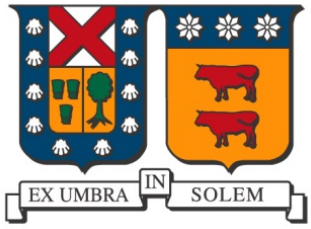
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO DE CHILE - CHILE



"DESARROLLO DE *FRAMEWORK* EN PYTHON PARA COMPARACIÓN DE MODELOS *NEURAL INFORMATION RETRIEVAL*"

ALFREDO IGNACIO SILVA CELPA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Marcelo Mendoza
Profesor Correferente: Ricardo Ñanculef

Septiembre - 2019

DEDICATORIA

Me gustaría dedicarles mi memoria a dos personas: A mi madre, Georgina Celpa, y a mi hermana, Claudia González, por jamás dejar de creer en mí, incluso si yo mismo no lo hice en algún momento, por acompañarme siempre y darme fuerzas para seguir adelante.

AGRADECIMIENTOS

Quiero agradecerle a todos mis amigos y familiares que me dieron palabras de ánimo durante este proceso, pero necesito destacar a dos de ellos: Carlos Andrade y Sebastían Olivares, porque de una u otra forma no me han dejado perder la cordura, al mismo tiempo que han apoyado todas mis locuras desde el momento en que me conocieron, gracias a ambos por tantos años de amistad.

También quiero destacar toda la ayuda brindada por el profesor Marcelo Mendoza, quién me ayudó no sólo con las partes más difíciles de digerir conceptualmente, sino que además me facilitó mucho material y los datos que fueron utilizados para la realización de este trabajo.

RESUMEN

Resumen—

En este trabajo se ha definido, desarrollado y utilizado un marco de trabajo, en el cual se pueden realizar comparaciones de modelos neurales para recuperación de información. En las pruebas experimentales se compararon ocho implementaciones de cinco modelos diferentes, sobre los que se obtuvieron valores para tres métricas de evaluación en aplicaciones de recuperación y extensión de consultas, que muestran en gran medida cómo mejorar un modelo básico de recuperación de información y cuáles son las cosas a tener en consideración para lograrlo. Este trabajo presenta un gran avance en el área, puesto a que debido a al poco tiempo que ha transcurrido desde la creación de estos modelos, todavía no se tiene demasiada información con respecto a su comportamiento o cómo se comparan entre ellos para diversas aplicaciones.

Palabras Clave— Recuperación de información; Representaciones densas; Motores de búsqueda; Marco de trabajo; Aprendizaje profundo.

ABSTRACT

***Abstract*—** In this work a framework has been defined, developed and used, in which comparisons of neural information retrieval models can be made. In the experimental tests, eight implementations of five different models were compared, on which three evaluations metrics were obtained for retrieval and query expansion applications. These tests largely show how to improve a basic information retrieval model and which are the things to consider to achieve it. This work presents a great advance in the area, since due to the short time that has elapsed since the creation of these models, there is still not much information regarding their behavior or how they compare to each other for various applications.

***Keywords*—** *Information retrieval*; *Dense representations*; *Search Engines*; *Framework*; *Deep Learning*.

GLOSARIO

AP - *Average Precision*: Métrica utilizada para evaluar un motor de búsqueda sobre una consulta.
AR@n - *Average Recall*: Métrica utilizada para evaluar un motor de búsqueda sobre varias consultas, pero fijando un rango en los *rankings* entregados.
AWE - *Average Word Embbeding*: Representación promedio de un texto combinando los *embeddings* de sus palabras.
BM25 - *Best Model #25*: Corresponde al mejor modelo (número 25) para calcular relevancia con parámetros ajustados a mano.
BOW - *Bag of Words*: Modelo de texto en que se mantienen los *tokens* que lo componen, pero no necesariamente en un orden específico.
CBOW - *Continuous Bag of Words*: Es un modelo con el que se calculan *embeddings* dependientes del contexto.
Corpus: Colección de documentos.
CRISP-DM - *Cross Industry Standard Process for Data Mining*: Es un método de trabajo específico para abordar problemas centrados fuertemente en los datos.
DCG - *Discounted Cumulative Gain*: Métrica de evaluación para motores de búsqueda que toma en concideración la posición en que aparecen los documentos relevantes.
Deep Learning: es un conjunto de algoritmos y arquitecturas de aprendizaje automático que intenta modelar abstracciones de alto nivel en datos.
Embedding: Representación vectorial densa de palabras.
IDF - *Inverse Document Frecuency*: Corresponde a un parámetro que toma en cuenta la cantidad total de documentos en un corpus y la cantidad de documentos en las que aparece un término específico.
IR - *Information Retrieval*: Área de la informática preocupada de recuperar información según las necesidades de un usuario.
LSTM - *Long short-term memory*: Arquitectura de redes neuronales recurrentes que es capaz de mantener cierto nivel de memoria con respecto a una secuencia de datos ordenados temporalmente.
MAP - *Mean Average Precision*: Promedio de *Average Precision* sobre varias consultas.
NDCG - *Normalized Discounted Cumulative Gain*: Versión normalizada de DCG
NIR - *Neural Information Retrieval*: Sub-área de IR en la que se utilizan métodos y modelos de *Deep Learning*.
SG - Skip-gram: Es un modelo con el que se calculan *embeddings* dependientes del contexo.
Tf - *Term Frecuency*: Parámetro calculado a partir de la frecuencia de un término específico dentro de un texto.
Token: Término procesado representativo de la palabra original.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Recuperación de información es un campo de estudios que ayuda a un usuario a encontrar información que satisfaga sus necesidades. El impacto de las aplicaciones que ha tenido este campo ha sido tan grande que ha moldeado la sociedad actual. En la actualidad, una persona difícilmente encontraría información certera sin utilizar algún motor de búsqueda web, como por ejemplo Google. Desde hace décadas existen modelos altamente efectivos, en los que se utilizan parámetros con valores ajustados empíricamente y representaciones dispersas de palabras y documentos.

Actualmente este problema está intentando ser abordado usando acercamientos basados en *Deep Learning*. Haciendo uso de distintas arquitecturas se pueden calcular representaciones más densas que tienen el potencial de entregar mucha más información sobre ellos. Estos acercamientos se han visto desarrollados en los últimos cinco o seis años cuando mucho, por lo que aún falta mucha experimentación utilizándolos para entender bien sus capacidades y falencias, además de entender cómo se ven afectados por pequeños ajustes en sus parámetros y cómo se pueden usar estos comportamientos para desarrollar modelos todavía más efectivos.

En este trabajo se propone, desarrolla y pone a prueba un *framework* en el que se pueden estandarizar comparaciones de distintos modelos basados en redes neuronales, utilizados en diferentes módulos y etapas de los motores de búsqueda.

Este trabajo está compuesto por cinco capítulos, en el primero de ellos se da una definición más extensa del problema, se identifican los objetivos del trabajo y el alcance que tendrá. En el segundo capítulo se presenta el marco conceptual sobre el que se basan los conceptos utilizados. Luego en el tercer capítulo se define la propuesta concreta del trabajo a realizar y se expone la metodología que se utilizará para su desarrollo. En el cuarto capítulo se expone la implementación y las pruebas realizadas sobre el *framework* junto a sus resultados y discusiones. Finalmente, en el quinto capítulo se entregan los conclusiones del trabajo realizado y los aportes que entrega al área de recuperación de información.

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

En este capítulo se define el contexto y el problema a resolver, además de precisar el objetivo general y los objetivos específicos; finalmente se limita el alcance del trabajo.

1.1. Contexto del problema

Recuperación de información es un campo de estudio cuyo propósito es ayudar al usuario a encontrar información muy específica desde una colección de documentos muy grande, a la cual se le llama Corpus. Lo anterior significa, en palabras simples, encontrar un conjunto ordenado de documentos que es relevante para la consulta del usuario. Resulta pertinente hacer la distinción entre recuperación de información y *Data retrieval* en bases de datos usando consultas tipo SQL, porque en éstas últimas los datos se encuentran altamente estructurados y guardados en tablas relacionales, mientras que la información en los documentos de un corpus no tiene una estructura simple (más allá de la sintáctica dependiente del idioma, para el caso de los documentos de texto), de manera que no existe un lenguaje estructurado para realizar consultas en recuperación de información [Baeza-Yates y Ribeiro-Neto, 1999].

Desde el año 2013 se han desarrollado acercamientos de recuperación de información basados en *Deep Learning*, llamados modelos *Neural Information Retrieval* (NIR), con el propósito general de crear representaciones vectoriales densas (*embeddings*) para las palabras pertenecientes al vocabulario de un corpus. Haciendo uso de dichas representaciones se pueden llevar a cabo las distintas tareas que se realizan en un motor de búsqueda como por ejemplo: recuperación, *ranking*, expansión de consultas, combinaciones de éstas y casos muchos más sofisticados como procesos *end-to-end*, en donde todos los procedimientos del motor son realizados con base en alguna arquitectura o aplicación de *deep learning*.

1.2. Identificación del problema

A pesar que los avances en informática se realizan a pasos agigantados en comparación con otras ciencias, NIR aún es una rama muy reciente de investigación, habiendo transcurrido apenas seis años desde la invención del primer modelo, es un tema actual de estudio sobre el cual aún se están realizando análisis básicos sobre distintos modelos, como por ejemplo el comportamiento de diversas arquitecturas y pruebas de concepto. La mayoría de los estudios realizados se han centrado en comparar NIR con métodos y modelos de representación clásicos de IR, además de comparar los resultados obtenidos entre modelos NIR cuando se les utiliza para alguna de las tareas anteriormente mencionadas, sin embargo no se han establecido corpus de prueba estándar, como lo son CIFAR o MNIST para *Deep Learning*, esto

provoca que sea sumamente difícil comparar resultados obtenidos entre distintas investigaciones, por más que éstas utilicen modelos NIR o arquitecturas similares. Prueba de esto puede verse en la gran colección de trabajos sobre distintos temas concernientes a NIR expuestos en [Onal *et al.*, 2018], los que además están realizados sobre una inmensa cantidad de corpus diferentes. Razón por la cual es bastante tentador definir un marco de trabajo en el que éstas comparaciones sean más fáciles de realizar y en el que además se tengan a disposición algunos corpus de prueba, métricas estándar y modelos embebidos.

1.3. Objetivos

- **Objetivo general**: Desarrollar un *framework* en el que se puedan comparar 5 modelos de *Neural Information Retrieval* distintos, para consultas y *corpus* en Inglés.
- **Objetivos específicos**:
    1. Implementar Skip-gram, CBOW, GLoVE, FastText y ELMo como modelos NIR y *Precision*, *Recall* y NDGC como medidas estándar.
    2. Definir y embeber *corpus* base para IR en el *framework*.
    3. Diseñar e implementar una arquitectura escalable para el *framework*.
    4. Comparar resultados sobre agregación de términos para expansión de consultas con uso explícito.

1.4. Alcance

Debido a la gran variedad de procesos que se pueden llevar a cabo utilizando métodos y arquitecturas de *deep learning*, es necesario limitar el alcance que tendrá la propuesta. El enfoque del trabajo se centra sobre representaciones densas generadas a partir de redes neuronales, que serán utilizadas para la recuperación de documentos. Tareas como *learn to combine*, *learn to rank* o *learn to expand*, entre otras, se encuentran fuera del alcance y serán mencionadas a lo sumo como ejemplos.

Además de realizar pruebas sobre recuperación de documentos, se usarán los *embeddings* en algoritmos básicos de expansión de consultas. Todo esto sobre tres componentes del corpus Tipster: *Associated Press* (AP), *Wall Street Journal* (WSJ) y *San Jose Mercury News* (SJMN), compuestos de 239302, 173252 y 90257 documentos, respectivamente, para una suma total de 502811 documentos.

Por último el *framework* estará programado en Python debido a la gran cantidad de herramientas y librerías para procesamiento de lenguajes naturales que se tienen a disposición en este lenguaje de programación y también tomando en cuanta el hecho de que es interpretado, por lo que el *framework* podrá ser utilizado en diversos sistemas operativos.

CAPÍTULO 2
MARCO CONCEPTUAL

En este capítulo se definen conceptos claves en el enfoque clásico de recuperación de información, así como también modelos modernos que utilizan métodos y arquitecturas de *Deep Learning*. Además se describe una metodología de trabajo específica para problemas de minería de datos.

2.1. Enfoque clásico de IR

En ciencias de la ingeniería informática, Recuperación de información (*Information retrieval*), IR de ahora en adelante, es un campo de estudios que ayuda a un usuario a encontrar información que satisfaga sus necesidades. IR se ha basado en recuperación de documentos, remarcando un documento como unidad básica. Ahora bien, en la práctica un documento puede ser definido como cualquier unidad de información multimedia, desde una página *web* hasta archivos de música o vídeos, pero desde un punto de vista clásico un documento se entiende como un archivo de texto plano.

En la figura 1 se puede ver la arquitectura general de un motor de búsquedas, donde el usuario ingresa una consulta (*user query*) a través del módulo de operaciones de consulta (*query operations module*), el cual preprocesa la consulta y genera una versión que el motor puede trabajar, luego el módulo de recuperación usa un índice invertido de documentos (*document index*) para recuperar aquellos documentos que contengan por lo menos uno de los términos que componen la consulta inicial, a continuación computa la puntuación de relevancia por documento y genera un *ranking* utilizando dicha puntuación. Por último, los documentos son presentados al usuario en el orden del *ranking* calculado [Baeza-Yates y Ribeiro-Neto, 1999].

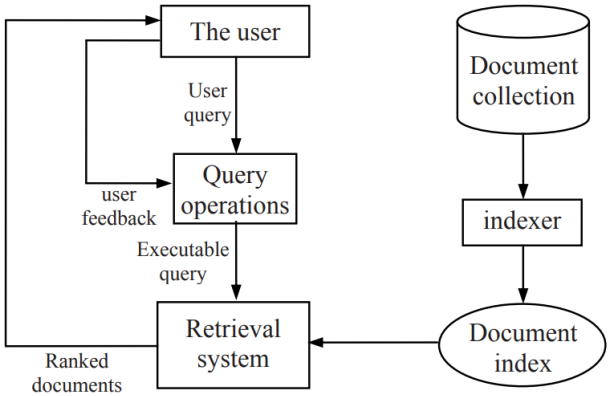


Figura 1: La arquitectura general de un motor de búsquedas
Fuente: [Liu, 2011].

El módulo de operaciones de consulta puede implementar protocolos que varían desde funciones relativamente simples a bastante complicadas para el preprocesamiento de las consultas, dependiendo de las necesidades del motor de búsqueda en sí mismo. En general se maneja la eliminación de *stopwords*, que corresponden a palabras que por si solas no tienen sentido alguno, como por ejemplo las preposiciones. Además puede tener rutinas de *stemming* o lematización, ambos son métodos que intentan llevar las palabras a su raíz léxica, ya sea por medio de expresiones regulares y reglas gramaticales o recortando las partes que cambian en sus conjugaciones, esto con el fin de reducir el vocabulario que el motor deberá manejar. Posteriormente el proceso de recuperación de documentos busca dentro del corpus documentos en los que aparezca alguno de los términos de la consulta y se les asigna una importancia, generalmente basada en la frecuencia de términos que tengan en común. Será el sistema de recuperación el encargado de asignarle un puntaje de relevancia a los documentos recuperados y de presentarlos de forma ordenada al usuario [Baeza-Yates y Ribeiro-Neto, 1999].

