como la probabilidad de un camino a partir de la raíz y termina en la unidad de hoja en cuestión. En cada unidad interna (incluyendo la unidad raíz), se deben asignar las probabilidades de ir a la izquierda y de ir a la derecha. Se define la probabilidad de ir a la izquierda en una unidad interna n como:

$$p(n, left) = \sigma({v'_{n(w,j)}}^T \cdot h) \quad (2\text{-}29)$$

Que se determina tanto por la representación vectorial de la unidad interna como por el valor de salida de la capa oculta (que luego se determina por la representación vectorial de la palabra o palabras de entrada). Y la probabilidad de ir a la derecha en la unidad n es:

$$p(n, right) = 1 - \sigma({v'_{n(w,j)}}^T \cdot h) = \sigma(-{v'_{n(w,j)}}^T \cdot h) \quad (2\text{-}30)$$

Siguiendo el camino de la raíz a w_2 en la **Figura 5**, se calcula la probabilidad de que w2 sea la palabra de salida, así:

$$\begin{aligned} p(w_2 = w_O) &= p(n(w_2, 1), left) \cdot p(n(w_2, 2), left) \cdot p(n(w_2, 3), right) \\ &= \sigma({v'_{n(w_2,1)}}^T \cdot h) \cdot \sigma({v'_{n(w_2,2)}}^T \cdot h) \cdot \sigma(-{v'_{n(w_2,3)}}^T \cdot h) \end{aligned} \quad (2\text{-}31)$$

Que es exactamente lo que se define en la ecuación (2-28). Además, se verifica que:

$$\sum_{i=1}^{V} p(w_i = w_O) = 1 \quad (2\text{-}32)$$

Convirtiendo a softmax jerárquica en una distribución multinomial bien definida entre todas las palabras.

Ahora se debe derivar la ecuación de actualización de parámetros para las representaciones vectoriales de las unidades internas. Para simplificar, primero se presenta para el modelo de contexto de una palabra y luego se facilita extender las ecuaciones de actualización a los modelos CBOW y Skip-gram.

Para simplificar la notación, primero se definen las siguientes abreviaciones sin introducir ambigüedad:

$$[\![\cdot]\!] = [\![n(w, j+1) = ch(n(w, j))]\!] \quad (2\text{-}33)$$

$$v'_j := v'_{n_{w,j}} \quad (2\text{-}34)$$

Para una instancia de entrenamiento, la función de error es definida así:

$$E = -\log\big(p(w = w_O|w_I)\big) = -\sum_{j=1}^{L(w)-1} \log(\sigma([\![\cdot]\!] {v'_j}^T h)) \quad (2\text{-}35)$$

Tomando la derivada de E con respecto de $v'_j h$ se obtiene:

$$\frac{\partial \mathrm{E}}{\partial v'_j h} = (\sigma\left([\![\cdot]\!] {v'_j}^T h\right) - 1)[\![\cdot]\!] \quad (2\text{-}36)$$

$$= \begin{cases} \sigma\left(v_j^{'T} h\right) - 1 & (\llbracket \cdot \rrbracket = 1) \\ \sigma\left(v_j^{'T} h\right) & (\llbracket \cdot \rrbracket = -1) \end{cases} \tag{2-37}$$

$$= \sigma\left(v_j^{'T} h\right) - t_j \tag{2-38}$$

Donde $t_j = 1$ si $\llbracket \cdot \rrbracket = 1$ y $t_j = 0$ en caso contrario [49].

A continuación, se toma la derivada de E con respecto a la representación vectorial de la unidad interna $n(w, j)$ y se obtiene:

$$\frac{\partial \mathrm{E}}{\partial v_j^{'}} = \frac{\partial \mathrm{E}}{\partial v_j^{'} h} \cdot \frac{\partial v_j^{'} h}{\partial v_j^{'}} = (\sigma\left(v_j^{'T} h\right) - t_j) \cdot \mathrm{h} \tag{2-39}$$

Que da como resultado la siguiente ecuación de actualización:

$$v_j^{'(new)} = v_j^{'(old)} - \eta(\sigma\left(v_j^{'T} h\right) - t_j) \cdot \mathrm{h} \tag{2-40}$$

Para $j = 1,2, \ldots L(w) - 1$. Se puede entender $\sigma\left(v_j^{'T} h\right) - t_j$ como el error de predicción para la unidad interna $n(w, j)$. La tarea para cada unidad interna es predecir si debe seguir al hijo izquierdo o al hijo derecho en la caminata aleatoria $t_j = 1$ significa que debe seguir al hijo izquierdo, $t_j = 0$ significa que debe seguir al hijo derecho.

Ahora, $\sigma\left(v_j^{'T} h\right)$ es el resultado de la predicción. Para una instancia de entrenamiento, si la predicción de la unidad interna está muy cerca de la verdad, entonces su representación vectorial v_j' se moverá muy poco; De lo contrario v_j' se moverá en una dirección apropiada (más cerca o más lejos de h) para reducir el error de predicción para esta instancia. Esta ecuación de actualización se puede utilizar tanto para CBOW como para el modelo Skip-gram. Cuando se utiliza para el modelo Skip-gram, se necesita repetir este procedimiento de actualización para cada una de las palabras C en el contexto de salida.

Ahora, σ(x) en softmax jerárquico está descrito en la ecuación (2-28) de la siguiente forma:

$$p(w = w_O) = \prod_{j=1}^{L(w)-1} \sigma(\llbracket n(w, j+1) = ch\big(n(w, j)\big) \rrbracket \cdot v_{n(w,j)}^{'T} h)$$

El código original de Word2Vec realiza el cálculo del gradiente a través de esta ecuación. Se puede denotar $({v'_{n(w,j)}}^T h)$ como "simi", entonces σ(simi) se puede volver a escribir como:

$$\sigma(simi) = \frac{e^{(1-código)*simi}}{1+e^{simi}} \qquad (2\text{-}41)$$

El valor de código es dado por el árbol de Huffman para indicar la ruta que se debe tomar para llegar a la palabra que se desea entrenar. Cuando el código = 1, $\sigma(simi) = \frac{1}{1+e^{simi}}$ y cuando el código = 1, $\sigma(simi) = \frac{e^{simi}}{1+e^{simi}}$

Usualmente se busca la derivada de $\sigma(simi)$ después de darle un registro. Por lo tanto,

$$ln\sigma(simi) = (1 - código) * simi - ln(1 + e^{simi}) \qquad (2\text{-}42)$$

La derivada respecto a "simi" es el gradiente:

$$g = (1 - código) - \frac{e^{simi}}{1+e^{simi}} = 1 - código - f \qquad (2\text{-}43)$$

Se debe tener en cuenta que el valor de f se encuentra en una tabla exponencial previamente definida.

Para dar una ilustración de lo anterior, a continuación se presenta un ejemplo del proceso realizado por Word2Vec para obtener vectores de palabras utilizando la arquitectura CBOW con varias palabras de contexto y Softmax Jerárquico tal como lo hace el código fuente de Word2Vec. Para ello se utiliza el mismo corpus de entrada del ejemplo con una palabra de contexto (véase sección 2.2.1.1) pero teniendo en cuenta que Word2Vec necesita la etiqueta “</s>” para delimitar el final de sentencia, el corpus de entrada se presenta así:

“El perro vio al gato </s>
El perro persiguió al gato </s>
El gato subió al árbol </s>”

El vocabulario se establece con nueve palabras (puesto que se debe agregar la etiqueta de fin de sentencia aunque no sea entrenada). La **Figura 6** Muestra el vocabulario organizado mediante un árbol de Huffman determinado mediante la frecuencia de cada palabra en el corpus. La matriz W se inicializa aleatoriamente y W' se inicializa en ceros. Y se establece una taza de aprendizaje de 0.025.

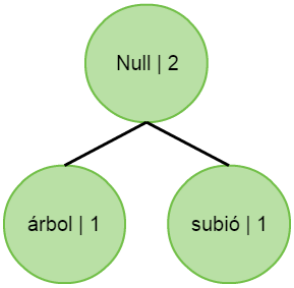
Para determinar el árbol de Huffman se realizan los siguientes pasos:

1) Primero se crea una lista con el vocabulario ordenado por la frecuencia de cada palabra (palabra | frecuencia) incluyendo la etiqueta de fin de sentencia que tiene una frecuencia 3 así:

Figura 6 Ejemplo de elaboración de árbol de Huffman 1.

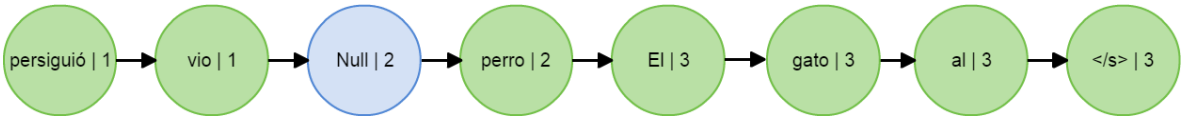
2) Se seleccionan los primeros dos nodos de la lista y se conforma un árbol con su nodo raíz con un nodo nulo que registra la suma de las frecuencias que contienen los dos primeros nodos tal como se indica a continuación:

Figura 7 Ejemplo de elaboración de árbol de Huffman 2.



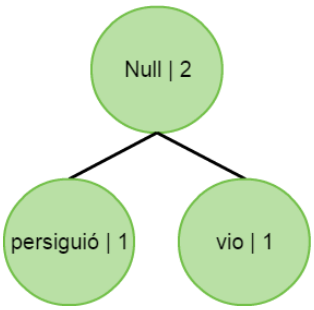
3) Luego, el nodo nulo se incluye en la lista ordenada de la siguiente forma:

Figura 8 Ejemplo de elaboración de árbol de Huffman 3.



Se repite el paso 2) (seleccionar los primeros dos nodos y formar el árbol) así:

Figura 9 Ejemplo de elaboración de árbol de Huffman 4.

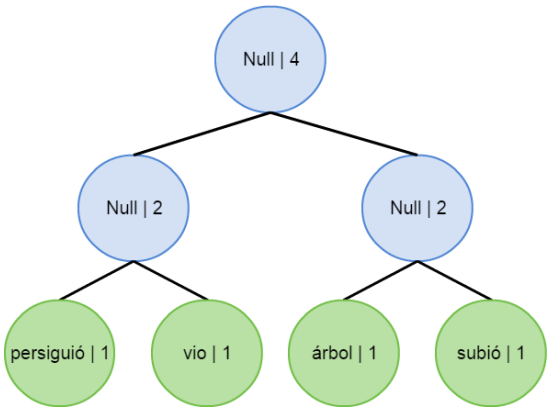


Se repite el paso 3) así:

Figura 10 Ejemplo de elaboración de árbol de Huffman 5.

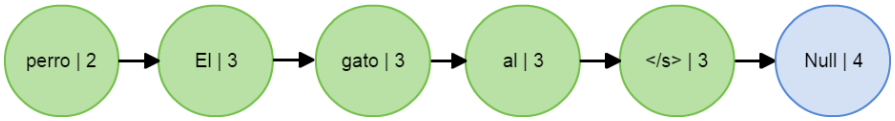
Se itera el paso 2) de la siguiente forma:

Figura 11 Ejemplo de elaboración de árbol de Huffman 6.



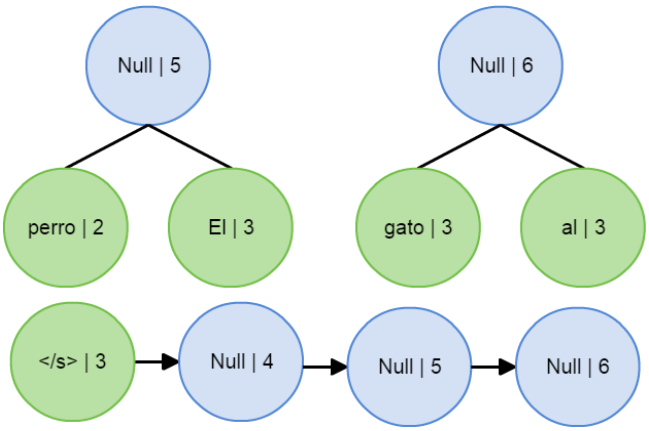
Se repite en el paso 3) como se muestra a continuación (teniendo en cuenta que se trata de una lista ordenada):

Figura 12 Ejemplo de elaboración de árbol de Huffman 7.



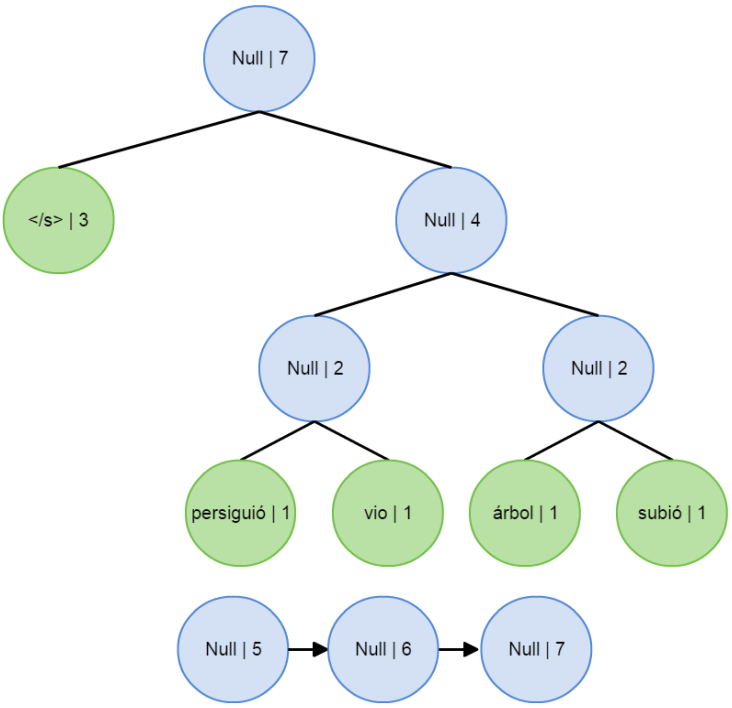
Se itera el paso 2) y el paso 3) hasta tener los siguientes árboles:

Figura 13 Ejemplo de elaboración de árbol de Huffman 8.



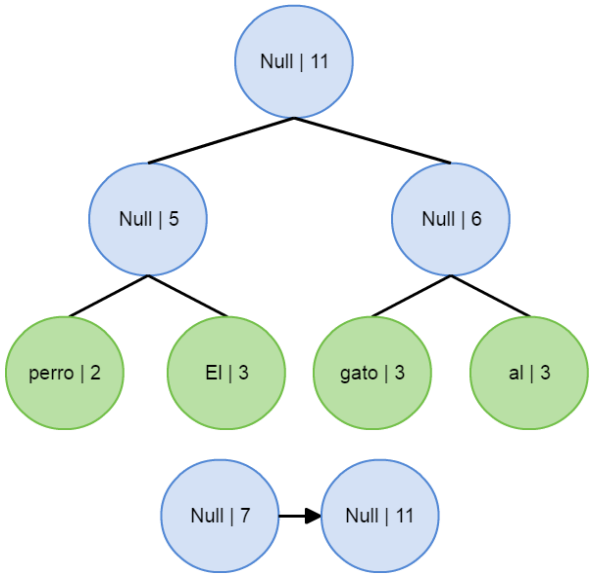
Se repite el paso 2) y 3) así:

Figura 14 Ejemplo de elaboración de árbol de Huffman 9.



