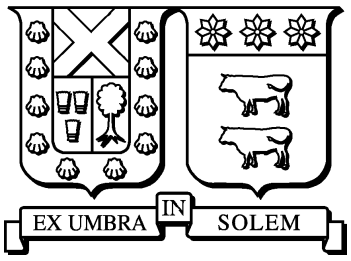
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INFORMÁTICA

SANTIAGO – CHILE



"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA BASADO EN MODELAMIENTO DE SECUENCIAS PARA HASHING SEMÁNTICO DE TEXTO"

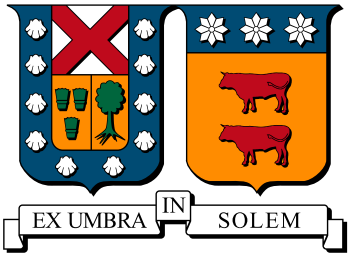
DAVID GUSTAVO FRANKENBERG CASTRO

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: RICARDO ÑANCULEF

JUNIO 2019

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO – CHILE



"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA BASADO EN MODELAMIENTO DE SECUENCIAS PARA HASHING SEMÁNTICO DE TEXTO"

DAVID GUSTAVO FRANKENBERG CASTRO

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: RICARDO ÑANCULEF
PROFESOR CORREFERENTE: MARCELO MENDOZA

JUNIO 2019

MATERIAL DE REFERENCIA, SU USO NO INVOLUCRA RESPONSABILIDAD DEL AUTOR O DE LA INSTITUCIÓN

Dedicado a mis padres,

Danya y Gabriel,

cuyo amor y apoyo incondicional

me ha permitido llegar hasta aquí.

Agradecimientos

Ha llegado el momento de agradecer a todos aquellos que de una forma u otra han contribuido a formarme como persona. Parte de este trabajo también les pertenece.

Para comenzar mis padres, Gabriel y Danya. Gracias por su paciencia, entrega y dedicación. Gracias por inculcarme los valores que ahora poseo y por preocuparse siempre de mi educación. A mis hermanos, Daniel y Ariel. Gracias por hacer más llevaderos los momentos difíciles y por ayudar a forjar mi sentido del humor. A mi familia en general, a los que están y a los que ya no, las palabras no alcanzan, gracias.

A continuación, quisiera agradecer a todas aquellas personas encargadas de mi educación. A mis morim del Instituto Hebreo Dr. Chaim Weizmann por sentar las bases. A la Universidad Técnica Federico Santa María y muy especialmente al departamento de informática (el mejor de todos). A sus funcionarios y profesores, por brindarme sus conocimientos y por hacer de mi estadía en la universidad una grata experiencia. Quiero agradecer especialmente al profesor Ricardo Ñanculef, por su guía y ayuda en la elaboración de esta memoria.

Finalmente, pero no por ello menos importante, quisiera agradecer a mis amigos. A mis amigos de la universidad, los integrantes de tarao y futuros colegas, quiero agradecerles el apoyo y los buenos momentos vividos durante la carrera. Pero por sobre todo quiero agradecerles el hacer de mí una persona más extrovertida. Espero que sigamos con los asados en la casa del Jorge. A los amigos que conocí en la universidad, la calaca flaca, quiero agradecerles el demostrarme que es posible mantener una amistad más allá de la universidad, y las partidas de Smash.

A toda esta gente y a la que no alcance a mencionar, mis más sinceros agradecimientos.

Resumen

Esta memoria aborda el problema de *hashing* para búsqueda por similitud sobre colecciones de documentos escritos. Escasos trabajos han abordado este problema formulando un modelo que aproveche la naturaleza secuencial de los textos. En particular y hasta donde sabemos, aún no se ha estudiado el uso de redes neuronales recurrentes, especializadas en el procesamiento de secuencias. Debido a lo anterior se plantea la hipótesis de que el uso de este tipo de redes puede mejorar el desempeño de las técnicas actuales de *hashing* en términos de precisión en la recuperación.

Para comprobar la hipótesis formulada se implementó una arquitectura de tipo *encoder-decoder* para el procesamiento de secuencias y técnicas de *word embedding* para la representación de texto. Los resultados obtenidos revelan un desempeño excepcional del método sobre conjuntos de datos conformados por documentos breves y un desempeño aceptable en colecciones de documentos extensos.

En base a los resultados obtenidos, se concluye que la efectividad de considerar la estructura secuencial de los textos escritos en tareas de *hashing* es inversamente proporcional a la extensión de los documentos de la colección. Además, se proponen diversas modificaciones a la arquitectura implementada para mejorar su desempeño sobre colecciones de documentos extensos.

Palabras Clave : Hashing; Búsqueda por Similitud; Búsqueda y Recuperación de Información; Redes Neuronales Artificiales; Aprendizaje Automático; Sequence to Sequence Learning; Word Embeddings.

Abstract

This work addresses the problem of hashing for similarity search on collections of text documents. few works have addressed this problem formulating a model that takes advantage of the sequential nature of the texts. As far as we know, the use of recurrent neural networks specialized in the processing of sequences has not yet been studied. Therefore, we hypothesized that the use of this type of networks can improve the performance of current hashing techniques in terms of precision in recovery.

To verify this hypothesis, an encoder-decoder architecture was implemented. For the process we use a recurrent neural networks and word embedding techniques for text representation. The obtained results reveal an exceptional performance of the method on data sets made up of short documents and an acceptable performance in collections of extensive documents.

Based on our experimental results, we conclude that the effectiveness of considering the sequential structure of the texts in hashing tasks is inversely proportional to the extension of the documents in the collection. In addition, various modifications to the architecture are proposed to improve its performance on extensive document collections.

Keywords : *Hashing; Similarity Search; Information Retrieval; Artificial Neural Networks; Machine Learning; Sequence to Sequence Learning; Word Embeddings.*

Índice de Contenidos

Agradecimientos IV

Resumen V

Abstract VI

Índice de Contenidos VII

Lista de Tablas X

Lista de Figuras XI

Glosario XIV

Introducción 1

1. Antecedentes 3

1.1. Aprendizaje Automático 4

1.1.1. Algoritmos de Aprendizaje 4

1.1.2. Tipos de Algoritmos 6

1.1.3. Generalización, Ajuste y Capacidad 7

1.1.4. Flujo de Trabajo en Aprendizaje Automático 11

1.2. Redes Neuronales Artificiales 14
1.2.1. Funciones de Activación 15
1.2.2. Arquitecturas de Red 17
1.3. Procesamiento del Lenguaje Natural 21
1.3.1. Modelos de Lenguaje 22
1.3.2. Representación de Texto 24
1.4. Word Embeddings 26
1.4.1. Basados en Frecuencia 26
1.4.2. Basados en Predicción 30
1.5. Hashing para Búsqueda por Similitud 35
1.5.1. Búsqueda por Similitud 35
1.5.2. Hashing 37
1.6. Métricas de Evaluación 40

2. Estado del Arte **43**
2.1. Repaso Histórico 43
2.2. Técnicas Relacionadas 46

3. Propuesta **50**
3.1. Preentrenamiento 50
3.2. Entrenamiento y Predicción 53

4. Implementación **57**
4.1. Datasets 57
4.2. Preentrenamiento 59
4.3. Entrenamiento 86
4.4. Predicción y Evaluación 97

Conclusiones **106**

A. Anexo **109**

Bibliografía **111**

Índice de cuadros

1.1. Tabla de contingencia para clasificación 42

4.1. Tabla de descripción de *datasets* 58

4.2. Tabla de cobertura de GloVe sobre los *train sets* 86

4.3. Tabla de resultados 100

4.4. Tabla comparativa de resultados para 20 Newsgroups 101

4.5. Tabla comparativa de resultados para SearchSnippets 102

4.6. Tabla comparativa de *embeddings* para SearchSnippets 104

4.7. Tabla comparativa de *embeddings* para 20 Newsgroups 104

Índice de figuras

1.1. Ejemplo de subajuste y sobreajuste en tareas de regresión 10

1.2. Relación típica entre capacidad y error 11

1.3. Diagrama de flujo de trabajo en aprendizaje automático 12

1.4. Ilustración de una neurona biológica 14

1.5. Ilustración de una neurona artificial 15

1.6. Red neuronal *feedforward* 19

1.7. Matriz término-documento 27

1.8. Ventana contextual 28

1.9. Ejemplo de coocurencia en ventana contextual 29

1.10. Modelo skip-gram 32

1.11. Conjuntos en métricas de evaluación. 41

2.1. Diagrama de la arquitectura de HRNH 47

2.2. Diagrama de la arquitectura de SRH 48

3.1. Diagrama de la arquitectura propuesta 56

4.1. Desempeño de un recuperador aleatorio 60

4.2. Reducción de dimensionalidad 20 Newsgroups - tf-idf 61

4.3. Reducción de dimensionalidad SearchSnippets - tf-idf 62

4.4. Histogramas de distancia L_1 sobre tf-idf 63

4.5. *Heatmap* distancia L_1 sobre Newsgroups - tf-idf 64

4.6. *Heatmap* distancia L_1 sobre SearchSnippets - tf-idf 65

4.7. Weiss: mP@100 en función de ϵ. 67

4.8. Reducción de dimensionalidad 20 Newsgroups - Weiss 69

4.9. Reducción de dimensionalidad SearchSnippets - Weiss 69

4.10. *Heatmap* distancia de Hamming sobre 20 Newsgroups - Weiss 70

4.11. *Heatmap* distancia de Hamming sobre SearchSnippets - Weiss 71

4.12. Wang: mP@100 en función de k 73

4.13. Wang: mP@100 en función de σ 74

4.14. Reducción de dimensionalidad 20 Newsgroups - Wang 76

4.15. Reducción de dimensionalidad SearchSnippets - Wang 76

4.16. *Heatmap* distancia de Hamming sobre 20 Newsgroups - Wang 77

4.17. *Heatmap* distancia de Hamming sobre SearchSnippets - Wang 78

4.18. Wang Supervisado: mP@100 en función de a 79

4.19. Wang Supervisado: mP@100 en función de b 80

4.20. Wang Supervisado: mP@100 en función de k 81

4.21. Wang Supervisado: mP@100 en función de σ 81

4.22. Reducción de dimensionalidad 20 Newsgroups - Wang Supervisado 82

4.23. Reducción de dimensionalidad SearchSnippets - Wang Supervisado 83

4.24. *Heatmap* distancia de Hamming sobre 20 Newsgroups - Wang Supervisado 84

4.25. *Heatmap* distancia de Hamming sobre SearchSnippets - Wang Supervisado 85

4.26. Gráfico de frecuencia de palabras de los *training sets* 87

4.27. Gráfico de frecuencia de palabras perdidas de los *training sets* 88

4.28. Distribución de probabilidad de las componentes del *embedding* 89

4.29. Ejemplo de *shift* y *one-hot encoding* sobre un *prehash* 91

4.30. Diagrama del modelo de entrenamiento 94

4.31. *Accuracy* y *Loss* del entrenamiento para 20 Newsgroups 95

4.32. *Accuracy* y *Loss* del entrenamiento para SearchSnippets 96

4.33. Diagrama del modelo de inferencia 98

A.1. Wang: mP@100 en función de c_{ij} 109

A.2. Desempeño de Wang utilizando los parámetros seleccionados 110

A.3. Desempeño de Wang Supervisado utilizando los parámetros seleccionados . 110

Glosario

A

Algoritmo: Conjunto no ambiguo y finito de instrucciones que permiten ejecutar una tarea.

Aprendizaje Automático (*Machine Learning*): Campo dentro de la inteligencia artificial que busca otorgar a las computadoras la habilidad de aprender algo para lo que no han sido explícitamente programadas.

B

***Backend*:** Parte de un sistema informático a la que el usuario no accede directamente, generalmente responsable de almacenar y manipular datos.

***Bag of Words* (Bolsa de Palabras):** Método de representación de texto. Se le llama bolsa porque descarta cualquier información sobre el orden o la estructura de las palabras en un documento, solo considerando la ocurrencia de términos dentro de un vocabulario.

***Big Data*:** Termino que describe a los conjuntos de datos extremadamente grandes, cuyo análisis generalmente se hace mediante técnicas computacionales que permiten revelar patrones, tendencias y relaciones.

C

Ciencias de la Computación (*Computer Science*): Ciencia que estudia el proceso de obtención y manipulación de información a partir de datos mediante el uso de algoritmos, así como la implementación de estos procesos en sistemas computacionales.

***Clustering*:** Proceso de partición de un conjunto de datos en un grupo de subclases significativas llamadas *clusters*.

Colección: Conjunto de documentos.

D

***Dataset*:** Conjunto de datos.

E

***Encoder-Decoder*:** Arquitectura de red en dos partes enfocada en tareas de *sequence to sequence learning*.

G

GloVe: Acrónimo para *Global Vectors*. Es una técnica de aprendizaje no supervisado basado en la coocurrencia de palabras y enfocada en la obtención de una representación vectorial para las palabras (*Word Embedding*).

H

***Hash*:** Cadena de bits que sirve de representación de un objeto más complejo.

***Hashing*:** Proceso de transformación de un objeto complejo a una cadena binaria de longitud fija (*hash*).

Hiperparámetros: Parámetros de configuración de un modelo.

I

***Information Retrieval* (Búsqueda y Recuperación de Información):** Actividad que consiste en encontrar información en colecciones de datos no estructurados para satisfacer una necesidad de información.

Inteligencia Artificial (IA): Capacidad de un sistema informático para realizar tareas comúnmente asociada con la inteligencia humana.

K

Keras: Librería de código abierto para `Python` que provee una API de alto nivel para la implementación de redes neuronales.

L

LSTM: Siglas para *Long Short Term Memory*. Es un tipo especifico de red neuronal recurrente capaz de capturar dependencias a largo plazo.

M

Modelo de Entrenamiento: Modelo utilizado para entrenar la red.

Modelo de Inferencia: Modelo utilizado para realizar la predicción.

Modelo: Función matemática que genera predicciones al encontrar patrones en los datos. Los más conocidos son para tareas de clasificación y regresión.

N

NLP: Siglas para Procesamiento del Lenguaje Natural. Es el campo dentro del aprendizaje automático que trata con el lenguaje natural utilizado por los seres humanos.

NP-*Hard*: Clase de complejidad de los problemas de decisión que son intrínsecamente más difíciles que los que puede resolver una máquina de Turing no determinista en tiempo polinomial.

O

***One-Hot Encoding*:** Codificación de la categoría de un objeto mediante el uso de un vector entero.

***Overfitting* (Sobreajuste):** Fenómeno que ocurre cuando un modelo se ajusta a las peculiaridades de los datos de entrenamiento disminuyendo su poder de generalización.

P

***Padding*:** Es un proceso de normalización de datos en que se rellenan los espacios vacíos utilizando un carácter específico.

PCA: Siglas para Análisis de Componentes Principales. Es una técnica de reducción de dimensionalidad.

***Precision*:** Métrica de evaluación. Corresponde al cociente entre los documentos relevantes recuperados y el total de documentos recuperados.

***Prehash*:** Hashes generados para el conjunto de entrenamientos que se utilizan para entrenar la red.

Python: Lenguaje de programación de alto nivel de propósito general, ampliamente utilizado en aprendizaje automático.

R

Redes Neuronales Artificiales (ANN): Algoritmo de aprendizaje automático bioinspirado, compuesto por capas de neuronas artificiales.

Redes Neuronales Recurrentes (RNN): Redes Neuronales Artificiales con al menos un ciclo que habilita la retroalimentación de la red.

S

***Sequence to Sequence Learning*:** Categoría de aprendizaje que intenta entrenar un modelo para mapear secuencias de entradas a secuencias de salida de diferente largo.

***Similarity Search*:** Tarea de encontrar objetos similares bajo cierta métrica de distancia o medida de similitud.

SpH: Siglas para *Spectral Hashing*.

***Stemming*:** Técnica de preprocesado para texto que intenta acercar palabras relacionadas a un término común.

Stop Words: Palabras sin relevancia semántica que aparecen con gran frecuencia en los textos, tales como artículos, pronombres y preposiciones.

T

TensorFlow: Librería de código abierto para computación numérica y aprendizaje automático.

TF-IDF: Siglas para frecuencia de termino y frecuencia inversa de documento. Es una técnica que evalúa la relevancia de una palabra dentro de un documento.

THC: Sigla para *Text Hashing with Convolutional Neural Networks*.

t-SNE: Sigla para *t-distributed Stochastic Neighbor Embedding*. Es una técnica de reducción de dimensionalidad.

Tuning: Proceso de sintonización de parámetros.

W

Wang Supervisado: Método de preentrenamiento supervisado basado en TCH.

Wang: Método de preentrenamiento no supervisado basado en TCH.

Weiss: Método de preentrenamiento no supervisado basado en SpH.

***Word Embedding*:** Familia de modelos de lenguaje y de selección de características cuyo objetivo es el mapeo contextual de palabras hacia un espacio continuo de menor dimensionalidad.

Introducción

Durante la última década se ha producido un **incremento espectacular en el volumen de datos** generados a nivel mundial. Para poner esto en perspectiva, según un informe publicado por IBM en 2011, el 90 % de los datos a nivel mundial habían sido creados en los últimos dos años [6]. Por su parte un informe elaborado en noviembre de 2018 por la firma de inteligencia de negocios *International Data Corporation* (IDC) en conjunto con el fabricante de discos duros Seagate [2], pronostica un crecimiento de la producción mundial de información, desde los 33 zettabytes en 2018 a los 175 zettabytes en 2025.

Este enorme volumen de información a dado origen al termino *Big Data*, siendo una de las diciplinas encargadas de obtener utilidad de toda esta información, el *Information Retrieval*. Uno de los problemas más conocidos dentro de este campo consiste la búsqueda de elementos similares dentro de una colección de datos. Para tratar este **problema de búsqueda por similitud**, los métodos tradicionales utilizan estructuras de datos basadas en árboles tales como *KD-trees*, *R-trees* y *VP-trees*. Por desgracia este tipo de técnicas no son capaces de escalar a grandes volúmenes de datos de alta dimensionalidad, por lo que otros métodos como *hashing* han crecido en popularidad.

Las técnicas de ***hashing*** permiten representar objetos complejos en cadenas de bits compactas que preservan la similitud entre elementos. Si bien existen dos enfoques principales para abordar el problema de *hashing*, hoy en día la variante **basada en datos** es la más popular debido a la abundancia de **técnicas de aprendizaje automático** que permiten aprender la función de *hash* directamente a partir de los datos.

Es dentro de este marco en el que surge el problema de *hashing* para búsqueda por similitud sobre colecciones de textos escritos abordado en esta memoria. Hasta ahora todas las técnicas de *hashing* estudiadas ignoran el hecho de que el orden de las palabras influye en el significado semántico de una oración, ya sea empleando técnicas de representación de texto que no consideran el orden de los términos como *bag of words* o utilizando arquitecturas que de lleno ignoran esta propiedad. Dado lo anterior **se plantea la hipótesis de que el uso de técnicas de aprendizaje automático especializadas en el procesamiento de secuencias, como las redes neuronales recurrentes, puede mejorar el desempeño de las técnicas actuales de *hashing*** en términos del *precision* de la recuperación.

Por lo tanto, **el objetivo principal de esta memoria es evaluar la conveniencia del uso de un modelo de secuencias (*sequence model*) para realizar *hashing* semántico de texto**. Para lograr esto se abordará la representación de un texto como una secuencia de vectores reales o "palabras" obtenidas a través de la utilización de un ***word embedding***. El modelo de secuencias a utilizar son las redes neuronales recurrentes organizadas en una arquitectura de tipo *encoder-decoder* habitualmente empleada en tareas de traducción automática.

Finalmente se comparará el desempeño de la técnica propuesta con el rendimiento mostrado por otros métodos de *hashing* y se determinará la validez de la hipótesis planteada por esta memoria.

La implementación se realizara sobre `Python 3.5` utilizando las interfaces proveídas por la librería `Keras` y empleando `TensorFlow` como *backend* de redes neuronales.

Capítulo 1

Antecedentes

El propósito de este capítulo es presentar al lector aquellos términos, conceptos e ideas necesarias para leer y comprender esta memoria. Los antecedentes presentados constituyen una breve introducción a aquellos campos de las ciencias de la computación en cuya intersección se encuentra este trabajo. Las áreas abordadas incluyen aprendizaje, redes neuronales artificiales, procesamiento del lenguaje natural, *word embeddings*, búsqueda y recuperación de información, *hashing* para búsqueda por similitud y métricas de evaluación. Si ya posee conocimientos de las áreas anteriormente descritas siéntase libre de dirigirse al siguiente capítulo.

1.1. Aprendizaje Automático

Mejor conocido como ***Machine Learning*** es el campo dentro de la inteligencia artificial que busca "otorgar a las computadoras la habilidad de aprender algo para lo que no han sido explícitamente programadas"[66]. A nivel práctico los **algoritmos de aprendizaje** buscan patrones en colecciones de datos con la finalidad de hacer predicciones y tomar decisiones.

Si bien los fundamentos básicos de la inteligencia artificial y el aprendizaje automático fueron desarrollados a fines de la década de los 40's y principios de los 50's, no fue sino hasta comienzos de este siglo que ambos cobraron verdadera relevancia debido al aumento en el poder de procesamiento y por sobre todo al enorme incremento en la cantidad de datos disponibles [55].

1.1.1. Algoritmos de Aprendizaje

Aunque intuitiva, la definición anterior no especifica lo que se entiende por "aprendizaje" por lo que se requiere una definición más formal para este concepto.

> Un programa de computador se dice que aprende de la experiencia E con respecto a una tarea T y una medida de desempeño P, si es que su desempeño en T, medido por P, mejora con la experiencia E. [54]

Para comprender mejor esta definición de algoritmo de aprendizaje es necesario profundizar en sus tres elementos fundamentales:

1. **La tarea T:** Es aquello que se desea lograr, siendo el aprendizaje el proceso mediante el cual se adquiere la habilidad de desempeñar la tarea. Usualmente se describe en términos de como el sistema de aprendizaje debería procesar un **ejemplo**. Un ejemplo es una **colección de características** cuantificadas del objeto o evento que se desea tratar, generalmente se representa como un vector $\boldsymbol{x} \in \mathbb{R}^n$ en donde la entrada x_i representa la i-ésima característica. Normalmente las técnicas de aprendizaje automático

son aplicadas a problemas que son demasiado difíciles de resolver mediante programas específicos escritos y diseñados por humanos. Por lo general es posible agrupar estos problemas en categorías siendo algunas de las más habituales las tareas de clasificación, regresión, traducción, detección de anomalías, entre otras.

2. **La medida de desempeño P:** Es una medida cuantitativa del desempeño del sistema de aprendizaje al realizar T. Generalmente P es específica a T. Por ejemplo, en las tareas de clasificación la medida de desempeño más utilizada es la exactitud del modelo. Es importante notar que la elección de una métrica de desempeño es crítica ya que determina que tan bien el modelo a aprendido a realizar la tarea T en base a la información que se le ha otorgado. Usualmente se busca que los algoritmos de aprendizaje sean capaces de realizar T incluso sobre datos que no hayan sido vistos anteriormente, para lograr esto se evalúa su desempeño sobre un conjunto de prueba el cual es diferente al conjunto de datos utilizado para el entrenamiento del modelo.

3. **La experiencia E:** Es la fuente del aprendizaje del algoritmo. Por lo general se dice que un algoritmo aprende al experimentar un **set de datos** completo. Un set de datos o *dataset* es una colección de ejemplos. En base a la experiencia que el modelo es capaz de obtener en el proceso, es posible distinguir dos tipos de algoritmos de aprendizaje, los supervisados y los no supervisados, los cuales se revisarán a detalle más adelante.

Bajo esta definición un ejemplo de algoritmo de aprendizaje podría ser un sistema de detección de fraude crediticio. En este contexto, la tarea T del sistema consiste en determinar si una transacción es fraudulenta o legítima. La medida de desempeño P podría ser el porcentaje de transacciones fraudulentas detectadas. Luego el sistema aprende si es que el porcentaje de transacciones fraudulentas que son detectadas aumenta en el tiempo. Por su parte, la experiencia E correspondería al conjunto de transacciones ya procesadas, ya que una vez procesada se sabe a ciencia cierta si una transacción es legítima o no, por lo que se puede usar esta información para que el sistema aprenda [17].