2.1.1. Modelos de IR

En un modelo de IR las definiciones más importantes son: la forma en que se representan de consultas y documentos; y cómo se calcula la relevancia de un documento con respecto a una consulta. En los enfoques clásicos generalmente se definen los términos que le dan sentido semántico a las consultas y documentos, el conjunto de todos los términos del corpus es llamado el vocabulario de la colección. Cada documento será descrito a partir del vocabulario, de forma que se le asigna un peso a cada término y se les asigna valor nulo a los términos que no aparecen en el documento. Se debe notar que a estos enfoques no les importa el orden en el que aparecen los términos y que el cálculo del peso cambiará según el modelo de *ranking* implementado por el motor de búsqueda. Debido a lo anterior un corpus puede quedar representado como una matriz de gran tamaño donde sus dimensiones corresponden a la cantidad de documentos que componen el corpus y al tamaño del vocabulario de la colección [Liu, 2011].

Uno de los esquemas más connotados para el cálculo de los pesos es *Term frecuency* (Tf), que utiliza la frecuencia de cada término en el documento y la utiliza para calcular su peso, generalmente este valor se normaliza como en la ecuación 1, el problema de este cálculo es que el modelo es ciego al largo del documento.

$$tf_{ij} = \frac{f_{ij}}{max\left(f_{1j}, f_{2j}, ..., f_{|V|j}\right)}, \tag{1}$$

Donde i corresponde al número del término, j corresponde al número del documento y f_{ij} es la frecuencia del término i en el documento j, por último $|V|$ es el tamaño del vocabulario.

Posteriormente, se introdujo a la ecuación el factor *inverse document frecuency* (idf) con la intención de solucionar uno de los efectos más grandes en texto y que ha causado más

de un dolor de cabeza: la ley de Zipf, ésta ley dicta que la distribución de frecuencia de las palabras en un texto tienen un comportamiento potencial, donde las palabras en la posición i muestran una frecuencia aproximadamente igual a $\frac{1}{i}$ la frecuencia de la primera palabra. Uniendo ambos se generó uno de los modelos más conocido en IR: Tf-idf, cuya fórmula para los pesos se muestra en la ecuación 2 y se puede aplicar similarmente a las consultas, aunque generalmente se aplican cambios para suavizar los vectores que las representan.

$$w_{ij} = tf_{ij} \times idf_i,\ idf_i = \log \frac{N}{df_i}, \tag{2}$$

Donde N es el número total de documentos y df_i corresponde al número de documentos en que aparece el término i [Liu, 2011].

Una vez calculados los pesos de los términos se deben establecer nociones de similitud entre los vectores que representan las consultas y los que representan documentos, como por ejemplo el producto punto entre ambos (ecuación 3) o la similitud coseno (ecuación 4), que tiene buenas propiedades, ya que equivale a 0 cuando son perpendiculares entre sí y asigna valor 1 cuando tienen la misma dirección.

$$sim\left(d_j, q\right) = \langle d_j \cdot q \rangle \tag{3}$$

$$coseno\left(d_j, q\right) = \frac{\langle d_j \cdot q \rangle}{||d_j|| \times ||q||} \tag{4}$$

También existe la posibilidad de definir funciones de similitud utilizando probabilidades de relevancia con fundamentos en la teoría estadística, es decir, los modelos de IR estiman la probabilidad de que el documento sea relevante a la consulta y utilizan ésta probabilidad para generar el *ranking* de documentos, aunque en este documento no se describirán en profundidad.

Al emplear los valores obtenidos del modelo Tf-idf, en conjunto a constantes empíricas y la introducción de variables para el largo de los documentos se creó el modelo de relevancia Okapi, también conocido como BM25 (*Best Model* #25), cuya fórmula se muestra en la ecuación 5. Es necesario mencionar que al ser una función basada en intuiciones y valores empíricos existen muchas variaciones de la misma que son utilizadas en la práctica [Singhal, 2001].

$$okapi(d_j, q) = \sum_{t_i, \in q, d_j} ln \frac{N - df_i + 0{,}5}{df_i + 0{,}5} \times \frac{\left(k_1 + 1\right) f_{ij}}{k_1 \left(1 - b + b \frac{dl_i}{avdl}\right) + f_{ij}} \times \frac{\left(k_2 + 1\right) f_{iq}}{k_2 + f_{iq}}, \tag{5}$$

Donde t_i es el término, $f_i j$ es la frecuencia del término en el documento, $f_i q$ es la frecuencia del término en la consulta, N es el número de documentos en el corpus, df_i es la cantidad de documentos que contienen el término t_i, dl_j es el largo del documento y $avdl$ es el promedio

del largo de los documentos en el corpus. Además de las variables anteriormente definidas, se tienen los parámetros empíricos k_1 (entre $1{,}0 - 2{,}0$), b (usualmente $0{,}75$) y k_2 (entre 1-1000) [Singhal, 2001].

2.1.2. *Relevance Feedback*

Con el objetivo de mejorar la efectividad de la recuperación, se han desarrollado diversas técnicas. *Relevance Feedback* es una de ellas, y se refiere al proceso donde el usuario identifica de una u otra forma algunos documentos relevantes e irrelevantes para su consulta dentro de la lista inicial de documentos recuperados. La identificación que realiza el usuario no necesariamente debe ser explícita, sino que el sistema podría guardar información de enlaces visitados, la cantidad de visitas hechas por el usuario o los enlaces completamente ignorados por él.

Con esta información el sistema procede a generar una nueva consulta, llamada consulta expandida, agregándole nuevos términos que sean representativos de los documentos marcados por el usuario. El sistema podría incluso generar un modelo de clasificación específico para un usuario identificado. El proceso de *Relevance Feedback* puede ser repetido hasta que el usuario quede satisfecho [Raghavan, 2007].

Una de las muchas variantes de *Relevance Feedback* es *Pseudo-relevance Feedback*, en la cual el usuario no se ve envuelto en la mejora de la consulta. La idea básica de este proceso es extraer términos de los documentos con mejor *ranking* utilizando alguna métrica, clásicamente se utiliza la frecuencia, y se agregan estos términos a la consulta original formando una nueva consulta con la que se hará nuevamente el proceso de recuperación. Al igual que *Relevance Feedback*, este proceso puede ser repetido hasta que el usuario esté satisfecho con los resultados. Claramente este proceso asume que los documentos en las mejores posiciones del *ranking* serán relevantes, aunque al expandir la consulta se pueden recuperar documentos relevantes que no fueron considerados en la primera ronda de recuperación, además de lo anterior, se puede notar que la efectividad de este proceso esta fuertemente ligada a la calidad de términos escogidos durante la expansión [Raghavan, 2007].

2.1.3. Expansión de Consultas

Como ya fue mencionado en la sección anterior, expansión de consultas es un método utilizado por los motores de búsqueda para intentar mejorar su eficacia. Éstos procesos se implementan con la intención de aminorar la falta de información que le entregan las consultas al sistema, las cuales suelen estar compuestas de unos pocos términos claves, esto en conjunto a la ambigüedad inherente de los lenguajes naturales genera que el motor de búsquedas se vuelva propenso a errores [Carpineto y Romano, 2012].

Uno de los contratiempos más críticos para IR es el problema de discrepancia de términos,

también conocido como el problema del vocabulario: los buscadores y los usuarios generalmente no utilizan las mismas palabras, ya sea por conjugaciones, cambios gramaticales, sinonimia y/o polisemia. Sinonimia se refiere al concepto de que distintas palabras tienen significados similares, por ejemplo: “abreviar” y “resumir”, lo que puede devenir en fallas al momento de recuperar un documento, llevando a bajas significativas de *recall*. Mientras que polisemia son aquellas palabras con más de un significado, por ejemplo: “banco” (tipo de asiento) y “banco” (entidad financiera), causando que se puedan recuperar documentos irrelevantes, implicando una baja en *precision*. *Recall* y *precision* son dos métricas de las cuales se hablará más adelante. Por consiguiente se utiliza la expansión de consultas como un medio por el cual se intenta capturar mejor la intención del usuario o simplemente produce una consulta más útil para el sistema. La expansión de consultas automatizada tiene una larga historia en IR, viendo sus primeras apariciones en la década de 1960 y desarrolladas aun más en la siguiente en técnicas como por ejemplo: *vector feedback*, *term-term clustering* y análisis comparativo de distribuciones de términos [Carpineto y Romano, 2012].

En años recientes, se han introducido diferentes métodos para la expansión de consultas haciendo uso de vectores densos que representan a las palabras, también conocidos como *embeddings*, de los cuales se hablará a fondo más adelante. Algunos de estos métodos agregan términos basándose en la similitud de éstos hacia la representación de la consulta en el espacio de los *embeddings*, otros utilizan la frecuencia de dichos términos y valores calculados empíricamente para recalcular los pesos en la nueva consulta. Se tiene constancia de trabajos que utilizan *pseudo-relevance feedback* e información mutua como base para extraer los términos candidatos a expandir la consulta original [Onal *et al.*, 2018].

2.2. *Embeddings*

Neural Information Retrieval (NIR), es un acercamiento a IR utiliza *Deep Learning* para generar vectores densos de representación, también conocidos como *embeddings*, al contrario de la representación vista en el modelo Tf-idf en el que los vectores son dispersos, es decir, tienen muchos más valores nulos que pesos calculados. Los *embeddings* tienen un conjunto de características que definen a la palabra y que se ven representadas en las distintas dimensiones de los vectores, por ejemplo una de sus dimensiones podría significar: ‘Es un animal’ y otra podría ser ‘Tiene escamas’, por lo que tendrán múltiples valores no nulos en su representación. Es necesario notar que ésta interpretación solo queda explícita en *embeddings* hechos a mano, pero en la práctica al estar calculados por máquinas su interpretación es mucho menos evidente. Gracias a los *embeddings* se puede definir un concepto de distancia entre términos, de esta forma palabras relacionadas como “mango” y “banana” aparecerán más cerca en el espacio vectorial que generado, que de la palabra “perro” [Mitra y Craswell, 2018].

Un efecto colateral de estas representaciones es que se puede definir una suerte de operación semántica entre palabras a nivel matemático, operaciones donde por ejemplo se puede obtener el siguiente resultado: $Rey - Hombre + Mujer \approx Reina$, que se puede apreciar

de forma visual en la figura 2.

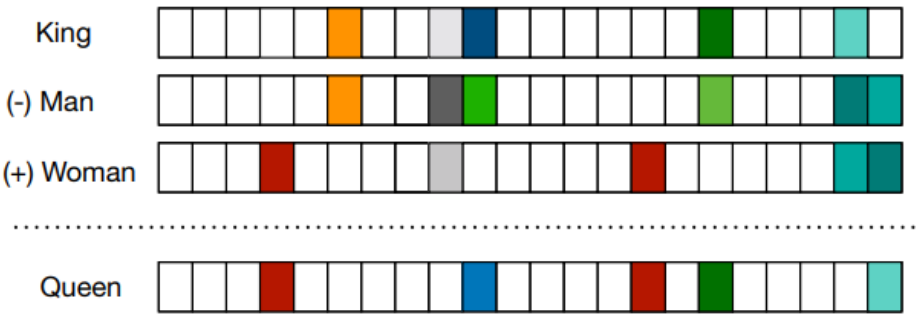


Figura 2: Ejemplo de matemática semántica
Fuente: [Young *et al.*, 2017].

Haciendo uso de estas operaciones de alto nivel se pueden definir distancias entre consultas y documentos, permitiendo la posibilidad de que además de apariciones explícitas de palabras en la consulta y los documentos, se consideren palabras fuertemente relacionadas a la ellas, pero sin que formen parte de la consulta o que los documentos potencialmente relevantes que han sido ignorados por el problema del vocabulario, ahora puedan ser valorados por el motor de búsqueda como respuesta a la consulta del usuario original y participen en el proceso de *ranking*. Durante los últimos cinco años se han propuesto múltiples modelos de redes neuronales para generar las *embeddings* de las palabras en el vocabulario de los motores de búsqueda, una de las primeras en dar buenos frutos fue ***Word2Vec*** [Mitra y Craswell, 2018].

2.2.1. Word2Vec

En *Word2Vec*, las características de una palabra son aprendidas utilizando a los términos vecinos de la misma dentro de una ventana móvil de tamaño fijo en el texto. Para el entrenamiento de estos modelos se requiere que las representaciones de los términos en la entrada sean del tipo *one-hot vector*, que corresponden a vectores de dimensionalidad igual al tamaño del vocabulario que será reconocido y en los cuales todos sus valores son nulos excepto en la dimensión que corresponde al término, es decir, en el vector del término i-ésimo, la dimensión i será la única con valor 1. *Skip-gram* fue la primera arquitectura ideada en el paradigma *Word2Vec*, este es un modelo de red neuronal simple con sólo una capa escondida, donde la entrada del modelo corresponde a la palabra de la que se desea generar el *embedding* y la salida debe ser una de las palabras en su vecindario. En la figura 3, se muestra la arquitectura de la red neuronal correspondiente al modelo *Skip-gram*, cuya función de pérdida entrenamiento se presenta en la ecuación 6 [Mikolov *et al.*, 2013b].

$$L_{skip-gram} = -\frac{1}{|S|}\sum_{i=1}^{|S|}\sum_{-c\leq j\leq +c, j=0} \log\left(p\left(t_{i+j}|t_i\right)\right), \tag{6}$$

donde,

$$\left(p\left(t_{i+j}|t_i\right)\right) = \frac{exp\left(\left(W_{out}\overrightarrow{\nabla}_{t_{i+j}}\right)^T\left(W_{in}\overrightarrow{\nabla}_{t_i}\right)\right)}{\sum_{k=1}^{|T|} exp\left(\left(W_{out}\overrightarrow{\nabla}_{t_k}\right)^T\left(W_{in}\overrightarrow{\nabla}_{t_i}\right)\right)} \tag{7}$$

S es el conjunto de todas las ventanas sobre el texto de entrenamiento y c es el número de vecinos que se quieren predecir en alguno de los costados del término t_i. Se debe mencionar que las dos matrices W_{in} y W_{out}, corresponden a los parámetros que pueden ser aprendidos en la red neuronal y que dentro del paradigma *Word2Vec* generalmente sólo se utiliza la matriz W_{in} como el conjunto de *embeddings*.

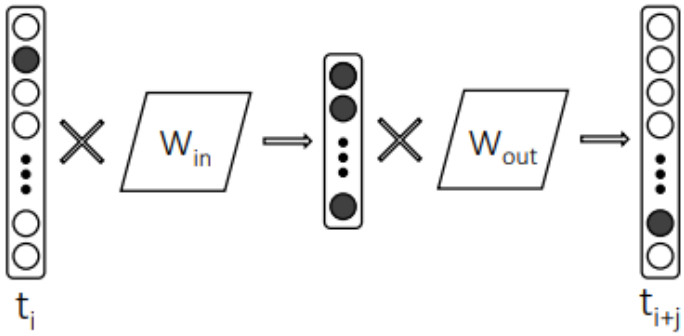


Figura 3: Arquitectura de Skip-gram
Fuente: [Mitra y Craswell, 2018].