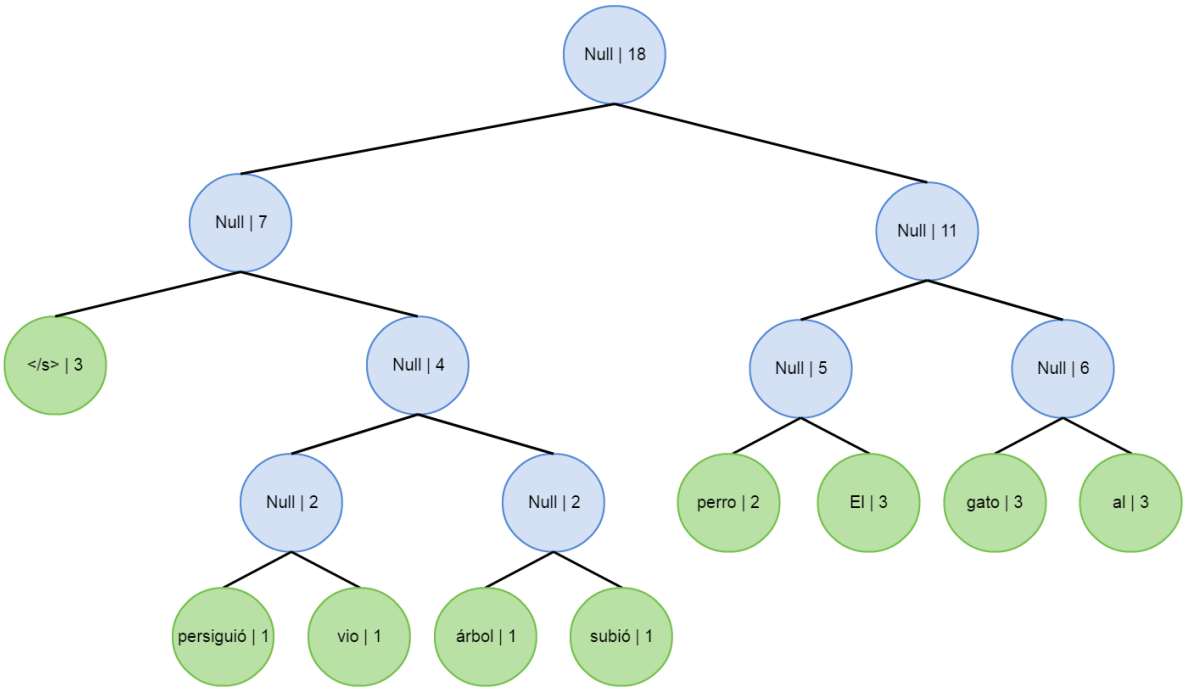
Nuevamente se itera en los pasos 2) y 3) así:

Figura 15 Ejemplo de elaboración de árbol de Huffman 10.



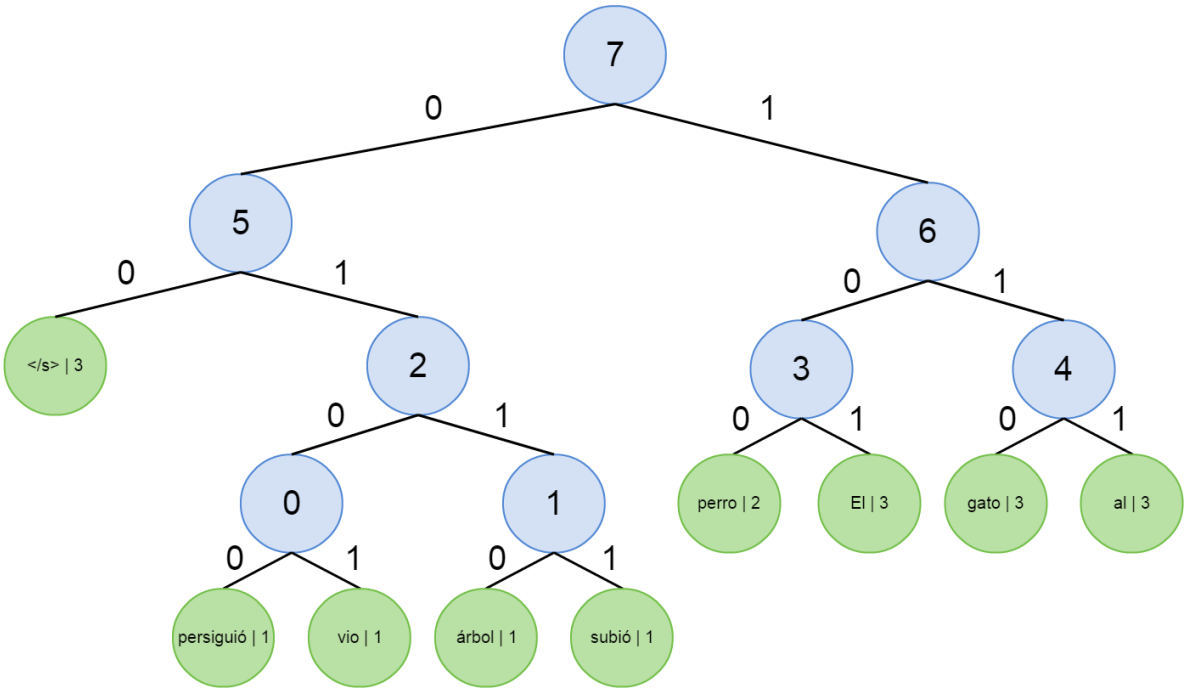
Luego se toman los dos últimos nodos de la lista ordenada y se realiza el paso 2) dando como resultado la estructura básica del árbol de Huffman, así:

Figura 16 Ejemplo de elaboración de árbol de Huffman 11.



Por último se agregan los caminos desde la raíz a cada hoja asignando el valor de 0 a la rama izquierda de cada nodo y el valor de 1 a la rama derecha de cada nodo y se enumeran los nodos nulos desde el nivel inferior hasta el nivel superior (raíz).

Figura 17 Árbol de Huffman. Ejemplo de Word2Vec con CBOW y Softmax jerárquico.



Para este ejemplo se va a entrenar la palabra “subió” con las palabras de contexto “El”, “gato”, “al” y “árbol”.

En este punto del entrenamiento se tiene la matriz W y W' definidas de la siguiente forma:

$$W = \begin{bmatrix} 0{,}1334229 & 0{,}1473134 & -0{,}1276754 \\ -0{,}10977 & 0{,}04556564 & 0{,}1009994 \\ 0{,}03143913 & 0{,}006683974 & -0{,}1204391 \\ 0{,}07385782 & -0{,}1456022 & 0{,}04212655 \\ -0{,}02523252 & -0{,}03199855 & -0{,}1108595 \\ -0{,}06288071 & 0{,}08583041 & 0{,}1009308 \\ 0{,}09889694 & 0{,}05329777 & 0{,}05160285 \\ -0{,}126473 & -0{,}1368285 & 0{,}1656012 \\ 0{,}1266404 & 0{,}1024776 & -0{,}01961908 \end{bmatrix} \begin{matrix} </s> \\ al \\ gato \\ El \\ perro \\ subió \\ árbol \\ vio \\ persiguió \end{matrix}$$

$$W' = \begin{bmatrix} 0{,}0007028893 & -0{,}000487299 & -0{,}0002296803 \\ -0{,}002291337 & 0{,}00121236 & 0{,}002448936 \\ -0{,}0002467792 & -0{,}002495237 & -0{,}003611555 \\ 0{,}004801504 & -0{,}004534638 & 0{,}00861265 \\ -0{,}005522867 & 0{,}002791227 & 0{,}009414396 \\ 0{,}005335759 & -0{,}0006692969 & -0{,}00157922 \\ 0{,}001916379 & -0{,}004865601 & -0{,}00617595 \\ 0{,}0001589228 & 0{,}006733424 & -0{,}003654487 \\ 0 & 0 & 0 \end{bmatrix}$$

Primero se calcula el vector h promediando los vectores de las palabras de contexto dando como resultado:

$$+ \begin{matrix} 0{,}07385782 & -0{,}1456022 & 0{,}04212655 & \boldsymbol{El} \\ 0{,}03143913 & 0{,}006683974 & -0{,}1204391 & \boldsymbol{gato} \\ -0{,}10977 & 0{,}04556564 & 0{,}1009994 & \boldsymbol{al} \\ 0{,}09889694 & 0{,}05329777 & 0{,}05160285 & \text{á}\boldsymbol{rbol} \end{matrix}$$

$$= 0{,}09442389 \quad -0{,}04005482 \quad 0{,}0742897$$

$$(0{,}09442389 \quad -0{,}04005482 \quad 0{,}0742897)/4$$

$$= 0{,}02360596 \quad -0{,}01001371 \quad 0{,}01857242$$

$$h = [0{,}02360596 \quad -0{,}01001371 \quad 0{,}01857242]$$

El gradiente se calcula de acuerdo a la ecuación (2-43) teniendo en cuenta que la ruta para la palabra “subió” en el árbol de Huffman es $[0 \quad 1 \quad 1 \quad 1]$ (códigos).

$$g = 1 - código_0 - f$$

$$g = 1 - 0 - 0{,}4910008$$

$$g = 0{,}5089992$$

El gradiente (multiplicado por la taza de aprendizaje) para el primer código es g = 0,01272498 y el error $[-1{,}800106\mathrm{E}-06 \quad 8{,}730412\mathrm{E}-05 \quad -4{,}95106\mathrm{E}-05]$, Luego se actualizan los pesos de la matriz W' (propagación de la capa interna a la de salida) tomando el código que corresponde al nodo raíz del árbol de Huffman (se parte de la raíz hasta el nodo de la palabra objetivo) que para este caso es 7.

$$w'^{(new)}_{0,7} = w'^{(old)}_{0,7} + \eta \cdot g \cdot h_0 = 0{,}0001589228 + 0{,}01272498 * 0{,}02360596 = \mathbf{0{,}0004593081}$$

$$w'^{(new)}_{1,7} = w'^{(old)}_{1,7} + \eta \cdot g \cdot h_1 = 0{,}006733424 + 0{,}01272498 * -0{,}01001371 = \mathbf{0.006605999}$$

$$w'^{(new)}_{2,7} = w'^{(old)}_{2,7} + \eta \cdot g \cdot h_2 = -0{,}003654487 + 0{,}01272498 * 0.01857242 = \mathbf{-0.003418153}$$

$$W' = \begin{bmatrix} 0{,}0007028893 & -0{,}000487299 & -0{,}0002296803 \\ -0{,}002291337 & 0{,}00121236 & 0{,}002448936 \\ -0{,}0002467792 & -0{,}002495237 & -0{,}003611555 \\ 0{,}004801504 & -0{,}004534638 & 0{,}00861265 \\ -0{,}005522867 & 0{,}002791227 & 0{,}009414396 \\ 0{,}005335759 & -0{,}0006692969 & -0{,}00157922 \\ 0{,}001916379 & -0{,}004865601 & -0{,}00617595 \\ \mathbf{0{,}0004593081} & \mathbf{0.006605999} & \mathbf{-0.003418153} \\ 0 & 0 & 0 \end{bmatrix}$$

Así sucesivamente hasta hacer la propagación de la matriz W' mediante cada código de la ruta del árbol de Huffman y actualizar el valor del error de propagación. Por último, se hace la propagación de la capa oculta a la de entrada sumando a cada fila el error de propagación para actualizar los valores de la matriz W teniendo:

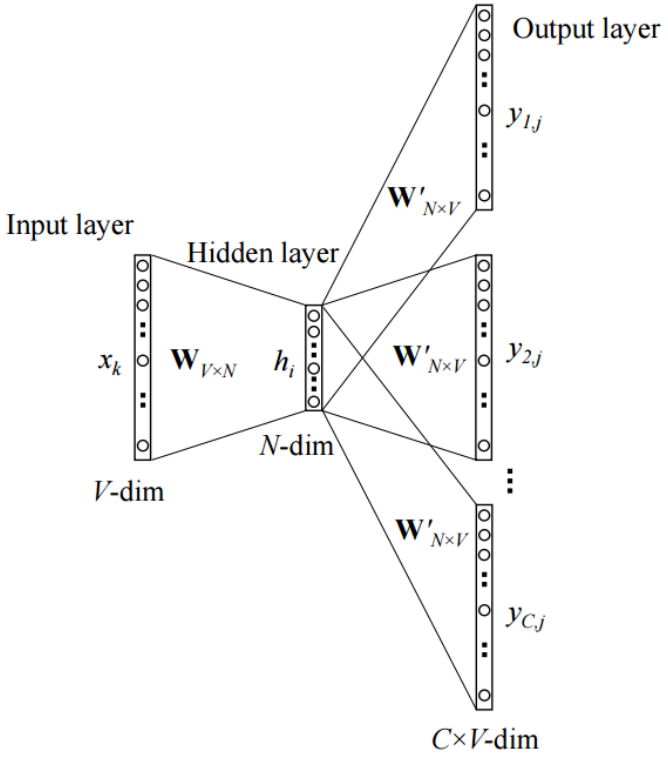
$$W = \begin{bmatrix} 0{,}1334229 & 0{,}1473134 & -0{,}1276754 \\ \mathbf{-0{,}1098066} & \mathbf{0{,}04567696} & \mathbf{0{,}1009836} \\ \mathbf{0{,}03140259} & \mathbf{0{,}006795291} & \mathbf{-0{,}1204548} \\ \mathbf{0{,}07382128} & \mathbf{-0{,}1454909} & \mathbf{0{,}04211082} \\ -0{,}02523252 & -0{,}03199855 & -0{,}1108595 \\ -0{,}06288071 & 0{,}08583041 & 0{,}1009308 \\ \mathbf{0{,}0988604} & \mathbf{0{,}05340909} & \mathbf{0{,}05158712} \\ -0{,}126473 & -0{,}1368285 & 0{,}1656012 \\ 0{,}1266404 & 0{,}1024776 & -0{,}01961908 \end{bmatrix} \begin{matrix} </s> \\ al \\ gato \\ El \\ perro \\ subió \\ árbol \\ vio \\ persiguió \end{matrix}$$

Hay que tener en cuenta que la matriz que contiene las representaciones de palabras es la matriz W y no la matriz W'. Además nótese que solo se actualizan las filas correspondientes a las palabras de contexto.

2.2.3 Continuous Skip-gram Model

El modelo continuo Skip-gram es un método eficaz para el aprendizaje de alta calidad de representaciones vectoriales distribuidas que capturan un gran número de relaciones precisas entre la sintaxis y la semántica de las palabras [48]. El modelo Skip-gram [22] que se muestra en la **Figura 7** es contrario al modelo CBOW, la palabra objetivo se encuentra ahora en la capa de entrada y las palabras de contexto están en la capa de salida [22, 48, 49, 52].

Figura 18 Modelo Skip-gram. Tomado de [49].



Se sigue utilizando v_{w_I} para denotar el vector de entrada de la única palabra en la capa de entrada, y como resultado se tiene la misma definición de la salida de la capa oculta h como en la Ecuación (2-1) (de nuevo esto significa que h copia una fila de la matriz de pesos de entrada W asociada con la palabra de entrada w_I). Recordar que la definición de h fue:

$$h = W_{(k,.)}^T := v_{w_i}^T \qquad (2\text{-}44)$$

Ahora, en la capa de salida, en lugar de emitir una distribución multinomial, se producen C distribuciones multinomiales. Cada salida se calcula usando la misma matriz de la capa oculta a la de salida como se indica a continuación:

$$p(w_{c,j} = w_{O,c}|w_I) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u_{j'})} \quad (2\text{-}45)$$

Donde $w_{c,j}$ es la j-ésima palabra en el c-ésimo panel de la capa de salida; $w_{O,c}$ es la c-ésima palabra real en las palabras de contexto de salida; w_I es la única palabra de entrada; $y_{c,j}$ es la salida de la j-ésima unidad en el c-ésimo panel de la capa de salida; $u_{c,j}$ es la entrada de la j-ésima unidad en el c-ésimo panel de la capa de salida [49]. Debido a que los paneles de capa de salida comparten los mismos pesos, por lo tanto:

$$u_{c,j} = u_j = {v'_{wj}}^T \cdot h \quad para\ c = 1,2,\dots,C \quad (2\text{-}46)$$

Donde v'_{wj} es el vector de salida de la j-ésima palabra en el vocabulario (v'_{wj} es de nuevo una columna de la matriz de peso de la capa de salida a la oculta W') [49].