1.1.2. Tipos de Algoritmos

Como fue mencionado anteriormente es posible construir una taxonomía de algoritmos de aprendizaje en base al uso que hacen estos de la experiencia. Si bien existen numerosas categorías, en general es posible clasificar cualquier algoritmo de aprendizaje en alguno de estos tres grupos:

- **Aprendizaje supervisado:** El algoritmo experimenta un *dataset* etiquetado, esto quiere decir que cada ejemplo de entrenamiento viene acompañado de una **etiqueta** correspondiente a la respuesta que se espera del sistema. Con esto se busca que el algoritmo reconozca patrones que le permitan predecir el valor de la etiqueta sobre datos sin rotular. Un ejemplo de algoritmo de aprendizaje supervisado es el sistema de detección de fraude crediticio presentado en la sección anterior. De manera más formal se dice que un algoritmo supervisado observa ejemplos de un vector aleatorio $\mathbf{x}$ y su vector asociado $\mathbf{y}$, para aprender a predecir $\mathbf{y}$ en función de $\mathbf{x}$, usualmente estimando la probabilidad condicional $p(\mathbf{y} \mid \mathbf{x})$ [35]. Este enfoque basado en la utilización de ejemplos etiquetados es la forma más común en la que se construyen los sistemas de aprendizaje automático y se utilizan en su mayoría para realizar tareas de regresión y clasificación. Algunos algoritmos de aprendizaje de este tipo son la regresión logística, las máquinas de soporte vectoriales y los árboles de decisión [13].

- **Aprendizaje no supervisado:** El algoritmo experimenta un *dataset* sin etiquetas. El objetivo es modelar la estructura o distribución subyacente en los datos para aprender propiedad útiles sin la necesidad de guía humana. Esta clase de algoritmos son frecuentemente utilizados sobre datos transaccionales, por ejemplo, pueden usarse para segmentar grupos de clientes con atributos similares para dirigir una campaña de marketing [9]. De manera más formal se dice que un algoritmo no supervisado busca conocer la distribución de probabilidad que genera dicho conjunto de datos, esto significa que el algoritmo observa ejemplos de un vector aleatorio $\mathbf{x}$ e intenta implícita o explícitamente aprender la distribución de probabilidad $p(\mathbf{x})$ o alguna propiedad interesante de dicha distribución [35]. Esta clase de enfoque se utiliza principalmente

para tareas de agrupamiento (*clustering*) y asociación. Algunos ejemplos de algoritmos de aprendizaje no supervisado son el análisis de componentes principales (PCA), el algoritmo *Apriori* y el algoritmo de agrupamiento *k-means*.

- **Aprendizaje Semisupervisado:** Está a medio camino entre el aprendizaje supervisado y no supervisado, por lo que el algoritmo experimenta un *dataset* que está parcialmente etiquetado. Por lo general la cantidad de elementos no etiquetados es muy superior a los etiquetados, siendo el objetivo principal aprender una representación de los elementos de manera tal que los ejemplos de una misma clase tengan una representación similar. Dentro de este contexto el aprendizaje no supervisado ayuda a determinar cómo los ejemplos se agrupan en el espacio de representación [36]. De manera más formal se supone que la distribución de probabilidad $p(\mathbf{x})$ obtenida a partir de los ejemplos no etiquetados aporta información útil para estimar $p(\mathbf{y} \mid \mathbf{x})$. De no cumplirse este supuesto, el uso de aprendizaje semisupervisado no mejora el desempeño sobre el uso únicamente de aprendizaje supervisado [25]. Un ejemplo de la utilización de este tipo de aprendizaje consiste en aplicar PCA sobre los datos como técnica de preprocesado antes de emplear un clasificador sobre la proyección resultante. Cabe señalar que existen otras categorías de aprendizaje relacionadas a esta clase como son el aprendizaje transductivo y el inductivo.

Es importante destacar que elegir entre usar un algoritmo supervisado, no supervisado o semisupervisado depende tanto del problema que se quiera resolver como de la estructura y cantidad de datos disponibles. Dicho lo anterior, es habitual que se mezclen varios enfoques para generar modelos más precisos, utilizando por ejemplo algoritmos de aprendizaje no supervisado para tareas de reducción de dimensionalidad.

1.1.3. Generalización, Ajuste y Capacidad

El desafío principal en aprendizaje automático es generar modelos capaces de realizar predicciones correctas sobre datos que no han sido vistos con anterioridad, en oposición a los datos que se usan para entrenar el modelo [5]. A esta capacidad se le conoce con el nombre de **generalización** y es una característica indispensable para generar sistemas de aprendizaje

automáticos robustos y fiables. Pese a esto, asegurar que el modelo sea capaz de generalizar adecuadamente no es una tarea sencilla. Por lo general para testear la capacidad de generalización de un modelo lo que se hace es dividir el conjunto de datos en dos subconjuntos diferentes los cuales tienen una finalidad bien definida:

1. **Conjunto de entrenamiento (*training set*):** Es el subconjunto que se utiliza para entrenar al modelo. Durante la fase de entrenamiento el algoritmo aplica una medida de distancia entre sus predicciones y los valores correctos contenidos en los ejemplos. A esta diferencia se le conoce como **error de entrenamiento**, siendo esta la cantidad que el algoritmo intenta minimizar en sucesivas iteraciones.

2. **Conjunto de prueba (*test set*):** Es el subconjunto que se utiliza para probar el modelo. La diferencia entre las predicciones del modelo y los valores verdaderos recibe el nombre de **error de prueba**. Esta cantidad es utilizada para definir el **error de generalización** del modelo que es el valor esperado del error para una nueva entrada.

La validez de este enfoque para asegurar la capacidad de generalización del modelo subyace en una serie de supuestos sobre el conjunto de datos. En primer lugar, se asume que tanto el conjunto de prueba como el de entrenamiento se encuentran idénticamente distribuidos, generados por una misma distribución de probabilidad, siendo esta distribución estacionaria, es decir, que no cambia dentro del conjunto de datos. En segundo lugar, se asume que los ejemplos en cada conjunto son independientes entre sí. Estos dos supuestos abreviados como **i.i.d** (independientes e idénticamente distribuidos), permiten describir el proceso de generación de datos mediante la distribución de probabilidad sobre un ejemplo [4, 35].

Dado lo anterior, se creería que la esperanza del error de prueba y la esperanza del error de entrenamiento fueran iguales ya que ambos provienen de la misma distribución de probabilidad. Claramente en la práctica esto no es así ya que los parámetros que definen la distribución son estimados utilizando únicamente el conjunto de entrenamiento, siendo el **ajuste** la medida de que tan bien esta estimación se aproxima a la función objetivo [12]. Debido a esto se tiene que el valor esperado del error de prueba sea en todo momento mayor o igual al valor esperado del error de entrenamiento. De esta manera un buen algoritmo de aprendizaje

debería ser capaz de minimizar el error de entrenamiento manteniendo la brecha entre ambos errores al mínimo. Es precisamente aquí en donde las deficiencias en el ajuste provocan la alteración de alguno de estos factores. Estas alteraciones constituyen los dos principales riesgos en tareas de aprendizaje, que son el **subajuste** (*underfitting*) y el **sobreajuste** (*overfitting*).

- **Subajuste:** Ocurre cuando el modelo no es capaz de ajustarse adecuadamente a los ejemplos de entrenamiento. Esto se traduce en un alto error de entrenamiento y por ende el modelo no es capaz de generalizar sobre nuevos datos. A menudo esto significa que el modelo empleado es demasiado sencillo como para explicar las observaciones, aunque también puede deberse a falta de ejemplos de entrenamiento.

- **Sobreajuste:** Ocurre cuando el modelo se adapta a las peculiaridades de los datos con los que fue entrenado. Esto se traduce en un bajo error de entrenamiento acompañado de un fuerte incremento en la diferencia entre este error y el error de prueba, por lo que el modelo no es capaz de generalizar sobre nuevos datos. Por lo general son los modelos no paramétricos y los modelos no lineales los que son más propensos a verse afectados por el sobreajuste debido a que estos modelos son más flexibles a la hora de aprender la función objetivo [12].

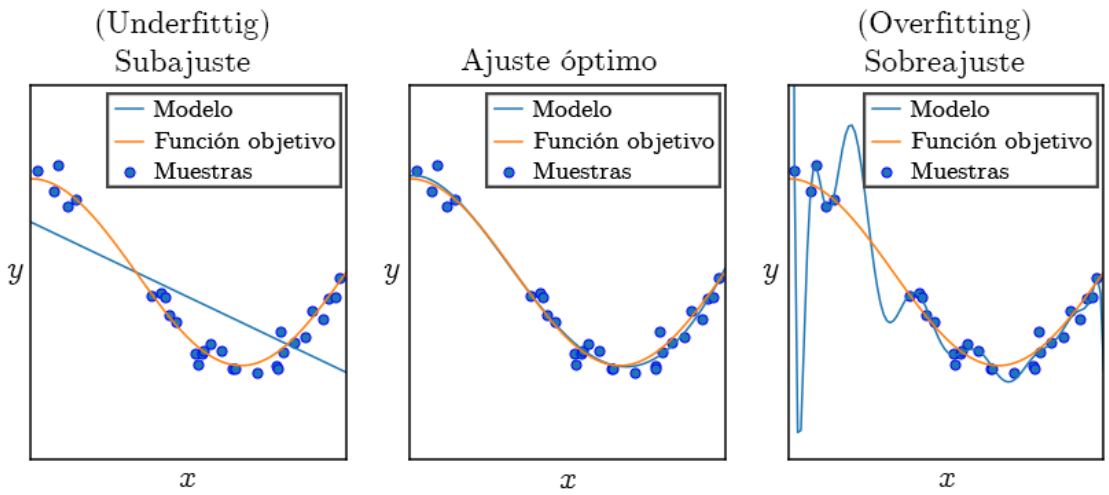


Figura 1.1: Estos tres modelos ejemplifican los problemas relacionados con el ajuste en una tarea de regresión. En todos los casos se entreno al modelo utilizando el mismo conjunto de entrenamiento, generado a partir de la función objetivo con la introducción de ruido. En el cuadro de la izquierda se utilizó un regresión lineal, se observa que el modelo no es capaz de ajustarse a los datos, por lo que se esta ante un caso de subajuste. En el cuadro central se utilizó un polinomio de cuarto grado, el cual se ajusta correctamente a la función objetivo. En el cuadro de la derecha se utilizo un polinomio de quinceavo grado, se observa que el modelo se ajusta al ruido desviándose de la función objetivo, por lo que se esta ante un caso de sobreajuste [16].

Debido a que el subajuste es relativamente fácil de detectar, la gran mayoría de esfuerzos en aprendizaje automático se han centrado en evitar el sobreajuste. A través de los años han surgido numerosas metodologías enfocadas en alcanzar este fin, tales como la validación cruzada, detención anticipada, poda, regularización, entre otros [15]. En general todas las técnicas buscan de una forma u otra alterar hasta cierto punto la **capacidad** del modelo. La capacidad del modelo hace referencia a su habilidad de ajustarse a una gran variedad de funciones. Si la capacidad de un modelo es demasiado limitada este tendrá problemas para ajustarse al conjunto de entrenamiento provocando un caso de subajuste. Por el contrario, si un la capacidad de un modelo es demasiado elevada este tenderá a aprender los detalles y el ruido asociado al conjunto de entrenamiento, provocando un caso de sobreajuste [35].

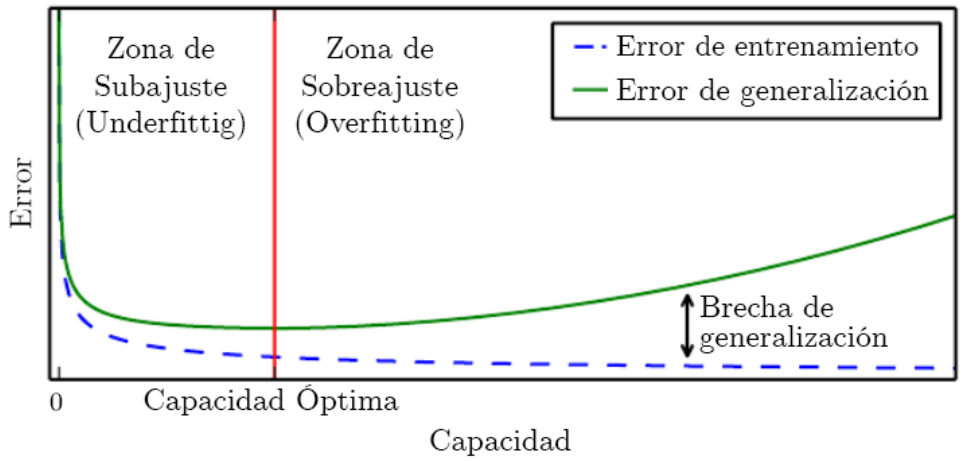


Figura 1.2: Relación típica entre capacidad y error. La línea roja marca la capacidad óptima para el modelo. A la izquierda ambos errores son elevados por lo que el modelo se encuentra subajustado. A la derecha el error de generalización aumenta a un ritmo mucho mayor al de la disminución del error de entrenamiento, por lo que el modelo se encuentra sobreajustado [35].

Finalmente, si bien la presión fundamental en aprendizaje automático es generar modelos que se ajusten correctamente a las observaciones, también se busca que el ajuste de los datos se realice de la manera más simple posible. Este principio se ve reflejado en la navaja de Ockham para el aprendizaje automático que dicta:

> Cuanto menos complejo sea un modelo de aprendizaje automático, más probable será que un buen resultado empírico no se deba simplemente a las peculiaridades de la muestra. [4]

1.1.4. Flujo de Trabajo en Aprendizaje Automático

En las secciones anteriores se ha revisado principalmente el concepto de algoritmo de aprendizaje. Sin embargo, los algoritmos de aprendizaje son solo un componente dentro de los sistemas de aprendizaje, específicamente son el componente encargado de generar el modelo que será utilizado por el sistema para realizar sus predicciones.

Para construir un sistema de aprendizaje automático primero es necesario evaluar el problema que se desea resolver. Como podrá haber deducido es fundamental que este pueda ser

resuelto mediante el descubrimiento de patrones en los datos. Además, es importante definir el objetivo, o en términos previos la "tarea", que se supone que el sistema debe realizar y evaluar si es que existen otros enfoques más directos o simples que permitan resolver el problema. En el caso de utilizar algoritmos de aprendizaje supervisado también es crucial contar con un conjunto de datos lo suficientemente numeroso como para entrenar el modelo de manera adecuada.

Una vez completada la fase anterior es posible comenzar a construir el sistema de aprendizaje. A continuación, se presenta un diagrama que ilustra un flujo de trabajo típico utilizado para construir un sistema de aprendizaje automático [10].

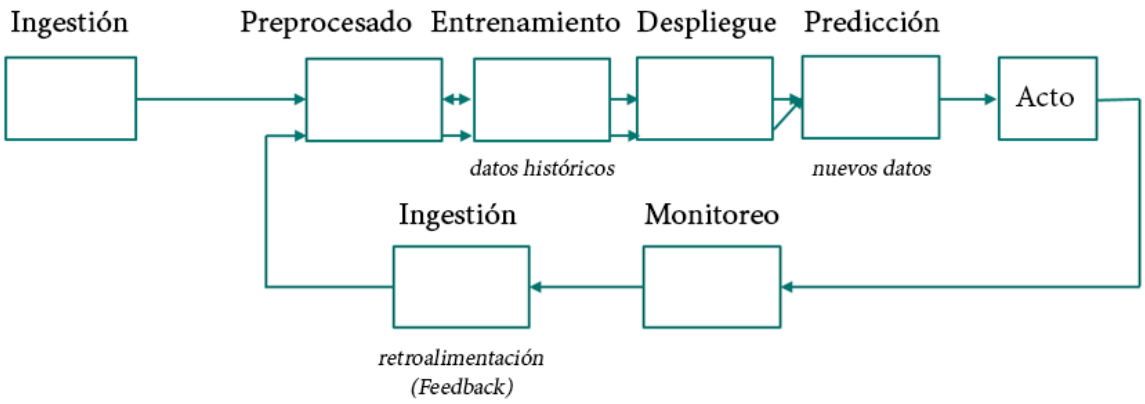


Figura 1.3: Diagrama de flujo de trabajo típico para un sistema de aprendizaje automático [8].

Cabe destacar que el diagrama 1.3 representa solamente un flujo típico, por lo que no es aplicable a todos los sistemas de aprendizaje automático, pudiendo estos contar con más o menos fases que las aquí expuestas.

En primer lugar, esta la fase de **ingestión** en la que se le proporciona al sistema los datos de entrenamiento, seguida de una fase de **preprocesamiento** de datos en dónde estos son formateados de manera de que se adecuen de mejor manera a la necesidades del modelo. Algunas tareas típicas de preprocesamiento incluyen la normalización, la reducción de redundancia, entre otras. A continuación, viene la fase de **entrenamiento** llevada a cabo por los algoritmos de aprendizaje automático. Esta es una fase crítica por lo que es importante evitar caer en los problemas de subajuste o sobreajuste tratados en la sección anterior. En

cuarto lugar, esta la fase de **despliegue** en la que el modelo generado en la fase de entrenamiento es implementado en un ambiente de producción y expuesto a nuevos datos con la finalidad de realizar **predicciones**.

En algunas unas ocasiones se requiere que el sistema **actúe** en consecuencia a la predicción realizada. En sistemas autónomos además existe una fase **monitoreo** en donde se mide el efecto de la acción realizada. Esto con la finalidad de generar un canal de **retroalimentación** o *feedback* para el sistema el cual permite mejorar el modelo continuamente.

En síntesis, los sistemas de aprendizaje automático permiten automatizar el análisis de datos mediante la construcción de modelos. Para ello hacen uso de una arquitectura **iterativa** que permite aprender de los datos de manera adaptativa mejorando la confiabilidad del sistema [9].

1.2. Redes Neuronales Artificiales

Una **red neuronal artificial** (ANN) es un **algoritmo de aprendizaje automático bioinspirado**, que trata de imitar, hasta cierto punto, el funcionamiento de un cerebro biológico. En una red neuronal biológica, cada neurona recibe impulsos eléctricos a través de sus dendritas. Si la neurona es excitada más allá de un **umbral**, envía una señal eléctrica por su axón. Esta última es recibida por las dendritas de otras neuronas, con las que el axón está en contacto mediante unas estructuras denominadas sinapsis [44].

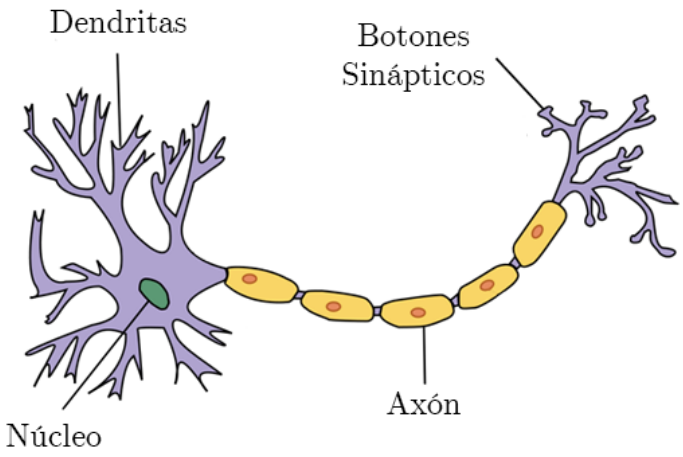


Figura 1.4: Ilustración de una neurona biológica [3].

Para imitar este comportamiento se plantea un modelo idealizado de neurona. En términos generales se dice que una **neurona artificial** es una función no lineal parametrizada [32]. Las variables x_i de la función son llamadas **entradas** (*inputs*), mientras que los parámetros w_i son denominados **pesos** (*weights*). Esta función recibe el nombre de **función de activación** siendo su salida (*output*), por lo regular, una combinación no lineal entre las entradas x_i y los pesos w_i. Por su parte la activación o no de una neurona viene dado por una **función de preactivación** la cual corresponde a una combinación lineal de x_i y w_i. Esta función a veces recibe el nombre de potencial debido a su símil con el modelo biológico.

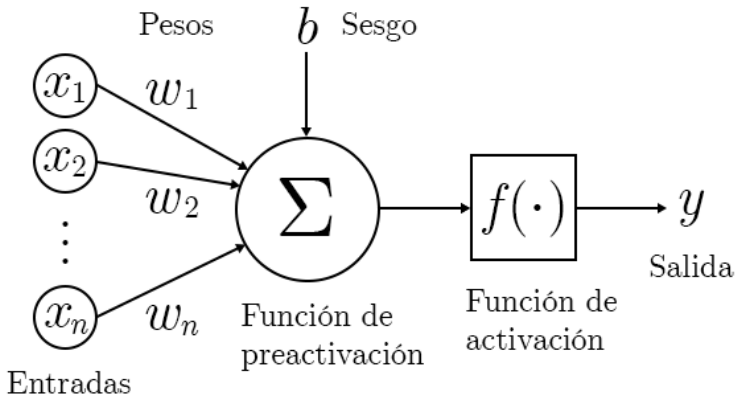


Figura 1.5: Neurona artificial como modelo idealizado y simplificado de una neurona biológica.

Como puede apreciarse en la figura 1.5, la función de preactivación recibe como argumento el vector de entrada $\boldsymbol{x}$. Formalmente, es posible definir esta función o **potencial** $v(\cdot)$ como:

$$v(\boldsymbol{x}) = b + \sum_{i=1}^{n} w_i x_i \tag{1.1}$$

en donde b es un parámetro llamado **sesgo**. Luego se observa que el umbral de activación θ de una neurona es igual al inverso aditivo del sesgo, es decir $\theta = -b$. Finalmente, para completar el modelo de neurona artificial es necesario definir la función de activación $f(\cdot)$, la cual depende de la función $v(\cdot)$ para producir su salida. Este es el tema central de la siguiente sección.

1.2.1. Funciones de Activación

La **función de activación** $f(\cdot)$ de una neurona k, es la encargada de calcular el valor que será transmitido a aquellas neuronas conectadas con k, lo que siguiendo el esquema de la figura 1.5, corresponde a la **salida** y de la neurona. Normalmente las neuronas se nombran según su función de activación. Dado lo anterior se presenta un listado con los tipos neuronas más comunes:

- **Neurona lineal:** Como su nombre lo indica su función de activación es lineal. Es la más simple de las funciones de activación, pero también una de la más limitadas.

La salida de esta neurona es simplemente el valor del potencial $v(\cdot)$ alcanzado por la neurona.

$$f(\boldsymbol{x}) = v(\boldsymbol{x}) \tag{1.2}$$

Por lo general esta función solo es utilizada en situaciones muy específicas como problemas de regresión o en la capa salida.

- **Neurona de umbral binario:** Su función de activación es la función escalón de Heaviside. La salida es 1 si el potencial supera cierto umbral y 0 en cualquier otro caso.

$$f(\boldsymbol{x}) = H(v(\boldsymbol{x})) = \begin{cases} 1 & \text{si} \quad v(\boldsymbol{x}) \geq 0 \\ 0 & \text{si} \quad v(\boldsymbol{x}) < 0 \end{cases} \tag{1.3}$$

Es el tipo de neurona utilizada en los inicios de las redes neuronales, específicamente es la neurona utilizada en el **perceptrón** de Rosenblatt o perceptrón Alfa.

- **Neurona lineal rectificada:** La función de activación de este tipo de neurona es una combinación de las dos anteriores. La salida de la función es 0 si el potencial se encuentra bajo el umbral y es $v(\boldsymbol{x})$ sí lo supera.

$$f(\boldsymbol{x}) = \begin{cases} v(\boldsymbol{x}) & \text{si} \quad v(\boldsymbol{x}) \geq 0 \\ 0 & \text{si} \quad v(\boldsymbol{x}) < 0 \end{cases} \tag{1.4}$$

Esta función es muy utilizada en **redes convolucionales** y en redes profundas en general. Se usa en tareas como visión computarizada, reconocimiento de voz, entre otras.

- **Neurona sigmoidea:** Su función de activación es la función sigmoidea. La salida de esta neurona es la evaluación del potencial $v(\cdot)$ en la función sigmoidea.

$$f(\boldsymbol{x}) = \frac{1}{1 + e^{-v(\boldsymbol{x})}} \tag{1.5}$$

Una de las grandes ventajas de esta función es que es derivable en todo su dominio lo que facilita enormemente la tarea de aprendizaje.