Luego de *Skip-gram*, se definió un modelo con entrada y salida intercambiadas, en el que la entrada corresponderá a los vecinos del término, definidos por la ventana deslizante del modelo, mientras que la salida corresponderá al término del cual se quiere calcular el *embedding*. A este modelo se le llamó CBOW (*Continuous Bag of Words*) y su arquitectura se puede apreciar en la figura 4. CBOW crea sólo un ejemplo de entrenamiento con la suma de los *one-hot vectors* que corresponden a los términos vecinos y espera obtener como salida el *one-hot vector* ($\overrightarrow{v_{t_i}}$) del término cuyo *embedding* se quiere calcular. En contraste con *Skip-gram* que genera $2c$ ejemplos de entrenamiento al crear los pares entre todos los vecinos del término y el término objetivo, CBOW sólo generará un ejemplo de entrenamiento por combinación término-vecinos, pero por ésta misma razón será más rápido al entrenar. En la ecuación 8 se presenta la función de pérdida para el entrenamiento de CBOW, donde se mantiene la nomenclatura vista en la ecuación 6 [Mikolov *et al.*, 2013a].

$$L_{CBOW} = -\frac{1}{|S|}\sum_{i=1}^{|S|}\log\left(p\left(t_i|t_{i-c},...,t_{i-1},t_{i+1},...,t_{i+c}\right)\right) \tag{8}$$

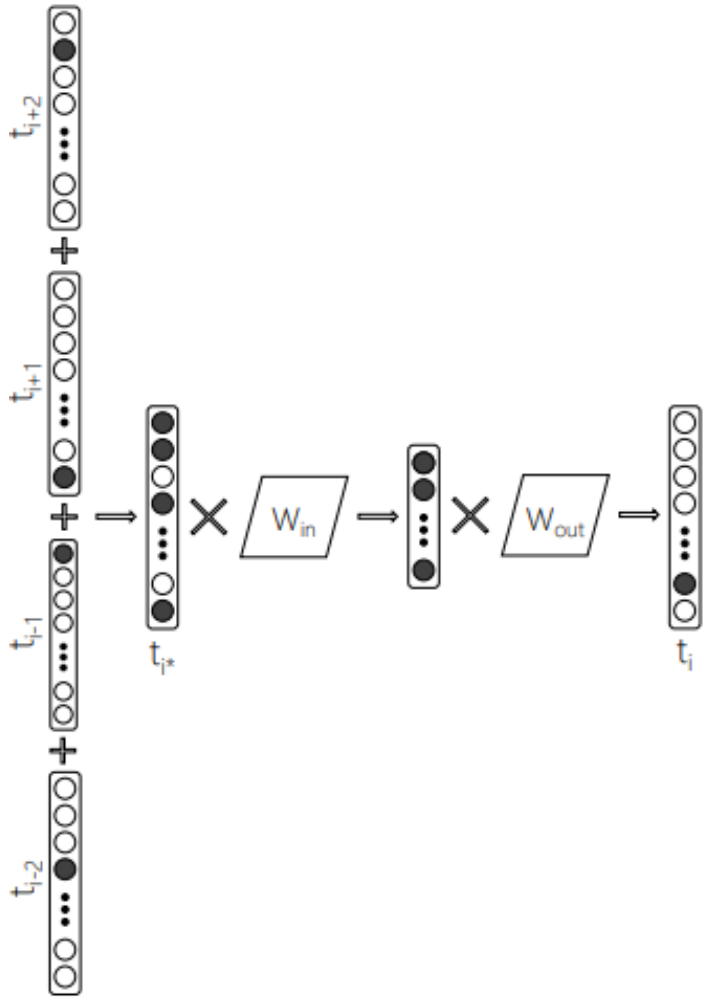


Figura 4: Arquitectura de CBOW
Fuente: [Mitra y Craswell, 2018].

2.2.2. GloVE

Como ya fue mencionado los modelos *Word2Vec*, se entrenan utilizando pares término-vecinos de forma individual. GloVe es un modelo en el que se utilizan las bases conceptuales de *Word2Vec*, proponiendo agregar todos los ejemplos de entrenamiento, tal que x_{ij} es la frecuencia del par $\langle t_j, t_j \rangle$ en toda la colección de entrenamiento, lo que genera un cambio en la función de pérdida como se muestra a continuación:

$$L_{skip-gram} = -\sum_{i=1}^{|T|}\sum_{j=1}^{|T|} x_{ij} \log\left(p\left(t_j|t_i\right)\right) \tag{9}$$

$$= -\sum_{i=1}^{|T|} x_i \sum_{j=1}^{|T|} \frac{x_{ij}}{x_i} \log\left(p\left(t_j|t_i\right)\right) \tag{10}$$

$$= -\sum_{i=1}^{|T|} x_i \sum_{j=1}^{|T|} \overline{p}\left(t_j|t_i\right) \log\left(p\left(t_j|t_i\right)\right) \tag{11}$$

$$= -\sum_{i=1}^{|T|} x_i H\left(\overline{p}\left(t_j|t_i\right), \left(p\left(t_j|t_i\right)\right)\right), \tag{12}$$

Donde $H\left(...\right)$ es el error *cross-entropy* entre la probabilidad de co-ocurrencia real $\overline{p}\left(t_j|t_i\right)$ y la que se predice con el modelo $p\left(t_j|t_i\right)$. La función 12 es similar a la pérdida de entrenamiento que se utiliza para GloVe (ecuación 13), pero reemplazando *cross-entropy* por un error cuadrado y aplicando una función de saturación $f\left(...\right)$ sobre la frecuencia de co-ocurrencia real [Pennington *et al.*, 2014].

$$L_{GloVe} = -\sum_{i=1}^{|T|}\sum_{j=1}^{|T|} f\left(x_{ij}\right)\left(\log\left(x_{ij} - \overrightarrow{v_{w_i}}^T\overrightarrow{v_{w_j}}\right)\right)^2, \tag{13}$$

donde,

$$f(x) = \begin{cases} \left(x/x_{max}\right)^{\alpha} & si\ x \leq x_{max} \\ 1, & en\ otro\ caso \end{cases} \tag{14}$$

GloVe se entrena utilizando AdaGrad, un algoritmo de optimización basado en gradientes con parámetros adaptables de aprendizaje. De manera similar a *Word2Vec*, GloVe también calcula dos matrices W_{in} y W_{out}, pero a diferencia de *Word2Vec* define los *embeddings* como la suma de los vectores en ambas matrices correspondientes a cada término del vocabulario [Mitra y Craswell, 2018]. Este cambio en el cálculo de los *embeddings* origina una variación en la forma que se agrupan los términos en el espacio vectorial. Si los vectores se computan utilizando sólo la matriz de entrada, los *embeddings* tenderán a juntar términos típicamente similares, es decir, que son de un tipo parecido, por ejemplo: ***Yale*** y ***Harvard***. Mientras que si se combinan las matrices de entrada y salida los *embeddings* agruparán términos tópicamente relacionados, por ejemplo: ***Yale***, **facultad** y **graduados** [Mitra *et al.*, 2016].

2.2.3. FastText

Ya se han mencionado tres implementaciones distintas que generan *embeddings* tomando como unidad mínima los términos. *Facebook* tomó un dirección diferente en este sentido,

proponiendo un modelo llamado *FastText*, el cual interpreta cada término como una composición de n-gramas al nivel de los caracteres, es decir, como una secuencia de n caracteres que conforman la palabra, donde la suma de la representación de cada carácter corresponde al *embedding* del término. Este cambio de perspectiva permite computar *embeddings* para palabras que están fuera del vocabulario del corpus, cosa que los modelos anteriormente mencionados son incapaces de hacer [Bojanowski *et al.*, 2017].

La implementación de *FastText* fue propuesta como una extensión al modelo Skip-gram con ejemplos negativos, cuya función de pérdida se puede ver en la ecuación 15, en el que además se toma en consideración la información que entrega la morfología de la palabra, separándola en una bolsa de n-gramas. Por ejemplo, la palabra *where* con $n = 3$ se separa en: $< wh,\ whe,\ her,\ ere,\ re >$ y la secuencia especial $where$, se debe notar que la secuencia $< her >$ corresponde a la palabra *her* y que es distinta del tri-grama her formado para la palabra *where*. También, como se puede apreciar, $<$ y $>$ son caracteres agregados que permiten distinguir prefijos de sufijos. En la práctica se extraen todos los n-gramas con $n \geq 3$ y $n \leq 6$.

$$L_{SG\ Negative\ Sampling} = \sum_{t=1}^{T} \left[\sum_{c \in C_t} l\left(s\left(w_t, w_c\right)\right) + \sum_{n \in N_{t,c}} l\left(-s\left(w_t, n\right)\right) \right], \tag{15}$$

Donde $N_{t,c}$ es el conjunto de ejemplos negativos tomados desde el conjunto de entrenamiento, l es la función de pérdida logística $l : x \rightarrow \log\left(1 + e^{s(w_t,n)}\right)$, $s\left(w_t, w_c\right)$ es el producto escalar entre los vectores del término y su contexto, w_t es el término objetivo y w_c es un término vecino dentro de su ventana de contexto [Bojanowski *et al.*, 2017].

Ahora bien, para generar la representación del término como la suma de sus n-gramas, se utiliza la función de puntuación 16, que reemplaza a la definida con anterioridad.

$$s\left(w, c\right) = \sum_{g \in G_w} z_g^T v_c, \tag{16}$$

Donde G_w corresponde al conjunto de n-gramas que aparecen en el término w y z_g es el vector de representación asociado a sus n-gramas.

Este cambio en el modelo permite compartir representaciones entre palabras, facultando el aprendizaje apropiado para términos raros. Finalmente una palabra es representada por un índice en el diccionario de términos y el conjunto de n-gramas que compone la palabra [Bojanowski *et al.*, 2017].

2.2.4. ELMo

Hasta el momento todos los modelos expuestos tienen la capacidad de generar un único *embedding* por término, por lo que no pueden resolver problemas de polisemia. Para intentar atacar este dilema se ideó un modelo que sale del paradigma impuesto por *Word2Vec*, en el que se utilizan arquitecturas de capas densas para realizar los cálculos de *embeddings*. En cambio estos se derivan de LSTM bidireccionales, que corresponden a arquitecturas de redes neuronales recurrentes especializadas en trabajar secuencias de datos, recorriéndolas en ambas direcciones, por consiguiente el nombre bidireccional y que pueden mantener cierto nivel de memoria con respecto a sus estados anteriores. Además de lo anterior, se le acoplaron objetivos de modelos de lenguaje a su entrenamiento y se le otorgó el nombre ELMo, proveniente de *Embeddings from Language Models*[Peters *et al.*, 2018].

Las representaciones generadas por ELMo se forman a partir de combinaciones lineales de los vectores pertenecientes a todas las capas internas de la arquitectura, esto se puede apreciar gráficamente en la figura 5.

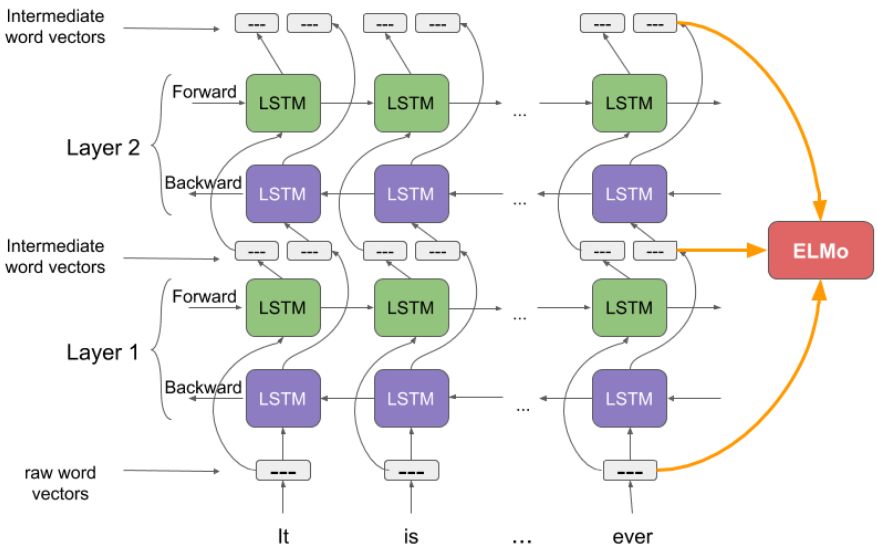


Figura 5: Formación de representaciones ELMo
Fuente: Analytics Vidhya.

Combinar los vectores de ésta manera permite una representación de las palabras que puede ser utilizada en varios niveles. Se ha mostrado que la capa de más alto nivel puede capturar información contextual, por ejemplo se puede usar para desambiguar una palabra, mientras que la capa más baja toma información sintáctica, con la que podrían realizarse tareas tipo NER (*Named Entity Recognition*). A diferencia de los otros modelos, los *embeddings* ELMo son calculados en función a una oración completa, de manera que un mismo término utilizado en dos oraciones de contextos diferentes tendrá dos representaciones distintas, una

para cada contexto, solucionando el problema de polisemia que se planteó en un principio [Peters *et al.*, 2018]. Por ejemplo si se le entregan dos oraciones al modelo: **Hoy adopté un gato** y **José y yo estábamos jugando al gato**, se calcularán dos *embeddings* diferentes para la palabra **gato** [Peters *et al.*, 2018].

Ahora bien, dada una secuencia de N términos, $(t_1, t_2, ..., t_N)$, la capa *forward* de la arquitectura calcula la probabilidad de la secuencia modelando la probabilidad del término t_k dado el historial $(t_1, ..., t_{k-1})$:

$$p\left(t_1, t_2, ..., t_N\right) = \prod_{k=1}^{N} p\left(t_k | t_1, t_2, ..., t_{k-1}\right) \tag{17}$$

Los modelos de lenguaje computan una representación independiente del contexto x_k^{LM}, que luego se pasará por L capas *forward* de LSTM, a continuación cada capa entrega una representación dependiente del contexto $\overrightarrow{h}_{k,j}^{LM}$, donde $j = 1, ..., L$. La representación entregada por la capa exterior L es utilizada para el cálculo del siguiente término t_{k+1} con una capa tipo Softmax. Este proceso se ve invertido en las capas *backward*, donde se requiere la predicción del término anterior:

$$p\left(t_1, t_2, ..., t_N\right) = \prod_{k=1}^{N} p\left(t_k | t_{k+1}, t_{k+2}, ..., t_N\right) \tag{18}$$

Una vez se tienen ambas direcciones calculadas se combinan maximizando log-verosimilitud:

$$\sum_{k=1}^{N} \left(\log p\left(t_k | t_1, ..., t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_S\right) + \log p\left(t_k | t_{k+1}, ..., t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_S\right)\right), \tag{19}$$

Donde Θ_x es la representación de independiente de contexto, Θ_S es el parámetro entregado por la capa Softmax y $\overrightarrow{\Theta}_{LSTM}, \overleftarrow{\Theta}_{LSTM}$ corresponden a las representaciones calculadas en ambas direcciones, *forward* y *backward* respectivamente.

Teniendo todo esto, ELMo agregará una combinación específica por tareas a la representación final. Por cada término t_k, una arquitectura de L capas computa un conjunto de $2L + 1$ representaciones:

$$R_k = \left\{ x_k^{LM}, \overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} | j = 1, ..., L \right\} = \left\{ h_{k,j}^{LM} | j = 0, ..., L \right\}, \tag{20}$$

Donde $h_{k,0}^{LM}$ es la representación independiente de contexto y $h_{k,j}^{LM} = \left[\overrightarrow{h}_{k,j}^{LM}; \overleftarrow{h}_{k,j}^{LM}\right]$ para cada capa de LSTM. ELMo colapsa todas las capas en R en un solo vector, $ELMo_k = E\left(R_k; \Theta_c\right)$. En el caso más simple ELMo sólo selecciona la última capa, $E\left(R_k\right) = h_{k,L}^{LM}$. De

forma más general, se usa un peso específico para la tarea, como se puede ver en la ecuación 21.

$$ELMo_k^{task} = E\left(R_k; \Theta^{task}\right) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM}, \tag{21}$$

Donde s^{task} son pesos normalizados por Softmax y el parámetro escalar γ^{task} permite escalar el vector ELMo completo. En la práctica, γ es importante para ayudar al proceso de optimización [Peters *et al.*, 2018].

2.2.5. *Average Word Embedding*

Habiendo mencionado todos los modelos generadores de *embeddings* que se cubrirán en este trabajo, sólo falta definir cómo utilizarlos para representar consultas y documentos.