Teniendo en cuenta todo esto, las ecuaciones de actualización de parámetros no son tan diferentes del modelo CBOW de una palabra de contexto. La función de pérdida se cambia a:

$$\mathrm{E} = -\log p(w_{O,1}, w_{O,2}, \dots, w_{O,c}|w_I) \quad (2\text{-}47)$$

$$= -\log \prod_{c=1}^{C} \frac{\exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} \exp(u_{j'})} \quad (2\text{-}48)$$

$$= -\sum_{c=1}^{C} u_{c,j_c^*} + C \cdot \log \sum_{j'=1}^{V} \exp(u_{j'}) \quad (2\text{-}49)$$

Donde j_c^* es el índice de la actual c-ésima palabra de contexto de salida en el vocabulario V [49].

Ahora bien, tomando la derivada de E con respecto a la entrada neta de cada unidad en el panel de la capa de salida (es decir, $u_{c,j}$), se obtiene:

$$\frac{\partial \mathrm{E}}{\partial u_{c,j}} = \mathrm{y}_{c,j} - \mathrm{t}_{c,j} := \mathrm{e}_{c,j} \quad (2\text{-}50)$$

Que de nuevo es el error de predicción en la unidad (igual que en la ecuación (2-8)). Ahora, sea $\mathrm{EI} = \{EI_1, EI_2, \dots, EI_V\}$ (un vector V-dimensional) definido como la suma de los errores de predicción sobre todas las palabras de contexto, es decir:

$$EI_j = \sum_{c=1}^{C} \mathrm{e}_{c,j} \quad (2\text{-}51)$$

A continuación, se toma la derivada de E con respecto a la matriz de la capa de salida a la oculta llamada W', obteniendo.

$$\frac{\partial \mathrm{E}}{\partial \mathrm{w}'_{ij}} = \sum_{c=1}^{C} \frac{\partial \mathrm{E}}{\partial \mathrm{u}_{cj}} \cdot \frac{\partial u_{cj}}{\partial \mathrm{w}'_{ij}} = \mathrm{EI}_j \cdot \mathrm{h}_i \quad (2\text{-}52)$$

Así se obtiene la ecuación de actualización para la matriz de la capa de salida a la oculta W'.

$$w'^{(new)}_{ij} = w'^{(old)}_{ij} - \eta \cdot EI_j \cdot h_i \quad (2\text{-}53)$$

O

$$v'^{(new)}_{wj} = v'^{(old)}_{wj} - \eta \cdot EI_j \cdot \mathrm{h} \quad (2\text{-}54)$$

2.2.4 Optimización de la eficiencia computacional II

Hasta el momento, se han presentado los modelos CBOW y skip-gram. Para estos modelos, existen dos representaciones vectoriales para cada palabra en el vocabulario: el vector de entrada v_w y el vector de salida v'_w. Entrenar los vectores de entrada es computacionalmente barato; Pero el entrenamiento de los vectores de salida es muy costoso. A partir de las ecuaciones de actualización (2-26) y (2-53), se puede actualizar v'_w, para cada instancia de entrenamiento, iterando a través de cada palabra w_j en el vocabulario, calcular su entrada neta u_j, la probabilidad de predicción y_j (o $y_{c,j}$, para skip-gram), su error de predicción e_j (o EI_j para skip-gram) y utilizar su error de predicción para actualizar el vector de salida v'_j.

Hacer estos cálculos para todas las palabras y para cada instancia de entrenamiento se vuelve demasiado costoso, por lo cual es poco práctico para vocabularios grandes o corpus de formación grandes. Una solución a este problema, es disminuir el número de vectores de salida que deben actualizarse por cada instancia de entrenamiento. Para ello se puede hacer uso de métodos de optimización computacional como: softmax jerárquico o muestreo negativo.

2.2.4.1 Muestreo Negativo (Negative Sampling)

La idea de muestreo negativo es más directa que softmax jerárquico (vea 2.2.2.1), con el fin de hacer frente a la dificultad de tener demasiados vectores de salida que necesitan ser actualizados por iteración, e implica que solo se actualice una muestra de ellos.

La palabra de salida (es decir, la verdadera, o muestra positiva) debe mantenerse en el grupo de muestra y se necesita muestrear algunas palabras como muestras negativas (de ahí “muestreo negativo”). Para realizar el proceso de muestreo es necesaria una distribución probabilística que puede elegirse arbitrariamente y se denota como $P_n(w)$, se puede determinar una buena distribución empíricamente.

En Word2Vec, en lugar de utilizar una forma de muestreo negativo que produce una distribución multinomial posterior bien definida, los autores sostienen que el siguiente

objetivo de entrenamiento simplificado es capaz de producir vectores de palabras de alta calidad.

$$E = -\log(\sigma({v'_{wO}}^T h)) - \sum_{w_j \in W_{neg}} \log(\sigma(-{v'_{wj}}^T h)) \quad (2\text{-}55)$$

Donde w_O es la palabra de salida (es decir, la muestra positiva) y v'_{wO} es el vector de salida; h es el valor de salida de la capa oculta (en el modelo Skip-gram $h = v_{wI}$; y en CBOW $h = \frac{1}{C}\sum_{C=1}^{C} V_{WC}$) y $W_{neg} = (w_i \mid \mathrm{i} = 1, \ldots, \mathrm{K})$ es el conjunto de palabras que se muestrean basado en $P_n(w)$, y por lo tanto son las muestras negativas [48, 49].

Para determinar las ecuaciones de actualización de los vectores de palabras mediante el muestreo negativo, se calcula la derivada de E con respecto a la entrada neta de la unidad de salida w_j de la siguiente manera:

$$\frac{\partial \mathrm{E}}{\partial {v'_{wj}}^T h} = \begin{cases} \sigma\left({v'_{wj}}^T h\right) - 1 & si\ (w_j = w_O) \\ \sigma\left({v'_{wj}}^T h\right) & si\ (w_j \in W_{neg}) \end{cases} \quad (2\text{-}56)$$

$$= \sigma\left({v'_{wj}}^T h\right) - t_j \quad (2\text{-}57)$$

Donde t_j es la etiqueta de la palabra w_j. Esto es, $t = 1$ cuando w_j es una muestra positiva; de lo contrario $t = 0$. A continuación, se toma la derivada de E con respecto al vector de salida de la palabra w_j.

$$\frac{\partial \mathrm{E}}{\partial v'_{wj}} = \frac{\partial \mathrm{E}}{\partial {v'_{wj}}^T h} \cdot \frac{\partial {v'_{wj}}^T h}{\partial v'_{wj}} = (\sigma\left({v'_{wj}}^T h\right) - t_j) \cdot \mathrm{h} \quad (2\text{-}58)$$

Que como resultado obtiene la siguiente ecuación de actualización:

$${v'_{wj}}^{(new)} = {v'_{wj}}^{(old)} - \eta(\sigma\left({v'_{wj}}^T h\right) - t_j) \cdot \mathrm{h} \quad (2\text{-}59)$$

Que sólo debe aplicarse a $w_j \in \{w_O\} \cup W_{neg}$ en lugar de aplicarse a cada palabra en el vocabulario. Lo cual demuestra que con muestreo negativo se puede ahorrar una cantidad significativa de esfuerzo computacional por iteración. Esta ecuación puede ser usada en cualquiera de los dos modelos, tanto en el CBOW como para el Skip-gram. Para el modelo Skip-gram, se debe aplicar la ecuación (2-59) para una palabra contextual a la vez.

Ahora, para ilustrar el comportamiento de Word2Vec con el modelo Skip-gram y el muestreo negativo se presenta un ejemplo del entrenamiento de vectores de palabras tal como lo hace el código fuente de Word2Vec mediante el corpus del anterior ejemplo (véase sección 2.2.2.1).

“El perro vio al gato </s>
El perro persiguió al gato </s>
El gato subió al árbol </s>”

El vocabulario se establece con nueve palabras (puesto que se debe agregar la etiqueta de fin de sentencia, aunque no sea entrenada). La matriz W se inicializa aleatoriamente y W' se inicializa en ceros. Y se establece una taza de aprendizaje de 0.025.

Para este ejemplo, por ser de muestreo negativo, la muestra positiva es la palabra “perro” (también llamada palabra de entrada) y la muestra negativa corresponde a las palabras de contexto “El”, “vio” y “al”. Por tratarse de Skip-gram, primero predice el valor que corresponde a la palabra “El” dada la palabra “perro” y de acuerdo al error actualiza los pesos de las matrices, luego predice el valor de la palabra “vio” dada la palabra “perro” y actualiza los pesos de las matrices de acuerdo al valor del error y por último hace lo mismo prediciendo el valor de la palabra “al” dada la palabra “perro”. Esto es:

$$W = \begin{bmatrix} 0{,}1334229 & 0{,}1473134 & -0{,}1276754 \\ -0{,}1092682 & 0{,}04555257 & 0{,}1007029 \\ 0{,}0313619 & 0{,}007044474 & -0{,}1201172 \\ 0{,}07394918 & -0{,}1452128 & 0{,}04165141 \\ -0{,}02504476 & -0{,}03190613 & -0{,}1105245 \\ -0{,}06273905 & 0{,}08597819 & 0{,}1008453 \\ 0{,}09896851 & 0{,}05324809 & 0{,}05149333 \\ -0{,}1267548 & -0{,}1365255 & 0{,}1656545 \\ 0{,}126709 & 0{,}1030019 & -0{,}02013143 \end{bmatrix} \begin{matrix} </s> \\ al \\ gato \\ El \\ perro \\ subió \\ árbol \\ vio \\ persiguió \end{matrix}$$

Para la palabra “El” dado “perro” se tiene un gradiente g = -0,01235001 y un error $[3{,}888924E-06 \quad 4{,}954351E-06 \quad 1{,}716213E-05]$. Se actualizan los pesos de W:

$$w_{0,3}{}^{(new)} = w_{0,3}{}^{(old)} + e_0 = 0{,}07394918 + 3{,}888924E-06 = \mathbf{0,07395307}$$
$$w_{1,3}{}^{(new)} = w_{1,3}{}^{(old)} + e_1 = -0{,}1452128 + 4{,}954351E-06 = \mathbf{-0,1452079}$$
$$w_{2,3}{}^{(new)} = w_{2,3}{}^{(old)} + e_2 = 0{,}04165141 + 1{,}716213E-05 = \mathbf{0,04166857}$$

$$W = \begin{bmatrix} 0{,}1334229 & 0{,}1473134 & -0{,}1276754 \\ -0{,}1092682 & 0{,}04555257 & 0{,}1007029 \\ 0{,}0313619 & 0{,}007044474 & -0{,}1201172 \\ \mathbf{0{,}07395307} & \mathbf{-0{,}1452079} & \mathbf{0{,}04166857} \\ -0{,}02504476 & -0{,}03190613 & -0{,}1105245 \\ -0{,}06273905 & 0{,}08597819 & 0{,}1008453 \\ 0{,}09896851 & 0{,}05324809 & 0{,}05149333 \\ -0{,}1267548 & -0{,}1365255 & 0{,}1656545 \\ 0{,}126709 & 0{,}1030019 & -0{,}02013143 \end{bmatrix} \begin{matrix} </s> \\ al \\ gato \\ El \\ perro \\ subió \\ árbol \\ vio \\ persiguió \end{matrix}$$

Ahora para la palabra “vio” dado “perro” se tiene un gradiente g = -0,01227502 y un error $[2{,}304397E-05 \quad -4{,}525107E-05 \quad 1{,}297937E-05]$. Se actualizan los pesos de W:

$$W = \begin{bmatrix} 0{,}1334229 & 0{,}1473134 & -0{,}1276754 \\ -0{,}1092682 & 0{,}04555257 & 0{,}1007029 \\ 0{,}0313619 & 0{,}007044474 & -0{,}1201172 \\ 0{,}07395307 & -0{,}1452079 & 0{,}04166857 \\ -0{,}02504476 & -0{,}03190613 & -0{,}1105245 \\ -0{,}06273905 & 0{,}08597819 & 0{,}1008453 \\ 0{,}09896851 & 0{,}05324809 & 0{,}05149333 \\ \mathbf{-0{,}1267317} & \mathbf{-0{,}1365707} & \mathbf{0{,}1656675} \\ 0{,}126709 & 0{,}1030019 & -0{,}02013143 \end{bmatrix} \begin{matrix} </s> \\ al \\ gato \\ El \\ perro \\ subió \\ árbol \\ vio \\ persiguió \end{matrix}$$

Luego para la palabra “al” dado “perro” se tiene un gradiente g = -0,01235001 y un error $[6{,}673054E-06 \quad -6{,}211358E-05 \quad 5{,}675475E-05]$. Se actualizan los pesos de W:

$$W = \begin{bmatrix} 0{,}1334229 & 0{,}1473134 & -0{,}1276754 \\ \mathbf{-0{,}1092615} & \mathbf{0{,}04549046} & \mathbf{0{,}1007597} \\ 0{,}0313619 & 0{,}007044474 & -0{,}1201172 \\ 0{,}07395307 & -0{,}1452079 & 0{,}04166857 \\ -0{,}02504476 & -0{,}03190613 & -0{,}1105245 \\ -0{,}06273905 & 0{,}08597819 & 0{,}1008453 \\ 0{,}09896851 & 0{,}05324809 & 0{,}05149333 \\ -0{,}1267317 & -0{,}1365707 & 0{,}1656675 \\ 0{,}126709 & 0{,}1030019 & -0{,}02013143 \end{bmatrix} \begin{matrix} </s> \\ al \\ gato \\ El \\ perro \\ subió \\ árbol \\ vio \\ persiguió \end{matrix}$$

De esta forma continua el entrenamiento hasta procesar todas las palabras del corpus y todas las iteraciones dadas.

Los resultados de Word2Vec han demostrado recientemente que los vectores de palabra capturan muchas regularidades lingüísticas, por ejemplo, en las operaciones vectoriales: vector (“Paris”) - vector ("Francia") + vector (“Italia”) da como resultado un vector que está muy cerca del vector (“Roma”), y el vector (“rey”) - vector ("hombre") + el vector ("mujer") está próximo del vector ("reina") [22].

En el área de generación automática de resúmenes, Word2Vec ha sido poco utilizado debido a su reciente aparición. En el 2014 [27] Word2Vec se utilizó para generar dos representaciones de la matriz de términos: una mediante la distancia euclidiana de los vectores y otra con la distancia de coseno, e implementando un algoritmo de

agrupamiento en tópicos de opiniones. En el 2015 [26], Word2Vec se usó sumando los vectores de las palabras contenidas en las oraciones para obtener así los vectores de oraciones y posteriormente aplicar algoritmos de agrupamiento para obtener las oraciones centrales en los *k* grupos. En otros estudios en NLP, Word2Vec ha sido utilizado para el análisis de sentimientos y representaciones de palabras bilingües.