- **Neurona tangente hiperbólica:** Su función de activación es la tangente hiperbólica evaluada en el potencial.

$$f(\boldsymbol{x}) = \tanh(v(\boldsymbol{x})) = \frac{1 - e^{-2v(\boldsymbol{x})}}{1 + e^{-2v(\boldsymbol{x})}} \tag{1.6}$$

Esta función a diferencia de las anteriores acota la salida entre -1 y 1.

- **Neurona binaria estocástica:** Todas las neuronas vistas hasta ahora comparten la propiedad de generar una salida **determinista**. Sin embargo, existen ocasiones en la que es preferible una salida **estocástica**. A ese tipo de neuronas se les suele llamar neuronas estocásticas. Dentro de este tipo de neuronas la más popular es la neurona binaria estocástica. Esta utiliza la misma función sigmoide de la neurona sigmoidea. No obstante, lo que calcula es la probabilidad de que se produzca una salida binaria.

$$p(\mathbf{y} = 1) = \frac{1}{1 + e^{-v(\boldsymbol{x})}} \tag{1.7}$$

Existen más tipos de neuronas como por ejemplo la softmax, la gaussiana, entre otras. Sin embargo, las aquí mencionadas son las más recurrentes en la literatura [56].

1.2.2. Arquitecturas de Red

Una arquitectura de red describe la forma en que las **neuronas** se **organizan** y se **conectan** entre sí. Esta organización determina lo que la red neuronal es capaz de hacer. Dado lo anterior, se tiene que para distintas clases de problemas se suelen utilizar diferentes arquitecturas. A continuación, se presenta la arquitectura de red más común.

Redes Neuronales *Feedforward*

Las redes neuronales de tipo ***feedforward*** constituyen el arquetipo clásico de red neuronal. Fueron el primer modelo en desarrollarse y todavía sirven de base para el resto de las arquitecturas. Como **máquina de aprendizaje**, el objetivo de la red es aproximar una función $y = f^*(\boldsymbol{x})$ mediante la utilización de **ejemplos**. Esto es construir una función $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w})$ y **aprender** el valor de los **parámetros** $\boldsymbol{\theta}$ y $\boldsymbol{w}$ para los cuales $f(\cdot)$ se aproxima de mejor manera a $f^*(\cdot)$. Para construir $f(\cdot)$, esta clase de modelos hace uso de la **composición en cadena** de funciones más simples. Por ejemplo, sería posible definir $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$ en donde $f^{(1)}(\cdot)$ es llamada la primera **capa** de la red, $f^{(2)}(\cdot)$ la segunda y así sucesivamente. La cantidad de funciones o capas que forman la cadena es lo que se denomina **profundidad** (*depth*) del modelo [37]. En base a su ocupación dentro de la red es posible distinguir tres tipos de capas diferentes:

- **Capa de entrada (*input layer*):** Es la encargada de **recibir** los datos a procesar por la red. Esta no realiza **ningún** tipo de **transformación** sobre la entrada, limitándose a proveer de información a la capa siguiente.

- **Capa oculta (*hidden layer*):** Es la capa que sigue a la de entrada. Pese a que pueden existir **numerosas capas** ocultas dentro de una red, el **teorema de aproximación universal** [41] establece que estas no proveen de un mayor **poder de generalización** del que se puede alcanzar con una **única capa**. Debido a esto es común que se hable de una única capa oculta independientemente de la profundidad del modelo. A grande rasgos esta **capa oculta** aplica una serie de **transformaciones** sobre la entrada con el objetivo de producir algo que pueda ser utilizado por la capa de **salida**. Más adelante se profundizará en el funcionamiento de esta capa.

- **Capa de salida (*output layer*):** Es la **última capa** de la red. Por tanto, es la que realiza la **tarea** encomendada al modelo, ya sea clasificación, predicción o alguna otra. Para llevar a cabo este procedimiento, la capa de salida hace uso de los ***outputs*** generados por la **capa oculta**. Dependiendo del problema que la red busca resolver, se suelen utilizar diferentes **funciones de activación** en la **capa de salida**; como por ejemplo la sigmoide en tareas de clasificación, la softmax en clasificación multiclase, la lineal en tareas de regresión, etc [39].

Una de las características más relevantes de las **redes neuronales *feedforward*** es que la información fluye a través de la red en una **única** dirección. Eso quiere decir que **no existe** ningún canal de **retroalimentación** en el que salidas intermedias del modelo sean utilizadas para almacenar información de estados anteriores de la red. Debido a esto se dice que las redes neuronales de tipo *feedforward* no poseen memoria. Cuando a una red neuronal se le permite contar con esos canales de retroalimentación se convierte en una **red neuronal recurrente**. Se hablará de este tipo de redes en secciones posteriores. Debido a esta propiedad y a su organización en forma de capas; es que se suele representar a las redes neuronales *feedfoward* como un **grafo acíclico dirigido**, como el que se presenta a continuación.

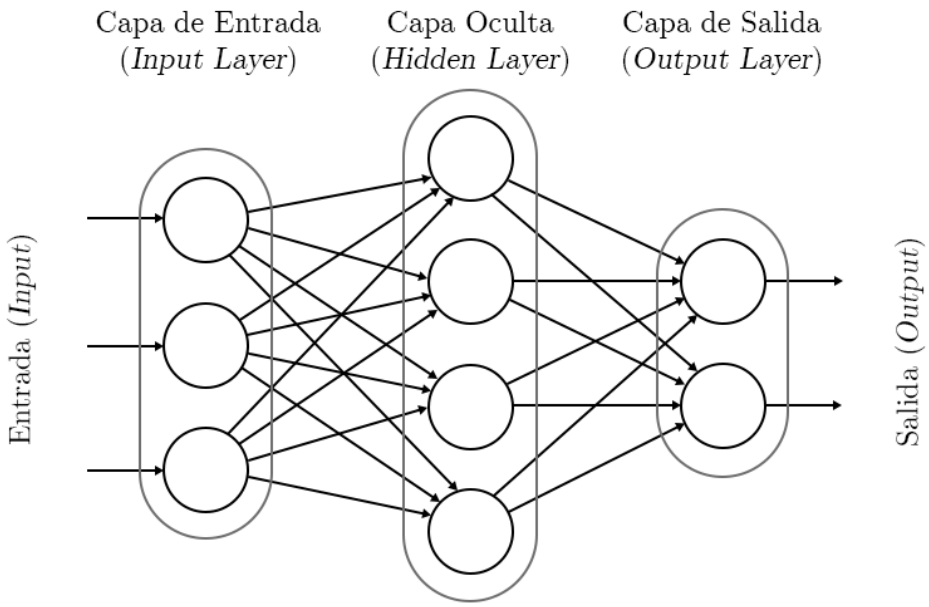


Figura 1.6: Esquema clásico de una red neuronal *feedforward*. Se observa como las neuronas se encuentran organizadas en capas y comunicadas entre sí solamente a las capas adyacentes de manera de que no existen ciclos. La información así solo fluye en la dirección del flujo.

Cada **capa oculta** dentro de la red es típicamente una **función vectorial multivariada** es decir una **función vector-vector**. Para simplificar este hecho a menudo se esquematiza la capa como un **conjunto** de **funciones vector-escalar** actuando en paralelo. Estas funciones vector-escalar corresponden a las **neuronas** presentadas al inicio de esta sección (véase Figura 1.5). El número de neuronas en las capas ocultas determina la **anchura** (*width*) del modelo. Tanto el número de capas como el número de neuronas y su función de activación son determinados por el **hiperparámetro** $\boldsymbol{\theta}$ presentado al comienzo de esta subsección.

Durante la **fase de entrenamiento** de la red, se utilizan los **ejemplos** proporcionados por el **conjunto de entrenamiento** para determinar el **parámetro** $\boldsymbol{w}$ del modelo. Esto es **aprender** los **pesos** asociados a cada neurona de la red, que en conjunto permiten aproximar $f^*(\cdot)$. Para lograr esto, cada **ejemplo** $\boldsymbol{x}$ viene acompañado de una **etiqueta** y que especifica directamente el valor deseado $y \approx f^*(\boldsymbol{x})$ que se espera sea retornado por la **capa de salida**. Sin embargo, en ninguna parte del ejemplo se especifica el comportamiento esperado del resto de capas de la red, por lo que es el **algoritmo de aprendizaje** empleado el que debe determinar cómo utilizar las capas intermedias para lograr este cometido. Es debido a esto que se dice que la

capa está oculta [37]. El algoritmo de entrenamiento más utilizado en redes neuronales es el algoritmo ***backpropagation*** o de **retro propagación**. En resumen, es posible definir a una **red neuronal *feedforward*** como una **función no lineal** de sus entradas, organizada como una cadena (composición) de transformaciones, capa una de las cuales esta compuesta por las funciones individuales de sus **neuronas**.

1.3. Procesamiento del Lenguaje Natural

El **procesamiento del lenguaje natural** (NLP) es el campo dentro de la inteligencia artificial que busca otorgar a las computadoras la capacidad de entender, interpretar y manipular el lenguaje semiestructurado utilizado por los seres humanos. Para alcanzar este objetivo se sirve de diferentes disciplinas tales como la lingüística, las ciencias de la computación y la estadística [60]. El NLP enmarca múltiples tareas de gran relevancia en el mundo moderno como son la traducción automática, el análisis de sentimientos, el reconocimiento del habla, entre otros [11]. Dependiendo de la aplicación que se este desarrollado es posible identificar diferentes procedimiento o fases que componen las tareas de NLP. No obstante, la mayoría de aplicaciones posee alguna de estas tres fases:

- **Modelado del lenguaje (*language modelling*):** Consiste en la creación de un **modelo estadístico** del lenguaje que será usado en etapas posteriores. Es estadístico ya que se centra en la distribución de probabilidad de las palabras en un determinado lenguaje.
- **Etiquetado gramatical (*part-of-speech tagging*):** Consiste en etiquetar las palabras según su función morfológica dentro de la oración. Esta fase utiliza el modelo generado en la etapa anterior en conjunto con parámetros adicionales para clasificar las palabras como sustantivo, adjetivo, verbo, etc.
- **Análisis gramatical (*parsing*):** Consiste en la generación de un árbol de análisis (*parse tree*) que permita evidenciar la relación entre los diferentes componentes que conforman una sentencia. Esta fase es especialmente importante para tratar la ambigüedad presente en el lenguaje natural. Para elaborar el árbol se utilizan gramáticas de libre contexto en conjunto con los resultados obtenidos en las fases anteriores.

Debido a los alcances de esta memoria solo se tratará la fase de **modelado del lenguaje**, ya que es un elemento significativo en el desarrollo de los ***word embeddings*** y por ende en el avance de las **técnicas de representación de texto**.

1.3.1. Modelos de Lenguaje

Entre las principales tareas del NLP se encuentra el desarrollo de **modelos matemáticos** que permitan conocer la estructura del lenguaje. Esta no es una tarea sencilla ya que a diferencia de los lenguajes formales los cuales se definen mediante un conjunto de reglas previamente establecidas, los **lenguajes naturales** emergen a partir de las interacciones humanas, por lo que no existe un conjunto de reglas formales que permitan identificar de manera unívoca si una sentencia pertenece o no al lenguaje [22]. No obstante, contar con **modelos de lenguaje** altamente precisos es una necesidad dado el gran número de tareas dentro de NLP que hacen uso de estos.

De manera formal los **modelos de lenguaje** se basan en el **aprendizaje** de la **distribución de probabilidad conjunta** de que una secuencia de palabras ocurra dentro de un texto. Uno de los modelos más populares es el denominado ***n*-grama** [28] que toma en consideración secuencias conformadas por n palabras consecutivas. De esta manera se intenta calcular la probabilidad de ocurrencia de una palabra w_i dada las $n-1$ palabras previas.

$$p(\, w_i \mid w_{i-1},\ \cdots,\ w_{i-n+1}\,) \tag{1.8}$$

Una forma habitual de estimar esta probabilidad en colecciones de documentos es mediante el **conteo de frecuencias**. Para ilustrar esta idea suponga que se desea estimar la probabilidad de que ocurra la palabra "café" dada la secuencia "tomó una taza de" es decir:

$$p(\, café \mid tomo\ una\ taza\ de\,) \tag{1.9}$$

Esto es equivalente a preguntarse ¿Qué tan a menudo la sentencia "tomo una taza de" va acompañada por la palabra "café" en los documentos de la colección? Este planteamiento surgido a partir de un enfoque **frecuentista** de la probabilidad se traduce en la siguiente expresión:

$$p(\, café \mid tomo\ una\ taza\ de\,) = \frac{C(tomo\ una\ taza\ de\ café)}{C(tomo\ una\ taza\ de\,)} \tag{1.10}$$

En donde la función $C(\cdot)$ retorna la **frecuencia** de la sentencia en la colección de documentos. Si bien este acercamiento funciona bien en dominios acotados, falla a la hora de **generalizar** sus estimaciones a dominios más generales.

El fenómeno anterior se debe a que el lenguaje humano es creativo por lo que siempre es posible generar nuevas sentencias que no están presentes en ningún corpus. Este hecho es en donde radica la dificultad del modelado del lenguaje, dado que una secuencia de palabras utilizada para probar el modelo puede ser completamente diferente a todos los ejemplos dentro del conjunto de entrenamiento [46]. Para generalizar estas estimaciones siguiendo el enfoque frecuentista es necesario eliminar la posibilidad de que una secuencia de palabras tenga frecuencia cero. Para lograr esto es posible replantear el problema de la probabilidad conjunta de una secuencia como el número de veces que dicha secuencia aparece con relación al número total de secuencias de igual longitud. Siguiendo el ejemplo anterior seria, dada todas las posibles secuencias de 5 palabras ¿Cuántas de ellas son "tomo una taza de café"?

Siguiendo esta lógica rápidamente se observa que la **factibilidad** del modelo queda restringido a la selección del parámetro n. Para ejemplificar esto suponga que se desea modelar la distribución probabilidad conjunta de una secuencia de 5 palabras sobre un **vocabulario** de 100000, el número de posibles combinaciones es de $10^{25} - 1$ el cual coincide con el número de **parámetros libres** que el modelo debe aprender. Esta **explosión combinatoria** producto del incremento del parámetro n es una expresión de la llamada **maldición de la dimensionalidad** y es la raíz del problema de generalización anteriormente descrito.

Si bien existen ciertos supuestos que permiten mejorar considerablemente el rendimiento de este método como puede ser la **asunción de Márkov**, que establece que la probabilidad de una palabra solo depende de aquella que la precede, por lo que la ecuación 1.8 puede aproximarse como:

$$p(w_i \mid w_{i-1}, \cdots, w_{i-n+1}) \approx p(w_i \mid w_{i-1}) \tag{1.11}$$

Otra alternativa es modelar el problema utilizando **variables continuas** que, a diferencia de las discretas, permiten que la **función de probabilidad conjunta** que se desea aprender posea propiedades deseables como es ser una función continuamente diferenciable al menos de manera local. Este enfoque posee grandes ventajas al aislar a la distribución de probabilidad en cierta medida de la enorme influencia que tiene el cambio de una variable aleatoria discreta en la distribución de probabilidad debido a la combinatoria. Es aquí en donde surge el concepto de ***word embedding*** esbozado por primera vez por Hinton [42] en 1986, pero cuyo concepto moderno fue introducido por Bengio y compañía en el 2003 [21]. En la siguiente

sección encontrará un análisis detallado de esta técnica.

1.3.2. Representación de Texto

Debido a su naturaleza la forma más eficiente de tratar con el lenguaje natural es a través de **textos escritos**. Dado lo anterior es importante generar **representaciones de texto** que permitan a las computadoras trabajar con él. De forma clásica un texto pasa por una serie de técnicas de **preprocesamiento** [29] que permite eliminar las partes de este que son irrelevantes para el análisis. Las tareas de preprocesamiento más comunes de texto son:

- **Normalización:** Se intenta eliminar elementos del texto que pudiesen representar problemas para el análisis que se realizara más adelante. Generalmente en esta etapa se eliminan caracteres extraños u otros símbolos que se consideran irrelevantes para la tarea a realizar. Típicamente también se llevan todas las palabras a minúsculas de manera de evitar que dos palabras se consideren diferentes solo por la presencia de mayúsculas.
- **Tokenización:** Se divide el texto formando una cadena de palabras.
- **Eliminación de *stop words*:** Se eliminan del texto aquellas palabras que son comunes a cualquier texto y por ende poseen una baja relevancia semántica. Típicamente se trata artículos y pronombres.
- **Lematización/*Stemming*:** Si bien ambos mecanismos persiguen el objetivo de llevar a las palabras a su raíz, lo hacen de manera diferente. La Lematización consiste en utilizar alguna fuente de conocimiento externa para encontrar la raíz real, mientras que el *Stemming* consiste en realizar operaciones que acerquen a las palabras a un término común; un ejemplo claro de esto es eliminar las ultimas letras de un verbo, ya que estás son normalmente utilizadas para definir el tiempo de este.

Una vez realizados los procedimientos anteriores se procede a utilizar una **estructura de representación**. A continuación, se presentan las dos técnicas de representación mas utilizadas en la literatura:

- **Bolsa de palabras (*bag of words*):** Es una representación que describe el texto como ocurrencias de términos conocidos. En otras palabras, se observa el histograma del texto contando el número de veces que aparece un término en determinada oración. La mayor ventaja de este método es su simpleza y en general ha mostrado buenos resultados. Sin embargo, tiende a generar el esparcimiento de la representación ya que en la mayoría de los textos la ocurrencia de palabras es muy baja comparada con el número de palabras en el vocabulario, también ignora el orden de las palabras y las descontextualiza, lo que genera una pérdida de significado.

- ***Word embeddings*:** Es una técnica relativamente reciente que ha propiciado el avance de numerosas tareas de NLP. La idea es representar cada palabra como un *vector de valores reales*. En la siguiente sección se analiza en profundidad esta técnica.

1.4. Word Embeddings

Los ***word embeddings*** son una familia de **modelos de lenguaje** y de selección de características cuyo objetivo es el mapeo contextual de palabras hacia un **espacio continuo** [46]. En otras palabras, la idea consiste en **representar las palabras utilizando vectores reales** cuyas similitudes relativas se correlacionan con la información semántica y sintáctica de las palabras contenidas en una oración.

$$café = \begin{bmatrix} 0{,}1 & 1{,}3 & \cdots & 0{,}4 \end{bmatrix}^T \tag{1.12}$$

Este enfoque se sustenta en la noción de que la **información contextual** por si sola constituye una **representación viable** de los términos lingüísticos, por lo que no es necesario la utilización de repositorios de conocimiento como diccionarios o tesauros [1]. Este acercamiento tan en contraposición con la lingüística tradicional tiene sus raíces en la lingüística estructuralista y en particular en los trabajos de varios investigadores durante la década de los 50's, de entre los que destaca Harris [40] cuya hipótesis radica en la asunción de que palabras pertenecientes a contextos similares tienden a tener significados similares.

Dado lo anterior surgen dos grandes categorías de *embeddings* [67], los basados en coocurrencia también conocidos como ***embeddings* basados en frecuencia** y los ***embeddings* basados en predicción** también conocidos como *neural word embeddings* debido a la preponderancia de esta técnica de aprendizaje.

1.4.1. Basados en Frecuencia

Estos métodos tienen sus raíces en el campo de la **búsqueda y recuperación de información** (IR). Esta disciplina se centra en la búsqueda de información sobre colecciones de datos no estructurados como puede ser el lenguaje natural escrito.

Como su nombre lo indica, estos métodos se basan en la **frecuencia** con que las palabras aparecen dentro de un contexto. De manera más formal se basan en el **análisis estadístico de la coocurrencia** de términos dentro de un **contexto**, siendo este último un parámetro determinante de lo que puede generalizar el modelo. Dependiendo del contexto, es posible clasificar

los modelos como **sintagmáticos** o **paradigmáticos** [46]. Los primeros generalmente consideran como contexto mayores extensiones del texto, capturando relaciones sintácticas entre términos, como por ejemplo la relación entre “bote” y “agua”. Por su parte los modelos paradigmáticos tienden a capturar relaciones del tipo “bote-nave” al basarse mayoritariamente en las palabras y en los contextos locales en que estas ocurren.

En relación con lo anterior otra categorización se basa en las herramientas utilizadas para generar el *embedding* [46], distinguiendo entre aquellos métodos basados en **factorización matricial** de aquellos basados en muestreo por **ventana corrediza**.

A continuación, se presentan algunos métodos de vectores de palabras basados en frecuencia ordenados de acuerdo con su dificultad:

- **Vector de Conteo:** Este método se basa en representar una palabra mediante un **vector** que almacena la **frecuencia del término** en cada uno de los documentos de la colección o si se lo desea en alguna otra subdivisión de éste. En términos de IR corresponde a las filas de la **matriz término-documento**.

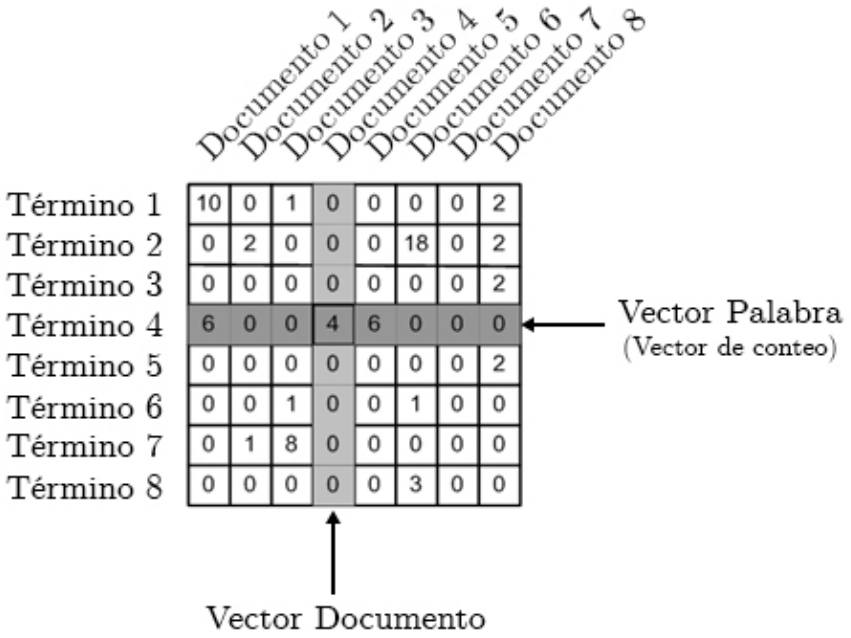


Figura 1.7: Matriz termino-documento que muestra la frecuencia de cada palabra en cada uno de los documentos de la colección [7].

Si bien este método es sencillo de entender posee numerosos problemas. En primer lugar, no es capaz de generalizar relaciones entre términos más allá del contexto que dio origen a la representación, además los vectores suelen ser altamente dimensionales siendo al mismo tiempo poco densos lo que dificulta el trabajo.

- **Vector tf-idf:** Son las siglas en inglés para **frecuencia de término** (tf) y **frecuencia inversa de documento** (idf). En conjunto representan una **medida de la relevancia relativa** de una palabra dentro de una colección, penalizando a aquellas que son comunes a todos los documentos. Su utilización como vector de palabras es parecida a los vectores de conteo, pero a diferencia de este último no solo considera la frecuencia de un término dentro de un documento sino también la frecuencia de este en toda la colección.

$$\text{tf}(t, d) = \frac{\text{frecuencia del término } t \text{ en el documento } d}{\text{número de términos en el documento } d} \tag{1.13}$$

$$\text{idf}(t, D) = \log\left(\frac{\text{número total de documentos dentro de la colección } D}{\text{número de documentos en } D \text{ que contienen el término } t}\right) \tag{1.14}$$

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) \tag{1.15}$$

- **Matriz de Coocurrencia:** Es la variante más representativa de los métodos basados en frecuencia. Su forma más tradicional es la llamada **matriz de coocurrencia con ventana contextual fija**, en donde se distinguen dos elementos principales:

  1. **Coocurrencia:** Se define como el número de veces que un par de palabras w_1 y w_2 aparecen juntas dentro de una **ventana contextual**.
  2. **Ventana Contextual:** Corresponde a la cantidad de palabras en ambas direcciones que conforman el **contexto** de la palabra objetivo, para ejemplificar esto considere la siguiente ilustración:



Figura 1.8: Ventana contextual de largo 2 en torno a la palabra objetivo *"fox"*.

Una vez definida la **colección** y el largo de la **ventana contextual**, se procede a construir la matriz de coocurrencia. Para ejemplificar esto considere el siguiente ejemplo [7].