A pesar de que existen modelos sofisticados que son capaces de condensar oraciones, párrafos o documentos en vectores densos con procedimientos similares a los observados con anterioridad, es bastante común utilizar el promedio de los *embeddings* de los términos pertenecientes al texto que se quiere representar, ya sea una consulta o un documento [Mitra y Craswell, 2018]. A esta representación se le conoce como *Average Word Embedding* (AWE) y se ha empleado en trabajos como [Mitra *et al.*, 2016, Kiros *et al.*, 2014]. En la ecuación 22 se puede apreciar el cálculo de AWE.

$$AWE\left(d\right) = AWE\left(\begin{bmatrix} W_{11} \\ W_{12} \\ \\ W_{1n} \end{bmatrix} + \begin{bmatrix} W_{21} \\ W_{22} \\ \\ W_{2n} \end{bmatrix} + ... + \begin{bmatrix} W_{m1} \\ W_{m2} \\ \\ W_{mn} \end{bmatrix}\right) = \begin{bmatrix} \frac{W_{11}+W_{21}+...+W_{m1}}{m} \\ \\ \\ \frac{W_{1n}+W_{2n}+...+W_{mn}}{m} \end{bmatrix}, \tag{22}$$

Donde d es un documento o consulta de largo m con *embeddings* de dimensionalidad n.

2.3. Métricas de efectividad

Ya se ha definido como trabaja un motor de búsqueda, sus componentes y procedimientos principales, las representaciones clásicas y representaciones basadas en NIR, pero aún no se ha definido como evaluarlos.

En IR generalmente no existe una decisión en cuanto a la relevancia o irrelevancia de un documento para una consulta. En su lugar, se crea un *ranking* de los documentos ordenado

en orden decreciente según el puntaje de relevancia, de la siguiente forma:

$$R_q :< d_1^q, d_2^q, ..., d_N^q >, \tag{23}$$

Donde R_q es el *ranking* para la consulta q, $d_1^q \in D$ corresponde al documento más relevante para la consulta y $d_N^q \in D$ es el documento más irrelevante para q, siendo D el conjunto de todos los documentos del corpus de tamaño N [Liu, 2011].

Suponiendo que se tiene un conjunto ordenado $D_q \subseteq D$, que corresponda a los documentos realmente relevantes a la consulta q, se pueden definir las métricas de *Precision*, *Recall* y NDCG.

2.3.1. *Precision* & *Recall*

Recall en la posición i del *ranking* es una métrica de evaluación, denotada por $r\,(i)$ y definida como la fracción de documentos relevantes desde d_1^q hasta d_i^q en R_q. Sea el número de documentos relevantes s_i existentes en $[d_1^q, d_i^q]$, entonces se tiene la ecuación 24.

$$r\,(i) = \frac{s_i}{|D_q|} \tag{24}$$

Mientras que *Precision* en la posición i del *ranking*, es la fracción de documentos relevantes entre d_1^q y d_i^q, utilizando la misma nomenclatura que en la definición anterior, se tiene la ecuación 25 [Liu, 2011].

$$p\,(i) = \frac{s_i}{i} \tag{25}$$

Tabla 1: Ejemplo *Precision* y *Recall* hasta la posición 10, sabiendo que en el corpus hay ocho documentos relevantes para la consulta.

Fuente: [Liu, 2011].

Rank i	$+/-$	$p\,(i)$	$r\,(i)$
1	+	$1/1 = 100\,\%$	$1/8 = 13\,\%$
2	+	$2/2 = 100\,\%$	$2/8 = 25\,\%$
3	+	$3/3 = 100\,\%$	$3/8 = 38\,\%$
4	-	$3/4 = 75\,\%$	$3/8 = 38\,\%$
5	+	$4/5 = 80\,\%$	$4/8 = 50\,\%$
6	-	$4/6 = 67\,\%$	$4/8 = 50\,\%$
7	+	$5/7 = 71\,\%$	$5/8 = 63\,\%$
8	-	$5/8 = 63\,\%$	$5/8 = 63\,\%$
9	+	$6/9 = 67\,\%$	$6/8 = 75\,\%$
10	+	$7/10 = 70\,\%$	$7/8 = 88\,\%$

Ahora bien, no siempre es práctico utilizar una tabla para comparar puesto a puesto éstas métricas, por lo que se hace necesario calcular un sólo valor por métrica para comparar diferentes algoritmos de recuperación en una consulta q. Para esto se puede obtener *Average Precision* (AP), ecuación 26 .

$$p_{avg} = \frac{\sum_{d_i^q \in D_q} p\left(i\right)}{|D_q|} \tag{26}$$

Por último, en caso de que se necesiten evaluar los resultados para múltiples consultas se puede utilizar *Mean Average Precision* (MAP), ecuación 27 [Liu, 2011].

$$MAP = \frac{\sum_{q \in Q} p_{avg}\left(q\right)}{|Q|}, \tag{27}$$

Donde Q es el conjunto de todas las consultadas a evaluar.

Un resultado similar se puede conseguir si se pondera *Recall*, pero sólo se encuentra bien definida si se calcula el promedio a en una posición específica del *ranking*, por ejemplo: $AR@10$ correspondería al promedio de *recall* obtenida hasta la posición 10.

2.3.2. NDCG

Discounted Cumulative Gain(DCG_q) es una métrica que consiste en considerar los primeros i puestos del *ranking* para una consulta y evaluar que tan bien ordenados están los documentos con respecto a la relevancia que presentan. Existen varias versiones de DCG_q que pueden ser definidas, la siguiente es una de ellas:

$$DCG_q = \sum_{\langle i,d \rangle \in R_q} \frac{2^{rel_q(d)} - 1}{\log_2\left(i+1\right)}, \tag{28}$$

Donde $rel_q\left(d\right)$ es alguna función que define el puntaje de relevancia de un documento d para una consulta q [Mitra y Craswell, 2018].

Se puede definir entonces el DCG_q ideal ($IDCG_q$) como $IDCG_q = max\ DCG_q$, que se puede calcular fácilmente si se supone que los documentos están ordenados de forma perfecta en el *ranking*. Con esto se define la versión normalizada de DCG (NDGC) cuya fórmula se puede apreciar en la ecuación 29 [Wang *et al.*, 2013].

$$NDCG_q = \frac{DCG_q}{IDCG_q} \tag{29}$$

Es necesario mencionar que en NDCG no importa la base del logaritmo, ya que al realizar la normalización se cancelará el efecto usar una base arbitraria, en [Wang *et al.*, 2013] utilizan

logaritmo natural para realizar todos los cálculos. Además NDCG queda bien definida si la lista de relevancias con las que se cuenta tiene valores binarios.

2.4. CRISP-DM

Cross-Industry Standard Process for Data Mining (CRISP-DM), es un modelo general del proceso que define enfoques comuniques que utilizan los expertos en minería de datos. CRISP-DM divide los procesos en seis fases: comprensión del negocio, comprensión de los datos, preparación de los datos, modelado, evaluación e implementación. Las relaciones entre sus procesos se muestran en la figura 6 [1] [Chapman *et al.*, 2000]. A continuación se describen las seis fases de CRISP-DM:

- **Comprensión del Negocio**: En ésta fase se espera obtener un claro entendimiento del problema que se intenta resolver, cuáles son sus impactos y definir los objetivos para enfrentarlo.
- **Comprensión de los datos**: Para ésta fase se deben inspeccionar, describir y evaluar los datos disponibles.
- **Preparación de los datos**: Corresponde a la fase en la que los datos deben ser procesados de manera que se encuentren en el estado necesario para trabajar con ellos.
- **Modelado**: En ésta fase se utilizan diversas herramientas de modelado para dar apoyo a las decisiones que se tomarán.
- **Evaluación**: Se debe evaluar la calidad de los modelos generados en la fase anterior.
- **Implementación**: Integrar los nuevos modelos al sistema que ya se tenga implementado.

[1] Ésta imagen es una versión con mejor resolución y a color del diagrama que se muestra en la fuente original: [Chapman *et al.*, 2000].

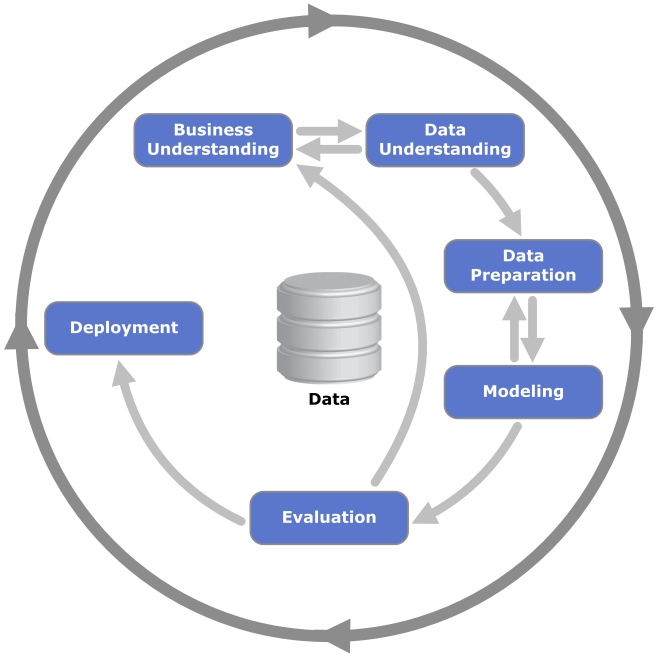


Figura 6: Diagrama que muestra la relación entre diferentes fases de CRISP-DM. Fuente: Wikipedia.

CAPÍTULO 3
PROPUESTA DE SOLUCIÓN

En este capítulo se define la metodología de trabajo que se utiliza en la propuesta de solución, incluyendo la preparación de los datos de los cuales se dispone, la estructura e implementación de la solución y el diseño experimental de las comparaciones en las que se utilizará el *framework* propuesto.

3.1. Metodología de Trabajo

Debido a que IR es un área de la informática cuyos métodos y procedimientos se encuentran fuertemente ligadas al conjunto de datos del cual se dispone junto con necesitar herramientas cercanas a minería de datos, es altamente deseable utilizar la metodología de trabajo CRISP-DM.

Como se expuso en la sección 2.4, se deben seguir los seis pasos que presenta CRISP-DM, pero debido a su naturaleza iterativa se unirán algunos de ellos y se alterará un poco el orden en el que se exponen para una mejor comprensión de la versión final que adopta la solución.

1. **Comprensión del Negocio**: Ésta fase queda expuesta en el capítulo 2, donde se definen y explican todos los conceptos que son necesarios para comprender la solución y elementos que se encuentran dentro de su alcance.
2. **Comprensión y Preparación de los datos**: Los detalles de ésta fase se encuentran expuestos en la sección 3.1.1, en la cuál se comentan los meta-datos de los corpus utilizados y los procesos a los cuales deben ser sometidos para poder embeberlos al *framework*. En el capítulo 4 se muestra la meta-data de los corpus post-procesamiento.
3. **Modelado e Implementación**: En la sección 3.1.2 se encuentra detallado el modelo del *framework* y en el capítulo 4 se presenta su implementación.
4. **Evaluación**: Por último con el fin de probar el *framework*, en la sección 3.1.3 se describe el diseño experimental de las comparaciones a realizar y en el capítulo 4 se muestran los resultados obtenidos a partir de ellas.

3.1.1. Comprensión y preparación de los datos

Los datos de los que se dispone corresponden a tres particiones del corpus Tipster, también conocido como *Text Research Collection Volume* (TREC). Tipster es un corpus muy grande

con información real en Inglés y sin procesamiento alguno más allá de la estructura en la que se ordenan los documentos. Fue creado en un intento de impulsar significativamente el estado del arte en IR y extracción de información en datos reales[2], por lo que es ideal para este trabajo.

Específicamente se dispone de todos los volúmenes de tres particiones distintas: *Associated Press* (AP), *Wall Street Journal* (WSJ) y *San Jose Mercury News* (SJMN). Los datos sobre éstas particiones se pueden apreciar en la tabla 2. Debido al tamaño de cada partición, cada una será tratada como un corpus a partir de ahora.

Tabla 2: Meta-data de los corpus sin procesar.
Fuente: Elaboración propia.

Corpus	Años	# Documentos	# Apróx. Palabras (Millones)	Tamaño (KB)
SJMN	1991	90257	45	293729
WSJ	$1987 - 1992$	173252	81	520703
AP	$1988 - 1990$	239302	114	753397

Cada corpus está separado por años en distintos archivos. Estos archivos tienen una estructura basada en marcas tipo HTML, donde se ordenan los documentos pertenecientes a ese grupo junto a sus datos, algunos de ellos son: la ID del documento, el texto que lo compone, la ciudad y fecha de publicación, datos de *copyright* entre otros. Gracias a esta estructura, la información necesaria se puede obtener fácilmente haciendo uso de la librería *Beautiful Soup*[3], la cual permite recortar, mantener o eliminar secuencias de texto con marcas tipo HTML. De ésta manera se pueden separar los documentos en distintos archivos de texto plano, utilizando sus ID como nombres de archivo para poder identificarlos rápidamente, esto permite que se puedan definir subconjuntos de documentos para realizar pruebas sin la necesidad de cargar todo Tipster a memoria principal.

Además de los archivos anteriormente descritos, se dispone de 150 consultas de prueba, junto a registros en los que se mantienen sus conjuntos de relevancia binaria con respecto a todos los documentos del corpus Tipster, con el siguiente formato:

```
101 0 AP880212-0047 1
101 0 AP880219-0139 0
101 0 AP880219-0166 0
        ...
173 0 WSJ920204-0068 0
173 0 WSJ920204-0104 1
173 0 ZF108-093-674 0
```

[2]Fuente:`https://catalog.ldc.upenn.edu/LDC93T3A`

[3]La documentación de ésta librería puede encontrarse en: `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`

Donde el primer campo de cada fila indica el ID de la consulta, el tercer campo indica el ID del documento y el cuarto campo indica si el documento es relevante (1) o no lo es (0).

Ya que Tipster se compone de documentos reales que fueron publicados entre las décadas de 1980 y 1990 en medios formales, se puede esperar que el texto en ellos tenga muy pocas faltas de ortografías o símbolos fuera de lo común. Por ende, se es posible suponer que el preprocesamiento que necesitan los corpus consta de pasos básicos, como por ejemplo: remoción de *stopwords*, llevar todo el texto a minúsculas, procesos de lematización o *stemming* y *tokenization* en general. Luego, estas versiones digeridas de los documentos pueden ser almacenadas de manera que los procedimientos no necesiten ser repetidos cada vez que se necesiten cargar los documentos a memoria principal para generar los índices invertidos. Es pertinente notar que el preprocesamiento debe ser repetido para las consultas, de manera que los términos puedan ser reconocidos por el sistema, pero no es necesario almacenar las consultas procesadas, puesto a que su cantidad y extensión es muy pequeña en comparación con los documentos, por ende no tomará demasiado tiempo hacerlo mientras se utiliza el motor de búsqueda.

3.1.2. Estructura e implementación del *framework*

A raíz de la naturaleza iterativa que impone la metodología CRISP-DM, se pueden utilizar prácticas ágiles para la implementación del *framework*, de forma que se tenga la capacidad de dividir el sistema en módulos que pueden ser agregados de manera gradual. La siguiente es una lista de módulos necesarios para obtener el alcance deseado para la solución:

- **Operaciones de consulta**: Este módulo debe procesar las consultas y generar consultas ejecutables por el sistema. Además será requerido que pueda implementar alguna forma de paralelización con el fin de aprovechar de la mejor manera posible los recursos computacionales de los que se dispongan y acelerar los tiempos de ejecución del *framework*.
- **Sistema de recuperación**: El sistema de recuperación corresponde al módulo central en todo motor de búsquedas. Es el responsable de recuperar los documentos del corpus y generar el *ranking* de relevancia que será presentado al usuario. Para lograr esto es necesario implementar un método o clase que genere un índice invertido del corpus que se quiera utilizar. También será necesario crear métodos para mantener una tabla de largos de documentos. Por último se debe definir una función de *ranking* para poder ordenar los documentos recuperados.
- **Métricas de evaluación**: En este módulo se deben desarrollar métricas pertinentes para la evaluación de distintos motores de búsqueda implementados con el *framework*, dichas métricas pueden programarse utilizando Numpy[4].