Existen diferentes librerías para implementar Word2Vec, entre ellas:

- Gensim. Librería gratis para Python que puede ejecutarse en Linux, Windows y Mac OS X y en cualquier otra plataforma que ejecute Python 2.6+ y NumPy. Requiere la instalación de Python, NumPy y SciPy.
- Deeplearning4j. Implementa una forma distribuida de Word2Vec para Java y Scala. Los requisitos de esta librería son: Java (versión de desarrollador) 1.7 o superior (solo soportado en versiones de 64-Bits), Apache Maven, IntelliJ IDEA o Eclipse y Git.
- Para .NET Framework se encuentra una implementación gratuita en Github de Word2Vec en C#, que se puede ejecutar en Windows con Visual Studio 2010 o superior [53].

2.3 Representación de los documentos

2.3.1 Modelo de Espacio Vectorial

El modelo de espacio vectorial es un grupo de documentos representados mediante vectores de un espacio vectorial [30]. Cada vector de documento denotado por $\vec{d}_j$ representa las m características del conjunto de documentos que, por lo general, definen la frecuencia de ciertas palabras o términos en el documento. Cada característica (o componente) del vector es un escalar.

El conjunto de documentos se representa de la siguiente forma, $\mathbf{D} = \{\boldsymbol{d_1}, \boldsymbol{d_2}, \dots, \boldsymbol{d_o}\}$, para un número $\boldsymbol{o}$ de documentos. Cada documento $\boldsymbol{d_j}$ (para $\boldsymbol{j} = \mathbf{1}, \dots, \boldsymbol{o}$) está compuesto por un conjunto de oraciones así, $\boldsymbol{d_j} = \{\boldsymbol{s_1}, \boldsymbol{s_2}, \dots, \boldsymbol{s_p}\}$, donde $\boldsymbol{p}$ es el total de oraciones que componen al documento $\boldsymbol{d_j}$. De la misma forma se puede representar el conjunto de documentos como el total de oraciones de todos los documentos que lo componen, $\boldsymbol{D} = \{\boldsymbol{s_1}, \boldsymbol{s_2}, \dots, \boldsymbol{s_n}\}$, donde $\boldsymbol{s_j}$ es la j-ésima oración en $\boldsymbol{D}$, asimismo $\boldsymbol{n}$ es el total de oraciones en el conjunto de documentos, $\boldsymbol{s_j} \in \boldsymbol{D} \leftrightarrow \boldsymbol{s_j} \in \boldsymbol{d_j} \in \boldsymbol{D}$**.** También se puede representar $\boldsymbol{T} = \{\boldsymbol{t_1}, \boldsymbol{t_2}, \dots, \boldsymbol{t_m}\}$ como el conjunto de términos (palabras) que se encuentran en $\boldsymbol{D}$**,** y $\boldsymbol{m}$**,** es el número total de términos [21].

Teniendo en cuenta lo anterior, cada oración $\vec{\boldsymbol{s}}_{\boldsymbol{j}}$ en el espacio vectorial, es representada por uno o más términos $\boldsymbol{t_i}$; los términos pueden ponderarse o no con un peso $\boldsymbol{w_{ij}}$ de acuerdo a su relevancia. En la **Figura 8** se muestra un espacio tridimensional donde cada oración es identificada por tres diferentes términos, éste espacio puede extenderse a m dimensiones dependiendo del número m de términos diferentes en la oración, en conclusión, cada oración $\vec{\boldsymbol{s}}_{\boldsymbol{j}}$ es representada por un vector *m*-dimensional así $\vec{\boldsymbol{s}}_{\boldsymbol{mj}} = (\boldsymbol{w_{1j}}, \boldsymbol{w_{2j}}, \dots, \boldsymbol{w_{ij}}, \dots, \boldsymbol{w_{mj}})$ [54]. Donde cada $\boldsymbol{w_{ij}}$ es el peso del *i-ésimo* término de la j-ésima oración.

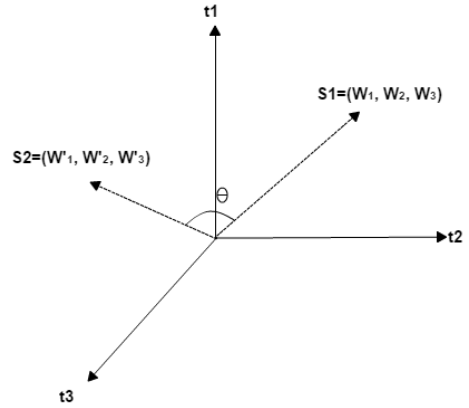
Cada término del conjunto de documentos $\boldsymbol{t_i}$ representa una dimensión independiente en el espacio vectorial, por lo general los vectores se encuentran en los cuadrantes positivos, debido a que a los términos les son asignados valores positivos [55].

2.3.2 Técnicas de ponderación de términos

2.3.2.1 Booleana

Determina mediante unos y ceros la existencia o ausencia respectiva de un término en la oración [56], como se muestra en la ecuación (2-60).

Figura 19 Representación de oraciones en el espacio vectorial tridimensional. Adaptado de [54].



$$w_{ij} = \begin{cases} 1, & si\ el\ término\ \boldsymbol{i}\ aparece\ en\ la\ oración\ \boldsymbol{j} \\ 0, & en\ caso\ contrario \end{cases} \quad (2\text{-}60)$$

2.3.2.2 Frecuencia del término

La frecuencia del término (TF), es asignada acuerdo al número de veces que aparece cierto término en la oración dando mayor o menor relevancia, como se muestra a continuación en la ecuación (2-61).

$$w_{ij} = f_{ij} \quad (2\text{-}61)$$

Donde f_{ij} es la frecuencia del i-ésimo término en la oración j.

2.3.2.3 Frecuencia Inversa de la Oración

Como se indica en [56], los problemas que puede tener TF se presentan cuando cierto término aparece en casi todas las oraciones de la colección haciendo que el método no sea útil para determinar las oraciones relevantes. Una alternativa a este problema es la frecuencia inversa de la oración (ISF) [19]. La cual utiliza la ecuación (2-62) para determinar la importancia de los términos de una oración $\vec{\boldsymbol{s}}_{\boldsymbol{j}}$.

$$w_{ij} = \log \frac{N}{n_j} \quad (2\text{-}62)$$

Donde N corresponde al número total de oraciones y n_j es el número de oraciones en las que está el término i.

2.3.2.4 Ponderación basada en la frecuencia relativa de un término

La frecuencia relativa de un término (TF-ISF) [30], es la combinación de las definiciones de TF e ISF para determinar el peso de cada término de la oración $\vec{s}_j$. La frecuencia relativa de un término atribuye mayor relevancia a los términos menos frecuentes en el conjunto de oraciones, pero a su vez más frecuentes en la oración, como se ve en la ecuación (2-63).

$$w_{ij} = (f_{ij}) \left(\log \frac{N}{n_j} \right) \tag{2-63}$$

Donde w_{ij} es el peso del i-ésimo término de la oración $\vec{s}_j$.

2.3.3 Medidas de Similitud

Para los cálculos realizados en los algoritmos de generación automática de resúmenes se hace necesario determinar la similitud entre las diferentes frases representándolas como vectores en un modelo de espacio vectorial [21].

2.3.3.1 Similitud de Coseno

Siendo $\vec{s}_i, \vec{s}_j$ dos vectores de oración *m*-dimensionales diferentes del vector cero, a los cuales se calculó los pesos de sus términos mediante TF-ISF (véase ecuación (2-63)). Entonces el ángulo φ o la similitud coseno entre $\vec{s}_i$ y $\vec{s}_j$ es definida en el intervalo [0, 1] como se muestra en la ecuación (2-64):

$$simcos(\vec{s_i}, \vec{s_j}) = \frac{\sum_{k=1}^{m} w_{ki} \; w_{kj}}{\sqrt{\sum_{k=1}^{m} w_{ki}^2 \sum_{k=1}^{m} w_{kj}^2}} \tag{2-64}$$

2.3.4 Representación de Documentos por Medio de Matrices

Existen diversas formas de realizar una representación en el espacio multidimensional del conjunto de vectores de oraciones y la similitud entre estas, las más conocidas y usadas en investigaciones previas son la Matriz de Términos por Oraciones y la Matriz de similitud de Cosenos.

2.3.4.1 Matriz de Términos por Oraciones

La matriz de términos por oración también llamada matriz TF-ISF es una matriz de mxn donde cada fila contiene los m términos, cada columna representa las n oraciones y el elemento ij denotado como w_{ij} corresponde al peso del i-ésimo término en la oración j,

calculado mediante a la ecuación (2-63); la Matriz se representa por la ecuación (2-65) [57, 58].

$$Matriz\ TF-ISF=\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1j} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2j} & \dots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{i1} & w_{i2} & \dots & w_{ij} & \dots & w_{in} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mj} & \dots & w_{mn} \end{pmatrix} \quad (2\text{-}65)$$

2.3.4.2 Matriz de Similitud de Cosenos

La matriz de similitud de cosenos representada en la ecuación (2-66) es una matriz cuadrada de nxn, donde n es el número total de oraciones, $\boldsymbol{a_{ij}}$ representa la similitud coseno entre la oración i y la oración j (véase ecuación (2-64)) [58].

$$MatrizDeSimilitud=\begin{pmatrix} 1_{11} & a_{12} & \dots & a_{1j} & \dots & a_{1n} \\ a_{21} & 1_{22} & \dots & a_{2j} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ a_{i1} & a_{i2} & \dots & 1_{ij} & \dots & a_{in} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nj} & \dots & 1_{nn} \end{pmatrix} \quad (2\text{-}66)$$

2.3.5 Vector centroide

El Vector Centroide es la representación del conjunto de oraciones como un solo vector, en el cual cada componente es el peso promedio del término i. Para calcular el vector centroide se utiliza la Matriz TF-ISF (ecuación (2-65)) [30]. El término i es el vector fila i de la Matriz TF-ISF como se ve en ecuación (2-67).

$$\overrightarrow{término_i}=(w_{i1},w_{i2},\dots,w_{ij},\dots,w_{in}) \quad (2\text{-}67)$$

Para obtener el peso promedio $\overline{w_i}$ se suma la fila del término i y se divide por el número total de términos(m) así: $\overline{w_i}=\frac{1}{m}\sum_{k=1}^{n} w_{ik}$. Por lo tanto, el vector centroide se define en la ecuación (2-68):

$$\overrightarrow{vectorCentroide}=(\overline{w}_1,\overline{w}_2,\dots,\overline{w}_i,\dots,\overline{w}_m) \quad (2\text{-}68)$$

3 WORD2VEC EN LA GENERACIÓN AUTOMÁTICA DE RESÚMENES DE MÚLTIPLES DOCUMENTOS

En éste capítulo se describe cómo se utiliza Word2Vec (véase sección 2.2) como mecanismo de representación de los documentos para la generación automática de resúmenes genéricos extractivos de múltiples documentos con los algoritmos LexRank, LSA y MCMR (para conocer detalles de estos algoritmos vea el Anexo A).

3.1 Adaptación del modelo propuesto en Word2Vec para representar oraciones de múltiples documentos.

3.1.1 Entrenamiento de palabras para obtención de vectores

Primero se identificaron cuatro formas diferentes de obtener vectores de palabras entrenadas a través de Word2Vec. La primera utilizando el gran compendio de tres millones de palabras con su respectivo vector de trecientas (300) dimensiones, proporcionado por Google inc. llamado GoogleNews-vectors-negative300 [59]. Se trata de un gran corpus de palabras previamente entrenadas con Word2Vec, con su respectivo vector. La segunda consiste en crear un gran corpus de oraciones para el posterior entrenamiento que integre todas las frases de los tres grupos de documentos de DUC (DUC2005 [60], DUC006 [61] y DUC2007 [62], véase sección 2.1.1.3) sin aplicar stemming ni lematización ni sinónimos (ver sección 2.1.1.1) con el fin de entrenar el vocabulario total. La tercera aplicando lematización y búsqueda de sinónimos a las oraciones de los tres grupos de documentos de DUC (DUC2005, DUC006 y DUC2007) sin aplicar stemming para luego ser entrenadas. Y la cuarta aplicando stemming a las oraciones de los tres grupos de documentos de DUC (DUC2005, DUC006 y DUC2007) sin aplicar lematización ni búsqueda de sinónimos para después entrenar las palabras. Para estas tres últimas, se generan los vectores de palabras usando Word2Vec de .NET Framework (véase sección 2.2).

3.1.1.1 Word2Vec con tres millones de palabras

Al tener un archivo con un tamaño aproximado de 3.6GB (GoogleNews-vectors-negative300, archivo con los 3 millones de palabras y vectores de 300 dimensiones), se debe tener en cuenta la optimización computacional. Primero se pensó en representar dicho archivo en una tabla hash (HashTable) para acceder al vector (valor) dada la palabra (llave); este mecanismo es rápido que en cuestión de tiempos pero debido a que las tablas hash son almacenadas en la memoria RAM, se imposibilitó este trabajo por las limitaciones de memoria con las que se cuenta. Por lo tanto, los vectores de palabras se almacenaron en una base de datos de Microsoft SQL Server 2014 para acceder a estos vectores en cualquier momento dada la palabra deseada. A continuación, en la **Figura 9** se muestra el diagrama de la base de datos.

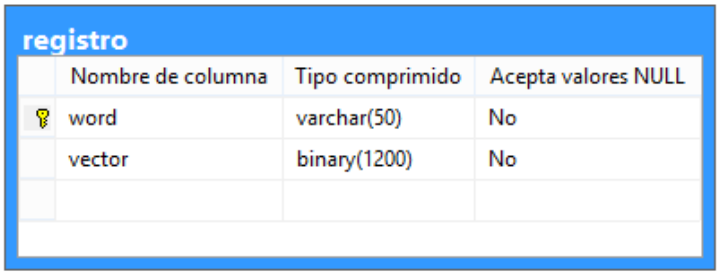
Formar la matriz TF-ISF a partir de Word2Vec conlleva dos dificultades adicionales. La primera tiene que ver con el contenido de las oraciones en los documentos de los datasets DUCs. Algunas noticias, al ser transformadas a texto plano, incluyen en las

oraciones palabras vacías que carecen de significado que luego de ser aplicado el pre-procesamiento del texto, aun así, permanecen. Esto se debe a que algunos datos representados en tablas terminan convirtiéndose en oraciones o simplemente palabras mal redactadas. Cabe aclarar que cuando se utiliza Word2Vec entrenado con los términos DUC, el algoritmo no hace uso de ninguna estrategia puesto que aunque sean palabras vacías, también son entrenadas. Para solucionar este inconveniente se establecieron las siguientes estrategias:

1) **Crear los vectores:** para ello se define un procedimiento que crea vectores a aquellas palabras que no los poseen. Para lograrlo, se busca el término en cuestión en todos los documentos y con un contexto de uno (una palabra atrás y una palabra adelante) se seleccionan todos los términos que se encuentran en su contexto, se suman todos los vectores del contexto y se divide entre el total de palabras halladas en el contexto. Esto tiene las siguientes limitaciones: i) la palabra es única en el conjunto de documentos, ii) la oración solo contiene esta palabra y es única en el documento iii) la palabra solo cuenta con una palabra de contexto. Es en estas situaciones en las que no se pueden crear los vectores nuevos y se debió hacer uso de la siguiente opción.

2) **Ignorar los términos:** No se tiene en cuenta el término para ser utilizado en las operaciones de dicha frase debido a que no tiene un vector en el Word2Vec de 3 millones de palabras. Es decir, la palabra que no tenga representación en el espacio vectorial se toma como una palabra vacía y no se agrega a la frase.