Sea el corpus sobre el cual se construirá la matriz de coocurrencia la siguiente oración:

"He is not lazy. He is intelligent. He is smart"

Luego se define una ventana contextual de largo dos, al analizar la coocurrencia del primer par de palabras *"He"* e *"Is"* se observa que ambos términos coocurren un total de cuatro veces en el texto:

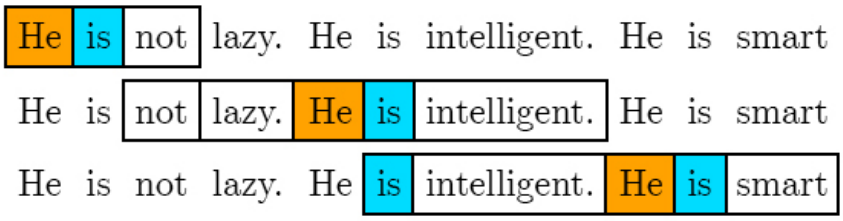


Figura 1.9: Ejemplo de coocurrencia de los términos *"He"* e *"Is"* utilizando la ventana contextual.

Para construir la matriz de coocurrencia simplemente se repite el proceso anterior para cada par de palabras.

$$\begin{array}{c} \\ \text{he} \\ \text{is} \\ \text{not} \\ \text{lazy} \\ \text{intelligent} \\ \text{smart} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{he} & \text{is} & \text{not} & \text{lazy} & \text{intelligent} & \text{smart} \end{array} \\ \begin{bmatrix} 0 & 4 & 2 & 1 & 2 & 1 \\ 4 & 0 & 1 & 2 & 2 & 1 \\ 2 & 1 & 0 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \tag{1.16}$$

De la matriz anterior se extraen algunas conclusiones. En primer lugar, se observa que la matriz resultante es **simétrica** y **dispersa**. Posee $|V|^2$ elementos siendo V el **vocabulario**, es decir el número de términos únicos dentro del corpus. Dado lo anterior, queda claro que incluso para tamaños modestos de V la matriz resultante es extremadamente

grande y difícil de manejar, por lo que es habitual reducir la matriz a un subconjunto de V mediante **técnicas de preprocesamiento** como la remoción de *stop words*, entre otras.

Finalmente, una vez obtenida la matriz de coocurrencia esta es **descompuesta** utilizando técnicas como PCA, SVD entre otras, de manera que los factores resultantes o la combinación de estos sean la representación vectorial de las palabras. Por ejemplo, usando PCA sobre la matriz del ejemplo es posible obtener N componentes principales desde los cuales se pueden escoger un subconjunto de k componentes de manera de generar una nueva matriz de dimensiones $V \times k$ la cual todavía contiene la mayoría de la información contenida en la matriz original.

En conclusión, los métodos de *embeddings* basados en frecuencia son métodos **deterministas** que generan los vectores de palabras a partir de la distribución de los términos a través del corpus o bien en métodos más avanzados mediante la descomposición de matrices. Los últimos poseen la ventaja de estar relacionados a un problema bien definido y estudiado en ciencias de la computación [67]. Sin embargo, requieren de grandes cantidades de memoria y poder de cómputo por lo que en general no escalan bien a no ser que se opte por una arquitectura distribuida.

1.4.2. Basados en Predicción

Estos son métodos **estocásticos** cuya meta es descubrir un conjunto base que permita representar palabras de forma eficiente. Por lo general estos modelos se basan en el uso de **redes neuronales** en donde el *embedding* corresponde a un set de pesos en alguna capa de la red. Existen muchos tipos de arquitectura de red que permiten capturar la información semántica y sintáctica de las palabras a partir de un conjunto de entrenamiento, tales como las redes neuronales recurrentes, las recursivas, las convolucionales, entre otras.

El termino de ***word embedding*** **neuronal** fue acuñado en 2003 por Bengio y su equipo [21]. Su propuesta consistía en combinar una **red neuronal *feedforward*** con un **modelo de lenguaje**. Esto es usar la red para predecir la siguiente palabra de una secuencia, siendo el

embedding un subproducto de este proceso. Bengio identificó tres capas diferentes en su modelo:

1. **Capa de *embedding*:** Es la capa que genera el *embedding*.
2. **Capa intermedia:** Una o más capas encargadas de producir una representación intermedia de la entrada.
3. **Capa softmax:** Corresponde a la capa de salida que genera la distribución de probabilidad sobre las palabras en el vocabulario.

El funcionamiento preciso de estas capas es irrelevante para efectos de esta memoria. No obstante, su mención es necesaria ya que estas todavía se encuentran presentes en las arquitecturas modernas. Si bien esta aproximación representó una verdadera innovación, no estaba exenta de problemas. El mayor problema con esta técnica reside en la capa de salida, debido a que el **costo computacional** de la función softmax es proporcional al número de términos en el vocabulario V el cual por lo general está en el orden de los cientos de miles [63].

Sin embargo, la verdadera revolución en el campo llegó diez años después de la mano de Tomas Mikolov y su equipo [53] que introdujeron un método simple y eficiente capaz de generar *embeddings* de gran calidad, conocido como **Word2Vec**.

Word2Vec

Más que un algoritmo, **word2vec** es un *framework* compuesto por dos arquitecturas diferentes para la creación de *embeddings*, ***skip-gram*** y ***continuous bag of words***. Estos modelos permiten solventar en gran medida los problemas derivados del excesivo costo computacional que afectaban a los planteamientos anteriores proveyendo de métodos rápidos, simples y eficaces para la creación de *embeddings* [30]. Si bien a grandes rasgos ambos modelos consisten en una red neural *feedforward* de dos capas enfocada en la predicción de palabras, difieren en la palabra que se busca predecir cómo se verá a continuación.

- **Skip-gram:** La tarea asignada a esta red es calcular la probabilidad de que una palabra w_k pertenezca a la ventana contextual (ver figura 1.8) dada una palabra objetivo w_j, es decir $p(w_k \mid w_j)$. Pese a esto, lo que realmente importa son los **pesos** de los arcos de la red. Los pesos entre la capa de entrada y la capa oculta constituyen una matriz $\boldsymbol{W}_{|V|\times d}$ que contiene los *embeddings* de las palabras objetivo, luego el *embedding* de w_j es el vector $\boldsymbol{v_j}$ correspondiente a la fila j de la matriz $\boldsymbol{W}$. Mientras que los pesos entre la capa oculta y la capa de salida constituyen una matriz $\boldsymbol{C}_{d\times|V|}$ que contiene los *embeddings* de las palabras de contexto, luego el *embedding* de w_k es el vector $\boldsymbol{c_k}$ correspondiente a la columna k de la matriz $\boldsymbol{C}$.

  Dado lo anterior se aprecia que la red aprende dos *embeddings* separados para cada palabra. Para obtener el *embedding* final existen diferentes opciones. Una alternativa es utilizar solamente los pesos asociados a la matriz $\boldsymbol{W}$, aunque también es posible sumar ambos *embeddings*. En ambos casos el tamaño final del *embedding* queda determinado por el ancho d de la capa oculta, siendo el tamaño de la entrada y la salida igual al tamaño del vocabulario V.

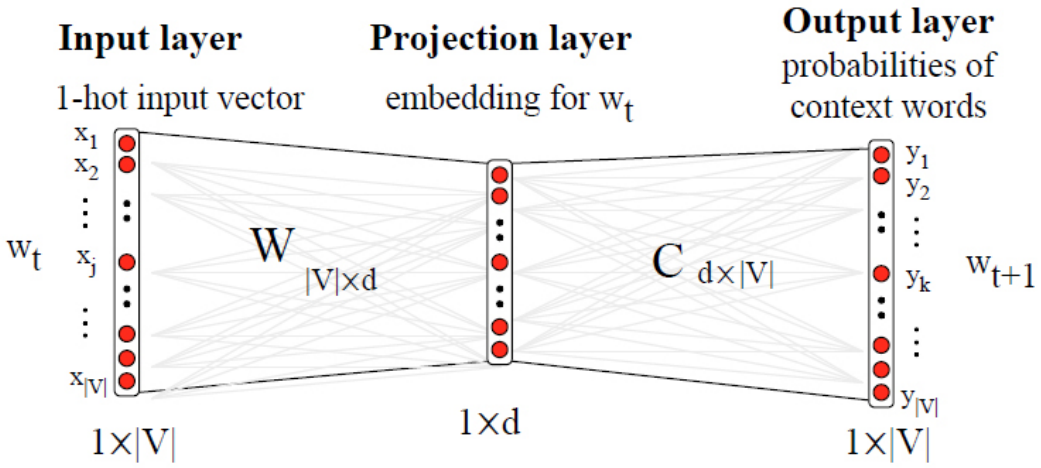


Figura 1.10: modelo skip-gram representado como una red neuronal [30].

Dicho lo anterior, el modelo calcula la probabilidad utilizando el producto punto ente $\boldsymbol{v_j}$ y $\boldsymbol{c_k}$. No obstante, es necesario normalizar esta medida para que refleje una probabilidad. Para lograr esto se utiliza la función softmax.

$$p(w_k \mid w_j) = \frac{\exp(\boldsymbol{c_k^T v_j})}{\sum_{i\in|V|}\exp(\boldsymbol{c_i^T v_j})} \tag{1.17}$$

En otras palabras la red realiza una regresión logística de manera dinámica.

- ***Continuous bag of words* (CBOW):** Si bien este modelo es similar *skip-gram*, la tarea asignada a la red es precisamente la tarea inversa; Predecir la palabra objetivo w_t a partir de las palabras restantes dentro de la **ventana contextual**. Como se vio en la sección sobre NLP, el objetivo de esta red no dista mucho al objetivo de los modelos de lenguaje. Sin embargo, a diferencia de estos últimos **CBOW** no se limita a usar únicamente los términos previos a la palabra objetivo; CBOW utiliza n palabras antes y después de la palabra objetivo para realizar su predicción. En otras palabras, la red máxima la siguiente función [64].

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(\, w_t \mid w_{t-n},\ \cdots,\ w_{t-1},\ w_{t+1},\ \cdots,\ w_{t+n}\,) \tag{1.18}$$

Es importante destacar que, aunque ambos modelos producen *embeddings* similares, difieren en su comportamiento por lo que dependiendo de la tarea que se desee realizar es mejor optar por alguno de los dos.

Gracias a la popularidad de word2vec surgieron nuevos métodos para generar *embeddings* basados en aprendizaje automático. Dentro de estas nuevas propuestas destaca **GloVe** [58] acrónimo de *Global Vectors for Word Representation* propuesto por Pennington y su equipo en 2014.

GloVe

A grandes rasgos este algoritmo es una extensión de word2vec que se aprovecha de la experiencia obtenida de los métodos basados en conteo [23]. A diferencia de word2vec, que se basa en el uso de una ventana contextual, **GloVe** construye de manera explícita una **matriz de coocurrencia** sobre el corpus analizado. De esta manera GloVe considera una dimensión más global del texto mientras aprende la representación.

Formalmente, GloVe trata de minimizar la siguiente función objetivo [57].

$$J_\theta = \frac{1}{2}\sum_{i,j=1}^{T} f(\, p(\, w_j \mid w_i\,))(v_i^T v_j - \log(p(\, w_j \mid w_i\,))^2 \tag{1.19}$$

En donde v_i y v_j son los vectores que representan a las palabras w_i y w_j, además de $f(\cdot)$ que es una función de ponderación que se define manualmente y que debe cumplir una serie de características descritas en [58]. No obstante, también en[58] recomiendan la utilización de la siguiente función:

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{si } \; x < x_{\max} \\ 1 & \text{en cualquier otro caso} \end{cases} \tag{1.20}$$

La ventaja de este método sobre otros es su **escalabilidad**, debido a que solo requiere construir una única vez la matriz de coocurrencia que se utiliza para obtener la probabilidad de $p(\, w_j \mid w_i\,)$.

1.5. Hashing para Búsqueda por Similitud

Esta sección trata sobre el problema de **búsqueda por similitud** abordado mediante técnicas de ***hashing***.

1.5.1. Búsqueda por Similitud

La **búsqueda por similitud** (*similarity search*) consiste en la tarea de encontrar elementos dentro de una colección que son cercanos a una consulta en base a una **métrica de distancia o similitud**. A estos objetos cercanos se les denomina **vecinos** (*neighbors*) por lo que este problema también se conoce con el nombre de búsqueda de vecinos cercanos. En base a la exhaustividad de la búsqueda realizada es posible clasificar el problema en tres subcategorías [69]:

1. **Búsqueda de vecinos exacta:** Dada una consulta q el objetivo es encontrar un subconjunto de vecinos $NN(q)$ de la colección X de manera tal $NN(q) = \arg\min_{x \in X} dist(q, x)$ en donde $dist(q, x)$ es la distancia entre q y x. Una generalización de esta tarea es la k-búsqueda de vecinos (*k-nearest neighbor search*) en donde se buscan los primeros k vecinos más cercanos.

2. **Búsqueda de vecinos aproximada:** A diferencia del punto anterior en donde los vecinos encontrados eran efectivamente los vecinos más cercanos, ahora la búsqueda es aproximada por lo que es posible marcar un vecino como el más cercano cuando en realidad no lo es. Siendo así el objetivo ahora consiste en encontrar los objetos x que cumplan con $dist(q, x) \leq (1 + \epsilon)\, dist(q, x^*)$ en donde x^* es el verdadero vecino más cercano y $\epsilon > 0$ es un factor arbitrario.

3. **Búsqueda aleatorizada de vecinos cercanos:** El objetivo de este enfoque es encontrar los vecinos cercanos de manera no determinista usando probabilidades.

Como se aprecia en los puntos anteriores, la **medida de distancia** juega un rol determinante a la hora de determinar los vecinos más cercanos. Algunas de las distancias más comúnmente

usadas se listan a continuación [18]:

- **Distancia Manhattan:** También conocida como distancia L_1. Sean $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ se define la distancia manhattan como:

$$L_1(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\| = \sum_{i=1}^{d} |x_i - y_i| \tag{1.21}$$

- **Distancia Euclidiana:** También conocida como distancia L_2. Sean $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ se define la distancia euclidiana como:

$$L_2(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2 = \sqrt{\sum_{i=1}^{d} |x_i - y_i|^2} \tag{1.22}$$

- **Distancia Minkowski:** Es una generalización del método utilizado para el cálculo de distancia en los dos puntos anteriores. También se le conoce como distancia L_n en donde el parámetro n determina la raíz utilizada. Sean $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ se define la distancia minkowski como:

$$L_n(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_n = \left(\sum_{i=1}^{d} |x_i - y_i|^n \right)^{\frac{1}{n}} \tag{1.23}$$

- **Distancia de Hamming:** Sean $\boldsymbol{x}, \boldsymbol{y}$ cadenas de bits del mismo largo d se define la distancia de hamming como:

$$\text{Hamming}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{p} x_i \oplus y_i \tag{1.24}$$

  En donde $\oplus$ es el operador de disyunción exclusiva también conocido como desigualdad material, definido como $p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$ donde p y q son proposiciones de verdad.

Por su parte existen ocasiones en que lo más adecuado es usar **medidas de similitud** en vez de métricas de distancia. Las medidas de similitud más frecuentes en la literatura son:

- **Índice de Jaccard:** Sean A y B conjuntos, se define el índice de Jaccard como:

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{1.25}$$

- **Similitud coseno:**Sean $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ se define la similitud de coseno como:

$$\cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| \, ||\boldsymbol{y}||} \tag{1.26}$$

Para afrontar este problema, se han desarrollado numerosos métodos centrados en el uso de **estructuras de datos eficientes**. De manera clásica se han propuesto métodos de indexación basados en árboles como *KD-trees*, *R-trees*, *VP-trees* entre otros. Sin embargo, se ha observado que estas técnicas **no escalan** bien a la hora de tratar con grandes volúmenes de datos de alta dimensionalidad. Es debido a esto que en las últimas décadas técnicas como **Hashing** han crecido en popularidad.

1.5.2. Hashing

El enfoque de esta técnica consiste en representar los objetos de la colección utilizando **cadenas de bits compactas** de manera que elementos similares en el espacio original difieran en solo unos pocos bits en el espacio de representación. Esta representación compacta recibe el nombre de **código *hash*** o simplemente *hash*. De manera formal es necesario definir una **función de *hash*** $h(\cdot)$ que mapee los objetos desde el espacio original al espacio de representación.

$$y = h(\boldsymbol{x}) \tag{1.27}$$

En donde $h(\cdot)$ es la función de *hash*, $\boldsymbol{x}$ es un objeto dentro del **espacio original** e y es el código *hash* asociado a $\boldsymbol{x}$. Usualmente para tareas de **búsqueda por similitud** se suelen utilizar múltiples funciones de *hash* para calcular y, luego se tiene que $y = [y_1 \; y_2 \; \cdots \; y_n]^T$ siendo $h(\boldsymbol{x}) = [h_1(\boldsymbol{x}) \; h_2(\boldsymbol{x}) \; \cdots \; h_n(\boldsymbol{x})]^T$. Existen dos estrategias básicas para utilizar códigos *hash* para abordar el problema de búsqueda por similitud, estos son [69]:

1. ***Hash table lookup:*** La **tabla de *hash*** es una estructura de datos compuesta por ranuras, cada una indexada por un código *hash*. Luego cada referencia y a un elemento x es almacenado en una ranura. A diferencia del enfoque clásico de *hashing* en donde se busca evitar **colisiones**, es decir que dos objetos del espacio de búsqueda sea mapeados por la función de *hash* a la misma ranura, en búsqueda por similitud se intenta

maximizar la probabilidad de colisión entre vecinos cercanos.

2. ***Fast distance approximation:*** Es el equivalente a la **búsqueda exhaustiva** de vecinos más cercanos, pero en el espacio de representación. Este enfoque explota las dos mayores ventajas que otorga la utilización de códigos *hash*. En primer lugar, es mucho más sencillo y rápido calcular la distancia entre los *hash* de los objetos que entre los objetos mismos, debido a que estos pertenecen a un espacio más reducido. En segundo lugar, el tamaño de los códigos *hash* generalmente son mucho menores que los propios objetos a los que representan en el espacio de entrada, por lo que permite realizar las búsquedas de forma más eficiente dentro de una máquina de cómputo real.

La clasificación anterior se basa en la forma en que se eligen a los vecinos más cercanos a una consulta. Sin embargo, tan importante como esto, es la función de *hash* a utilizar ya que es ésta la que determina la efectividad real del sistema. En base a este aspecto, es evidente que la **función de *hash*** debe **preservar la similitud** entre elementos del espacio original. Para alcanzar esta propiedad indispensable para el proceso de búsqueda existen numeras técnicas las cuales se pueden catalogar dentro de dos grandes familias [62]:

1. **Métodos independientes de los datos (*data-independent*):** Como su nombre lo indica, son **métodos independientes de los datos** que se desea mapear al espacio de representación. Por lo general son algoritmos que intentan explotar propiedades de alguna distribución de probabilidad para asegurar que las distancias en el espacio original de búsqueda sean en cierta medida preservadas en el espacio de representación. La más famosa de estas técnicas es el ***Locality Sensitive Hashing*** (LSH) que funciona como *framework* eficiente para la búsqueda aproximada de vecinos cercanos.

2. **Métodos basados en datos *(data-driven)*:** También conocidos como *Learning to Hash* son métodos basados en técnicas de **aprendizaje automático** que buscan aprender la función de *hash* a partir de los datos que se busca transformar. La más representativa de estas técnicas es el ***Semantic Hashing*** fundamentado enteramente en técnicas de aprendizaje no supervisado.

En general, se ha observado que los métodos basados en datos superan ampliamente a los

métodos independientes de datos incluso cuando la similitud del espacio original solo es preservada de manera implícita. Sin embargo, los métodos basados en aprendizaje no supervisado como *semantic hashing* poseen la desventaja de asumir que existe una medida confiable de distancia en el espacio original, supuesto que no siempre es adecuado. Por otra parte, los enfoques basados en aprendizaje supervisado no sufren de esta desventaja al optimizar de manera activa la preservación de la similitud.

1.6. Métricas de Evaluación

Para **evaluar la efectividad** de un sistema de recuperación de información, es necesario establecer ciertas **métricas de evaluación** que reflejen la calidad de las respuestas generadas por el sistema. Para llevar a cabo esta tarea, los enfoques tradicionales se han basado en el uso de **conjuntos de prueba** conformados por [20]:

1. Una **colección de documentos** (objetos).
2. Un conjunto de **necesidades de información**, generalmente expresadas en forma de consulta.
3. Un **set de etiquetas** que establece la relevancia de cada objeto de la colección respecto a una necesidad de información.

La noción de **relevancia** de un documento es el fundamento que sustenta la evaluación. De manera estándar se utiliza un sistema de **clasificación binaria** en donde los objetos de la colección son marcados como relevantes o no relevantes en función de una determinada necesidad de información. A esta clasificación ideal, dada por el conjunto de prueba, se le conoce con el nombre de ***gold standard*** o ***ground truth*** y sirve como punto de comparación contra el cual medir el desempeño de los sistemas de IR.

Dicho lo anterior, considere un requerimiento de información I sobre un conjunto de prueba que contiene R documentos relevantes en relación con I. Además, considere la existencia de un sistema de IR que responde a I generando un set de respuesta A que contiene los documentos recuperados del conjunto de prueba que a juicio del sistema son relevantes. Se observa que los documentos de A considerados relevantes por el *ground truth* son aquellos dentro de la intersección entre A y R. A este conjunto se le denominara como R_a (ver figura 1.11).

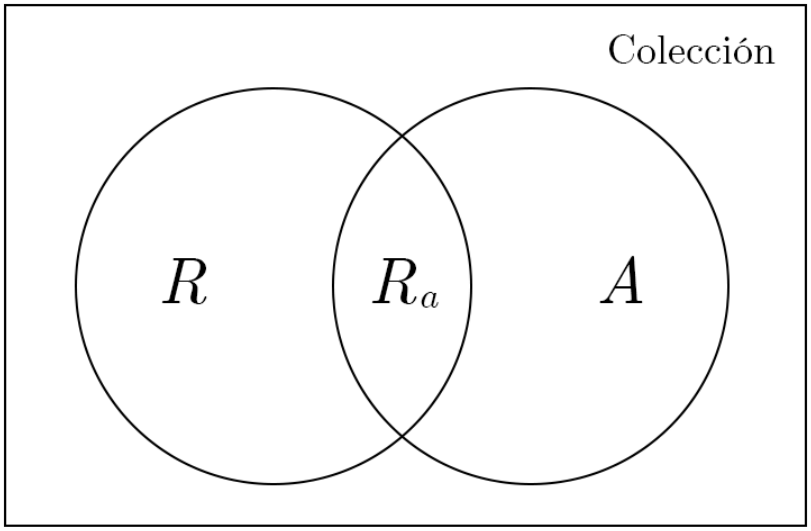


Figura 1.11: Diagrama de Venn de los conjuntos involucrados. R son los documentos relevantes. A son los documentos dentro de la respuesta. $R_a = A \cap R$ son los documentos dentro de la respuesta que son relevantes.

En base a los conjuntos introducidos anteriormente es posible definir intuitivamente dos métricas de evaluación para la respuesta A:

1. ***Precision:*** Es la fracción entre el número de documentos relevantes recuperados $|R_a|$ y el total de documentos recuperados $|A|$. También puede interpretarse como la probabilidad de que un documento sea relevante dado que fue recuperado, es decir $p(R \mid A)$.

$$\text{Precision} = \frac{|R_a|}{|A|} \tag{1.28}$$

2. ***Recall:*** Es la fracción ente el número de documentos relevantes recuperados $|R_a|$ y el total de documentos relevantes $|R|$. También puede interpretarse como la probabilidad de que un documento sea retornado dado que es relevante, es decir $p(A \mid R)$.

$$\text{Recall} = \frac{|R_a|}{|R|} \tag{1.29}$$

Desde un punto de vista **estadístico** [26], es posible reformular las definiciones anteriores utilizando criterios de clasificación (Tabla 1.1).

Cuadro 1.1: Sea d un documento cualquiera perteneciente a la colección. Es posible clasificar los documentos en función de su pertenecía al conjunto de respuesta y a su condición de relevancia. La clasificación y su relación con respecto a la notación anterior está dada por la siguiente tabla de contingencia.