[4]La documentación de ésta librería puede encontrarse en: `https://numpy.org/`

- **Métodos para el entrenamiento y la integración de *embeddings***: A través de estos métodos se deben poder calcular *embeddings* para el vocabulario de los corpus disponibles en el *framework*, esto puede ser logrado utilizando las siguientes librerías: Keras[5], Gensim[6] y/o bilm-tf[7]. Igualmente este módulo debe poder integrar *embeddings* ajenos al sistema, en caso de que se desee evaluar representaciones usando técnicas como *transfer learning* o modelos que no puedan ser entrenados sobre los corpus disponibles por diversas razones. Es necesario mencionar que este módulo puede implementarse gradualmente, ofreciendo mayor cantidad de modelos a medida que se avance en las iteraciones del *framework*.

- **Módulo de distancias semánticas**: Ya que toda ésta propuesta se centra en comparar modelos NIR, este modulo es esencial para el *framework* y debe poder ofrecer métodos para el calculo de distancias semánticas entre términos o conjuntos de ellos, junto a calcular representaciones de documentos y consultas basándose en los *embeddings* que se tengan a disposición gracias al módulo anterior.

- **Módulo de expansión de consultas**: Por medio de distintas técnicas como *pseudo-relevance feedback* y métodos para la selección de términos cercanos se puede implementar un módulo que expanda las consultas originales utilizando como base las funciones ofrecidas por los dos módulos anteriores. Por ende este debe ser uno de los últimos módulos a implementar.

3.1.3. Diseño Experimental

Para poder validar la funcionalidad del *framework* a continuación se presenta una serie de pasos para realizar las comparaciones entre los modelos anteriormente mencionados.

1. Debido a lo difícil que es lograr el objetivo de IR, es realmente complicado poder superar la base que sientan los modelos clásicos con parámetros ajustados experimentalmente, de manera que se debe establecer una *Strong Baseline* para los modelos NIR haciendo uso de un modelo simple de recuperación.

2. Sintonizar el parámetro que define la cantidad de términos en el que se expandirán las consultas.

3. Pruebas de expansión utilizando distintos modelos de *embeddings*.

4. Pruebas de recuperación utilizando AWE para representar las consultas y documentos.

[5]La documentación de ésta librería puede encontrarse en: `https://keras.io/`
[6]La documentación de ésta librería puede encontrarse en: `https://radimrehurek.com/gensim/`
[7]La documentación de ésta librería puede encontrarse en: `https://github.com/allenai/bilm-tf`

CAPÍTULO 4
VALIDACIÓN DE LA SOLUCIÓN

En este capítulo se presentan los resultados de la preparación de los corpus, la implementación del *framework* y los resultados de los experimentos propuestos en el capítulo anterior.

4.1. Preparación de los corpus

Debido a que los procedimientos realizados y resultados obtenidos son similares en los tres corpus, en ésta sección sólo se mostrarán las figuras correspondientes al corpus AP y las demás serán expuestas en el apéndice del trabajo.

Como se mencionó en el capítulo anterior, los corpus escogidos están compuestos por documentos reales publicados en distintos medios formales de comunicación, por lo que se esperaba que el texto en ellos tuviera un comportamiento normal, es decir, siguieran la ley de Zipf. Además se confiaba que las palabras al inicio de la distribución fuesen *stopwords*. Esto queda confirmado claramente en las figuras 7, A1 y A5.

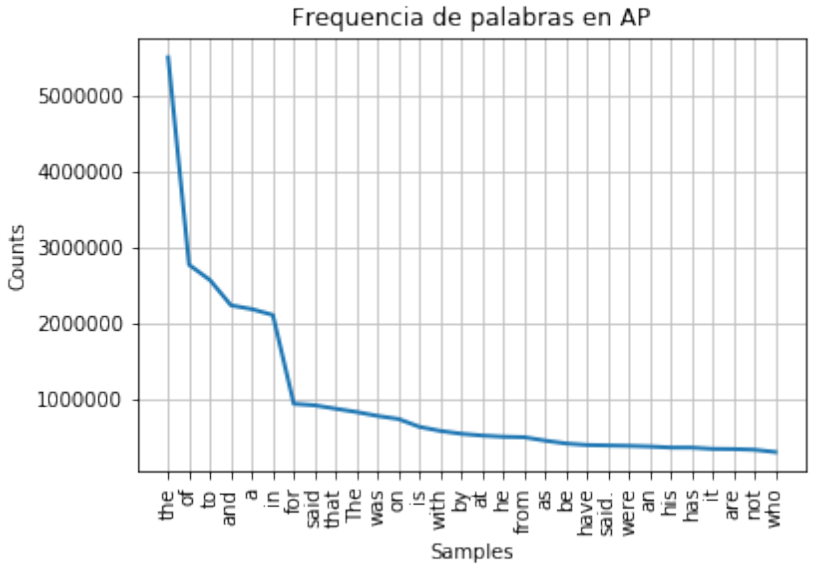


Figura 7: Distribución de frecuencia de las treinta palabras más frecuentes en AP.
Fuente: Elaboración propia.

También se realizó un *wordcloud* por cada corpus, presentados en las figuras 8, A2 y A6; en los cuales se puede apreciar las palabras con sentido propio que más se repiten en los corpus.

Con ellas se podrían definir tópicos predominantes en los corpus, por ejemplo al ver el *word-cloud* de WSJ, figura A6, se podría argumentar que el corpus tiene una alta concentración de documentos centrados en economía.

Figura 8: *Wordcloud* de AP.
Fuente: Elaboración propia.

Para finalizar el análisis de los corpus, se estudió la cantidad de palabras en los documentos que los componen. En la figura 9 se pueden apreciar los *boxplot* para los largos de documentos de los tres corpus. Además se presentan los datos exactos de mínimo, máximo y promedio, en la tabla 3 ya que son difíciles de visualizar en los gráficos.

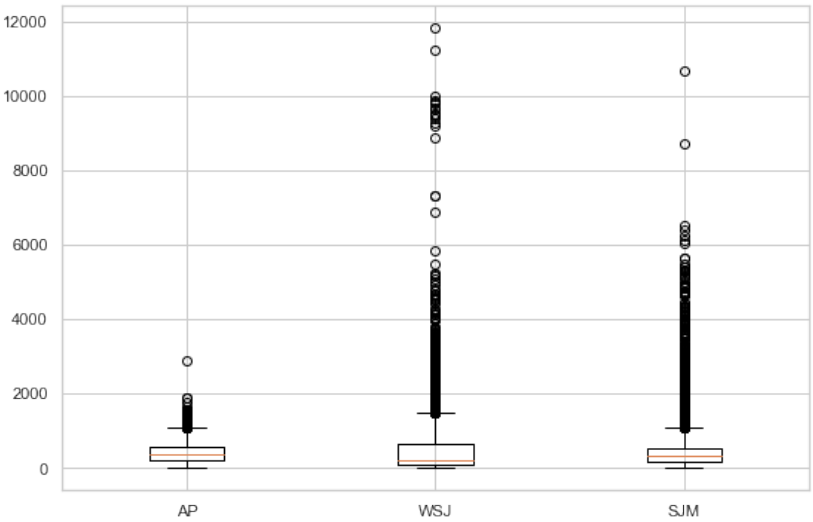


Figura 9: *Boxplot* para el largo de los documentos en los corpus antes de ser procesados.
Fuente: Elaboración propia.

Tabla 3: Valores notables para largos de documentos en corpus sin procesar. Fuente: Elaboración propia.

Corpus	Mín	Máx	Promedio
AP	1	2876	418.74
WSJ	1	11828	390.14
SJMN	1	10689	327.94

Con toda la información presentada anteriormente, se confirmó la necesidad de un procesamiento estándar como el que se propuso en la sección 3.1.1, es decir, remoción de *stopwords*, llevar todo el texto a minúsculas y realizar *stemming* o lematización. Durante el desarrollo se optó por utilizar *stemming*, debido a la simpleza del proceso en comparación a lematización y a que la librería NLTK de Python ofrece un muy buen conjunto de métodos para llevarlo a cabo. En particular se utilizó el módulo *SnowballStemmer*[8], el cual suele reportar buenos resultados. Así como también, se implementó la remoción de *stopwords* utilizando la lista entregada por NLTK, ya que para el idioma Inglés está bastante completa.

A continuación se presentan las figuras 10, A3 y A7 correspondientes a la frecuencia de distribución de términos en los corpus procesados y sus *wordcloud*, figuras 11, A4 y A8. En ellas se puede apreciar el efecto que tuvo el procesamiento en los corpus: los primeros treinta términos ya no son exclusivamente *stopwords*, no existe diferenciación por mayúsculas y las palabras con sentido siguen siendo representadas en los *wordclouds* de la misma manera, a través de sus *tokens*, por lo que se puede decir que el contexto y los tópicos de los documentos no se vieron afectados por los procedimientos realizados.

[8]La documentación de esta librería puede encontrarse en: `https://www.nltk.org/_modules/nltk/stem/snowball.html`

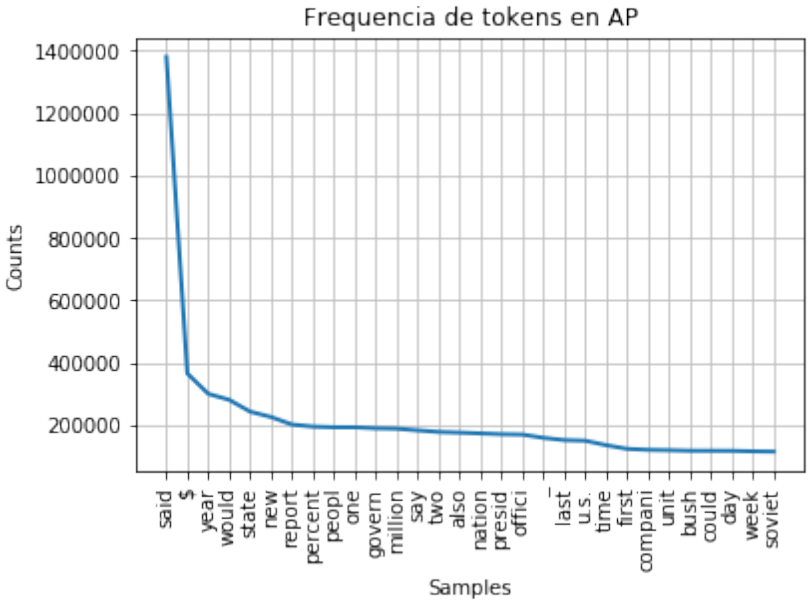


Figura 10: Distribución de frecuencia de los treinta términos más frecuentes en AP.
Fuente: Elaboración propia.

Figura 11: *Wordcloud* de AP post-procesamiento.
Fuente: Elaboración propia.

Finalmente, se presentan los *boxplot* para la cantidad de *tokens* en los documentos de cada corpus en la figura 12, y de manera similar a la presentación anterior se entregan los valores de mínimo, máximo y promedio en la tabla 4.

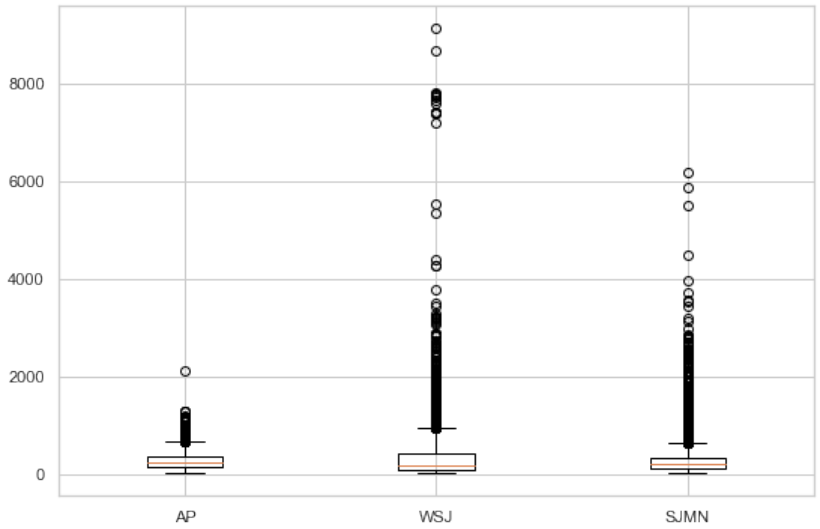


Figura 12: *Boxplot* para el largo de los documentos en los corpus después de ser procesados. Fuente: Elaboración propia.

Tabla 4: Valores notables para largos de documentos en corpus procesados. Fuente: Elaboración propia.

Corpus	Mín	Máx	Promedio
AP	1	2115	255.45
WSJ	1	9139	245.83
SJMN	1	6168	197.13

Estos datos pueden ser útiles para estimar la cantidad de memoria principal necesaria y los tiempos de ejecución aproximados en las pruebas experimentales.

4.2. Implementación del *framework*

A continuación se describirá la implementación de los módulos en el *framework*.

4.2.1. Modelo base

Como fue mencionado en la sección 3.1.2, la implementación del *framework* debía desarrollarse de manera iterativa e incremental, de ésta forma se necesitaba desarrollar un modelo base que siguiera las guías de funcionamiento para los motores de búsqueda clásicos.

Se partió desarrollando el módulo de operaciones de consulta, al cual se nombró *QueryParser*. Este sigue el mismo procesamiento que se le realizó a los corpus, pero lo aplica a las consultas en tiempo de ejecución. Las consultas deben ser entregadas al módulo en un archivo de texto plano, donde cada línea del archivo corresponde a una consulta diferente. Por ejemplo la consulta:"Design of the "Star Wars" Anti-missile Defense System", luego del procesamiento será registrada como: ['design', 'star', 'war', 'anti-missil', 'defens', 'system'].

Luego se desarrolló el módulo de recuperación y *ranking*. Para esto se crearon tres clases:

- *CorpusParser*, que se encarga de tomar el corpus que se indique y cargarlo a memoria principal para poder construir clases siguientes.
- *InvertedIndex*, la cual mantiene el índice invertido del vocabulario en una estructura de diccionario, cuyas llaves corresponden a los términos del vocabulario reconocido por el sistema y sus valores son listas con las ID de los documentos en los que aparecen.
- *DocumentLengthTable*, en la que se guarda la tabla de largos de documentos, estos datos serán utilizados posteriormente en la función de *ranking*.

Utilizando éstas tres clases se procedió a definir el estándar de recuperación para el modelo base. Para este procedimiento se obtienen todos los documentos que presenten la conjunción de los términos de la consulta, en otras palabras, se toman en cuenta sólo los documentos en los que se encuentren presentes todos los términos de la consulta. La razón por la cual se definió así fue intentar asegurar que el contexto general de los documentos correspondiera al contexto de la consulta. Finalmente, los documentos son puntuados según Okapi-BM25 con parámetros dentro de los estándares dispuestos en la sección 2.1.1.