Figura 20 Diagrama de base de datos de palabras y vectores de Word2Vec.



El segundo reto consistió en aprovechar el valor de las frases ya conformadas en Word2Vec de 3 millones. En el archivo proporcionado por Google inc. en [59] se encuentran una considerable cantidad de frases entrenadas como si fueran una palabra mediante el conector "_". Por ejemplo: la palabra "los" tiene su contexto definido al igual que la palabra "ángeles"; pero la palabra "los_ángeles" comparte un contexto muy diferente de las palabras por separado. Teniendo en cuenta que las frases contenidas en Word2Vec de 3 millones están definidas con un tamaño mínimo de dos palabras y máximo de cuatro palabras (los demás son términos independientes), se definió el **Algoritmo 5** presentado a continuación.

Algoritmo 5. Estrategia para aprovechar las frases conformadas en Word2Vec de 3 millones.

Paso 1	Inicializar posición: se inicializa la posición en cero para ubicarse en el primer término de la oración.
Paso 2	Inicializar N-grama: Se inicia con valor de cuatro para cubrir la primera opción que es tomar la frase más larga. Si el valor del N-grama más la posición supera en tamaño de la oración, se disminuye hasta encontrar un N-grama adecuado.
Paso 3	Formar frase: dependiendo el tamaño del N-grama, se conforma la frase concatenando el carácter "_" entre términos.
Paso 4	Buscar la frase: se consulta en la base de datos para saber si está la frase formada.
Paso 5	Determinar si sirve la frase: si la frase está en la base de datos, ésta se cuenta como un término para la conformación de la oración, en caso contrario se disminuye el tamaño del N-grama y se regresa al Paso 3. En caso que el N-grama sea menor que dos, de toma como un término independiente y se le dará el manejo correspondiente a los términos.
Paso 6	Actualizar posición: si después del Paso 5 resulta un término independiente, la posición se incrementa en 1 y se regresa al Paso 2 y si resulta una frase, la posición se incrementa dependiendo el tamaño del N-grama.
Paso 7	Revisar el criterio de parada: la oración termina de formarse cuando la posición sea igual al tamaño de la oración.

3.1.1.2 Word2Vec texto procesado sin lematización, sin sinónimos y sin stemming

Con el fin de evitar el problema de palabras presentes en los documentos de DUC (véase sección 2.1.1.3) y ausentes en GoogleNews-vectors-negative300, se decidió entrenar las palabras contenidas en los documentos de DUC. Para ello, la primera estrategia fue elaborar un corpus de entrenamiento que conserve las palabras con un mínimo procesamiento como lo son la conversión a minúsculas y eliminación de palabras vacías (véase sección 2.1.1.1). Se decidió involucrar en el corpus todas las frases de los documentos de DUC2005, DUC2006 Y DUC2007 para realizar un mejor entrenamiento de las palabras. Este procesamiento hace que la siguiente frase:

"*when mr bramble has testified he is expected to be followed by another dea agent who worked in panamá*".

Se modifique a la siguiente:

"mr bramble testified expected dea agent worked panama".

Luego de extraer todas las frases y agregarles la etiqueta de fin de sentencia "</s>" a cada oración en los documentos, se entrenaron las palabras con el software de Word2Vec de .Net Framework [53]. Dando como resultado un archivo que comprende el vocabulario total y el vector correspondiente a cada palabra del vocabulario. Posteriormente, estos vectores se almacenaron en una base de datos (diferente a la base de datos de tres millones de palabras) con el mismo diseño de base de datos de la **Figura 9**. Para hacer uso de dichos vectores en el momento de ejecutar los algoritmos de generación automática de resúmenes extractivos genéricos de múltiples documentos (véase Anexo A).

3.1.1.3 Word2Vec texto procesado con lematización y búsqueda de sinónimos

Teniendo en cuenta que al entrenar los vectores con un mínimo procesamiento del texto (texto procesado sin lematización, sin sinónimos y sin stemming, véase sección 3.1.1.2), se encontraban palabras que compartían la misma forma flexionada (es decir, en plural, en femenino, conjugada, etc) y tenían vectores muy diferentes entre sí, se decidió aplicar lematización y búsqueda de sinónimos al texto de entrenamiento para garantizar que palabras que comparten el mismo significado sean representadas en un mismo vector. Con este fin, se entrenaron las palabras comprendidas en los documentos de DUC2005, DUC2006 Y DUC2007 con un procesamiento del texto que involucra la conversión a minúsculas, eliminación de palabras vacías, lematización y búsqueda de sinónimos. Para el procesamiento de lematización y búsqueda de sinónimos se hizo uso de la herramienta NLTK 3.0 [34] (véase sección 2.1.1.1). Así frases como la siguiente:

"*when mr bramble has testified he is expected to be followed by another dea agent who worked in panamá*".

Se modifican de esta manera:

"*Mister bramble testify expect Drug_Enforcement_Administration agent work Panama*".

Al igual que en la sección 3.1.1.2, se agrega la etiqueta "</s>" de fin de sentencia. Y se procede a entrenar las palabras del vocabulario resultante haciendo uso de la misma herramienta (Word2Vec de .Net Framework [53]). Luego los vectores son almacenados en una nueva base de datos con el mismo diseño de la **Figura 9** para ser utilizados posteriormente en la generación automática de los resúmenes.

3.1.1.4 Word2Vec texto procesado con stemming

Con el mismo fin de la sección anterior (Sección 3.1.1.3), se procedió a entrenar las palabras comprendidas en los documentos de DUC2005, DUC2006 Y DUC2007 con un procesamiento del texto que involucra la conversión a minúsculas, eliminación de palabras vacías y stemming mediante la herramienta Lucene versión 2.9.2.2 [32]. Como resultado se obtiene que al procesar la siguiente frase:

"*when mr bramble has testified he is expected to be followed by another dea agent who worked in panamá*".

Se obtiene la siguiente:

"mr brambl testifi expect dea agent work panama".

De igual forma que en la sección 3.1.1.2, se agrega la etiqueta "</s>" de fin de sentencia a cada frase. Luego se entrenaron las palabras del vocabulario resultante con la misma herramienta (Word2Vec de .Net Framework [53]). Luego los vectores se almacenaron en una nueva base de datos con el mismo diseño de la **Figura 9** para posteriormente ser utilizados en la generación automática de los resúmenes.

3.1.1.5 Parámetros de entrenamiento

Se realizaron tres entrenamientos presentados anteriormente en las secciones 3.1.1.2, 3.1.1.4 y 3.1.1.5. Estos se realizaron bajo los mismos parámetros de entrenamiento de Word2Vec. Se empleó la arquitectura CBOW y el algoritmo Softmax Jerárquico. Para determinar el número de palabras de contexto se estableció una ventana de 5 unidades. Se realizaron 1000 iteraciones de entrenamiento. Se fijó una taza de aprendizaje inicial de 0.025, los vectores se crearon con un tamaño de 300 dimensiones y se hizo uso de 4 hilos de ejecución.

3.1.2 Representación de frases a partir de vectores

Los vectores de palabras formados con Word2Vec se utilizan para la creación de la matriz TF-ISF (véase sección 2.3.4.1). La matriz TF-ISF original es una matriz de m oraciones (filas) y n términos (columnas) que se crea dependiendo del cálculo del peso que tenga cada término en cada oración. Como en Word2Vec las palabras ya poseen un valor correspondiente a un punto en el espacio definido, fue necesario buscar una alternativa para sustituir los pesos de la matriz TF-ISF. Además, se debía tener en cuenta que las filas mantuvieran el mismo tamaño para así garantizar las propiedades de la matriz en la ejecución de los algoritmos. Para esto se determinaron las siguientes formas de hacerlo:

1) **Suma:** para determinar el vector que representa a la oración (frase) j, se toman todos los vectores de palabras que componen dicha oración y se suma posición a posición (dimensión a dimensión) y así obtener un vector que contiene las mismas dimensiones de todos los vectores de la oración. Por último, éste vector de oración se ubica en la fila j de la matriz TF-ISF. Ejemplo:

   Vector de palabra 1: $(0,1 \quad 0,4 \quad 0,2 \quad 0,3 \quad 0,2)$

   Vector de palabra 2: $(0,3 \quad 0,5 \quad 0,2 \quad 0,1 \quad 0,6)$

   Vector de frase: $(0,4 \quad 0,9 \quad 0,4 \quad 0,4 \quad 0,8)$

2) **Promedio:** el vector de la oración j, se determina de acuerdo al número de términos que contenga, para así formar un vector que sea el centroide a todos los vectores de la oración j y luego este vector promedio es la j-ésima fila de la matriz TF-ISF. Ejemplo:

   Vector de palabra 1: $(0,1 \quad 0,4 \quad 0,2 \quad 0,3 \quad 0,2)$

   Vector de palabra 2: $(0,3 \quad 0,5 \quad 0,2 \quad 0,1 \quad 0,6)$

   Vector de frase: $(0,2 \quad 0,45 \quad 0,2 \quad 0,2 \quad 0,4)$

3.2 LexRank con Word2Vec

El algoritmo LexRank con umbral (véase Anexo A) hace uso de la matriz de términos por oración (TF-ISF) y de la matriz de similitud de cosenos (véase sección 2.3.4.2) para representar los conjuntos de documentos de entrada. En este trabajo, se hace uso de la nueva matriz TF-ISF como se indicó en la sección 3.1.2 para calcular los valores de la nueva matriz de similitudes. Esto es, teniendo los vectores de cada oración en las filas de la matriz TF-ISF se evalúa la similitud coseno de cada oración con el resto de oraciones de los documentos hasta agotar todas las comparaciones. Obteniendo una matriz de $n \times n$ donde su elemento S_{ij} es la similitud coseno entre la oración i y la oración j para $i = 1,2,\dots,n$ y $j = 1,2,\dots,n$, en caso de i ser igual a j, el valor de S_{ij} es 1.

Teniendo conformada la matriz de similitudes, se aplica el algoritmo LexRank con umbral; sin embargo, para que la información de contexto que proporciona Word2Vec no sea desaprovechada, se hizo un cambio al Algoritmo. Este cambio es presentado en el **Algoritmo 6** a continuación.

Algoritmo 6. Cálculo de puntuación de LexRank modificado para Word2Vec.

```
Entrada:  Arreglo S de n oraciones, umbral t, valor del factor de amortiguamiento (dampingFactor)
Salida:   Arreglo L con los scores definidos por LexRank para cada oración
Arreglo   CosineMatrix[n][n];
Arreglo   L[n];
01 Para i=1 hasta n haga
02        suma=0
03        Para j=1 hasta n haga
04                  CosineMatrix[i][j] = idf-modified-cosine(S[i],S[j]);
05                  Si CosineMatriz[i][j] > t haga
06                            suma = suma + CosineMatrix[i][j];
08                  Si No
09                            CosineMatriz[i][j] = 0;
10                  Fin Si
11        Fin Para
15 Fin Para
16 Para i=1 hasta n haga
17        Para j=1 hasta n haga
18                  CosineMatriz[i][j] = CosineMatriz[i][j] / suma;
19        Fin Para
20 Fin Para
21 Para i=1 hasta n haga
22        Para j=1 hasta n haga
23                  CosineMatriz[i][j] = (dampingFactor/n) + (1- dampingFactor)* CosineMatriz[i][j];
24        Fin Para
25 Fin Para
26 L = PowerMethod(CosineMatrix, n, £ ); //Algoritmo 1 previamente presentado
27 retorne L;
```

Lo cual obliga a cambiar también el método de potencia (powerMethod) presentado en el **Algoritmo 1** del Anexo A y dando como resultado el nuevo **Algoritmo 7.**

Algoritmo 7. Método de Potencia adaptado para Word2Vec.

Entrada: Una matriz M estocástica, irreducible y aperiódica
Entrada: Tolerancia de error £
Entrada: Lista de frases
Salida: Vector propio p
01 sumatoria = 0
02 Para i=1 hasta tamaño de M haga
03 sumatoria = sumatoria + listaFrases[i].SimilitudCosenoAlDocumento
04 Fin para
05 Para i=1 hasta tamaño de M haga
06 $p_0 =$ listaFrases[i]. SimilitudCosenoAlDocumento/sumatoria
07 $t = 0$
08 repita
09 $t = t + 1$
10 $p_t = M^T P_{t-1}$
11 $\delta = ||p_t - p_{t-1}||$
12 hasta que $\delta <$ £;
13 retorne p_t

3.3 Análisis Semántico Latente con Word2Vec

El algoritmo denominado Análisis Semántico Latente o LSA (véase Anexo A) al igual que LexRank, utiliza la matriz TF-ISF para generar resúmenes de múltiples documentos. A la matriz TF-ISF se le aplica la descomposición de valores singulares (Singular Value Decomposition, SVD) para obtener los vectores latentes de frases. A partir de ahí se generan las matrices U, Σ, V^T y la matriz de frases latentes que en sus columnas contiene las frases de los documentos representadas en el espacio latente. A partir de estas matrices se desarrolla el nuevo algoritmo de LSA con Word2Vec para la selección de las frases que componen el resumen de la siguiente manera:

Primero se calcula un nuevo vector de documento llamado vector de documento latente (VDL):

$$VDL_{1xr} = VD_{1xm} x\, V_{mxr}\, x\, \Sigma^{-1}_{rxr} \qquad (3\text{-}1)$$

VDL es la transformación del vector de documento original (VD) al espacio latente dado por SVD.

Luego de la matriz de frases latentes se procede a seleccionar las frases que tengan mayor similitud al vector VDL siempre y cuando las frases que se seleccionan no sean muy parecidas a las ya seleccionadas. Para esto se introduce un umbral de selección de frases.

El **Algoritmo 8** presenta los cambios realizados al algoritmo LSA para la generación automática de resúmenes mediante una representación basada en Word2Vec.