	Relevante $d \in R$	**No relevante** $d \notin R$
Recuperado $d \in A$	TP $d \in R_a$ Verdadero Positivo *True Positive*	FP Falso Positivo *False Positive* **Error de tipo I**
No recuperado $d \notin A$	FN Falso Negativo *False Negative* **Error de tipo II**	VN Verdadero Negativo *True Negative*

Al utilizar este nuevo enfoque las medidas de evaluación *Precision* y *Recall* se reescriben como:

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN} \tag{1.30}$$

A partir de estos conceptos básicos, es posible definir métricas de evaluación más sofisticadas que, de ser necesarias, se introducirán más adelante. Dado que la búsqueda por similitud puede ser entendida como una tarea de IR las mismas métricas definidas para esta clase de tareas pueden ser empleadas con la salvedad de que la necesidad de información ya no se estructura como una consulta clásica, sino que es un elemento de la colección. También se dejará de lado el término documento a favor del término objeto o elemento.

Capítulo 2

Estado del Arte

En este capítulo se expone un breve repaso por los métodos de *hashing* más relevantes a lo largo del tiempo, además de incluir aquellas propuestas que debido a su relación conceptual con el planteamiento del método propuesto merecen ser revisadas.

2.1. Repaso Histórico

Como fue señalado en la sección de antecedentes ítem 1.5.2, las técnicas de *hashing* pueden ser divididas en **métodos independientes de los datos** (*data-independent*) y **métodos basados en datos** (*data-driven*) . Dentro de la primera categoría, una de las técnicas más populares históricamente ha sido ***Locality-Sensitive Hashing*** (LSH) [19]. En esta técnica una familia de funciones *hash* $\mathcal{F} : \mathbb{X} \rightarrow \mathbb{H}^1$ se diseñan para ser localmente sensibles, es decir, preservar una métrica de distancia o similitud en $\mathbb{X}$ con una cierta probabilidad. Finalmente, la función de *hash* $h : \mathbb{X} \rightarrow \mathbb{H}^B$ se obtiene concatenando B funciones independientes entre sí provenientes de $\mathcal{F}$. Un ejemplo clásico es la técnica de *random projections*, que particiona el espacio $\mathbb{X}$ utilizando B hiperplanos aleatorios. Es posible demostrar que esta técnica preserva aproximadamente la distancia Euclidiana con una alta probabilidad. También se han elaborado familias de funciones para otras métricas entre las que se incluyen L_1, Hamming,

Jaccard, similitud coseno, entre otras. No obstante, para poder utilizar esta técnica se hace necesario conocer de antemano métricas aplicables al espacio de características original. Además, **al no distinguir la distribución de los datos en el espacio original, esta técnica tiende a generar *hashes* de mayor longitud lo que limita su desempeño** en comparación con métodos más modernos. Este problema es extensible a cualquier enfoque de este tipo por lo que la mayoría de la investigación actual se centra en los métodos basados en datos.

Los **métodos basados en datos utilizan técnicas de aprendizaje automático supervisado, semisupervisado y no supervisado**, para generar *hashes* de manera eficiente. Quizás, dentro de este último grupo la técnica más conocida, y también una de las más antiguas, sea ***Semantic Hashing*** (SH) [65]. Esta técnica utiliza un modelo generativo profundo basado en el criterio de máxima verosimilitud para ajustar los parámetros de la red a la distribución de los datos, para luego en la capa superior generar los códigos *hash*. Experimentos han demostrado que *sematic hashing* supera ampliamente a LSH en términos de *Precision* y *Recall* en tareas de IR, incluso si el método preserva la similitud del espacio solo de manera implícita [71].

Otra técnica de *hashing* no supervisado es ***Spectral Hashing*** (SpH) [71] la cual es especialmente relevante para esta memoria ya que es la base para los métodos de preentramiento expuestos en el capítulo propuesta sección 3.1. Esta técnica plantea un enfoque mucho más simple que la propuesta anterior, ya que, aborda el problema como uno de partición de grafos en donde los nodos son los elementos del conjunto de entrenamiento y las aristas poseen unos pesos que representa la afinidad entre los elementos. Para calcular dicha afinidad se **utiliza un kernel gaussiano lo que es equivalente a asumir que la similitud entre elementos se correlaciona con la distancia euclidiana**. Finalmente, los *hashes* son obtenidos como soluciones a una versión relajada del problema anteriormente descrito. En ese mismo trabajo el autor comprueba que su método no posee diferencias significativas respecto a *semantic hashing* en términos de resultados en tareas de recuperación de información sobre *datasets* de imágenes de gran escala, pese a ser mucho más simple.

También es posible adaptar métodos independientes de los datos para que aprovechen la información contenida en el dataset. Un ejemplo de esto es ***Kernelized Locality-Sensitive Hashing*** (KLSH) [45] que extiende LSH para convertirlo en un método basado en datos.

Esto se logra proyectando los datos sobre hiperplanos que están definidos sobre una transformación del espacio original $\mathbb{X}$ mediante la utilización de *kernels*.

Algunos otros métodos no supervisados de hashing son; ***Iterative Quantization*** (ITQ) [34] que reduce la dimensionalidad de los datos de entrada aplicando técnicas como PCA y convierte a binario la salida minimizando el error de cuantización. ***Self-Taught Hashing*** (STH) [75] que descompone el proceso de aprender un *hash* en dos pasos: generar el código binario y aprender la función de *hash*. Este enfoque dio origen al denominado ***two-step hashing framework*** (TSH) [48] del cual han surgido propuestas como ***FastHash*** [47] el cual utiliza árboles de decisión para aprender la función de *hash* de manera supervisada, e incluso una versión supervisada de STH denominada **STHs** [74].

Si bien los métodos de aprendizaje no supervisado, superan en su gran mayoría a los métodos independientes de los datos, también adolecen de algunos problemas similares. En primer lugar, se tiene que **esta clase de técnicas asume la existencia de una métrica confiable de similitud o distancia en el espacio original, lo que no es siempre apropiado**, o en su defecto solo aseguran una preservación implícita de la similitud entre elementos. Gran parte de estos problemas se solucionan al utilizar técnicas supervisadas o semisupervisadas ya que el uso de etiquetas o de ejemplos de semejanza ayudan al método a determinar la métrica de similitud o distancia que debe optimizar, incluso si la información que aportan es implícita o está incompleta.

Los **métodos supervisados de *hashing*** preservan activamente la similitud entre elementos, aprendiendo al mismo tiempo la métrica de similitud y la función de *hash*. **Existen principalmente dos enfoques** ligados a la forma en que se etiqueta la información: ***pointwise*** y ***pairwise*** [62]. Los métodos *pointwise* utilizan marcadores de categoría sobre los elementos del conjunto, de manera que la similitud viene dada por la pertenencia a una misma categoría. Como ejemplo, los *dataset* utilizados en esta memoria utilizan este enfoque. Por su parte, las técnicas *pairwise* poseen ejemplos de pares de elementos etiquetados como similares o disímiles. Dentro de los métodos supervisados de *hashing*, existe una fuerte tendencia hacia a la utilización de redes neuronales como método de aprendizaje, lo que se debe a la gran flexibilidad, adaptabilidad y popularidad de han demostrado este tipo de técnicas en años recientes.

Algunas técnicas de *hashing* que utilizan aprendizaje supervisado mediante el uso de redes neuronales artificiales son: ***Deep Hashing*** en su variante supervisada (SDH) [33], la cual utiliza un enfoque *pairwise* y una red profunda, entrenada para minimizar la varianza entre los *hashes* de una misma clase y maximizarla entre clases diferentes; ***Multimodal Similarity-Preserving Hashing*** [52], que también utiliza un enfoque *pairwise* y redes neuronales siamesas para construir los *hashes*; ***Deep Video Hashing*** (DVH) [49], el cual utiliza capas convoluciones para extraer descriptores que luego utiliza en un proceso de fusión temporal mediante capas densas para generar códigos *hash* sobre video.

Ahora bien, pese a que en general los métodos supervisados de *hashing* obtienen mejores resultados que los métodos no supervisados, no siempre es posible disponer de datos etiquetados. Además, el proceso de etiquetado es costoso y normalmente conduce a conjuntos de datos más pequeños que dificultan la generalización del conocimiento extraído. Es debido a lo anterior que **las técnicas de aprendizaje semisupervisadas** cobran especial relevancia, ya que **permiten aprovechar el gran volumen de datos no etiquetados disponibles, junto a datos etiquetados para mejorar el poder de generalización de los algoritmos**.

Los **algoritmos semisupervisados** como su nombre lo indica, están enfocados en la orientación de una técnica no supervisada mediante el uso de un pequeño conjunto de datos etiquetados. Autores como Wang proponen un *framework* de trabajo (SSH) en [70] el que **dividen el proceso en dos pasos**. El primer paso consiste en el establecimiento de un **objetivo supervisado** de la misma forma que lo hacen los algoritmos supervisados anteriormente descritos. En segundo lugar, establece la utilización de un **regularizador** para minimizar la varianza y obtener los códigos *hash* utilizando para ello el *dataset* no supervisado.

2.2. Técnicas Relacionadas

La parte de la propuesta de esta memoria es usar **redes neuronales recurrentes** para realizar *hashing* sobre texto. A continuación, se presentan algunas propuestas que se han realizado utilizado este tipo de redes en la literatura.

Hierarchical Recurrent Neural Hashing for ImageRetrieval With Hierarchical Convolutional Features (HRNH) [50] es una técnica de *hashing* para búsqueda por similitud presentada el año 2017 que utiliza una jerarquía de redes neuronales recurrentes para obtener los códigos *hash*. Este método utiliza en primera instancia capas convolucionales preentrenadas (VGG-F ImageNet) para extraer mapas de características de la imagen. Estas características que componen una representación jerarquizada de la imagen original luego pasan a una red neuronal conformada por dos capas LSTM, más una capa densa y una capa de salida softmax. La primera capa LSTM tiene por objetivo generar un vector de contexto para cada mapa de características para que actúe como sumario de dicho mapa. A continuación, la secuencia de vectores de contexto se utiliza para alimentar la segunda capa LSTM cuya salida pasa a la capa densa, que el autor denomina *hash layer*, y que finalmente retorna los códigos *hash* usando una función de activación softmax.

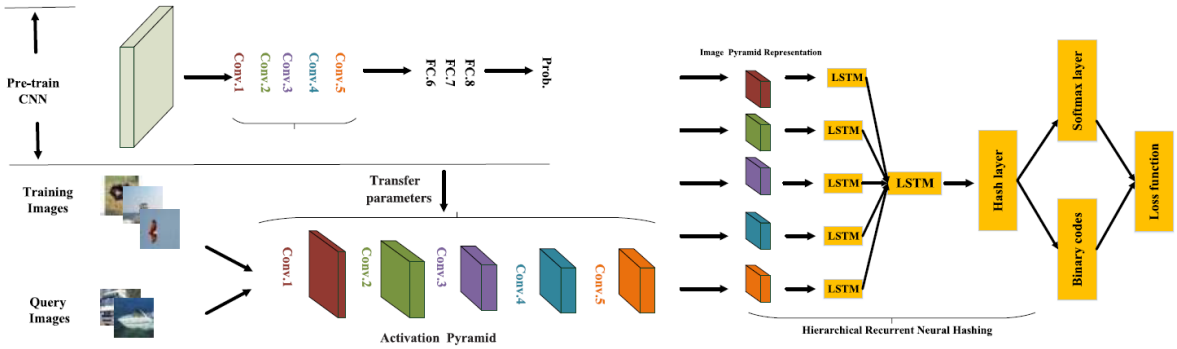


Figura 2.1: Arquitectura propuesta en HRNH. A la izquierda la jerarquía de capas de convolución, a la derecha las redes LSTM [50].

Para los autores, una de las grandes ventajas de este método es que las convoluciones iniciales extraen información relacionada con la distribución espacial de la información mientras que las ultimas extraen información semántica de la imagen. Si bien esta técnica está diseñada para recuperación de información en imágenes, también podría aplicarse a *hashing* de texto, si este es representado como una matriz de datos utilizando técnicas como *word embedding*. No obstante, ya no se podrían usar las capas convolucionales preentrenadas ya que están adaptadas para el reconocimiento de imágenes.

Supervised Recurrent Hashing for Large Scale Video Retrieval (SRH) [38] es otra técnica supervisada de *hashing* para búsqueda por similitud de vídeos presentada el año 2016 que

utiliza redes convolucionales y recurrentes para generar los códigos *hash*. Al igual que el método anterior esta arquitectura utiliza redes convolucionales para generar una representación de los datos. No obstante, a diferencia de la propuesta anterior este método está diseñado para operar sobre video, por lo que cada fotograma es introducido de manera independiente a esta capa. Toda esta información es luego enviada a una red LSTM. Luego una capa *Max-Pooling* integra todos los estados ocultos de la LSTM a un único vector de tamaño fijo. Luego para generar los *hashes* los vectores pasan a una capa densa con función de activación tanh.

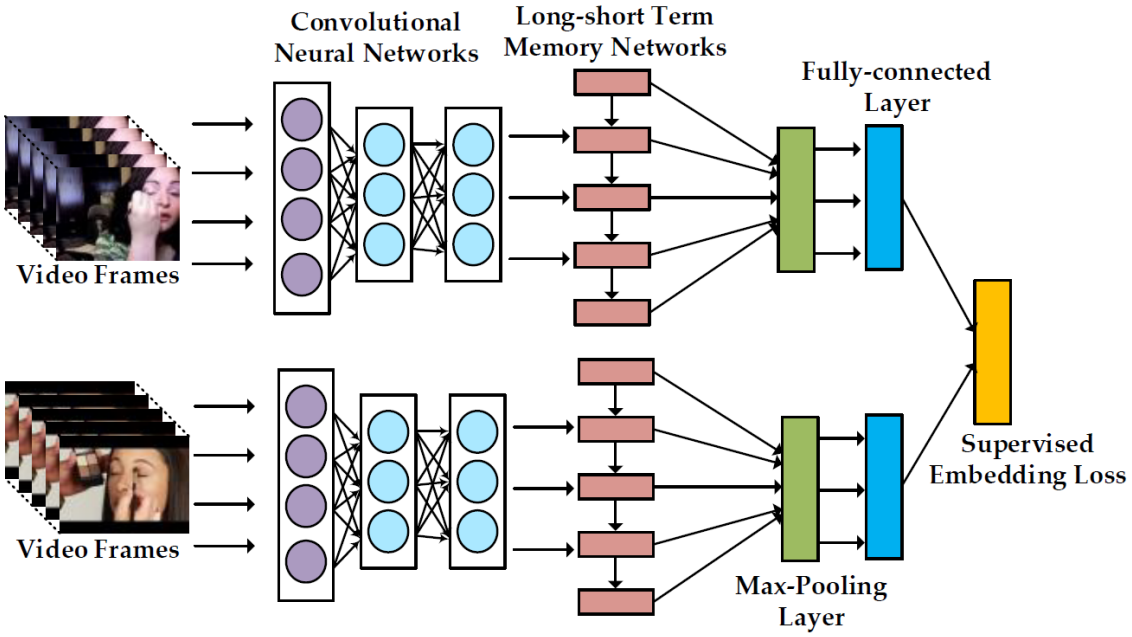


Figura 2.2: Arquitectura propuesta en SRH. Se observa el enfoque *pairwise* en el input del modelo que recibe una pareja de elementos similares [38].

Como se observa en la ilustración, el método utiliza un enfoque *pairwise* ya que se utilizan ejemplos de parejas de vídeos similares para entrenar la red. Finalmente, y dado que los *hashes* generados deben preservar la similitud entre los elementos, se introduce una última capa que predice la similitud entre éstos a partir de los *hashes* y que se puede entrenar a partir de la similitud real usando la función de pérdida cuadrática.

Los autores de este trabajo señalan que el **utilizar redes LSTM les permite capturar relaciones de corto y largo plazo entre los fotogramas de un video**, a diferencia de lo que

ocurre con descriptores generados a mano. Además, comentan que **el uso redes LSTM les permite mejorar de forma significativa el desempeño del método** en comparación a utilizar solo redes convolucionales. Asimismo, concluyen que la introducción de datos etiquetados mejora considerablemente el desempeño.

Los métodos anteriormente descritos son una muestra de como se han utilizado las redes neurales recurrentes para tareas de *hashing* en la literatura. La gran mayoría de las técnicas estudiadas utilizan redes LSTM para condensar la información obtenida por las capas convolucionales que las anteceden, en tareas enfocadas al *hashing* de video o imágenes. **Hasta donde se ha podido observar la propuesta de esta memoria es la primera en utilizar redes neurales recurrentes para el problema de *hashing* sobre textos escritos**.

Capítulo 3

Propuesta

En este capítulo se presenta la **propuesta de esta memoria que consiste en el uso de una red neuronal recurrente para *hashing* de texto**. Específicamente se propone la adaptación de una arquitectura *encoder-decoder* habitualmente utilizada en traducción automática para abordar el problema de *hashing* de texto. Al igual que en los problemas de traducción, en *hashing* se busca trasladar un texto de un domino a otro procurando mantener ciertos atributos del original. La mayor diferencia radica en el tamaño de la cadena de salida y del vocabulario utilizado para generarla. Considerando el problema de *hashing* se tiene un vocabulario binario y una cadena de salida fija. Además, es necesario contar con ejemplos de "traducciones" para entrenar la red, por lo que es necesario dividir el proceso en dos fases.

3.1. Preentrenamiento

El objetivo de esta fase es generar un código binario o ***prehash*** para cada elemento del conjunto de entrenamiento, el cual será utilizado más adelante durante la fase 2 para entrenar a la red neuronal. Es debido a lo anterior que los códigos creados durante esta fase deben preservar en la medida de lo posible la similitud entre elementos en el espacio original, a esta restricción se le conoce como ***locality preserving constraint***. Al igual que se hizo en [73] la técnica utilizada para calcular los *prehash* es una variante del ***Spectral Hashing*** propuesto

en [71] el cual cumple con la restricción anteriormente descrita.

En [71] Weiss y su equipo enumeran los requisitos que todo buen *hash* debe tener cumplir:

1. Ser **fácil de calcular** para nuevos elementos.
2. Ser **compacto**.
3. Cumplir con la ***locality preserving constraint***

Ignorando la primera restricción es posible plantear la elección de los códigos binarios como un **problema de optimización**. Sea $\boldsymbol{B}$ la matriz cuyas filas corresponden a los códigos binarios de dimensión k asociados a los n elementos del conjunto de entrenamiento siendo $\boldsymbol{b_i} = \boldsymbol{B}^T_{i,:}$ y sea $\boldsymbol{W}_{n\times n}$ la **matriz de afinidad** asociada a dichos n elementos, se plantea el siguiente problema de optimización:

$$\arg\min_{\boldsymbol{B}} \sum_{i,j=1}^{n} \boldsymbol{W}_{i,j} \left\|\boldsymbol{b_i} - \boldsymbol{b_j}\right\|^2 \tag{3.1}$$

$$\begin{aligned} \text{Sujeto a:} \quad & \boldsymbol{b_i} \in \{-1, 1\}^k \\ & \sum_{i=1}^{n} \boldsymbol{b_i} = 0 \\ & \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{b_i}\boldsymbol{b_i}^T = \mathrm{I} \end{aligned}$$

En donde la restricción $\sum_{i=1}^{n} \boldsymbol{b_i} = 0$ mantiene el balance entre los bits al forzar una distribución uniforme entre las filas de la matriz, mientras que la restricción $\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{b_i}\boldsymbol{b_i}^T = \mathrm{I}$ asegura que los bits no estén correlacionados.

Como se demuestra en [71], el problema anterior es equivalente a un **problema de partición de grafos**. Específicamente a encontrar los cortes mínimos que permiten obtener k particiones balanceadas para el grafo completo cuyos nodos son los n elementos de la colección y el peso de sus aristas esta dado por $\boldsymbol{W}_{i,j}$. Este problema es **NP-*hard*** por lo que para resolverlo se utiliza la **matriz laplaciana** $\boldsymbol{L}$ del grafo, la cual se define como $\boldsymbol{L} = \boldsymbol{D}–\boldsymbol{W}$ siendo $\boldsymbol{D}$ una

matriz diagonal de $n \times n$ en donde $\boldsymbol{D}_{i,i} = \sum_{j=1}^{n} \boldsymbol{W}_{i,j}$, y el método de **relajación espectral**, por lo que el problema anterior se puede reescribir como:

$$\arg\min_{\boldsymbol{B}} \operatorname{tr}(\boldsymbol{B}^T(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{B}) \tag{3.3}$$

$$\text{Sujeto a:} \quad \boldsymbol{B}^T 1 = 0$$
$$\boldsymbol{B}^T \boldsymbol{B} = \mathrm{I}$$

Si se excluye la restricción $b_i \in \{-1, 1\}^k$ la solución al problema anterior esta dada por los k **vectores propios** de la **matriz laplaciana** $\boldsymbol{D} - \boldsymbol{W}$ con **mínimos valores propios** (excluyendo la solución trivial, es decir el vector con valor propio igual a 0) luego:

$$\boldsymbol{B} = [v_2 \;\; v_3 \;\; \ldots \;\; v_{k+1}] \tag{3.5}$$

Donde v_2 denota el vector propio cuyo valor propio es el segundo valor positivo más pequeño, de igual forma con el tercero y así sucesivamente. Finalmente, para determinar los *prehash* del conjunto de entrenamiento se aplica una **función umbral** para obtener de esta forma una cadena binaria.

Ahora bien, dado que los *prehash* se genera en base a los vectores propios de $\boldsymbol{L}$ cobra especial relevancia el método utilizado para construir $\boldsymbol{W}$ ya que la matriz laplaciana surge a partir de esta última. Para efectos de esta memoria se han abordado 3 métodos para construir dicha matriz, siendo el primero de ellos el método original propuesto en [71] mientras que los dos restantes corresponden a la aproximación descrita en [73].

$$W_{i,j} = \exp\left(-\frac{\left\|x_i - x_j\right\|^2}{\epsilon^2}\right) \tag{3.6}$$

La ecuación anterior corresponde al **kernel gaussiano** utilizado en [71]. Las variables x_i y x_j son elementos del conjunto de entrenamiento representados como vectores en $\mathbb{R}^d$. Para obtener dicha representación vectorial se utiliza el vector documento de la matriz **tf-idf** de la colección, de la misma forma en que fue utilizada en [73]. Por su parte la constate ϵ define

la distancia en $\mathbb{R}^d$ a la que dos ítems pueden considerarse similares. Este tipo de kernel es ampliamente utilizado como medida de similitud debido a que tiende a 0 entre más disimiles sean x_i y x_j y a 1 entre más semejantes son.

$$W_{i,j} = \begin{cases} c_{ij} \cdot \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), & \text{si} \quad x_i \in \mathbf{N}_k(x_j) \ \vee \ x_j \in \mathbf{N}_k(x_i) \\ 0, & \text{en cualquier otro caso} \end{cases} \tag{3.7}$$

La ecuación anterior corresponde a una de las variantes del ***heat kernel*** utilizado en [73]. Aunque a primera vista similar al kernel gaussiano presentado anteriormente, este introduce un coeficiente de ajuste c_{ij} y un parámetro σ que cumple la función de ϵ del kernel anterior. No obstante, a diferencia de este último la función solo es activada cuando uno de los elementos pertenece a los k vecinos más cercanos del otro, lo que se denota mediante la función de vecindad $\mathbf{N}_k(\cdot)$.

Por ultimo en [73] se propone una variante al kernel anterior aprovechando la información proveniente de las **etiquetas t del dataset**.

$$W_{i,j} = \begin{cases} a \cdot \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), & \text{si} \quad [x_i \in \mathbf{N}_k(x_j) \ \vee \ x_j \in \mathbf{N}_k(x_i)] \ \wedge \ t_i = t_j \\ b \cdot \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), & \text{si} \quad [x_i \in \mathbf{N}_k(x_j) \ \vee \ x_j \in \mathbf{N}_k(x_i)] \ \wedge \ t_i \neq t_j \\ 0, & \text{en cualquier otro caso} \end{cases} \tag{3.8}$$

Dicho lo anterior se establece un mayor valor para c_{ij} al que denominaremos como a cuando ambos elementos x_i y x_j comparten una etiqueta y un valor menor b cuando no lo hacen.

3.2. Entrenamiento y Predicción

El objetivo de esta fase es generar ***hashes*** que permitan realizar una **búsqueda por similitud** dentro de la colección. Para lograr este objetivo se plantea una **arquitectura *encoder-decoder*** [68] para el tratamiento de secuencias mediante el uso de **redes neuronales recurrentes**, específicamente redes LSTM. Lo anterior se debe a que, a diferencia de otros métodos, la propuesta de esta memoria trata a los **documentos como una secuencia ordenada**

de palabras, entendiendo que el orden de estas impacta en el significado del texto.