El procedimiento completo de evaluación de consultas puede ser realizado en múltiples procesos haciendo uso de la librería *multiprocessing*[9] de Python. Para lograrlo se divide la lista de consultas que se le da como entrada al sistema y se reparten entre la cantidad de procesos que se indique, pero se debe tener cuidado al fijar el número de procesos deseados, puesto a que hacer esto consume una gran cantidad de recursos. Se escogió ésta forma de implementación, ya que la otra opción era utilizar *Threads*, pero en Python los *threads* no paralelizan realmente, debido al uso de GIL (*Global Interpreter Lock*) causando que solo se pueda tener sólo un hilo de procesamiento activo a la vez.

Finalmente, se agregaron métodos para calcular tres métricas de evaluación: MAP, AR@N y NDCG@N, cuya implementación solo requirió el uso de la librería Numpy[10] y siguen las fórmulas presentes en la sección 2.3. Es importante mencionar que para calcularlas se precisa de una lista con los conjuntos de relevancia para todas las consultas que se utilicen. Para el caso de las pruebas realizadas se utilizaron las relevancias booleanas que entrega Tipster,

[9]La documentación de ésta librería se puede encontrar en: `https://docs.python.org/3/library/multiprocessing.html`

[10]Documentación de Numpy: `https://numpy.org/devdocs/`

pero en ellas se consideran todas las particiones del corpus, por lo que no se esperan grandes valores de *recall* o MAP en los experimentos.

4.2.2. Implementación de *Embeddings*

Luego de terminar las iteraciones sobre el modelo base, se agregó el modulo de *embeddings*. En él se ofrecen funciones para calcular *embeddings Word2Vec* (Skip-gram o CBOW) y GloVe, aunque este último sólo pudo ser calculado sobre subconjuntos pequeños de documentos, debido a las grandes cantidades de memorias requeridas para mantener las matrices de co-ocurrencia de los términos. Tanto Skip-gram como CBOW, fueron implementados utilizando los modelos ofrecidos por Gensim[11], mientras que GloVe fue desarrollado utilizando una librería con el mismo nombre[12]. Junto a lo anterior, este módulo también ofrece la opción de integrar *embeddings* precalculados o entrenados sobre un corpus externo. Esto puede realizarse utilizando los métodos dispuestos en Gensim para cargar parámetros de modelos o bien entregando los *embeddings* como texto plano con el siguiente formato:

```
Termino_1 valor_1 valor_2 valor_3 ... valor_d \n
Termino_2 valor_1 valor_2 valor_3 ... valor_d \n
                    ...
Termino_N valor_1 valor_2 valor_3 ... valor_d \n
```

Por último, también se pueden escoger dos modelos precalculados sin la necesidad de integrarlos, puesto a que vienen cargados en el *framework*, estos son *FastText* y ELMo. El primero de ellos fue conseguido a través de la página oficial de *FastText*[13] y el segundo es dispuesto utilizando los valores disponibles en la librería Tensorflow-hub[14]. Es muy importante recordar que las evaluaciones de ELMo se realizan en tiempo de ejecución, pues el cálculo del *embedding* para un término es dependiente del contexto que tenga y debido a que su modelo está basado en tensores, los valores se procesan GPU, en particular, haciendo uso de métodos CUDA.

4.2.3. Implementación de recuperación por AWE y Expansión de Consultas

Por último, pero no menos importante, se realizó el desarrollo de los módulos AWE y Expansión de consultas. Para ello fue necesario crear una nueva clase, la cual se llamó “Dist_semantica”. Ella está encargada de mantener los valores de *embeddings* en memoria principal según el modelo que se se le indique. También contiene los métodos que permitirán calcular

[11]Para mayor información de Gensim: `https://pypi.org/project/gensim/`
[12]Para más información sobre la librería GloVe: `https://pypi.org/project/glove/`
[13]`https://fasttext.cc/docs/en/english-vectors.html`
[14]Para más información sobre Tensorflow-hub: `https://www.tensorflow.org/hub`

las representaciones AWE de documentos y consultas, la distancia coseno entre vectores y los primeros K términos más cercanos a una consulta.

Para realizar la recuperación por medio de AWE se utiliza la disyunción de los términos en la consulta, es decir, se recuperarán todos los documentos que contengan por lo menos uno de los términos de la consulta. Ésta diferenciación con respecto al modelo base es debido a que, en este caso, se debería poder decidir si el documento es cercano al contexto definido por la consulta utilizando los *embeddings*. Luego de la recuperación se utiliza la función distancia coseno entre el *embedding* de la consulta y la representación AWE de cada documento, ecuación 4, para ordenarlos de forma creciente y se entregan sólo los Top documentos más cercanos, donde Top es un parámetro con valor 50 por defecto, pero puede ser ajustado a gusto por el usuario. Luego los documentos de este conjunto son evaluados utilizando la función Okapi-BM25 para generar el *ranking* final que se devuelve al usuario. Este procedimiento se muestra de manera gráfica en la figura 13.

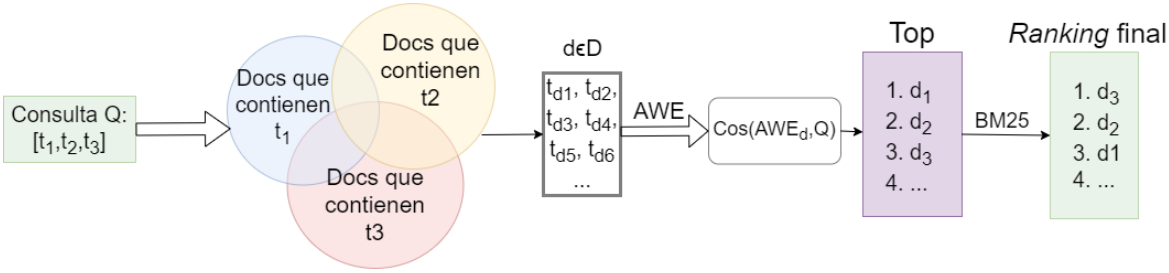


Figura 13: Diagrama de recuperación usando AWE
Fuente: Elaboración propia.

Por otro lado, para la implementación del módulo de expansión se utilizó un método basado en [Kuzi *et al.*, 2016]. En ese trabajo se definió, entre otros, un método simple inspirado en AWE, al cual llamaron *The Centroid Method* o en español El método del Centroide, que resulta ser simple y efectivo. El método del Centroide consiste en calcular la representación AWE de la consulta y luego verificar la distancia entre los términos candidatos a expandir utilizando la función definida en la ecuación 30.

$$S_{Cent}\left(t;q\right) = e^{coseno\left(\overrightarrow{t},\overrightarrow{q}_{Cent}\right)} \tag{30}$$

Ahora bien, para la selección de términos candidatos se utilizó *pseudo-relevance feedback*. La primera fase de expansión consiste en utilizar el modelo base del sistema para encontrar los primeros T documentos con mejor *ranking*, donde T es un parámetro con valor 10 por defecto, pero puede ser ajustado por el usuario. Luego se forma un conjunto con todos los términos que pertenezcan a éstos T documentos y se utiliza el método del centroide para ordenarlos de manera creciente, una vez completado ese ordenamiento se devuelven los primeros K términos más cercanos a la consulta y se usan como si correspondieran a una nueva consulta sobre una variante del modelo base, cuya diferencia es que la recuperación

de documentos se hará usando disyunción en vez de conjunción. Una vez que se obtiene el *ranking* de documentos para ésta fase, se ponderan con los resultados de la primera fase usando la fórmula expuesta en la ecuación 31. Se muestra este proceso de manera gráfica en la figura 14.

$$Rank_{final_{d_i,q}} = (1-\alpha) \cdot Rank_{d_i,q} + (\alpha) \cdot Rank_{d_i,q_{exp}}, \tag{31}$$

Donde α es una constante con valor $0{,}3$ por defecto, pero puede ser cambiada según las necesidades del usuario.

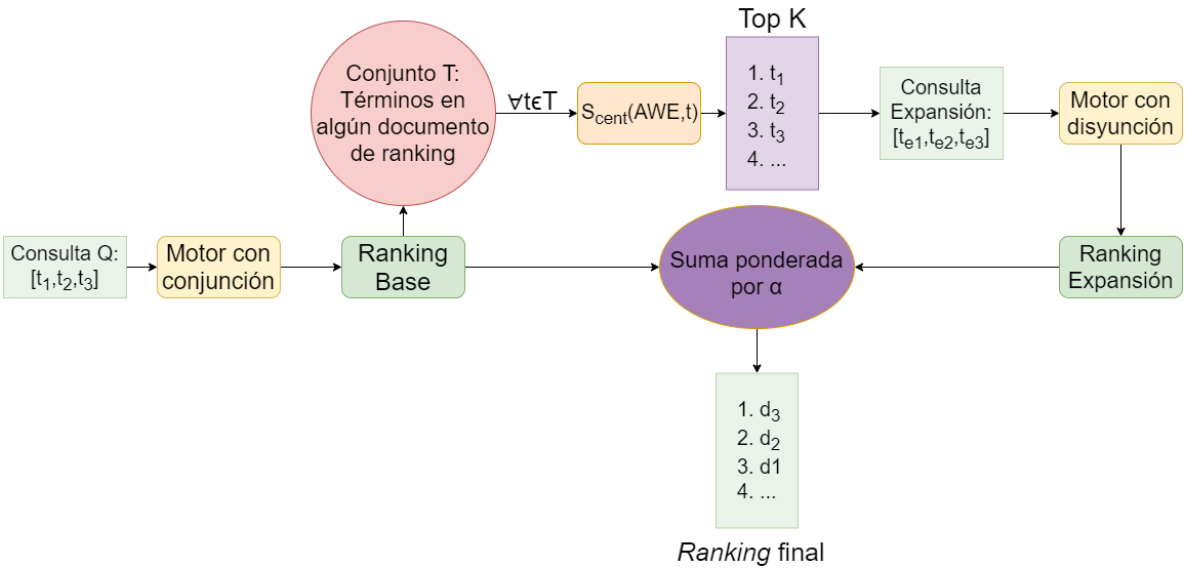


Figura 14: Diagrama de expansión de consultas
Fuente: Elaboración propia.

A modo de ejemplo se muestran a continuación los términos escogidos para la consulta de pruebas #133, utilizando uno de los *embeddings* Skipgrams sobre el corpus SJMN.

```
Query  #133:
['hubbl', 'space', 'telescop']
Query's Extension:
['nasa', 'spacecraft', 'orbit', 'wide-field', 'astronaut']
```

4.3. Resultados experimentales

Debido a lo complejo que es el problema presentado en Recuperación de información, ha sido muy difícil superar los modelos clásicos, que utilizan parámetros calculados empírica-

mente. Es por esto que se les suele llamar *Strong baseline*, en otras palabras, una barrera base difícil de superar. Por esto se deben establecer las métricas sobre los resultados entregados por el modelo base, estos valores se pueden ver en la tabla 5.

Es necesario aclarar que todos los resultados experimentales fueron obtenidos sobre los primeros diez puestos del *ranking*.

Tabla 5: Resultados del modelo base (*Strong Baseline*).
Fuente: Elaboración propia.

Corpus	MAP	AR@10	NDCG@10
AP	22.6672 %	1.4959 %	0.5920
WSJ	21.7948 %	1.5794 %	0.5401
SJMN	12.7336 %	0.3957 %	0.3009

Como se predijo en la sección 4.2.1, los valores de *average recall* no alcanzan a superar el 2 %. Además de éstas consideraciones para *recall*, se debe tener en cuenta que el valor MAP puede verse igualmente afectado por este fenómeno, puesto que las métricas tomarán dentro del conjunto de documentos relevantes aquellos que el modelo jamás podrá rescatar, pero en realidad esto no es nada grave, simplemente se deben comparar los resultados obtenidos para los mismos corpus. Finalmente, el valor NDGC es el más estable para los experimentos en los que sólo se toma en cuenta el *ranking* hasta una posición en particular, que es precisamente la forma en que presentan los resultados en este trabajo. Dicha estabilidad proviene de que considera todos los documentos del *ranking* que se le entrega, no sólo los documentos relevantes, así como su orden real y el orden perfecto en el que deberían aparecer.

Para evitar confusiones se debe aclarar que, a pesar de que NDCG es un valor específico para el *ranking* de una sola consulta, en las tablas de resultados presentadas en ésta sección y las siguientes, se muestra con el mismo nombre el valor promedio de NDCG para las 150 consultas evaluadas en las pruebas.

Por último, se puede discutir que los valores de *average recall* y NDCG son mucho menores para el corpus SJMN en comparación a sus pares, puesto a que las consultas evaluadas fueron las mismas para los tres corpus y probablemente tengan pocos documentos que sean relevantes para ellas. Este argumento se puede sustentar debido a que los resultados de todas las pruebas mostradas más adelante en el trabajo, presentan la misma tendencia.

Habiendo definido todo lo anterior se procede a presentar la lista de modelos NIR sobre los que se realizarán comparaciones:

- Skip-gram entrenado sobre Tipster, con ventana deslizante 5 y sin requerimiento de conteo mínimo (Skip-gram mc1).
- Skip-gram entrenado sobre Tipster, con ventana deslizante 5 y con un conteo de palabras mínimo 3 (Skip-gram mc3).

- CBOW entrenado sobre Tipster, con ventana deslizante 5 y sin requerimiento de conteo mínimo (CBOW mc1).
- CBOW entrenado sobre Tipster, con ventana deslizante 5 y con un conteo de palabras mínimo 3 (CBOW mc3).
- *Word2Vec* ofrecido por Google[15].
- Glove entrenado sobre Wikipedia[16].
- FastText[17].
- ELMo[18].

Se debe aclarar que conteo mínimo significa: la mínima cantidad de veces que debe aparecer un término en todo el corpus para ser reconocido en el vocabulario y por ende tener un *embedding*.

Los tamaños de vocabulario reconocido para los modelos mc3 y mc1, son 279738 y 783244, respectivamente. También se sabe que el vocabulario de Google W2V es de 3000000 de palabras, pero sólo 31433 se tienen en común con los modelos mc3, por último el tamaño del vocabulario de GloVe es de $\approx\ 400000$ términos y 115145 de ellos coinciden con mc3. Se omiten los tamaños de *FastText* y ELMo, ya que por definición pueden crear el *embedding* de cualquier término.

Además los primeros siete modelos tienen una dimensionalidad de 300, mientras que ELMo, por definición, tiene una dimensionalidad de 1024.

4.3.1. Resultados para Expansión de consultas

Antes de hacer cualquier prueba para expansión de consultas entre los modelos, primero se debe sintonizar la cantidad de términos en la expansión (parámetro K). Para esto se utilizó el modelo CBOW mc3 y el corpus SJMN. Los resultados se pueden apreciar en la tabla 6 y los tiempos de ejecución requeridos para estas pruebas se muestran en la figura 15, en la que se puede ver una relación aproximadamente lineal entre el tiempo de ejecución y la cantidad de términos en los que se expanden las consultas.

Tomando en cuenta el tiempo requerido para las pruebas, junto a los resultados obtenidos, para todo el resto de pruebas se utilizará $K = 5$, ya que este valor aumentó de gran forma

[15]Disponibles en `https://code.google.com/archive/p/word2vec/`

[16]Éstos *embeddings* están disponibles en StanfordNLP: `https://github.com/stanfordnlp/GloVe`

[17]Se usó el modelo entrenado sobre Wikipedia. Este modelo se encuentra disponible en: `https://fasttext.cc/docs/en/english-vectors.html`

[18]El modelo utilizado está disponible a través de *Tensorflow Hub*: `https://tfhub.dev/google/elmo/1`

MAP en comparación con el modelo base, al mismo tiempo que aumentó un poco el *recall* y NDCG. A pesar de que con valores más altos de K se puede apreciar un aumento de *recall* y MAP, el precio a pagar por ello es demasiado en cuanto a tiempos de ejecución y NDCG.

Tabla 6: Sintonización de la variable K sobre SJMN usando CBOW mc3.
Fuente: Elaboración propia.