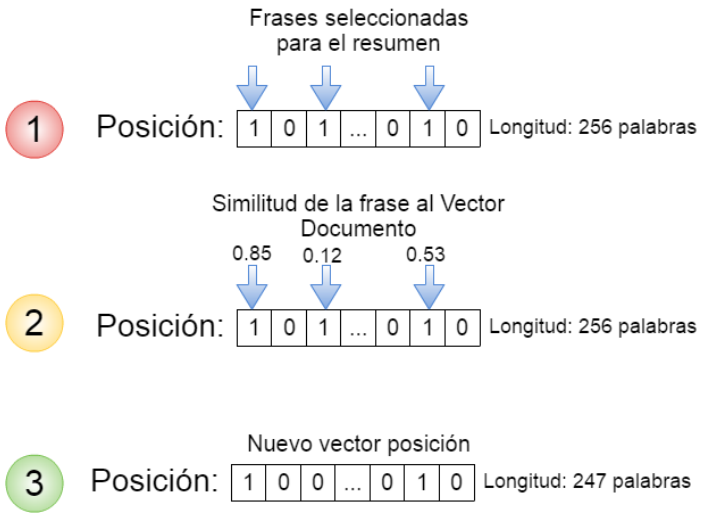
Algoritmo 8. LSA modificado para Word2Vec.

```
Entrada:  Arreglo S de n oraciones, umbral t
Salida:   Resumen de documentos.
01        SVD: se aplica la descomposición de valores singulares.
02        Cálculo del vector VDL: mediante la ecuación (3-1) y las matrices obtenidas en SVD.
03        Selección de frases:
04              De las frases latentes, incluya al resumen la más similar a VDL (similitud coseno)
05              Repita
06                  Incluir al resumen la frase más parecida y diferente a las que ya se incluyeron
07                  dado un umbral.
08              Hasta completar la máxima longitud del resumen.
09        Retornar resumen.
```

3.4 Máxima Cobertura y Mínima Redundancia con Word2Vec

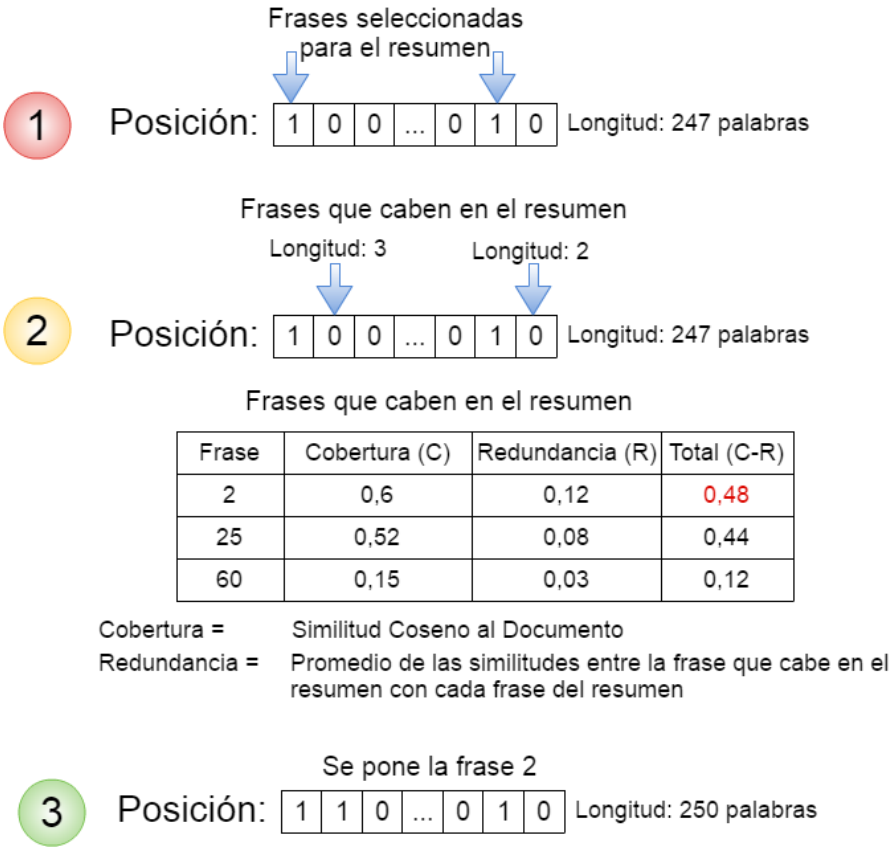
El Algoritmo Máxima Cobertura y Mínima Redundancia (MCMR, véase Anexo A) hace uso de la Matriz de Similitud de Cosenos (Véase sección 2.3.4.2) para el cálculo de la función objetivo. La implementación del algoritmo MCMR se realizó ciñéndose a la descripción dada por los autores. Sin embargo, estos no especifican cómo el algoritmo aplica la restricción del tamaño máximo del resumen (250 palabras) y si no se restringe, el algoritmo tiende a involucrar un número muy grande de frases en el resumen. Por lo cual se implementó una estrategia de reparación que consiste en quitar las frases que le dan menor cobertura al resumen y poner las que aporten mayor cobertura teniendo en cuenta el criterio de redundancia. Con esto se garantiza que cada partícula en el proceso iterativo tiene el tamaño adecuado. A continuación, la **Figura 10** muestra una descripción gráfica de la función Quitar.

Figura 21 Descripción de la función Quitar de MCMR.



A continuación, se presenta se presenta la **Figura 11** que describe la función poner de la etapa de reparación de la posición del algoritmo MCMR.

Figura 22 Descripción de la función Poner de MCMR.



4 EXPERIMENTACIÓN

4.1 Conjuntos de documentos de prueba

Los resúmenes obtenidos por los algoritmos presentados en el capítulo 3, se generaron a partir de las colecciones de documentos DUC2005 [60], DUC2006 [61] y DUC2007[62]. Tanto en DUC2005 como en DUC2006 y DUC2007, se planteó la tarea de generar resúmenes genéricos de múltiples documentos de noticias periodísticas en inglés en 250 palabras (longitud máxima del resumen) tomando un conjunto de 45 a 50 grupos o tópicos, donde cada tópico cuenta con unos 25 a 50 documentos. A continuación, en la **Tabla 2** se presenta una breve descripción de los conjuntos de datos.

Tabla 2. Descripción de los conjuntos de datos de prueba

Descriptor	DUC 2005	DUC 2006	DUC 2007
Número de Grupos	50	50	45
Número de Documentos	1593	1250	1125
Promedio documentos por grupo	31.86	25	25
Fuente de documentos	TREC: Los Angeles Times y Financial Times of London	AQUAINT: Associated Press, New York Times, y Xinhua newswire	AQUAINT: Associated Press, New York Times, y Xinhua newswire
Longitud esperada del resumen	250 palabras	250 palabras	250 palabras

4.2 Métricas de evaluación

Para la evaluación automática de la calidad de los resúmenes de este trabajo investigativo se hizo uso de la herramienta llamada ROUGE 1.5.5 desarollada en Perl por Lin Chin-Yew [47], la cual hace uso de las métricas más utilizadas en el estado del arte (ROUGE-1, ROUGE-2 y ROUGE-SU4) y se describen a continuación.

4.2.1 ROUGE-N (ROUGE-1 y ROUGE-2)

ROUGE-N está basado en N-gramas de co-ocurrencia para medir la similitud entre un resumen candidato y un conjunto de resúmenes de referencia (resúmenes ideales creados por expertos). Que para el caso de ROUGE-1 se denomina unigrama y para ROUGE-2 bigrama. ROUGE-N se calcula mediante la ecuación (4-1):

$$\text{ROUGE} - \text{N} = \frac{\sum_{S \in (ResúmenesDeReferencia)} \sum_{grama_n \in S} Conteo_{Coincidencia}(grama_n)}{\sum_{S \in (ResúmenesDeReferencia)} \sum_{grama_n \in S} Conteo(grama_n)} \qquad (4\text{-}1)$$

Donde n determina la longitud del n-grama, $grama_n$ y $Conteo_{Coincidencia}(grama_n)$ es el máximo número de n-gramas coincidentes entre un resumen candidato y un conjunto de resúmenes de ideales.

4.2.2 ROUGE-SU4

ROUGE-S está basado en estadísticas de co-ocurrencias de bigramas-skip (pares de palabras) y miden la superposición de bigramas-skip entre un resumen candidato y un conjunto de resúmenes de referencia.

Dada una oración X perteneciente al resumen ideal, de longitud m, y una oración Y del resumen a evaluar, de longitud n, el cálculo de la medida-F basada en bigramas-skip corresponde al cálculo de la métrica ROUGE-S como se muestra en las ecuaciones (4-2), (4-3) y (4-4):

$$R_{skip2} = \frac{SKIP2(X,Y)}{C(m,2)} \quad (4\text{-}1)$$

$$P_{skip2} = \frac{SKIP2(X,Y)}{C(n,2)} \quad (4\text{-}2)$$

$$F_{skip2} = \frac{(1+\beta^2)R_{skip2}P_{skip2}}{R_{skip2}+\beta^2 P_{skip2}} \quad (4\text{-}3)$$

Donde $SKIP2(X,Y)$ es el número de coincidencias de bigramas-skip entre X e Y, β es el encargado de verificar la importancia relativa de P_{skip2} y R_{skip2}, y C es la función de combinación que calcula la cantidad de bigramas-skip presentes en una oración.

La versión extendida se llama ROUGE-SU. Se puede obtener ROUGE-SU a partir de ROUGE-S añadiendo el conteo de unigramas como unidades y un marcador de comienzo de oración al comienzo de las oraciones a evaluar y de referencia.

4.3 Resultados y análisis

La implementación de los tres algoritmos para la generación automática de resúmenes empleados en el presente trabajo de grado (véase sección 3.2, 3.3 y 3.4) se realizó con el lenguaje de programación C# de la plataforma .NET.

4.3.1 Comparación de diferentes métodos de pre-procesamiento de texto

A continuación, se compara el rendimiento de los algoritmos para la generación automática de resúmenes con el método de representación basado en Word2Vec (véase secciones 3.2, 3.3 y 3.4) y el rendimiento que obtuvo la antigua forma de representación de los documentos basado en TDM.

Cabe recordar primero, que los métodos que utilizan Word2Vec no elaboran la matriz TF-ISF si no que crean una nueva matriz mediante las estrategias descritas en el capítulo 3. La comparación se realiza entre los diferentes métodos de pre-procesamiento del texto como lo son:

- TDM con stemming: Es el método de representación de documentos con el cual se han trabajado anteriormente todos los algoritmos de generación automática de resúmenes en el Grupo de I+D en Tecnologías de la Información (GTI) del Departamento de Sistemas de la Universidad del Cauca.

- Word2Vec con stemming: se realizó un procesamiento del texto que contiene la eliminación de palabras vacías, la conversión a minúsculas y stemming; para la conformación de la matriz de representación a partir de los vectores entrenados con Word2Vec con la misma técnica de pre-procesamiento.

- Word2Vec de 3 millones: se realizó un procesamiento básico del texto de los documentos que comprende la eliminación de palabras vacías y la conversión a minúsculas con el fin de trabajar con los tres millones de vectores previamente entrenados por Google (véase sección 3.1.1).

- Word2Vec con procesamiento de texto simple: con el texto de los documentos de DUC se entrenaron los vectores de palabras del vocabulario con Word2Vec luego de un procesamiento del texto que consistió en eliminación de palabras vacías y la conversión a minúsculas. Por último, se realizó la generación automática de los resúmenes con el mismo pre-procesamiento para obtener las mismas palabras entrenadas.

- Word2Vec con lematización y sinónimos: se hizo un procesamiento del texto que consiste en eliminación de palabras vacías, la conversión a minúsculas, lematización y búsqueda de sinónimos y a partir de ahí se formó la matriz de representación con los vectores entrenados con Word2Vec mediante el mismo procesamiento.

- TDM con lematización y sinónimos: para hacer una justa comparación también se formó la matriz TF-ISF y demás componentes de la TDM a partir del texto procesado mediante la eliminación de palabras vacías, la conversión a minúsculas, lematización y búsqueda de sinónimos.

La fórmula de mejora relativa da una comparación de resultados de los métodos evaluados y se presenta en la ecuación (4-5).

$$\frac{(método\ propuesto - otro\ método)}{otro\ método}\ x\ 100 \qquad (4\text{-}5)$$

La información de los resultados de la evaluación se organiza dependiendo de la medida de ROUGE-2 debido a que se encontró como la medida más estable. También cabe recalcar que el algoritmo de LSA con TDM con lematización y sinónimos supera el tiempo de pruebas. Por lo cual, se decidió descartar dichas pruebas. Además, como el tiempo de

ejecución del algoritmo MCMR es muy grande, se hicieron solamente las pruebas más prometedoras; tomando como prometedoras las tres mejores presentadas por LexRank en cada dataset.

4.3.2 Evaluación con DUC2005

En la **Tabla 3** se muestran los resultados de los mejores experimentos registrados en la evaluación de ROUGE-2 por cada algoritmo y método de representación que se utilizaron para los documentos de DUC2005. En negrita se presenta el mejor resultado.

Tabla 3 Evaluación con ROUGE de los métodos propuestos para DUC2005.

Algoritmo	Métodos	ROUGE-1	P	ROUGE-2	P	ROUGE-SU4	P
LexRank	**Word2Vec con lematización y sinónimos**	0,37778	**4**	**0,07711**	1	0,1338	**3**
LexRank	TDM con stemming	0,3862	1	0,077	2	0,13529	2
LexRank	Word2Vec con stemming	0,38523	2	0,0764	3	0,13663	1
LexRank	TDM con lematización y sinónimos	0,38247	3	0,0743	4	0,1336	4
LexRank	Word2Vec con procesamiento de texto simple	0,37491	7	0,07363	5	0,13143	5
MCMR	TDM con stemming	0,37573	6	0,06921	6	0,12781	7
LexRank	Word2Vec de 3 millones	0,37694	5	0,0679	7	0,12876	6
MCMR	Word2Vec con lematización y sinónimos	0,37036	8	0,06722	8	0,12535	8
MCMR	Word2Vec con stemming	0,36880	9	0,06485	9	0,12336	9
LSI	Word2Vec con stemming	0,35793	10	0,06174	10	0,11844	10
LSI	Word2Vec con lematización y sinónimos	0,35623	11	0,06085	11	0,11792	11
LSI	Word2Vec con procesamiento de texto simple	0,34079	12	0,05683	12	0,1101	13
LSI	Word2Vec de 3 millones	0,33807	13	0,05567	13	0,11027	12
LSI	TDM con stemming	0,33594	14	0,04735	14	0,10455	14

Tabla 4 Mejora en % del mejor método en DUC2005 (ver Tabla 3).

Algoritmo	Métodos	Mejora
LexRank	**Word2Vec con lematización y sinónimos**	**0**
LexRank	TDM con stemming	0,14%
LexRank	Word2Vec con stemming	0,92%
LexRank	TDM con lematización y sinónimos	3,78%
LexRank	Word2Vec con procesamiento de texto simple	4,72%
MCMR	TDM con stemming	11,41%
LexRank	Word2Vec de 3 millones	13,56%
MCMR	Word2Vec con lematización y sinónimos	14,71%
MCMR	Word2Vec con stemming	18,9%
LSI	Word2Vec con stemming	24,89%
LSI	Word2Vec con lematización y sinónimos	26,72%
LSI	Word2Vec con procesamiento de texto simple	35,68%
LSI	Word2Vec de 3 millones	38,51%
LSI	TDM con stemming	62,85%

Como se observa en la **Tabla 4**, el algoritmo LexRank y la representación basada en Word2Vec con lematización y sinónimos presenta el mejor rendimiento en comparación con los demás métodos. Además, se muestra que cuando Word2Vec es entrenado con los documentos de DUC (Word2Vec con procesamiento de texto simple, Word2Vec con

lematización y sinónimos y Word2Vec con stemming) se presentaron resultados superiores a la representación basada en Word2Vec de 3 millones a pesar de haber sido entrenado con un gran volumen de información.

4.3.3 Evaluación con DUC2006

En la **Tabla 5** se muestran los resultados de los mejores experimentos registrados en la evaluación de ROUGE-2 por cada algoritmo y método de representación que se utilizaron para los documentos de DUC2006. La columna P representa la posición que ocupa dependiendo de cada evaluación. En negrita se presenta el mejor resultado.

Tabla 5 Evaluación con ROUGE de los métodos propuestos para DUC2006.