No obstante, para que el supuesto anterior tenga sentido, es necesario que la representación utilizada para las palabras sea capaz de capturar la **similitud semántica entre términos**. Debido a esto se optó por utilizar un ***word embedding*** [46], convirtiendo un documento en una secuencia ordenada de vectores reales, es decir, en una matriz. A su vez, también se optó por utilizar un *embedding* preentrenado como GloVe [58], entrenado sobre 6 mil millones de *tokens* de la Wikipedia 2014 y Gigaword 5. Esto permite reducir el tiempo de entrenamiento de la red, sin mencionar el aumento en la calidad del *embedding* gracias al enorme corpus utilizado. Sin embargo, el uso de un *embedding* preentrenado puede llevar a problemas de cobertura, tal y como se analizará en el capítulo de implementación.

En cuanto a la arquitectura, el objetivo de esta debe ser convertir secuencias desde un dominio (matrices de documento) a una secuencia en otro dominio (códigos *hash*). A este tipo de problema se le conoce como ***sequence-to-sequence learning*** (Seq2Seq) y es ampliamente estudiado en campos como la traducción automática (MT) y la búsqueda de respuestas (QA) [43]. El modelo en cuestión consiste en una primera capa recurrente llamada ***encoder*** que procesa la secuencia de entrada (el *embedding* del documento) en su totalidad, retornando al final su propio estado interno. Este estado interno, denominado a veces como **vector de contexto**, sintetiza la información contenida en la secuencia de entrada y se utiliza para inicializar una segunda capa LSTM que recibe el nombre de ***decoder***. El propósito de esta última capa es aprender a predecir el siguiente termino de la secuencia objetivo, lo que equivale en este caso, a aprender el siguiente *bit* del *prehash* obtenido durante la fase de preentrenamiento. Para esto se utiliza como entrada la secuencia objetivo desplazada en una posición (*shift*) generando un proceso de entrenamiento llamado *teacher forcing* (entrenamiento reforzado) el cual optimiza la red para predecir la secuencia objetivo original.

Una vez entrenada la red es necesario optimizar el *decoder* para generar la secuencia *hash* de manera recursiva. Para esto se genera un nuevo modelo denominado **modelo de inferencia** que utiliza los pesos aprendidos durante el entrenamiento. El modelo de inferencia utiliza el vector de contexto obtenido por el *encoder* y un carácter especial de inicio, para generar el *hash* de un documento, un *bit* a la vez, de manera recursiva e incremental.

Finalizado el modelo de inferencia se pasa a la **fase de predicción y evaluación**, en donde se evalúa la calidad de los *hashes* generados por el modelo. Esto se logra generando un ***ground truth*** en base a las etiquetas de clase y los *hashes* generados por la red neuronal. Posteriormente se realiza una búsqueda por similitud tipo ***fast distance approximation*** [69] sobre los *hashes* del conjunto de entrenamiento, utilizando como consulta los *hashes* de los documentos del conjunto de prueba. Finalmente se evalúa el desempeño del método utilizando como métrica la ***mean precision*** sobre el ranking de los primeros k elementos recuperados. Para efectos de comparación con [73] se evaluaran los resultados con $k = 50, 100, 150$ y 200.

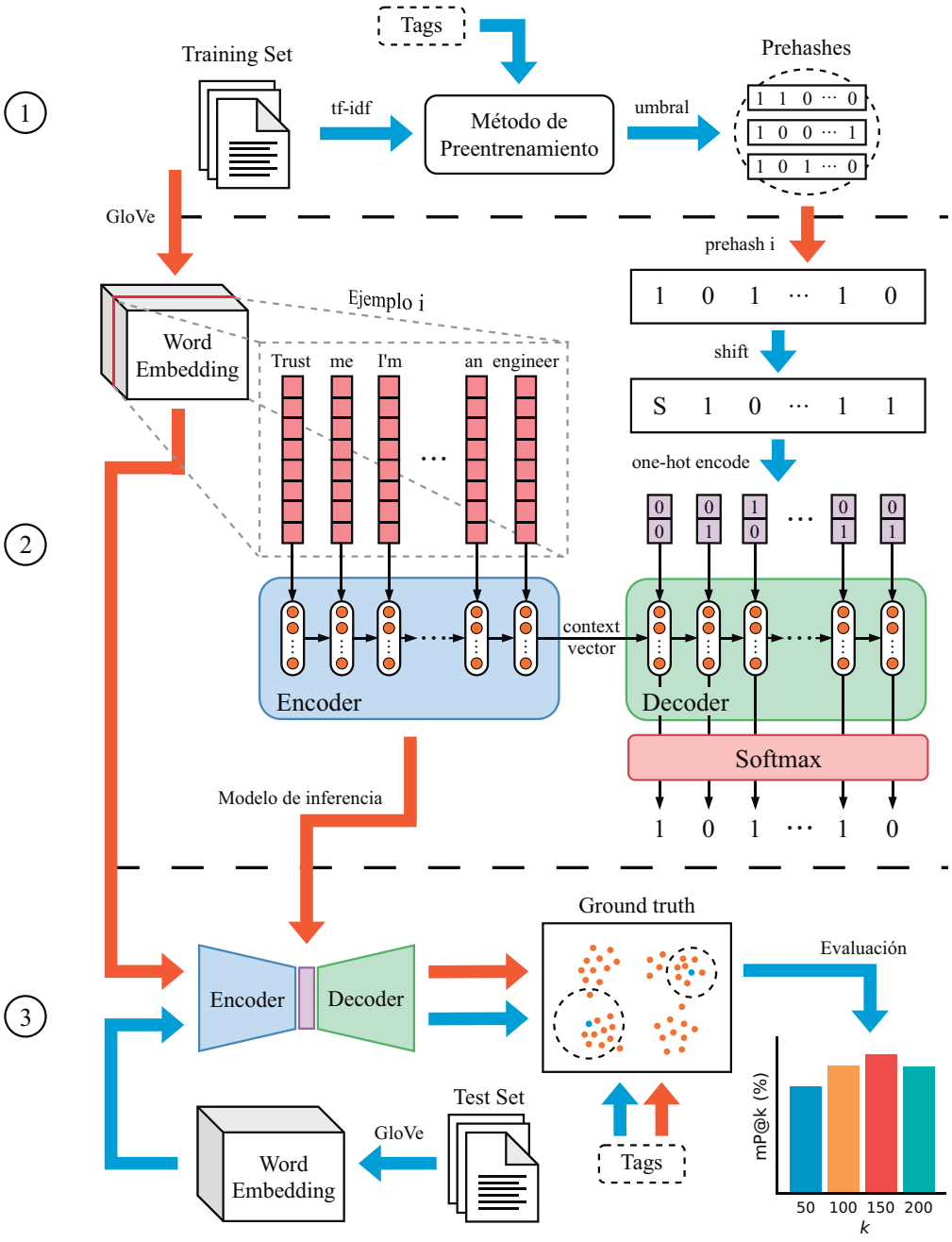


Figura 3.1: Propuesta de *hashing* de texto mediante redes neuronales recurrentes. El paso 1 corresponde a la fase de preentrenamiento, el 2 a la fase de entrenamiento de la red, siendo el paso 3 el apartado de predicción y evaluación del método.

Capítulo 4

Implementación

En este capítulo se presenta la implementación real de la propuesta señalada en el capítulo anterior junto a los resultados obtenidos.

4.1. Datasets

Para la realización de los experimento se emplearon los mismos conjuntos de datos utilizados en [73]; esto con la finalidad de comparar los resultados obtenidos con los resultados presentados en ese trabajo. Dicho lo anterior los *dataset* utilizados son **20 Newsgroups** y **SearchSnippets**, los cuales se detallan a continuación.

1. **20 Newsgroups:**: Es una colección de datos de aproximadamente 20000 entradas pertenecientes al repositorio newsgroups de la red Usenet. Estos documentos se encuentran distribuidos de manera aproximadamente uniforme a lo largo de 20 categorías diferentes, cada una perteneciente a un tópico en particular, siendo un *dataset* muy popular para tareas de clasificación y *clustering* de texto [61]. La versión del *dataset* utilizada para la elaboración de este trabajo es la variante *bydate* procesada por la doctora Ana Cardoso Cachopo [24], la cual realizó un proceso de *stemming* utilizando el algoritmo de Porter. Además, se removieron de la colección aquellos documentos con

más de 300 términos.

2. **SearchSnippets:** Es un dataset de aproximadamente 13000 elementos generado a partir de resultados de búsqueda en Google utilizando para ello frases predefinidas para 8 dominios diferentes [59]. Esta compuesto de textos pequeños de una o dos oraciones, donde las *stop words* fueron removidas.

A continuación, se presenta una tabla con información estadística relevante de ambos conjuntos de datos.

Cuadro 4.1: Información estadística de los *datasets* utilizados. C es el número de clases (etiquetas) correspondiente a cada conjunto de datos. $Train/Test$ corresponde al número de documentos pertenecientes al conjunto de entrenamiento (*train*) y al conjunto de prueba (*test*) respectivamente. $L(mean/max)$ corresponde a la cantidad promedio y máxima de palabras por documento en el *dataset*. $|V|$ representa la cardinalidad del vocabulario del *dataset* completo, es decir *train set* y *test set* combinados.

Dataset	C	$Train/Test$	$L(mean/max)$	$\|V\|$
20 Newsgroups	20	10443/6972	94,2/300	54370
SearchSnippets	8	10060/2280	18,1/50	29257

El objetivo de usar estos *datasets* es evaluar el desempeño del método implementado sobre colecciones de documentos de diferentes extensiones, cortos en el caso de SearchSnippets y de mayor extensión en 20 Newsgroups.

4.2. Preentrenamiento

Como fue mencionado en la sección homónima del apartado de propuesta, el objetivo de esta fase es generar un prehash para todos los elementos pertenecientes a los conjuntos de entrenamiento. Para lograr lo anterior se han utilizado las 3 variantes del *spectral hashing* descritas en aquella sección, las que pasarán a denominarse **Weiss** [71], **Wang** y **Wang supervisado** [73], haciendo referencia a los investigadores titulares de las respectivas publicaciones.

Dado lo anterior es necesario **seleccionar los hiperparámetros** correspondientes a cada método que permitan **maximizar su poder de discriminación**. Estos es seleccionar el parámetro ϵ en el caso de Weiss, el parámetro σ, el parámetro k y el coeficiente de ajuste c_{ij} en el caso de Wang, agregando los valores de a y b para Wang supervisado. Por su parte también es necesario seleccionar la longitud de las cadenas a utilizar, por lo que se estudiará el comportamiento de los métodos a 16, 32 y 64 **bits** respectivamente.

Para determinar la **calidad de los prehashes** generados se utilizó como **métrica de evaluación *mean precision*** sobre el top 100 de documentos recuperados (mP@100) en relación con la **distancia de hamming relativa** entre éstos y el documento de consulta. A su vez para realizar la búsqueda por similitud se realizó una partición aleatoria del conjunto de prehashes reservando el 20 % como documentos de consulta. Este proceso de partición aleatoria y búsqueda fue realizado tres veces para minimizar los posibles sesgos provenientes de la selección del conjunto de validación. Los resultados presentados corresponden al promedio de los resultados de mP@100 sobre las tres particiones anteriormente descritas.

Además, para otorgar sentido al nivel calidad de los resultados obtenidos se usó como punto de referencia el mP@100 alcanzado por un **recuperador aleatorio**. Esto quiere decir que para el caso de 20 Newsgroups si se seleccionasen los elementos del conjunto de recuperación de manera aleatoria, en promedio un $1/20$ de los documentos debiesen ser relevantes lo que equivale a un 5 %. De la misma manera para SearchSnippets el rendimiento promedio sería del 12,5 % dado que este *dataset* posee solo 8 categorías. Este razonamiento fue corroborado experimentalmente, tal y como se detalla continuación.

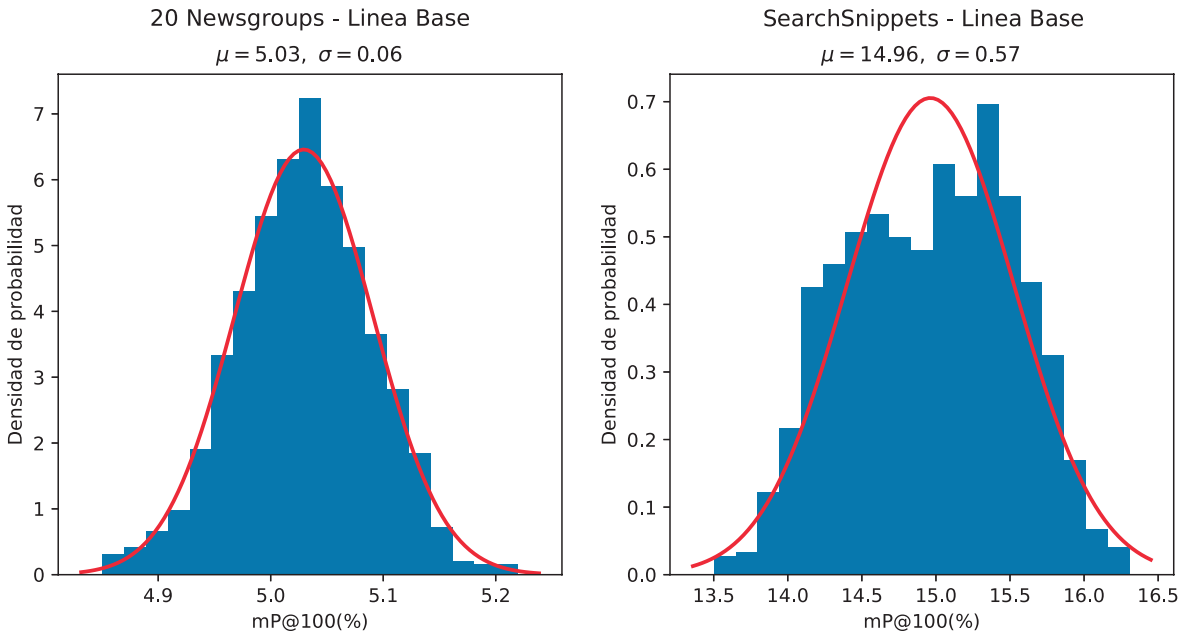


Figura 4.1: Distribución de los resultados de mP@100(%) alcanzado por un recuperador aleatorio después de 1000 repeticiones del proceso de partición aleatoria y recuperación. En ambos casos el histograma se encuentra normalizado para describir una función de densidad de probabilidad. Resultados para 20 Newsgroups a la izquierda y SearchSnippets a la derecha.

A partir de los gráficos anteriores es posible extraer algunas conclusiones. Para el caso de 20 Newsgroups la distribución se ajusta perfectamente al comportamiento esperado por lo que es posible afirmar con certeza que el rendimiento de un recuperador aleatorio es del 5 %. Sin embargo, en SearchSnippets la distribución se desvía ligeramente hacia la derecha por lo que el desempeño real se sitúa en torno al 15 % lo cual difiere del 12,5 % estimado. El motivo de esta diferencia radica en un desbalance en la cardinalidad de las clases lo que provoca un sesgo en los resultados.

No obstante, antes de presentar los resultados obtenidos, es importante **comprobar que la distancia euclidiana es una medida de similitud razonable dentro del espacio de representación tf-idf**. Este es un supuesto fundamental para el correcto funcionamiento de las técnicas de preentrenamiento propuestas, con excepción de Wang supervisado en donde las etiquetas cobran una mayor relevancia.

Para comenzar es necesario precisar cómo se construyeron los vectores tf-idf que representan a los elementos de la colección. Para realizar esto, se utilizó la función `TfidfVectorizer`

de la librería `Sklearn` la cual calcula el coeficiente idf introduciendo una variación respecto a la formulación expuesta en la ecuación 1.13 del capítulo antecedentes.

$$\text{idf}(t, D) = \log\left(\frac{\text{número total de documentos dentro de la colección } D}{\text{número de documentos en } D \text{ que contienen el término } t}\right) + 1 \qquad (4.1)$$

El propósito de sumar 1 en idf radica en evitar que la técnica ignore a aquellos términos que, de lo contrario, tendrían un valor idf igual a cero, es decir a aquellos que aparecen en todos los documentos de la colección [14].

Una vez obtenido los vectores tf-idf para ambos conjuntos de entrenamiento se observa que los documentos pertenecientes a 20 Newsgroups se representan utilizando vectores de 41876 dimensiones mientras que los de SearchSnippets utiliza solo 25117 dimensiones, lo que en ambos casos se aproxima al tamaño del vocabulario de su respectivo *train set*. Visualizar la distribución de los datos en espacios altamente dimensionales no es una tarea sencilla. Para solventar este problema se utilizan **técnicas de reducción de la dimensionalidad** como PCA [72] o t-SNE [51]. En este tipo de métodos, los datos son proyectados a un espacio de baja dimensionalidad, cuyas componentes intentan preservar la distribución del espacio original. Dicho lo anterior, se presentan los siguientes diagramas de dispersión.

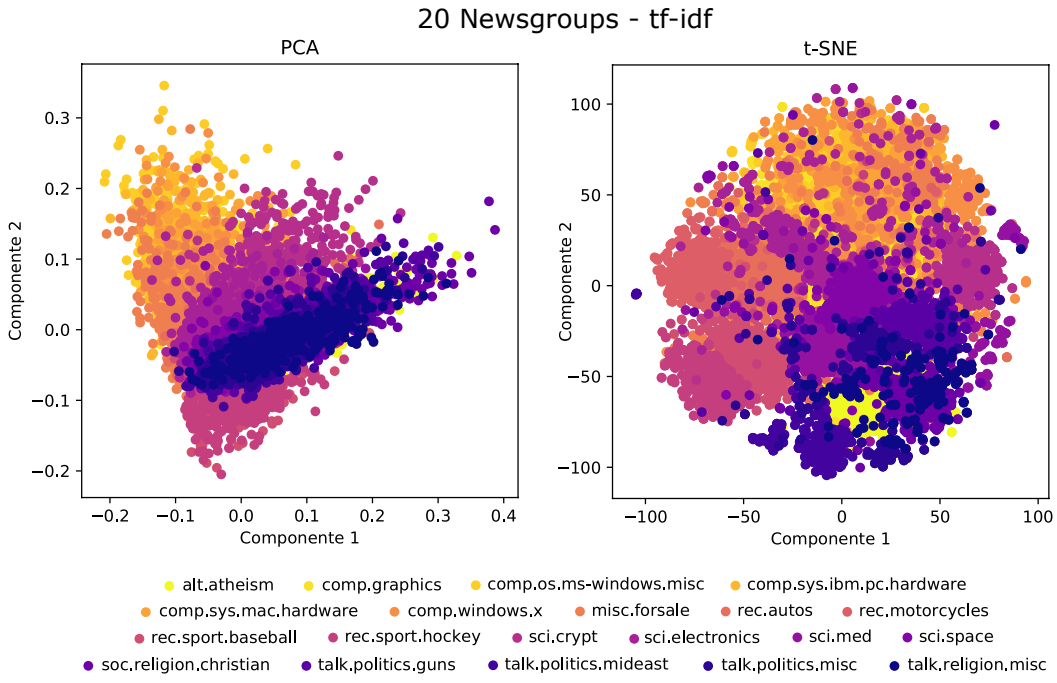


Figura 4.2: Diagrama de dispersión para 20 Newsgroups. Reducción mediante PCA y t-SNE.

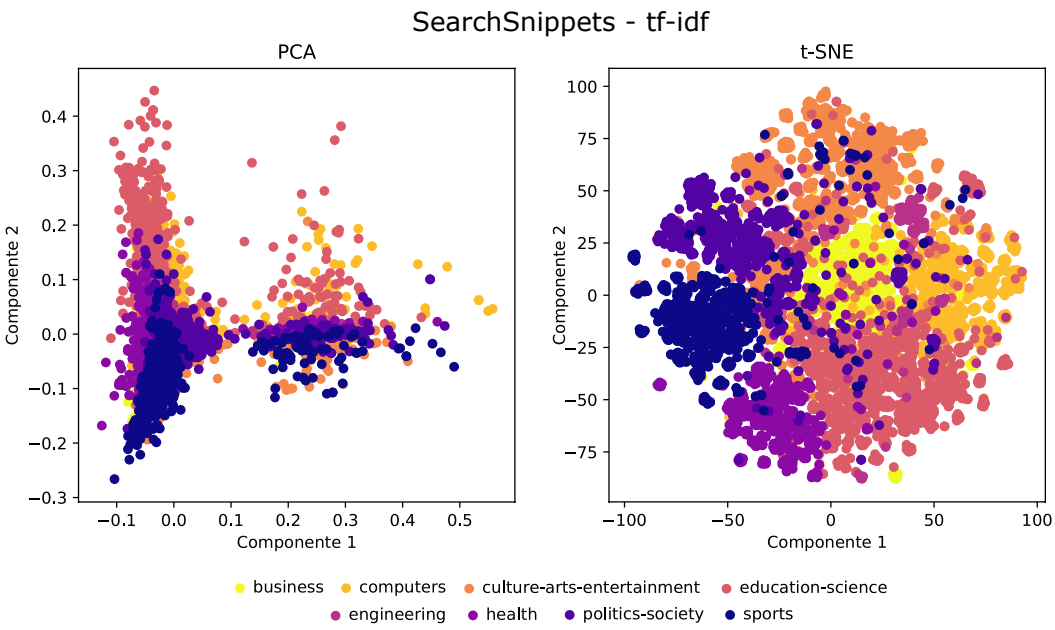


Figura 4.3: Diagrama de dispersión para SearchSnippets. Reducción mediante PCA y t-SNE.

En ambos casos se observa como PCA superpone las clases, quedando algunas completamente ocultas a diferencia de lo que ocurre con t-SNE donde prácticamente todas son visibles. También en ambos casos se observa que **los elementos pertenecientes a una misma clase se agrupan en una región del espacio** formando un *(*cluster). Estos grupos se encuentran muy cercanos entre si, por lo que parte de sus elementos se entremezclan difuminando la frontera entre clases. No obstante, los gráficos anteriores parecen indicar que los elementos pertenecientes a una misma clase se encuentran agrupados dentro del espacio tf-idf lo cual **indica que el uso de una métrica de distancia en este espacio podría ser un indicador viable de similitud entre elementos**.

Ya que el problema planteado se centra en la distancia euclidiana como medida de similitud en este espacio, es necesario estudiar la distribución de las distancias. Para esto se procedió a construir un histograma de las distancias entre documentos para cada caso. Esto además permite determinar el espacio de búsqueda para los parámetros ϵ y σ durante el proceso de sintonización.

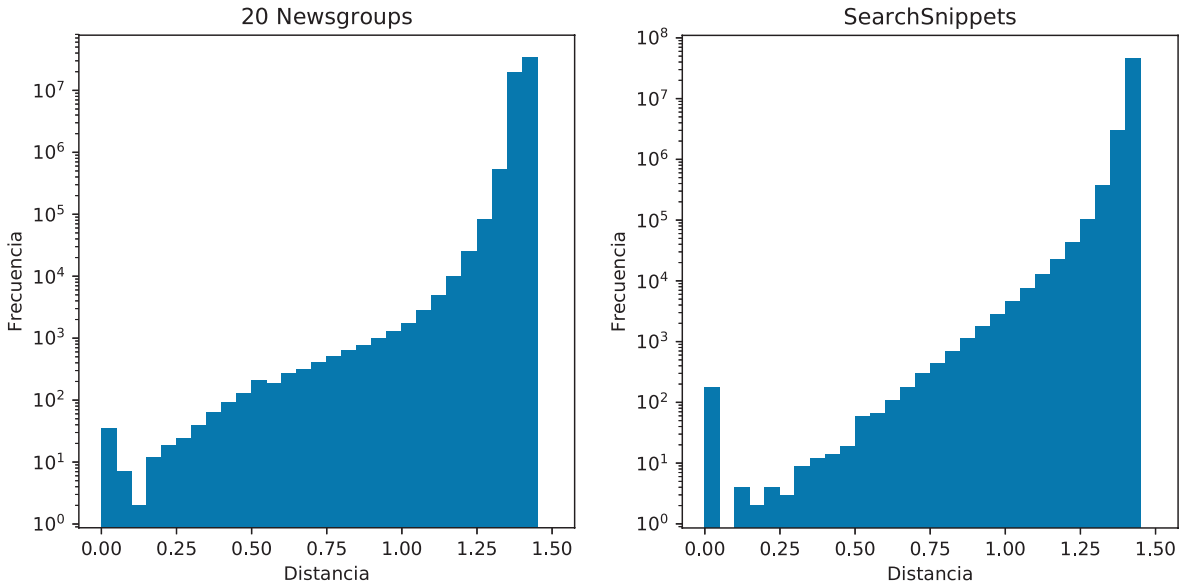


Figura 4.4: Histogramas de distancia euclidiana entre documentos como vectores tf-idf. En ambos casos se utilizan 30 intervalos en el rango 0 a 1.5 mientras que el eje de frecuencias está en escala logarítmica. Resultados para 20 Newsgroups a la izquierda y SearchSnippets a la derecha.