K	MAP	AR@10	NDCG@10
K=5	16.4350 %	0.4055 %	0.3143
K=10	13.9882 %	0.4500 %	0.3188
K=25	15.5460 %	0.4729 %	**0.3234**
K=50	**18.4814** %	**0.4946** %	0.3022

Figura 15: Tiempo de ejecución en función del parámetro K en expansión de consultas
Fuente: Elaboración propia.

Ahora que ya se ha definido la cantidad de términos en los que se expandirán las consultas, se procede a mostrar los resultados obtenidos para cada uno de los corpus.

Tabla 7: Resultados para expansión de consultas sobre el corpus AP.
Fuente: Elaboración propia.

Modelo	MAP	AR@10	NDCG@10
Strong Baseline	22.6672 %	1.4959 %	0.5920
SG mc1	24.1773 %	1.6557 %	0.6101
SG mc3	25.1842 %	1.6450 %	**0.6238**
CBOW mc1	**26.6824** %	1.5467 %	0.6117
CBOW mc3	25.4302 %	**1.6774** %	0.6125
Google W2V	22.2165 %	1.4424 %	0.5789
GloVe	22.7260 %	1.4370 %	0.5755
FastText	22.1853 %	1.5423 %	0.5812
ELMo	23.1704 %	1.6251 %	0.5959

En AP, los modelos que mejor comportamiento exhibieron fueron aquellos entrenados sobre Tipster, puesto a que todos ellos aumentaron varios puntos el valor promedio de NDCG, se podría discutir que los mejores entre ellos corresponden a SG mc3 y CBOW mc1, debido a que aumentaron bastante todas las métricas en comparación a los valores del modelo base consistentemente.

Se puede notar que los modelos con un vocabulario reducido en cuanto a los términos reconocidos por el corpus, Google W2V y Glove, mostraron un mal desempeño que se encuentra por debajo del *strong baseline* en AP. Muy probablemente a causa de la falta de *embeddings* reconocidos en estos modelos, los términos de las expansiones hayan sido de mala calidad, porque se obviaron palabras más cercanas a las consultas originales, pero de las que no se tenía conocimiento.

A pesar de que tanto FastText como ELMo son capaces de reconocer cualquier término, estos presentaron un comportamiento muy similar al modelo base. Probablemente se pueda decir que se comportan mejor que Glove y Google W2V debido a su habilidad de extender sus vocabularios, pero que obtienen peores resultados que SG y CBOW, porque su entrenamiento fue realizado sobre un corpus con un contexto general diferente al de AP. Aunque ELMo tampoco no estuvo bastante lejos de los mejores modelos, debido a que puede ajustar sus *embeddings* al contexto inmediato de las palabras.

Tabla 8: Resultados para expansión de consultas sobre el corpus WSJ.
Fuente: Elaboración propia.

Modelo	MAP	AR@10	NDCG@10
Strong Baseline	21.7948 %	1.5794 %	0.5401
SG mc1	22.5230 %	1.5758 %	0.5850
SG mc3	22.6858 %	**1.6286** %	0.5859
CBOW mc1	**24.4174** %	1.5061 %	0.5504
CBOW mc3	22.5445 %	1.5438 %	0.5681
Google W2V	23.7089 %	1.4543 %	0.5636
GloVe	21.0667 %	1.4982 %	**0.5919**
FastText	20.9705 %	1.4375 %	0.5506
ELMo	21.7948 %	1.4516 %	0.5401

Para este corpus se encontraron resultados similares que en AP, por lo que los argumentos expuestos con anterioridad aún se sostienen. Aunque se puede ver que Glove mostró el mejor valor de NDGC, obtuvo al mismo tiempo uno de los peores valores de *average recall*, así que probablemente haya compensado ésta falta con un mejor orden del *ranking* en las primeras posiciones.

Tabla 9: Resultados para expansión de consultas sobre el corpus SJMN.
Fuente: Elaboración propia.

Modelo	MAP	AR@10	NDCG@10
Strong Baseline	12.7336 %	0.3957 %	0.3009
SG mc1	13.1843 %	**0.4178** %	0.3019
SG mc3	14.7052 %	0.4118 %	0.3008
CBOW mc1	15.5548 %	0.4020 %	0.3124
CBOW mc3	**16.4350** %	0.4055 %	0.3143
Google W2V	14.6447 %	0.3500 %	0.2915
GloVe	15.0826 %	0.4017 %	0.3090
FastText	13.4299 %	0.3361 %	0.3021
ELMo	13.3356 %	0.4055 %	**0.3189**

En los resultados de SJMN se puede ver como CBOW tiene un comportamiento promedio superior a Skip-grams. Esto puede deberse a que en el entrenamiento se tenía una gran cantidad de ejemplos y no le faltó aprendizaje en comparación a SG. Éstas diferencias en cuanto al entrenamiento de ambos modelos W2V se explicaron en la sección 2.2.1.

Se puede notar la eficacia del modelo ELMo, comparable con CBOW para este corpus, que si bien bajó su *precision*, compensó ésta falencia con un aumento en NDCG y un comportamiento promedio en *average recall*, superando a casi todos los demás modelos en ambas

métricas. Probablemente el contexto general de su corpus de entrenamiento sea más cercano a SJMN que al de WSJ y AP.

Habiendo discutido los resultados en todos los corpus se puede argumentar sobre la importancia del contexto de entrenamiento para encontrar buenos términos de expansión, debido a que de manera constante los modelos entrenados sobre Tipster, en particular ambos CBOW, exhibieron mejores resultados que el resto.

4.3.2. Resultados para Recuperación por AWE

Para ésta sección se utilizó el parámetro Top por defecto, es decir, de todos los documentos recuperados por disyunción se calcula el *ranking* usando sólo los primeros 50 documentos más cercanos a la consulta, según la similitud coseno.

Debido a que los *embeddings* ELMo deben ser calculados durante el tiempo de ejecución, no es posible presentar resultados de recuperación por AWE utilizándolos. A diferencia de expansión de consultas, donde solo se deben obtener los *embeddings* de un número muy limitado de documentos (por defecto 10), la mayor parte de las consultas en recuperación por AWE requiere ordenar decenas de miles de documentos y cada llamada a ELMo demora como mínimo un segundo en el sistema donde se realizaron las pruebas, de esta forma sólo para obtener los valores de *embeddings* para cada documento a evaluar en una sola consulta, se requiere como mínimo la mayor parte de un día ejecutando una prueba, siendo que en ese tiempo la mayor cantidad de los modelos entregan los resultados sobre las 150 consultas que se utilizaron para las pruebas.

A continuación se presentan los resultados obtenidos para los otros siete modelos sobre los tres corpus utilizados:

Tabla 10: Resultados para recuperación por AWE sobre el corpus AP.
Fuente: Elaboración propia.

Modelo	MAP	AR@10	NDCG@10
Strong Baseline	22.6672 %	1.4959 %	0.5920
SG mc1	23.3658 %	**1.8990** %	0.6227
SG mc3	23.7273 %	1.8718 %	**0.6247**
CBOW mc1	23.4758 %	1.6766 %	0.5964
CBOW mc3	**25.0809** %	1.6858 %	0.6053
Google W2V	21.3067 %	1.1242 %	0.4653
GloVe	23.3103 %	1.2664 %	0.5007
FastText	23.2580 %	1.2912 %	0.5083

En los resultados para AP queda claramente expuesto el fenómeno que se describió al inicio de la sección 4.3. El modelo *FastText* ganó un poco en *precision*, al punto de compararse a los

modelos SG y CBOW, pero disminuyó excesivamente su *average recall* y NDCG en comparación, esto indica que su *precision* está basada en que se recuperaron muy pocos documentos relevantes, pero que posteriormente la función de *ranking* dejó colocados relativamente bien.

Nuevamente los mejores comportamientos encontrados los tienen los modelos SG y CBOW en sus dos variantes mc1 y mc3.

Tabla 11: Resultados para recuperación por AWE sobre el corpus WSJ.
Fuente: Elaboración propia.

Modelo	MAP	AR@10	NDCG@10
Strong Baseline	21.7948 %	1.5794 %	0.5401
SG mc1	25.8732 %	**2.1517** %	**0.7008**
SG mc3	25.6705 %	2.1027 %	0.6974
CBOW mc1	25.5916 %	1.8544 %	0.6505
CBOW mc3	**26.2069** %	1.8413 %	0.6380
Google W2V	21.9326 %	1.2710 %	0.5264
GloVe	23.0830 %	1.4608 %	0.5410
FastText	21.7688 %	1.1992 %	0.5160

Los resultados para WSJ siguen las mismas tendencias que los presentados la tabla 10, pero se marca aun más la superioridad de los modelos SG en cuanto a *recall*. De esta manera, se podría decir que SG tiene mejor comportamiento al evaluar cercanías de documentos completos por medio de AWE, pero CBOW tiene mejores resultados al evaluar cercanía a términos por la forma en que se realizan las comparaciones en expansión de términos. Esto podría explicarse debido al cambio en el objetivo de entrenamiento de los modelos: SG es entrenado intentando predecir el contexto a partir de un término, en cambio en CBOW el objetivo de entrenamiento es predecir una palabra usando el contexto inmediato en el que se encuentra.

Tabla 12: Resultados para recuperación por AWE sobre el corpus SJMN.
Fuente: Elaboración propia.

Modelo	MAP	AR@10	NDCG@10
Strong Baseline	12.7336 %	0.3957 %	0.3009
SG mc1	**21.0972** %	**0.5126** %	0.3400
SG mc3	21.0636 %	0.5051 %	**0.3416**
CBOW mc1	17.7376 %	0.4155 %	0.3173
CBOW mc3	17.3195 %	0.4120 %	0.3144
Google W2V	19.1746 %	0.2734 %	0.2435
GloVe	20.6850 %	0.3649 %	0.2870
FastText	15.8422 %	0.3025 %	0.2661

En el corpus SJMN se pueden ver las mismas tendencias expuestas en los otros corpus, por que ayuda a consolidar los argumentos realizados con anterioridad.

Era de esperarse el bajo desempeño de los modelos Google W2V y GloVe, ya que tienen una capacidad muy limitada de reconocer palabras en el vocabulario de los corpus, lo que afecta directamente a la calidad de representación AWE que pueden generar.

Habiendo discutido todos los resultados tanto para recuperación por AWE como para la expansión de consultas, se puede destacar que el comportamiento entre los modelos mc1 y modelos mc3 es muy similar, por lo que debería escogerse utilizar los modelos mc3 debido a que cuentan con un vocabulario más reducido cuyos *embeddings* deben mantenerse en memoria principal, ahorrando muchos recursos durante la ejecución de las pruebas. Esto pareciera indicar que se pueden obviar términos con una frecuencia demasiado baja sin sufrir una penalización excesiva por hacerlo.

Por último, se puede señalar que debido al mal comportamiento de *FastText*, la información morfológica de las palabras no necesariamente es más útil que la información contextual de los términos para los motores de búsqueda, al menos no a un nivel de procesos básicos como los que se han utilizado en este trabajo. Entonces a no ser que se utilicen métodos más avanzados y especializados en que se haga uso de dicha información, no se mejorarán los resultados obtenidos a partir de este modelo.

CAPÍTULO 5
CONCLUSIONES

Este trabajo tuvo como objetivo principal Desarrollar un *framework* en el que se puedan comparar 5 modelos de *Neural Information Retrieval* distintos, para consultas y corpus en Inglés. Con el fin de lograr este objetivo se definieron todos los conceptos dentro del alcance del trabajo, desde los modelos clásicos de recuperación de información hasta modelos del estado del arte para la representación de palabras en un sistema. Teniendo estos conceptos en consideración se generó una propuesta de trabajo basada en la metodología CRISP-DM con la que se logró desarrollar un *framework* que tiene embebido tres corpus diferentes (AP, WSJ y SJMN), los cuales forman parte de uno de los *datasets* más conocidos dentro del área: Tipster. Además de esto el *framework* entrega la posibilidad no solo de entrenar distintos tipos de modelos ajustando diversos parámetros, sino que también permite integrar y utilizar modelos externos dentro de las restricciones explicitadas en el alcance del trabajo. A raíz de lo anterior se reconoce que el objetivo principal fue cumplido a plenitud.

Para el desarrollo del sistema propuesto se plantearon distintos objetivos específicos. El primero de ellos fue implementar Skip-gram, CBOW, GloVe, *FastText* y ELMo como modelos NIR y *Precision*, *Recall* y NDCG como medidas estándar. Todos los modelos de *Neural Information Retrieval* mencionados pueden ser entrenados, integrados o importados al sistema, mientras que las tres métricas estándar de evaluación están implementadas de manera nativa en el *framework*.

El segundo objetivo específico fue definir y embeber un corpus base para IR en el *framework*, que como ya se ha mencionado en múltiples veces a lo largo del trabajo resultaron ser tres particiones de Tipster especialmente creado para impulsar el estado del arte en IR.

El tercer objetivo menciona la necesidad de diseñar e implementar una arquitectura escalable para el *framework*. Como se ha dicho a lo largo de este documento el sistema desarrollado está compuesto de distintos módulos sobre los que se pueden seguir construyendo métodos y clases para aumentar la capacidad de comparar aún más procesos y modelos. Además la arquitectura tiene implementados métodos para acelerar el procesamiento de las consultas y aprovechar los recursos de la máquina en donde se utilice.

Finalmente, se definió el cuarto objetivo específico: Comparar resultados sobre agregación de términos para expansión de consultas con uso explícito, el cual fue realizado a cabalidad en la sección 4.3.1, donde se entregan resultados exhaustivos sobre la comparación de ocho modelos NIR distintos utilizados para expandir las consultas haciendo uso explícito de los términos agregados a ellas.

Uno de los aportes más importantes de este trabajo es la proposición y posterior desarrollo de un marco de trabajo general en el que se puedan comparar distintos modelos NIR utilizando corpus bien definidos de prueba para ellos, en las que se pueden ajustar diversos

parámetros, aun más de los que se sintonizaron para las pruebas expuestas en este trabajo, mostrando el gran potencial del *framework*. Incluso era necesario contar con publicaciones tipo *survey* que reunieran una gran cantidad de investigaciones y trataran de organizarlos por métodos usados corpus sobre los que se realizaron, por lo que el presente trabajo supone un gran avance en el tema.

A lo largo de los experimentos expuestos en el capítulo anterior se pudieron desarrollar algunos argumentos interesantes, como por ejemplo: la importancia del conjunto de entrenamiento sobre el que se calculan los valores de los *embeddings* y cómo pueden afectar para mejor o para peor a los modelos en el uso de corpus específicos. Junto a lo anterior, se encontró que a pesar de ser modelos altamente similares Skip-grams y CBOW presentan diferencias de comportamiento dependiendo de los procesos en los que se utilicen y de la forma en la que se haga.

Es importante destacar que el alcance de este trabajo se centró en motores de búsqueda que utilizan la representación densa de palabras en base a los *embeddings* de distintos modelos, pero no menciona prácticamente nada sobre otros tipos de aplicaciones como: *learn to rank*, *learn to expand* y *end-to-end*, entre otras. Por lo que se propone implementar éstas técnicas sobre el *framework* o sobre uno similar como trabajo futuro. Se debe tener en cuenta que los algoritmos y procedimientos implementados son básicos en comparación a lo que se puede llegar a realizar en la actualidad, por lo que sería de sumo interés poder implementar soluciones más complejas que la desarrollada durante este trabajo.