Algoritmo	Métodos	ROUGE-1	P	ROUGE-2	P	ROUGE-SU4	P
LexRank	**Word2Vec con procesamiento de texto simple**	0,39552	4	**0,08827**	1	0,14334	2
LexRank	Word2Vec con lematización y sinónimos	0,39698	3	0,08686	2	0,14261	4
LexRank	Word2Vec con stemming	0,39481	5	0,08619	3	0,14208	5
LexRank	TDM con stemming	0,40311	2	0,0857	4	0,14376	1
LexRank	TDM con lematización y sinónimos	0,40424	1	0,08529	5	0,14294	3
LexRank	Word2Vec de 3 millones	0,39445	6	0,08528	6	0,14135	6
MCMR	Word2Vec con procesamiento de texto simple	0,38978	8	0,08087	7	0,13704	7
LSI	Word2Vec con lematización y sinónimos	0,38588	9	0,07946	8	0,13372	9
MCMR	TDM con stemming	0,39233	7	0,07907	9	0,13553	8
LSI	Word2Vec con stemming	0,38439	10	0,07736	10	0,13288	10
LSI	Word2Vec de 3 millones	0,37707	11	0,07281	11	0,12907	11
LSI	Word2Vec con procesamiento de texto simple	0,37527	12	0,0725	12	0,12819	12
MCMR	Word2Vec con lematización y sinónimos	0,3726	13	0,06562	13	0,12401	13
LSI	TDM con stemming	0,37	14	0,06348	14	0,12097	14

Tabla 6 Mejora en % del mejor método en DUC2006 (ver Tabla 5).

Algoritmo	Métodos	Mejora
LexRank	**Word2Vec con procesamiento de texto simple**	**0**
LexRank	Word2Vec con lematización y sinónimos	1,62%
LexRank	Word2Vec con stemming	2,41%
LexRank	TDM con stemming	2,99%
LexRank	TDM con lematización y sinónimos	3,49%
LexRank	Word2Vec de 3 millones	3,50%
MCMR	Word2Vec con procesamiento de texto simple	9,15%
LSI	Word2Vec con lematización y sinónimos	11,08%
MCMR	TDM con stemming	11,63%
LSI	Word2Vec con stemming	14,1%
LSI	Word2Vec de 3 millones	21,23%
LSI	Word2Vec con procesamiento de texto simple	21,75%
MCMR	Word2Vec con lematización y sinónimos	34,51%
LSI	TDM con stemming	39,05%

Como se observa en la **Tabla 6**, el algoritmo LexRank y la representación basada en Word2Vec con procesamiento de texto simple muestra el mejor rendimiento en comparación con los demás métodos. También se evidencia mejor rendimiento de las

representaciones basadas en Word2Vec con respecto a las basadas en TDM de acuerdo a cada algoritmo. Además, se muestra que cuando Word2vec es entrenado con los documentos de DUC (Word2Vec con procesamiento de texto simple, Word2Vec con lematización y sinónimos y Word2Vec con stemming) se presentaron resultados superiores a la representación basada en Word2Vec de 3 millones a pesar de haber sido entrenado con un gran volumen de información.

4.3.4 Evaluación con DUC2007

En la **Tabla 7** se muestran los resultados de los mejores experimentos registrados en la evaluación de ROUGE-2 por cada algoritmo y método de representación que se utilizaron para los documentos de DUC2007. La columna P representa la posición que ocupa dependiendo de cada evaluación. En negrita se presenta el mejor.

Tabla 7 Evaluación con ROUGE de los métodos propuestos para DUC2007.

Algoritmo	Métodos	ROUGE-1	P	ROUGE-2	P	ROUGE-SU4	P
LexRank	**Word2Vec con procesamiento de texto simple**	0,41801	**2**	**0,10584**	1	0,15809	**2**
LexRank	TDM con stemming	0,41521	5	0,10525	2	0,15538	5
LexRank	Word2Vec de 3 millones	0,42335	1	0,10378	3	0,16	1
LexRank	Word2Vec con stemming	0,41579	3	0,10374	4	0,15766	3
LexRank	Word2Vec con lematización y sinónimos	0,41619	4	0,10247	5	0,15656	4
LexRank	TDM con lematización y sinónimos	0,41154	6	0,09929	6	0,15164	6
MCMR	Word2Vec con procesamiento de texto simple	0,40694	8	0,09821	7	0,1514	7
MCMR	TDM con stemming	0,41002	7	0,09557	8	0,15027	8
MCMR	Word2Vec con stemming	0,40656	9	0,09459	9	0,14944	9
LSI	Word2Vec con lematización y sinónimos	0,40468	10	0,09254	10	0,14581	10
LSI	Word2Vec de 3 millones	0,39948	11	0,08719	11	0,14229	11
LSI	Word2Vec con procesamiento de texto simple	0,39386	12	0,0862	12	0,13904	12
LSI	Word2Vec con stemming	0,38439	14	0,07736	13	0,13288	14
LSI	TDM con stemming	0,39027	13	0,07711	14	0,13337	13

Tabla 8 Mejora en % del mejor método en DUC2007 (ver Tabla 7).

Algoritmo	Métodos	Mejora
LexRank	**Word2Vec con procesamiento de texto simple**	**0**
LexRank	TDM con stemming	0,56%
LexRank	Word2Vec de 3 millones	1,98%
LexRank	Word2Vec con stemming	2,02%
LexRank	Word2Vec con lematización y sinónimos	3,28%
LexRank	TDM con lematización y sinónimos	6,59%
MCMR	Word2Vec con procesamiento de texto simple	7,76%
MCMR	TDM con stemming	10,74%
MCMR	Word2Vec con stemming	11,89%
LSI	Word2Vec con lematización y sinónimos	14,37%
LSI	Word2Vec de 3 millones	21,39%
LSI	Word2Vec con procesamiento de texto simple	22,78%
LSI	Word2Vec con stemming	36,81%
LSI	TDM con stemming	37,25%

Como se observa en la **Tabla 8**, al igual que en DUC2006, el algoritmo LexRank y la representación basada en Word2Vec con procesamiento de texto simple muestra el mejor rendimiento en comparación con los demás métodos. Además, se observa un mejor rendimiento de las representaciones basadas en Word2Vec con respecto a las basadas en TDM de acuerdo a cada algoritmo. La representación basada en el Word2Vec de 3 millones tuvo un desempeño aceptable pero aun así se obtienen mejores resultados cuando los vectores de palabras son entrenados con los propios documentos de DUC. A su vez la representación basada en el Word2Vec de 3 millones presentó mejores calificaciones en ROUGE-1 Y ROUGE-SU4 cuando se aplicaron umbrales altos (mayores a 0,75).

Nótese que tanto en DUC2005, como en DUC2006 y DUC2007; el algoritmo LSI con la nueva representación basada en Word2Vec supera en todos los casos a dicho algoritmo con la representación tradicional (TDM).

También se evidencia que para DUC2006 y DUC2007; se lograron los mejores resultados cuando no se redujo el tamaño del vocabulario

4.3.5 Comparación con otros algoritmos del estado del arte

Para DUC2005 se seleccionó como mejor algoritmo a LexRank con Word2Vec con lematización y sinónimos y se estableció en la **Tabla 9** un ranking para compararlo con los algoritmos del estado del arte. Para DUC2006 Y DUC2007 se seleccionó como mejor algoritmo a LexRank con Word2Vec con procesamiento de texto simple y se determina su posición frente a los mejores algoritmos del estado del arte mediante un ranking descrito en la **Tabla 11** y **Tabla 13** respectivamente. Para DUC2005 y DUC2006 se hace la comparación de los resultados presentados en este trabajo de grado y los resultados de los algoritmos descritos en [43]. Y para DUC2007 de acuerdo a los algoritmos expuestos en [42]. La columna P representa la posición que ocupa dependiendo de cada evaluación. En negrita se presenta el mejor. La **Tabla 10**, **Tabla 12** y **Tabla 14** presentan la mejora relativa del mejor algoritmo del estado del arte para DUC2005, DUC2006 y DUC2007 respectivamente calculada mediante la ecuación (**4-5**).

Tabla 9 Comparación de resultados de DUC2005 con otros algoritmos del estado del arte. Adaptado de [43].

Método	ROUGE-1	P	ROUGE-2	P	ROUGE-SU4	P
LexRank + Word2Vec con lematización y sinónimos	0,37778	9	0,07711	7	0,1338	8
MA-MultiSumm	**0,4001**	**1**	**0,0868**	**1**	**0,1434**	**1**
DESAMC+DocSum	0,3937	2	0,0822	2	0,1418	2
PLSA	0,3913	3	0,0811	3	0,1389	5
LFIPP	0,3905	4	0,0804	4	0,1403	3
MCMR	0,3891	5	0,079	6	0,1392	4
iRANK	0,388	6	0,0802	5	0,1373	6
HybHSum	0,3812	8	0,0749	9	0,1354	7
SVR	0,3849	7	0,0757	8	0,1335	9
TMR	0,3775	10	0,0715	12	0,1304	12
LEX	0,376	11	0,0735	11	0,1316	11

HierSum	0,3753	12	0,0745	10	0,1324	10
SNMF+SLSS	0,3501	14	0,0604	14	0,1172	14
Centroid	0,3535	13	0,0638	13	0,1198	13
MMR	0,3479	15	0,0601	15	0,1134	15

Tabla 10 Mejora relativa del mejor algoritmo del estado del arte para DUC2005.

Método	ROUGE-1	ROUGE-2	ROUGE-SU4
MA-MultiSumm	0%	0%	0%
DESAMC+DocSum	1,62%	5,59%	1,12%
PLSA	2,24%	7,02%	3,23%
LFIPP	2,45%	7,96%	2,2%
iRANK	3,11%	8,22%	4,44%
MCMR	2,82%	9,87%	3,01%
LexRank + Word2Vec con lematización y sinónimos	5,9%	12,56%	7,17%
SVR	3,94%	14,66%	7,41%
HybHSum	4,95%	15,88%	5,9%
HierSum	6,6%	16,51%	8,3%
LEX	6,4%	18,09%	8,96%
TMR	5,98%	21,39%	9,96%
Centroid	13,18%	36,05%	19,69%
SNMF+SLSS	14,28%	43,7%	22,35%
MMR	15%	44,42%	26,45%

La **Tabla 9** y la **Tabla 10** muestran que el mejor algoritmo sigue siendo MA-MultiSumm para la generación automática de resúmenes de múltiples documentos extractivos y genéricos en DUC2005.

Tabla 11 Comparación de resultados de DUC2006 con otros algoritmos del estado del arte. Adaptado de [43].

Método	ROUGE-1	P	ROUGE-2	P	ROUGE-SU4	P
LexRank + Word2Vec con procesamiento de texto simple	0,39552	12	0,08827	11	0,14334	11
MA-MultiSumm	0,4195	5	0,0986	2	0,1526	4
DESAMC+DocSum	**0,43450**	**1**	**0,09890**	**1**	**0,1569**	**1**
PLSA	0,43280	2	0,09700	3	0,1557	2
LFIPP	0,42090	4	0,09340	4	0,1534	3
MCMR	0,4184	6	0,0928	5	0,1512	5
iRANK	0,40320	8	0,09120	9	0,145	9
HybHSum	0,43000	3	0,09100	10	0,151	6
SVR	0,40180	10	0,09260	6	0,1485	8
TMR	0,40630	7	0,09130	7	0,1504	7
LEX	0,40300	9	0,09130	8	0,1449	10
HierSum	0,40100	11	0,08600	12	0,143	12
SNMF+SLSS	0,39550	13	0,08550	13	0,1429	13
Centroid	0,38070	14	0,07850	14	0,133	14
MMR	0,37160	15	0,07570	15	0,1308	15

Tabla 12 Mejora relativa del mejor algoritmo del estado del arte para DUC2006.

Método	ROUGE-1	ROUGE-2	ROUGE-SU4
DESAMC+DocSum	0%	0%	0%
MA-MultiSumm	3,57%	0,3%	2,81%

PLSA	0,39%	1,95%	0,77%
LFIPP	3,23%	5,88%	2,28%
MCMR	3,84%	6,57%	3,76%
SVR	8,13%	6,8%	5,65%
TMR	6,94%	8,32%	4,32%
LEX	7,81%	8,32%	8,28%
iRANK	7,76%	8,44%	8,20%
HybHSum	1,04%	8,68%	3,9%
LexRank + Word2Vec con procesamiento de texto simple	9,85%	12,04%	9,4%
HierSum	8,35%	15%	9,72%
SNMF+SLSS	9,86%	15,67%	9,79%
Centroid	14,13%	25,98%	17,96%
MMR	16,92%	30,64%	19,95%

La **Tabla 11** y la **Tabla 12** muestran que el mejor algoritmo sigue siendo DESAMC+DocSum para la generación automática de resúmenes de múltiples documentos extractivos y genéricos en DUC2006.

Tabla 13 Comparación de resultados de DUC2007 con otros algoritmos del estado del arte. Adaptado de [42].

Método	ROUGE-1	P	ROUGE-2	P	ROUGE-SU4	P
LexRank + Word2Vec con procesamiento de texto simple	0,41801	8	0,10584	8	0,15809	8
MCMR	0,46850	3	0,12210	3	0,1753	4
MMR	0,37980	11	0,06920	11	0,1124	11
PPRSum	0,46730	4	0,11950	4	0,171	5
AdaSum	0,46540	5	0,11720	5	0,1692	6
PLSA-JS	0,45400	6	0,11680	6	0,1768	3
SVR	0,44670	7	0,11330	7	0,1652	7
PNR2	0,38670	10	0,08950	9	0,1291	9
SVM	0,39050	9	0,08670	10	0,1284	10
DDS	**0,47190**	**1**	**0,12370**	**1**	**0,1836**	**1**
CDS	0,47080	2	0,12280	2	0,1814	2

Tabla 14 Mejora relativa del mejor algoritmo del estado del arte para DUC2007.

Método	ROUGE-1	ROUGE-2	ROUGE-SU4
DDS	0%	0%	0%
CDS	0,23%	0,73%	1,21%
MCMR	0,72%	1,31%	4,73%
PPRSum	0,98%	3,51%	7,36%
AdaSum	1,39%	5,54%	8,51%
PLSA-JS	3,94%	5,9%	3,84%
SVR	5,64%	9,17%	11,13%
LexRank + Word2Vec con procesamiento de texto simple	12,89%	16,87%	16,13%
PNR2	22,03%	38,21%	42,21%
SVM	20,84%	42,67%	42,99%
MMR	24,24%	78,75%	63,34%

La **Tabla 13** y la **Tabla 14** muestran que el mejor algoritmo sigue siendo DDS para la generación automática de resúmenes de múltiples documentos extractivos y genéricos en DUC2007.

Nótese que la comparación con algoritmos del estado del arte se realiza también con los resultados reportados por el algoritmo MCMR. Sin embargo, no se hizo posible replicar los mismos resultados debido a la falta de información por parte de los autores y la no disponibilidad del código fuente del mismo. Lo cual genera incertidumbre en la veracidad de algunos resultados reportados.

5 CONCLUSIONES, TRABAJO FUTURO Y RECOMENDACIONES

5.1 Conclusiones

Como conclusión general se obtiene que tanto en DUC2006 como DUC2007 se encontró una representación basada en Word2Vec que logra obtener mejores resúmenes extractivos genéricos de múltiples documentos según las medidas ROUGE para los algoritmos LexRank, LSI y MCMR. En DUC2005 se encontró una representación basada en Word2Vec que obtiene mejores resúmenes según las medidas ROUGE para los algoritmos LexRank y LSI. Para MCMR fue incierto debido a que no se pudo realizar el número total de pruebas.

Pre-procesamientos de texto como stemming, lematización y la búsqueda de sinónimos permitieron reducir el vocabulario de los documentos de DUC puesto que a palabras con el mismo significado se les pudo asignar un mismo vector. Esto ayudó a que ciertas frases con significados similares, pero con vectores distantes pudieran tener una representación vectorial cercana. Lo que permitió encontrar buenos resultados con DUC2005. Sin embargo, para DUC2006 y DUC2007 se alcanzaron mejores resultados cuando no se redujo el vocabulario.