En el caso de 20 Newsgroups existen 54.522.903 aristas entre documentos cuyo valor oscila entre 0 y 1,41 quedando concentrado el 98,78 % en el intervalo 1,35 y 1,45. Por su parte SearchSnippets posee 50.596.770 aristas cuyo valor oscila también entre 0 y 1,41 quedando concentrado el 98,85 % en el intervalo 1,35 y 1,45. De los gráficos anteriores es posible concluir que una abrumadora mayoría de documentos se encuentra a una distancia euclidiana entre 1,35 y 1,45. Este hecho es una consecuencia de uno de los aspectos de la llamada **maldición de la dimensionalidad** la cual establece que las diferencias entre elementos se diluyen a medida que aumenta las dimensiones del espacio [31].

Además, considerando que los arcos interiores de las clases corresponden al 5 % y al 15 % de las aristas totales respectivamente, podría parecer en primera instancia que la distancia euclidiana no es un buen indicador de similitud en el espacio tf-idf. No obstante, para poder corroborar dicha afirmación es necesario analizar la distribución de las distancias y observar que no existe diferencia entre la distancia inter e intraclase. Para comprobar lo anterior se construyeron los siguientes mapas de calor (*heatmaps*) que representan diferentes aspectos de la distribución de distancias para cada *dataset*.

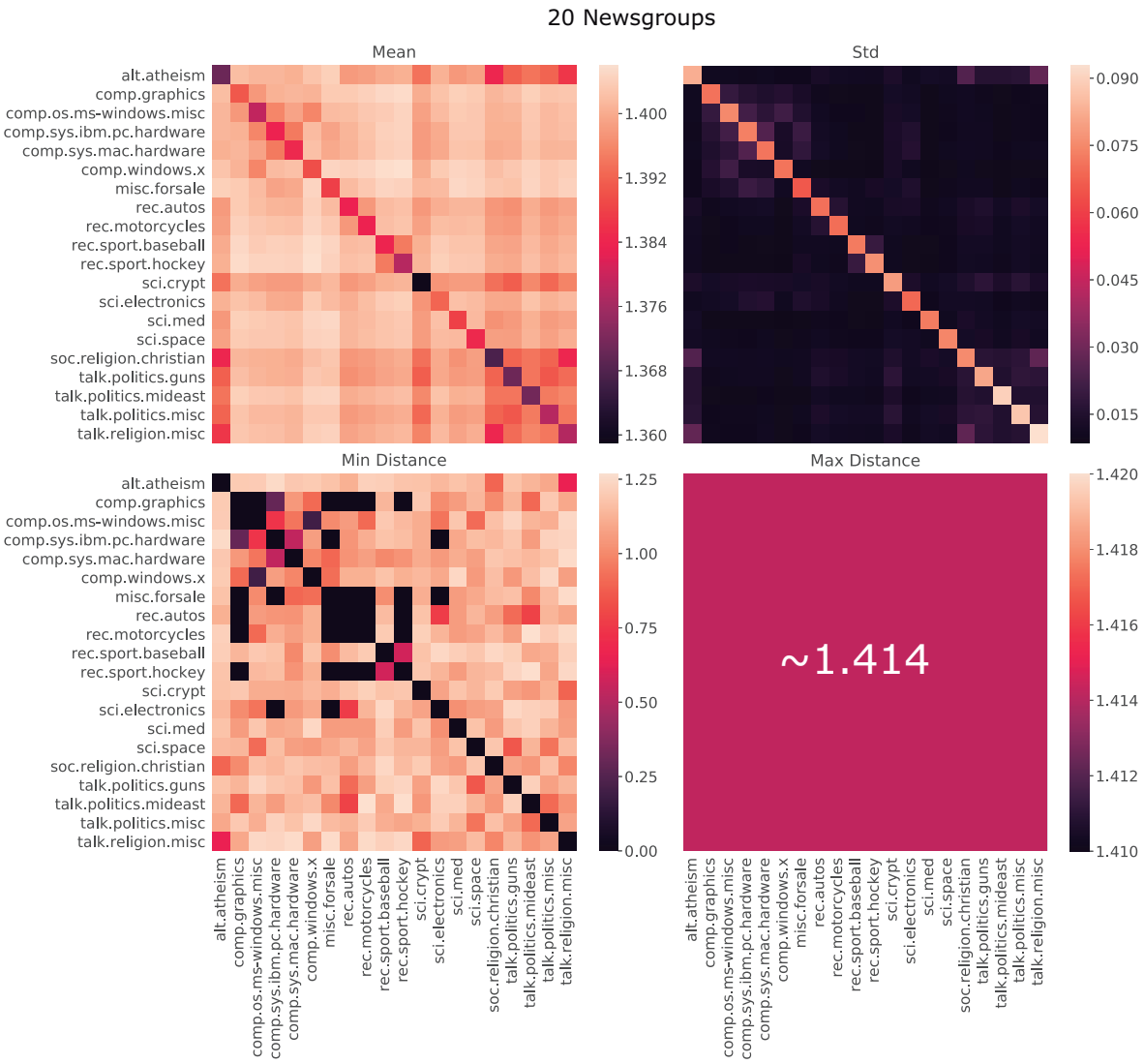


Figura 4.5: Mapa de calor de las distancias entre clases en el espacio tf-idf para 20 Newsgroups.

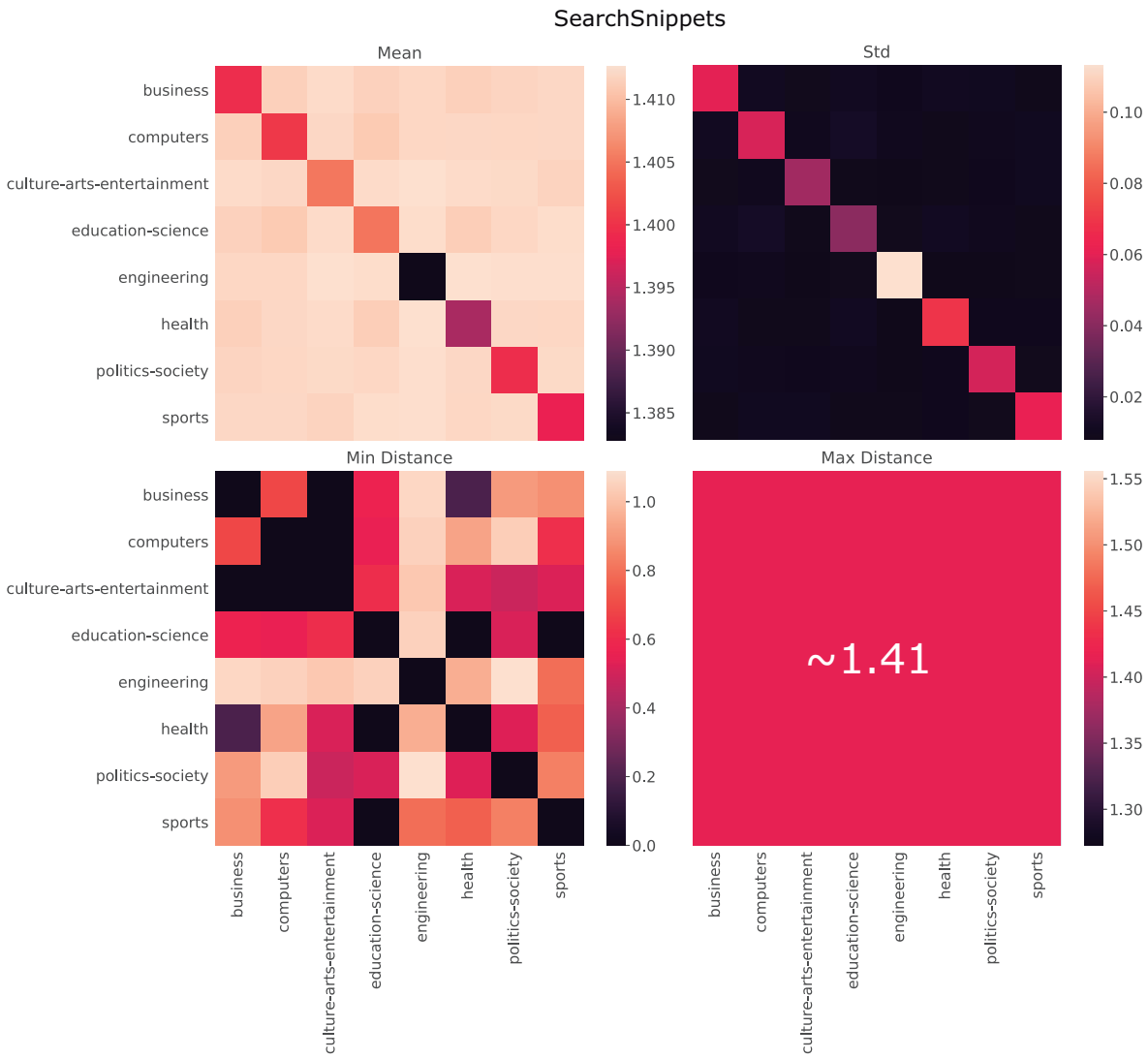


Figura 4.6: Mapa de calor de las distancias entre clases en el espacio tf-idf para SearchSnippets.

En ambos cuadros se observa el mismo comportamiento. La distancia promedio intraclase es menor que la distancia interclase. También la desviación estándar intraclase es más alta que la interclase. Esto se debe a que dentro de una misma clase hay una mayor concentración de documentos cercanos como puede intuirse al observar los *heatmaps* de distancia mínima. Con respecto a este último, llama la atención como **las clases que se confunden entre si son aquellas que, en efecto, poseen una cercanía semántica**, como pueden ser las categorías de `rec.autos` y `rec.motorcycles` de 20 Newsgroups que hablan de autos y motocicletas, o las categorías `education-science` y `health` de SearchSnippets que tratan de educación/-ciencia y salud respectivamente. Por otro lado, también llama la atención lo extremadamente

homogénea que es la matriz de distancia máxima ya todos los elementos se encuentran cercanos a 1,41 con diferencias ínfimas del orden de 10^{-15}, lo cual deja de sorprender cuando se toma en consideración que la enorme mayoría de distancias se encuentran próximas a ese valor, tal y como queda patente en la figura 4.4.

Con base a los resultados experimentales presentados anteriormente, es seguro afirmar que **la distancia euclidiana dentro del espacio tf-idf si captura hasta cierto punto la similitud temática entre los documentos de los *dataset***, por lo que tiene sentido aplicar los métodos de prehash propuestos.

Aclarado el punto anterior, se procede a presentar los resultados obtenidos divididos por método de preentrenamiento.

Weiss

Es el método original descrito en [71]. En él se utiliza un **kernel gaussiano** para generar la matriz $\boldsymbol{W}$.

$$W_{i,j} = \exp\left(-\frac{\left\|x_i - x_j\right\|^2}{\epsilon^2}\right) \tag{4.2}$$

Como puede apreciarse, la ecuación genera una matriz $\boldsymbol{W}$ densa a diferencia del resto de métodos que tienden a generar matrices dispersas las cuales son más eficientes en términos de almacenamiento. Cabe destacar que debido a que este método posee únicamente dos parámetros, uno explícito con ϵ y uno implícito con el número de bits, es posible realizar un proceso de sintonización más exhaustivo.

A continuación, se presenta los resultados obtenidos al variar el parámetro ϵ utilizando cadenas de 16, 32 y 64 bits sobre los conjuntos 20 Newsgroups y SearchSnippets respectivamente.

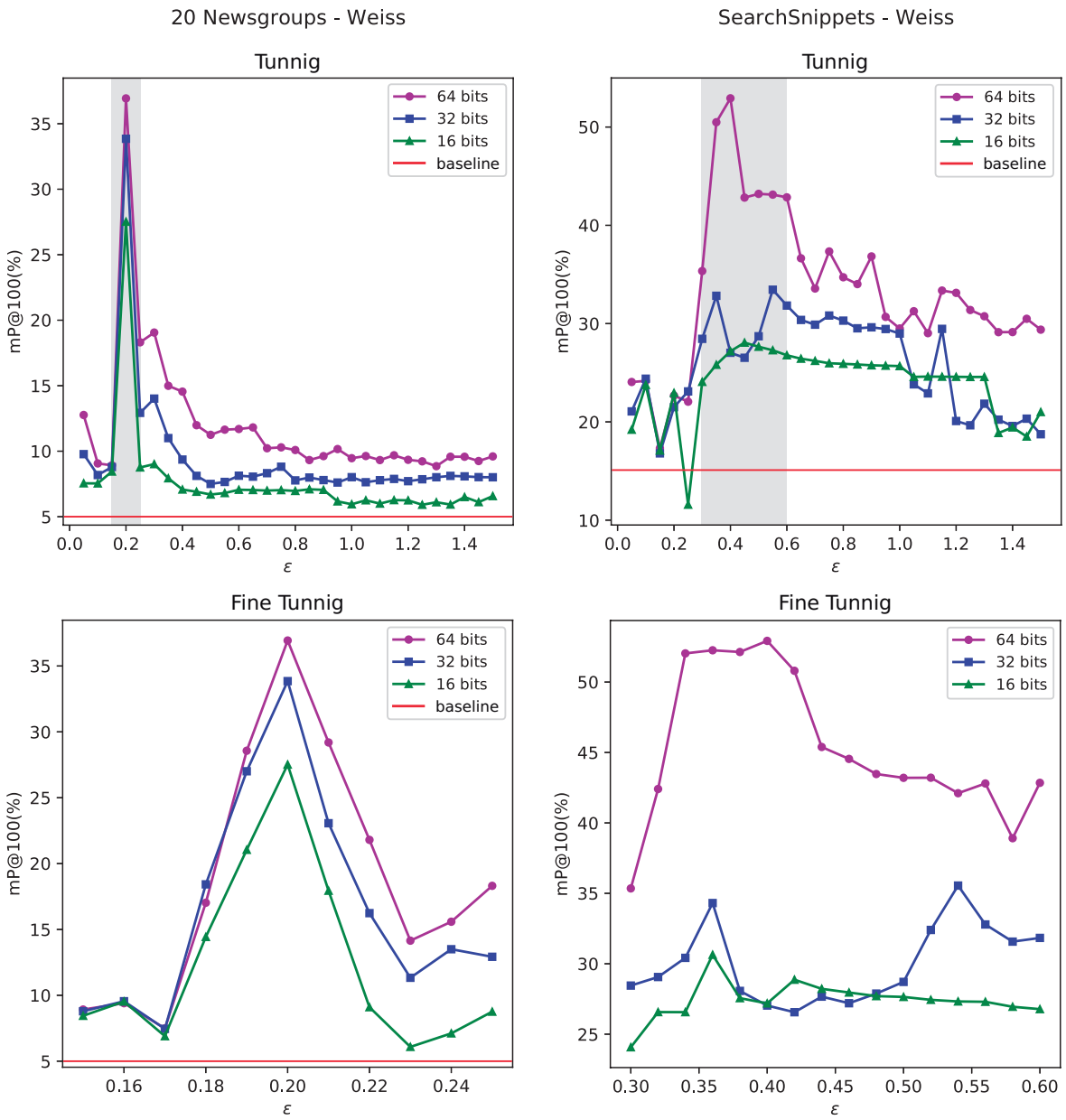


Figura 4.7: Resultados de mP@100 en función del parámetro épsilon utilizando cadenas de 16, 32 y 64 bits. La linea roja representa el rendimiento base de un recuperador aleatorio. Resultados del proceso de sintonización en el cuadrante superior y de la sintonización fina en el inferior. Resultados sobre 20 Newsgroups a la izquierda y Searchsnippets a la derecha.

Como se observa en el cuadro anterior el método es extremadamente **sensible a la selección del parámetro** ϵ pudiendo generar diferencias en los resultados de hasta 30 puntos porcentuales. Esta sensibilidad, caracterizada por la presencia de un claro óptimo global dentro de las gráficas, **coincide con el comportamiento teóricamente esperado**, ya que, al considerar

el parámetro ϵ como el radio de la esfera que delimita la vecindad entre elementos, se intuye la existencia de un radio óptimo en que la mayoría de los elementos dentro de la frontera sean efectivamente pertenecientes a una misma clase.

Dicho esto, se percibe que el mejor desempeño se alcanza con $\epsilon = 0,2$ y $\epsilon = 0,4$ para 20 Newsgroups y SearchSnippets respectivamente. Respecto al número de bits, se advierte que normalmente **una mayor cantidad contribuye a aumentar la calidad de los prehashes generados**. También en ambos casos el óptimo se logra al utilizar 64 bits, siendo mucho más marcada la diferencia de rendimiento en SearchSnippets.

En términos generales se tiene que para 20 Newsgroups este método alcanza un rendimiento máximo de 37 % lo cual equivale a 7,4 veces el desempeño base. Para SearchSnippets el máximo se sitúa en 53 % lo que es equivalente a 3,5 veces el rendimiento real de un recuperador aleatorio. Si bien el rendimiento general del método se posiciona por encima de un recuperador aleatorio, todavía se encuentra lejos del desempeño deseado, por lo que **estos prehashes no son una buena opción para la fase de entrenamiento**.

Para entender mejor como la distancia de hamming actúa como medida de similitud entre elementos, es conveniente comparar la distribución del conjunto de prehashes con la distribución de la representación tf-idf presentada al principio de este capítulo. Para lograr esto, se construyeron nuevos mapas de calor que reflejan aspectos de la distribución de la distancia de hamming relativa entre clases. Además de construir un nuevo diagrama de dispersión utilizando t-SNE con distancia de hamming para visualizar los datos en un espacio bidimensional. Todo esto utilizando los parámetros óptimos obtenidos para cada *dataset*.

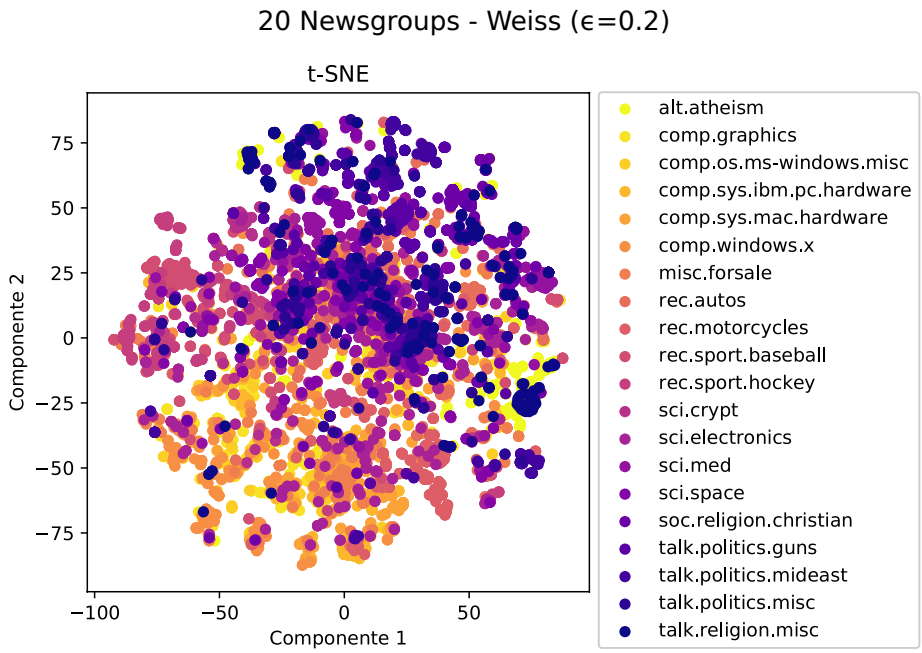


Figura 4.8: Diagrama de dispersión de los mejores prehashes obtenidos por Weiss para 20 Newsgroups. Reducción mediante t-SNE con distancia de hamming.

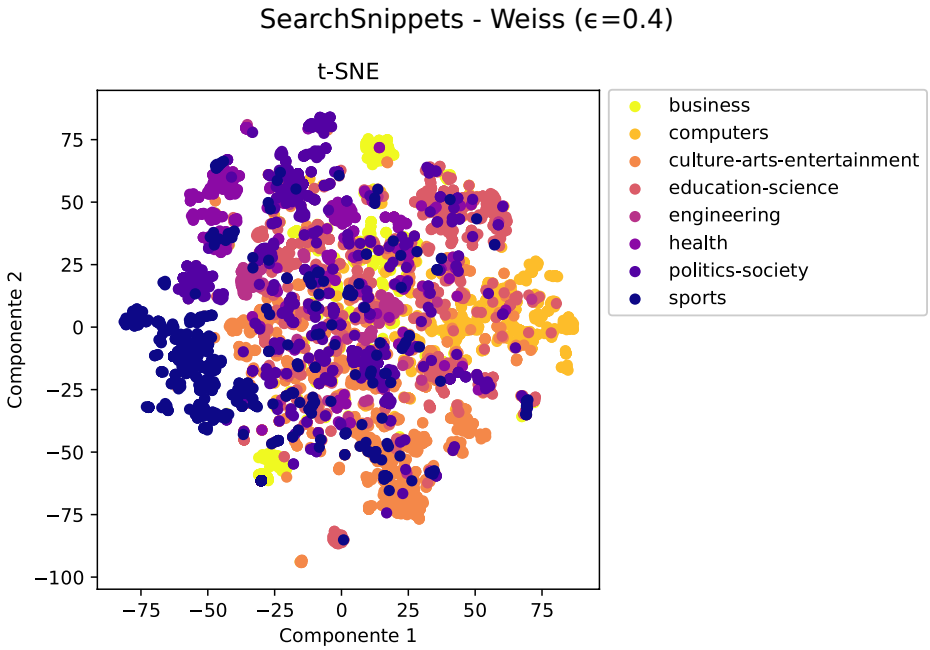


Figura 4.9: Diagrama de dispersión de los mejores prehashes obtenidos por Weiss para SearchSnippets. Reducción mediante t-SNE con distancia de hamming.

En ambos diagramas de dispersión se observa como **Weiss es incapaz de generar *clusters* cohesionados para cada clase**. Al comparar estos gráficos con las figuras 4.2 y 4.3 se observa que, si bien los datos se encuentran más dispersos en el espacio generando pequeños grupos de elementos similares, todavía existen demasiados elementos desperdigados. En base a esto es posible concluir que este método genera *clusters* menos definidos que los observados en el espacio tf-idf por lo que falla a la hora de capturar de manera efectiva la distribución de este espacio. Esta conclusión se refleja en los resultados obtenidos por los mapas de calor que se presentan a continuación.