Un ejemplo concreto del trabajo futuro propuesto es: cambiar la manera en la que se calculan las representaciones AWE de los documentos para poder utilizar el modelo ELMo, además de utilizar un método diferente al del centroide para definir la cercanía de la consulta al documento. Una mejor manera podría ser encontrar las instancias de los términos en cada documento, definir una ventana de contexto y evaluar todos estos contexto como si fueran sentencias en el cálculo de los *embeddings* ELMo, para luego calcular un AWE sólo a partir de las representaciones de estos *snippets* del documento en función de la consulta realizada. Finalmente, esa misma distancia puede ser utilizada para definir el *ranking* en el que se presentan los documentos, constituyendo un modelo *end-to-end* de recuperación.

Por último, pero no menos importante, se reconocen los múltiples conocimientos adquiridos a lo largo de la carrera que fueron un gran aporte para la elaboración del presente trabajo, cursos como por ejemplo Estadística computacional o Computación científica que entregan conocimientos básicos sobre las herramientas, librerías y conceptos base en el entendimiento a un nivel teórico. En relación a los ramos electivos que fueron tomados, se consideran indispensables Tecnologías de Búsqueda en la Web e Introducción a las Redes Neuronales Artificiales, ya que son cursos introductorios a las temáticas abordadas a lo largo de todo este proyecto y sientan muy buenas bases sobre el área en cuestión, así como también cuentan con experiencias en las que se pueden aplicar los conocimientos de forma práctica, ya sea a través de tareas, laboratorios e incluso competencias en proyectos.

ANEXOS

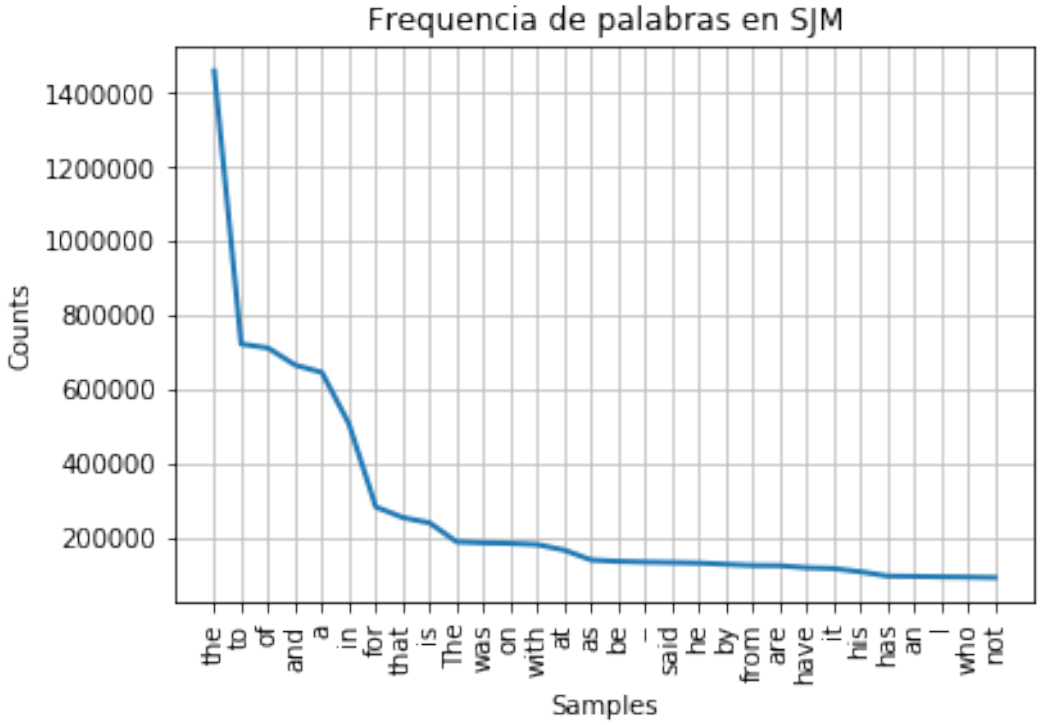


Figura A1: Distribución de frecuencia de las treinta palabras más frecuentes en SJMN.
Fuente: Elaboración propia.

Figura A2: *Wordcloud* de SJMN.
Fuente: Elaboración propia.

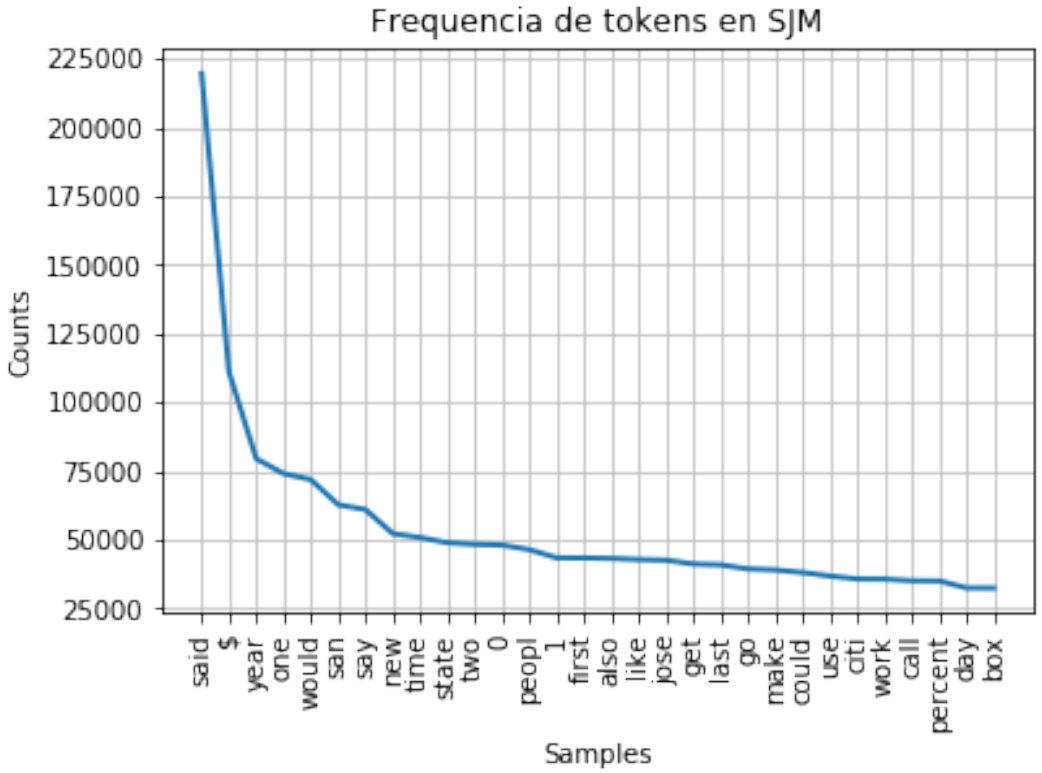


Figura A3: Distribución de frecuencia de los treinta términos más frecuentes en SJMN post-procesamiento.

Fuente: Elaboración propia.

Figura A4: *Wordcloud* de SJMN post-procesamiento.
Fuente: Elaboración propia.

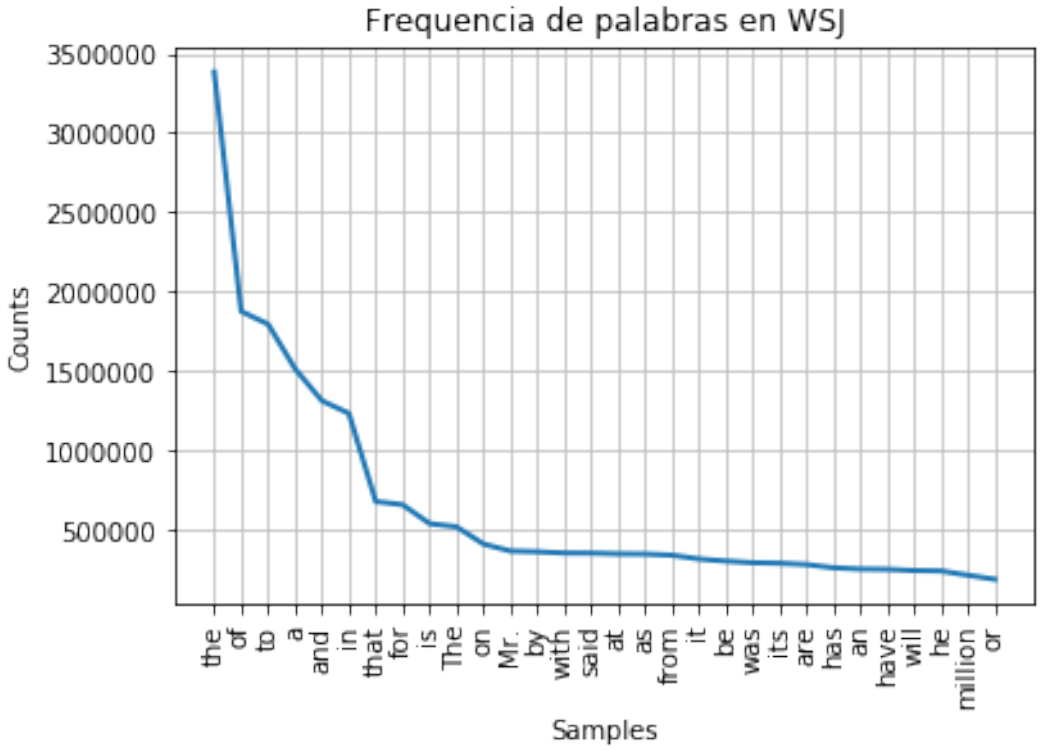


Figura A5: Distribución de frecuencia de las treinta palabras más frecuentes en WSJ.
Fuente: Elaboración propia.

Figura A6: *Wordcloud* de WSJ.
Fuente: Elaboración propia.

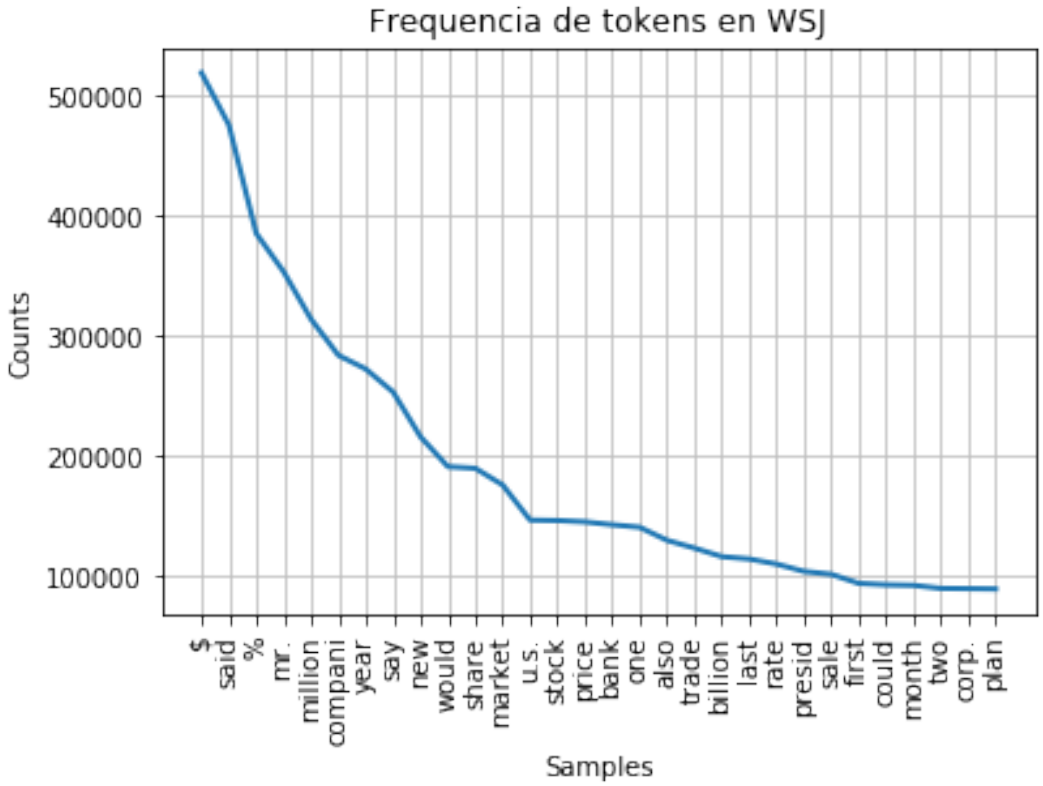


Figura A7: Distribución de frecuencia de los treinta términos más frecuentes en WSJ post-procesamiento.

Fuente: Elaboración propia.

Figura A8: *Wordcloud* de WSJ post-procesamiento.
Fuente: Elaboración propia.

REFERENCIAS BIBLIOGRÁFICAS

[Baeza-Yates y Ribeiro-Neto, 1999] Baeza-Yates, R. A. y Ribeiro-Neto, B. A. (1999). *Modern Information Retrieval*. ACM Press / Addison-Wesley.

[Bojanowski *et al.*, 2017] Bojanowski, P., Grave, E., Joulin, A., y Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

[Carpineto y Romano, 2012] Carpineto, C. y Romano, G. (2012). A survey of automatic query expansion in information retrieval. *ACM Comput. Surv.*, 44(1):1:1–1:50.

[Chapman *et al.*, 2000] Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., y Wirth, R. (2000). Crisp-dm 1.0 step-by-step data mining guide. Technical report, The CRISP-DM consortium.

[Kiros *et al.*, 2014] Kiros, R., Zemel, R. S., y Salakhutdinov, R. (2014). A multiplicative model for learning distributed text-based attribute representations. En *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pp. 2348–2356, Cambridge, MA, USA. MIT Press.

[Kuzi *et al.*, 2016] Kuzi, S., Shtok, A., y Kurland, O. (2016). Query expansion using word embeddings. En *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pp. 1929–1932, New York, NY, USA. ACM.

[Liu, 2011] Liu, B. (2011). *Web data mining: exploring hyperlinks, contents, and usage data*. Springer Science & Business Media, second edicin.

[Mikolov *et al.*, 2013a] Mikolov, T., Chen, K., Corrado, G. S., y Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Mikolov *et al.*, 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., y Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. En *NIPS*, pp. 3111–3119. Curran Associates, Inc.

[Mitra y Craswell, 2018] Mitra, B. y Craswell, N. (2018). An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, pp. 1–117.

[Mitra *et al.*, 2016] Mitra, B., Nalisnick, E. T., Craswell, N., y Caruana, R. (2016). A dual embedding space model for document ranking. *CoRR*, abs/1602.01137.

[Onal *et al.*, 2018] Onal, K. D., Zhang, Y., Altingovde, I. S., Rahman, M. M., Karagoz, P., Braylan, A., Dang, B., Chang, H.-L., Kim, H., McNamara, Q., Angert, A., Banner, E., Khetan, V., McDonnell, T., Nguyen, A. T., Xu, D., Wallace, B. C., de Rijke, M., y Lease, M. (2018). Neural information retrieval: at the end of the early years. *Information Retrieval Journal*, 21(2):111–182.

[Pennington *et al.*, 2014] Pennington, J., Socher, R., y Manning, C. D. (2014). Glove: Global vectors for word representation. En *In EMNLP*.

[Peters *et al.*, 2018] Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., y Zettlemoyer, L. (2018). Deep contextualized word representations. En *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

[Raghavan, 2007] Raghavan, H. (2007). *Tandem Learning: A Framework for Document Categorization*. Ir.

[Singhal, 2001] Singhal, A. (2001). Modern information retrieval: A brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–43.

[Wang *et al.*, 2013] Wang, Y., Wang, L., Li, Y., He, D., Liu, T., y Chen, W. (2013). A theoretical analysis of NDCG type ranking measures. *CoRR*, abs/1304.6480.

[Young *et al.*, 2017] Young, T., Hazarika, D., Poria, S., y Cambria, E. (2017). Recent trends in deep learning based natural language processing. *CoRR*, abs/1708.02709.