De los tres algoritmos estudiados (LexRank, LSI y MCMR), LexRank presentó las mejores calificaciones en los resúmenes generados, para DUC2005, DUC2006 y DUC2007; tanto con representaciones basadas en Word2Vec como representaciones basadas en TDM. Además, el algoritmo LSI con la nueva representación basada en Word2Vec supera en todos los casos a dicho algoritmo con la representación tradicional (TDM).

Cuando se ejecutaron representaciones basadas en Word2Vec y se realizó el cálculo del vector de cada frase mediante la suma y el promedio de los vectores, no se presentaron diferencias significativas en la calidad de los resúmenes excepto con el algoritmo LSI; para el cual siempre se obtuvieron mejores resúmenes cuando se aplicó promedio a los vectores de frase.

Al hacer la comparación entre los mejores resultados obtenidos con los algoritmos desarrollados y los demás algoritmos del estado del arte, se determinó que para la generación automática de resúmenes extractivos genéricos de múltiples documentos para DUC2005 el mejor algoritmo sigue siendo MA-MultiSumm; para DUC2006 el mejor algoritmo sigue siendo DESAMC+DocSum y para DUC2007 el mejor algoritmo sigue siendo DDS.

Para el desarrollo de algunos algoritmos del estado del arte se presenta la dificultad de repetir los experimentos y los resultados previamente publicados; debido a falta de información detallada en los artículos publicados (problemas asociados a la repetibilidad de los experimentos) y a la no disponibilidad de códigos fuente. Lo anterior impide avances más significativos en el área de investigación, debido a que se realizan esfuerzos en la obtención de los mismos resultados en lugar de realizar investigaciones a partir del trabajo ya desarrollado.

Para la elaboración de la matriz de representación basada en Word2Vec se deben realizar los diferentes cálculos de entrenamiento de los vectores de palabras y la obtención de los vectores de frases, lo cual conlleva a un consumo significativo de recursos computacionales. Por lo tanto, se hace necesario el uso de mecanismos que ayuden a aumentar el rendimiento computacional para mejorar la eficiencia en tiempo del proceso de generación automática de resúmenes.

5.2 Trabajo Futuro

Adaptar la representación basada en Word2Vec a los otros algoritmos de generación automática de resúmenes extractivos y genéricos de múltiples documentos del estado del arte. Para DUC2005 Y DUC2006 los algoritmos descritos en DESAMC+DocSum [43] y para DUC2007 los algoritmos reseñados en DDS [42] (debido a que a la fecha son los más representativos).

Desarrollar un método formal para la evaluación de la calidad de los vectores de palabras entrenados por Word2Vec. Debido a que la calidad de los mismos puede variar dependiendo del corpus de entrenamiento, el número de iteraciones y demás parámetros utilizados en el entrenamiento.

Implementar otros algoritmos metaheurísticos para el problema de la generación automática de resúmenes extractivos y genéricos de múltiples documentos. De tal modo que puedan utilizarse diferentes representaciones. Dado que en el Grupo de I+D en Tecnologías de la Información (GTI) del Departamento de Sistemas de la Universidad del Cauca se han implementado algoritmos para la resolución de diversos problemas presentando buenos resultados, se recomienda utilizar estos algoritmos en el área de la generación automática de resúmenes.

5.3 Recomendaciones

Proveer al programa de Ingeniería de Sistemas con equipos de altas prestaciones para desarrollar proyectos de este tipo.

6 BIBLIOGRAFÍA

[1] M. Mendoza, S. Bonilla, C. Noguera, C. Cobos, and E. León, "Extractive single-document summarization based on genetic operators and guided local search," *Expert Systems with Applications,* vol. 41, pp. 4158-4169, 2014.

[2] M. E. Mendoza, "Generación automática de resúmenes extractivos de múltiples documentos basada en algoritmos meméticos," *Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial, Universidad Nacional de Colombia, Bogotá D.C., Colombia,* 2015.

[3] A. Nenkova and K. McKeown, "Automatic Summarization," *Foundations and Trends in Information Retrieval,* vol. 5, pp. 103-233, 2011.

[4] M. Mendoza and E. Leon, "Una revisión de la generación automática de resúmenes extractivos," *UIS Ingenierías,* vol. 12, pp. 7-27, 2015.

[5] S. Soumya, G. Kumar, R. Naseem, and S. Mohan, "Automatic Text Summarization," in *Computational Intelligence and Information Technology*. vol. 250, V. Das and N. Thankachan, Eds., ed: Springer Berlin Heidelberg, 2011 pp. 787-789.

[6] J. Steinberger and K. Ježek, "Text Summarization: An Old Challenge and New Approaches," in *Foundations of Computational, IntelligenceVolume 6*. vol. 206, A. Abraham, A.-E. Hassanien, A. Leon F. de Carvalho, and V. Snášel, Eds., ed: Springer Berlin Heidelberg, 2009, pp. 127-149.

[7] S. Suneetha, "Automatic Text Summarization: The Current State of the art," *International Journal of Science and Advanced Technology,* vol. 1, pp. 283-293, 2011.

[8] E. Lloret and M. Palomar, "Text summarisation in progress: a literature review," *Artificial Intelligence Review,* vol. 37, pp. 1-41, 2012/01/01 2012.

[9] G. Ravindra, N. Balakrishnan, and K. R. Ramakrishnan, "Multi-document Automatic Text Summarization Using Entropy Estimates," in *SOFSEM 2004: Theory and Practice of Computer Science*. vol. 2932, P. Emde Boas, J. Pokorný, M. Bieliková, and J. Štuller, Eds., ed: Springer Berlin Heidelberg, 2004, pp. 289-300.

[10] D. R. Radev, H. Jing, M. Stys, and D. Tam, "Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user studies," *Inf. Process. Manage.,* vol. 40, pp. 919-938, 2004.

[11] D. Wang, S. Zhu, T. Li, Y. Chi, and Y. Gong, "Integrating clustering and multi-document summarization to improve document understanding," presented at the Proceedings of the 17th ACM conference on Information and knowledge management, Napa Valley, California, USA, 2008.

[12] L. Hennig, "Topic-based Multi-Document Summarization with Probabilistic Latent Semantic Analysis," in *RANLP*, 2009, pp. 144-149.

[13] D. Wang, S. Zhu, T. Li, and Y. Gong, "Multi-Document Summarization using Sentence-based Topic Models," in *ACL/IJCNLP (Short Papers)*, 2009, pp. 297-300.

[14] D. Wang and T. Li, "Weighted consensus multi-document summarization," *Inf. Process. Manage.,* vol. 48, pp. 513-523, 2012.

[15] D. Wang and T. Li, "Many are better than one: improving multi-document summarization via weighted consensus," in *SIGIR*, 2010, pp. 809-810.

[16] T. Li and C. H. Q. Ding, "Weighted Consensus Clustering," in *SDM*, 2008, pp. 798-809.

[17] D. Bollegala, N. Okazaki, and M. Ishizuka, "A Bottom-Up Approach to Sentence Ordering for Multi-Document Summarization," in *ACL*, 2006.

[18] A. Çelikyilmaz and D. Hakkani-Tür, "A Hybrid Hierarchical Model for Multi-Document Summarization," in *ACL*, 2010, pp. 815-824.

[19] G. Erkan and D. R. Radev, "LexRank: graph-based lexical centrality as salience in text summarization," *J. Artif. Int. Res.,* vol. 22, pp. 457-479, 2004.
[20] D. Ai, Y. Zheng, and D. Zhang, "Automatic text summarization based on latent semantic indexing," *Artificial Life and Robotics,* vol. 15, pp. 25-29, 2010// 2010.
[21] R. M. Alguliev, R. M. Aliguliyev, M. S. Hajirahimova, and C. A. Mehdiyev, "MCMR: Maximum coverage and minimum redundant text summarization model," *Expert Systems with Applications,* vol. 38, pp. 14514-14522, 2011.
[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv:1301.3781v3 [cs.CL],* 2013.
[23] H. Yang, Q. Hu, and L. He, "Learning Topic-Oriented Word Embedding for Query Classification," in *Advances in Knowledge Discovery and Data Mining: 19th Pacific-Asia Conference, PAKDD 2015, Ho Chi Minh City, Vietnam, May 19-22, 2015, Proceedings, Part I*, T. Cao, E.-P. Lim, Z.-H. Zhou, T.-B. Ho, D. Cheung, and H. Motoda, Eds., ed Cham: Springer International Publishing, 2015, pp. 188-198.
[24] B. Jiang, E. Xun, and J. Qi, "A Domain Independent Approach for Extracting Terms from Research Papers," in *Databases Theory and Applications: 26th Australasian Database Conference, ADC 2015, Melbourne, VIC, Australia, June 4-7, 2015. Proceedings*, A. M. Sharaf, A. M. Cheema, and J. Qi, Eds., ed Cham: Springer International Publishing, 2015, pp. 155-166.
[25] N. Rekabsaz, R. Bierig, M. Lupu, and A. Hanbury, "Toward Optimized Multimodal Concept Indexing," in *Semantic Keyword-based Search on Structured Data Sources: First COST Action IC1302 International KEYSTONE Conference, IKC 2015, Coimbra, Portugal, September 8-9, 2015. Revised Selected Papers*, J. Cardoso, F. Guerra, G.-J. Houben, M. A. Pinto, and Y. Velegrakis, Eds., ed Cham: Springer International Publishing, 2015, pp. 141-152.
[26] N. Khosla and V. Venkataraman, "Learning Sentence Vector Representations to Summarize Yelp Reviews," *NLP Group, Stanford University,* 2015.
[27] M. Kageback, O. Mogren, N. Tahmasebi, and D. Dubhashi, "Extractive summarization using continuous vector space models," *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC),* pp. 31-39, 2014.
[28] J. Steinberger and K. Ježek, "Automatic Text Summarization (The state of the art 2007 and new challenges)," in *Znalosti*, ed: Václav Snášel, 2008, pp. 1-12.
[29] D. Das and A. F. T. Martins, "A Survey on Automatic Text Summarization," pp. 1-31, 2007.
[30] C. D. Manning, P. Raghavan, and H. Schütze, *An introduction to information retrieval*. Cambridge: Cambridge University Press, 2009.
[31] D. Gillick, "Sentence boundary detection and the problem with the U.S," presented at the Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers, Boulder, Colorado, 2009.
[32] L. u. t. A. L. T. A. S. Foundation. (2011-2012). *Lucene*. Available: http://lucene.apache.org/
[33] U. Princeton. *WordNet*. Available: https://wordnet.princeton.edu/
[34] U. o. Pennsylvania. (2015). *NLTK Project*. Available: http://www.nltk.org/
[35] M. Gambhir and V. Gupta, "Recent automatic text summarization techniques: a survey," *Artificial Intelligence Review,* pp. 1-66, 2016// 2016.
[36] C. Yan-Min, W. Xiao-Long, and L. Bing-Quan, "Multi-document summarization based on lexical chains," in *2005 International Conference on Machine Learning and Cybernetics*, 2005, pp. 1937-1942 Vol. 3.

[37] J. Steinberger and M. Křišťan, "Lsa-based multi-document summarization," *Proceedings of 8th International Workshop on Systems and Control,* vol. 7, 2007.
[38] M. Xiao-Chen, Y. Gui-Bin, and M. Liang, "Multi-Document Summarization Using Clustering Algorithm," in *Intelligent Systems and Applications, 2009. ISA 2009. International Workshop on*, 2009, pp. 1-4.
[39] M. M. Ali, M. K. Ghosh, and A. Al-Mamun, "Multi-document Text Summarization: SimWithFirst Based Features and Sentence Co-selection Based Evaluation," in *Future Computer and Communication, 2009. ICFCC 2009. International Conference on*, 2009, pp. 93-96.
[40] X. Cai and W. Li, "A spectral analysis approach to document summarization: Clustering and ranking sentences simultaneously," *Information Sciences,* vol. 181, pp. 3816-3827, 2011.
[41] R. M. Alguliev, R. M. Aliguliyev, and C. A. Mehdiyev, "Sentence selection for generic document summarization using an adaptive differential evolution algorithm," *Swarm and Evolutionary Computation,* vol. 1, pp. 213-222, 2011.
[42] R. M. Alguliev, R. M. Aliguliyev, and N. R. Isazade, "CDDS: Constraint-driven document summarization models," *Expert Systems with Applications,* vol. 40, pp. 458-465, 2013.
[43] R. M. Alguliev, R. M. Aliguliyev, and N. R. Isazade, "DESAMC+DocSum: Differential evolution with self-adaptive mutation and crossover parameters for multi-document summarization," *Knowledge-Based Systems,* vol. 36, pp. 21-38, 2012.
[44] R. M. Aliguliyev, "CLUSTERING TECHNIQUES AND DISCRETE PARTICLE SWARM OPTIMIZATION ALGORITHM FOR MULTI-DOCUMENT SUMMARIZATION," *Computational Intelligence,* vol. 26, pp. 420-448, 2010.
[45] W. Meng, X. Wang, L. Chungui, and Z. Zengfang, "Multi-document Summarization Based on Word Feature Mining," *Proceedings of the 2008 International Conference on Computer Science and Software Engineering,* pp. 743-746, 2008.
[46] D. Harman and P. Over "The DUC summarization evaluations," *Proceedings of the second international conference on Human Language Technology Research, San Diego, California,* pp. 44-51, 2002.
[47] C.-Y. Lin, "ROUGE: a Package for Automatic Evaluation of Summaries," in *Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004), Barcelona, Spain*, 2004.
[48] Y. Goldberg and O. Levy, "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method," *arXiv:1402.3722v1 [cs.CL],* 2014.
[49] X. Rong, "word2vec Parameter Learning Explained," *arXiv:1411.2738v4,* 2016.
[50] c. d. Wikipedia. (2017, Producto tensorial. Available: https://es.wikipedia.org/w/index.php?title=Producto_tensorial&oldid=94254938
[51] F. Morin and Y. Bengio, "Hierarchical Probabilistic Neural Network Language Model," *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics,* pp. 246-252, 2005.
[52] T. Mikolov, K. Chen, G. Corrado, J. Dean, and I. Sutskever, "Distributed Representations of Words and Phrases and their Compositionality," *Neural Information Processing Systems,* 2013.
[53] Y. Abdullin. (2016). *Word2Vec aplicación de .Net Framework*. Available: https://github.com/eabdullin/Word2Vec.Net
[54] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM,* vol. 18, pp. 613-620, 1975.

[55] A. Singhal, "Modern Information Retrieval: A Brief Overview," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering,* vol. 24, pp. 35-42, 2001.
[56] R. M. Soto, R. A. García-Hernández, Y. Ledeneva, and R. C. Reyes, "Comparación de Tres Modelos de Texto para la Generación Automatica de Resúmenes," *Sociedad Española para el Procesamiento del Lenguaje Natural,* pp. 303-311, 2009.
[57] Y. Gong and X. Liu, "Generic text summarization using relevance measure and latent semantic analysis," presented at the Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, New Orleans, Louisiana, USA, 2001.
[58] S. I. Grossman, "Álgebra Lineal," *Editorial Offset, S.A. de C.V, Durazno No. 1 esq. Ejido, Col. Las Peritas, Tepepan Xochimilco, C.P. 16010 México D.F.,* 2001.
[59] Google, "Word2Vec," 2013.
[60] NIST. (2005, DUC 2005: Task, Documents, and Measures. Available: http://duc.nist.gov/duc2005/tasks.html
[61] NIST. (2005, DUC 2006: Task, Documents, and Measures. Available: http://duc.nist.gov/duc2006/tasks.html
[62] NIST. (2007, DUC 2007: Task, Documents, and Measures. Available: http://duc.nist.gov/duc2007/tasks.html