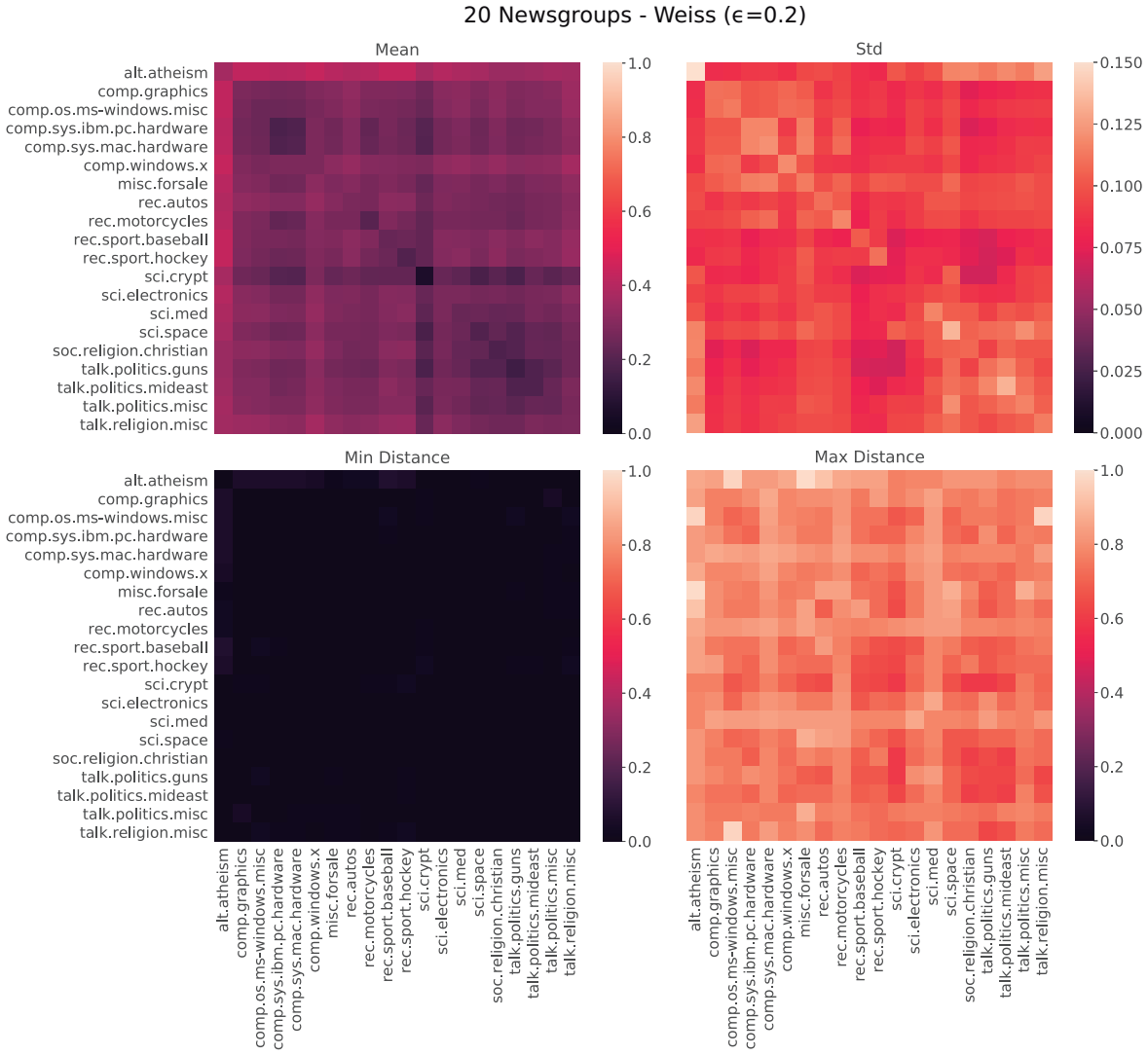


Figura 4.10: Mapas de calor de la distancia de hamming relativa entre clases, generados a partir de los mejores prehashes obtenidos por Weiss para 20 Newsgroups.

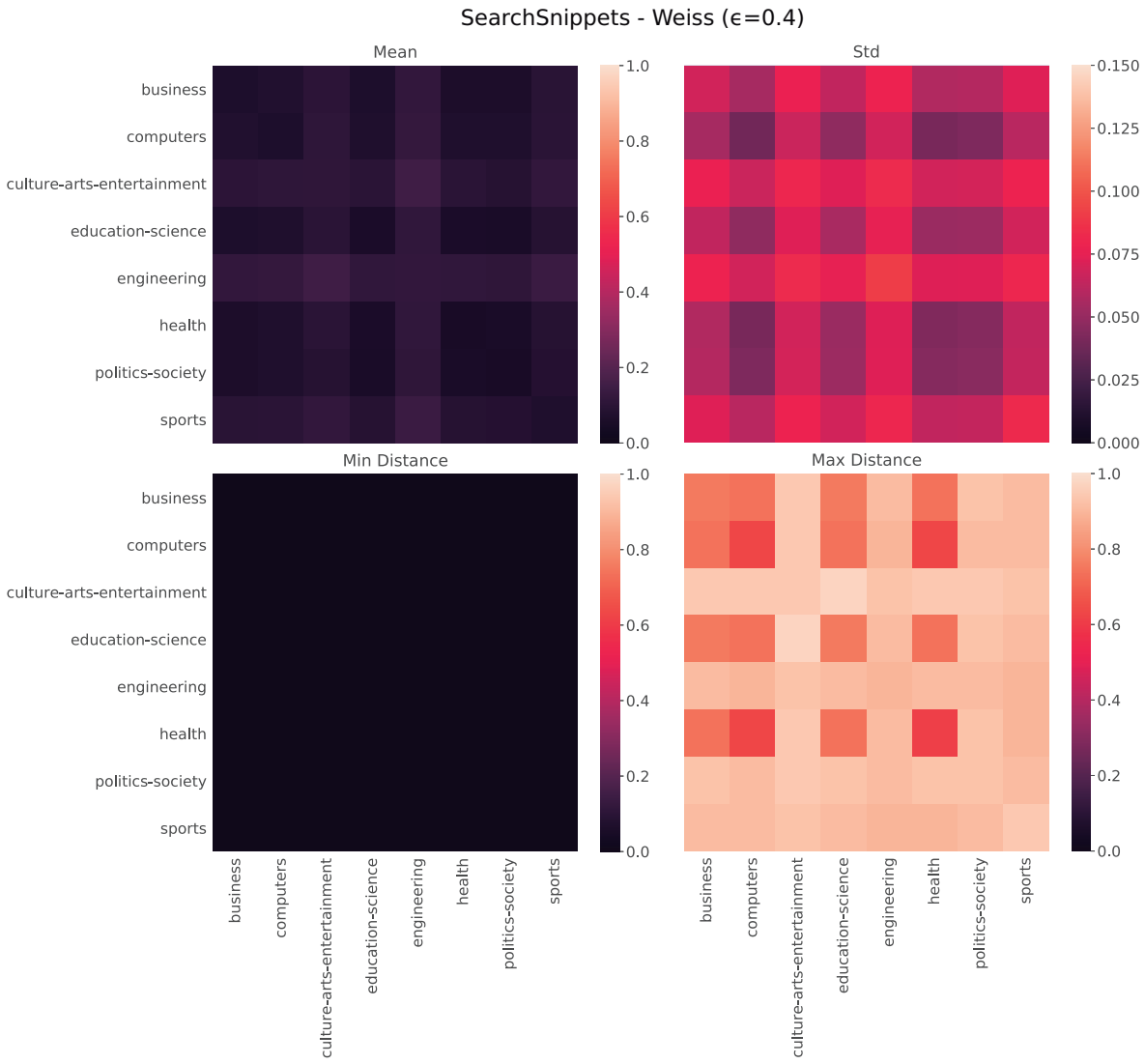


Figura 4.11: Mapas de calor de la distancia de hamming relativa entre clases, generados a partir de los mejores prehashes obtenidos por Weiss para SearchSnippets.

Tal y como se esperaba, ambos casos presentan una atenuación de la diagonal principal, es decir, las distancias intraclases no se diferencian significativamente de las intercalases, lo que se traduce en un pobre desempeño en el proceso de recuperación.

Wang

Es uno de los métodos propuestos en [73]. Al igual que Weiss es un método no supervisado, pero, a diferencia de éste, utiliza una **lista de vecinos más cercanos** y un ***heat kernel*** para construir la matriz $\boldsymbol{W}$.

$$W_{i,j} = \begin{cases} c_{ij} \cdot \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), & \text{si} \quad x_i \in \mathbf{N}_k(x_j) \ \vee \ x_j \in \mathbf{N}_k(x_i) \\ 0, & \text{en cualquier otro caso} \end{cases} \tag{4.3}$$

A diferencia de Weiss, la matriz $\boldsymbol{W}$ generada por esta ecuación es dispersa, lo que se traduce en un uso de memoria más eficiente. Además, como puede apreciarse este método posee 3 parámetros; el parámetro σ que cumple una función similar al parámetro ϵ de Weiss y el parámetro k que determina el tamaño de la vecindad. Por su parte el coeficiente c_{ij} no tiene ningún efecto en los resultados, como se expondrá más adelante.

Dicho lo anterior, se presentan los resultados del proceso de sintonización de parámetros, utilizando cadenas de 16, 32 y 64 bits sobre los conjuntos 20 Newsgroups y SearchSnippets respectivamente. El proceso de sintonización se realizó bajo un **supuesto de independencia** por lo que cada parámetro se seleccionó de manera independiente utilizando como base la configuración descrita en [73] es decir $c = 1$, $\sigma = 1$ y $k = 7$. Finalmente se considera al conjunto de parámetros obtenidos como la configuración óptima, vía previa validación experimental, para descartar que alguna de las configuraciones estudiadas durante el proceso de sintonización pudiese tener un mejor rendimiento.

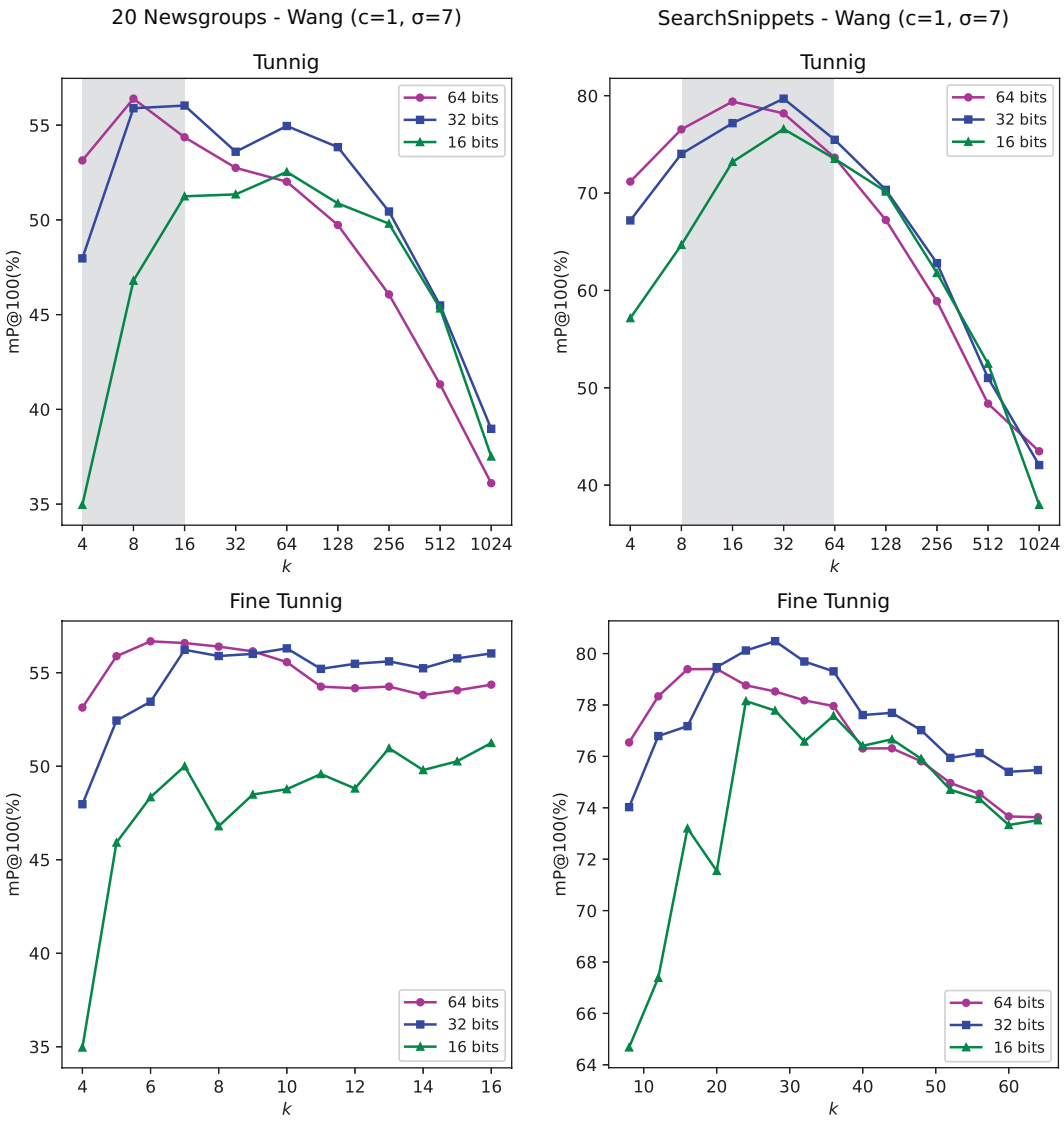


Figura 4.12: Resultados de mP@100 en función del tamaño de la vecindad k 16, 32 y 64 bits. Resultados del proceso de sintonización en el cuadrante superior y de la sintonización fina en el inferior. Resultados sobre 20 Newsgroups a la izquierda y Searchsnippets a la derecha.

Como se aprecia en el cuadro anterior el método es muy sensible a la selección del parámetro de vecindad k. Tal y como se esperaba el desempeño aumenta junto con k hasta alcanzar un valor máximo con $k = 6$ para 20 Newsgroups a 64 bits y $k = 28$ para SearchSnippets a 32 bits, antes de comenzar a descender. Cabe de destacar que a medida que el parámetro k aumenta, la matriz $\boldsymbol{W}$ se vuelve más densa llegando a un límite teórico cuando el tamaño de la vecindad es igual al número de documentos, en cuyo caso este método es indistinguible del de Weiss. También llama la atención como en muchos casos el rendimiento de las cadenas

de 32 bits sobrepasa a las de 64 bits sobre todo para SearchSnippets en donde constituyen el máximo global. Ahora bien, dado que este método es sensible a la selección del parámetro k es de esperar que la influencia del parámetro σ, el cual es equivalente al parámetro ϵ de Weiss, disminuya considerablemente, ya que ahora es la lista de los k vecinos más cercanos la que determina en mayor medida la similitud entre elementos.

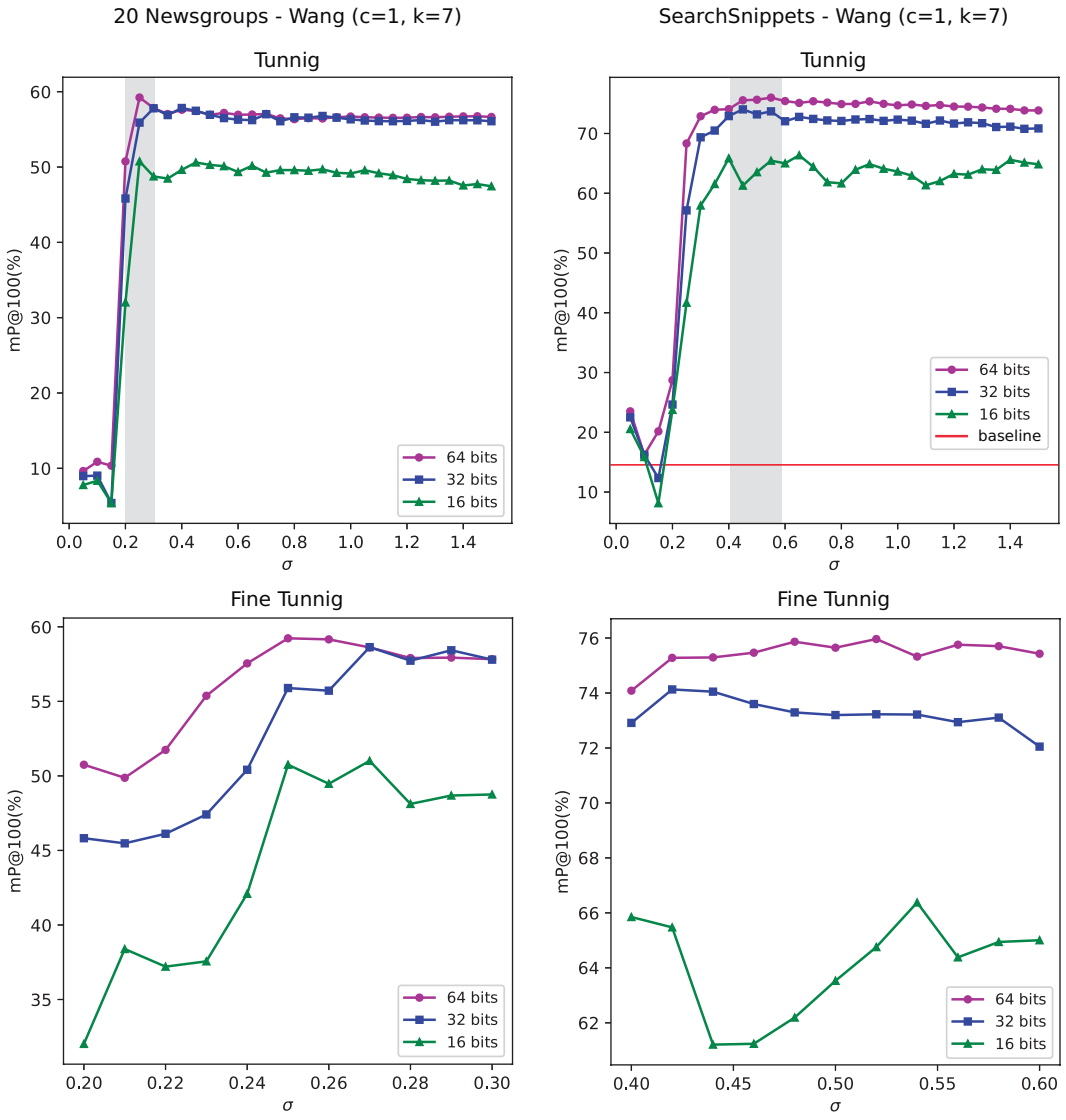


Figura 4.13: Resultados de mP@100 en función del parámetro σ a 16, 32 y 64 bits. La linea roja representa el rendimiento base de un recuperador aleatorio. Resultados del proceso de sintonización en el cuadrante superior y de la sintonización fina en el inferior. Resultados sobre 20 Newsgroups a la izquierda y SearchSnippets a la derecha.

En ambos casos se observa como el rendimiento aumenta abruptamente a partir de cierto valor, para continuar de manera estable con una leve tendencia a la baja. En base a esto se concluye que el método no es sensible, a partir de cierto valor, al parámetro σ obteniendo el mejor desempeño con $\sigma = 0,25$ para 20 Newsgroups y $\sigma = 0,52$ para SearchSnippets ambos a 64 bits.

Respecto al coeficiente de ajuste c_{ij} se observa que no tiene ningún impacto en el desempeño del método tal y como puede apreciarse en la figura A.1 del apartado de anexo. Esto se debe a que multiplicar un matriz por una constante no tiene ningún efecto en los vectores propios por lo que los prehashes resultantes son siempre los mismos. No obstante este coeficiente cobra relevancia en Wang supervisado con sus posibles valores a y b.

Al validar experimentalmente los parámetros seleccionados se observa un rendimiento inferior al obtenido por una de las configuraciones durante el proceso de sintonización, lo que pone en duda el supuesto de independencia anteriormente establecido. Debido a lo anterior se variaron los parámetros a mano, utilizando como base el mejor caso observado en la sintonización, hasta obtener un desempeño superior al de todas las configuraciones estudiadas. Finalmente, los mejores parámetros encontrados son $\sigma = 0,27$ y $k = 16$ para 20 Newsgroups con cadenas de 64 bits, logrando un mP@100 del 61,83 %, y $\sigma = 0,52$ y $k = 28$ para SearchSnippets con cadenas de 32 bits, logrando un mP@100 del 81,16 %. Estos resultados se encuentran en la figura A.2 del apartado anexo.

A continuación, se presentan los mapas de calor y los diagramas de dispersión que permiten entender mejor como los *prehashes* generados segmentan el espacio.

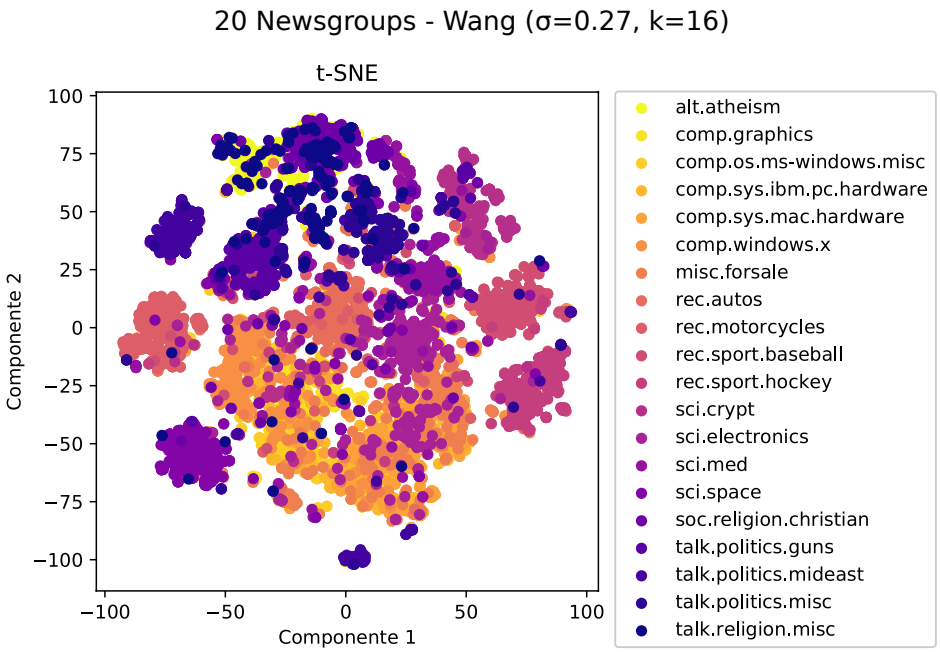


Figura 4.14: Diagrama de dispersión de los mejores prehashes obtenidos por Wang para 20 Newsgroups. Reducción mediante t-SNE con distancia de hamming.

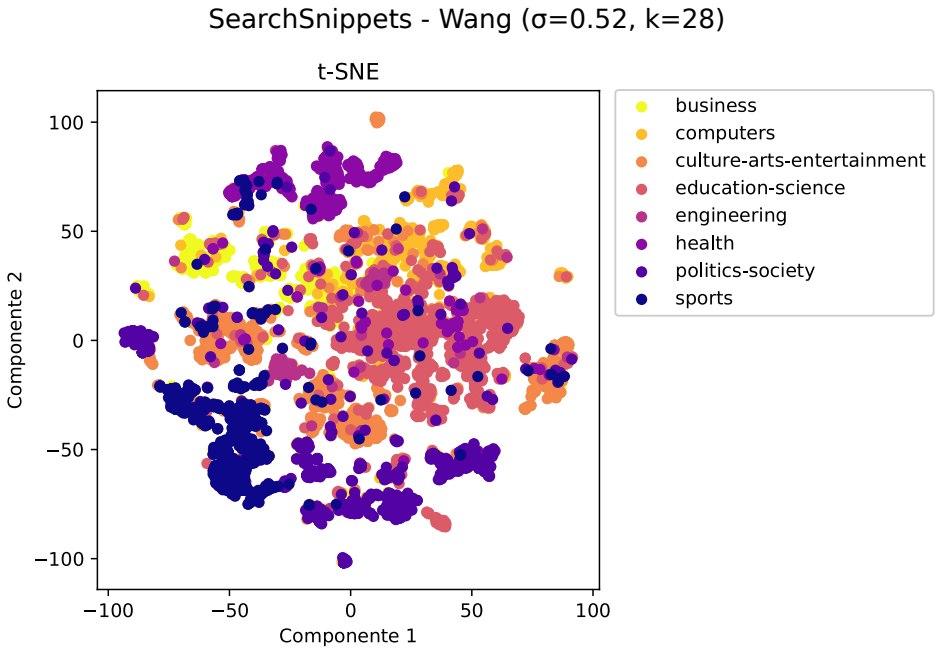


Figura 4.15: Diagrama de dispersión de los mejores prehashes obtenidos por Wang para SearchSnippets. Reducción mediante t-SNE con distancia de hamming.

En ambos diagramas de dispersión se observa una **mayor compartimentación de las clases**

que las observadas en los diagramas 4.8 y 4.9 generados por Weiss. Esto es esperable si se tiene en cuenta que este método representa una mejora de entre 25 y 30 puntos porcentuales sobre el método anterior. De la misma forma es de esperar que la distribución de distancias presentadas en los mapas de calor tienda a destacar más la diagonal de las matrices, presentando una distancia intraclase menor, mayor varianza intraclase junto a mínimos más marcados, tal y como se observa en la distribución en el espacio tf-idf (figuras 4.5 y 4.6).

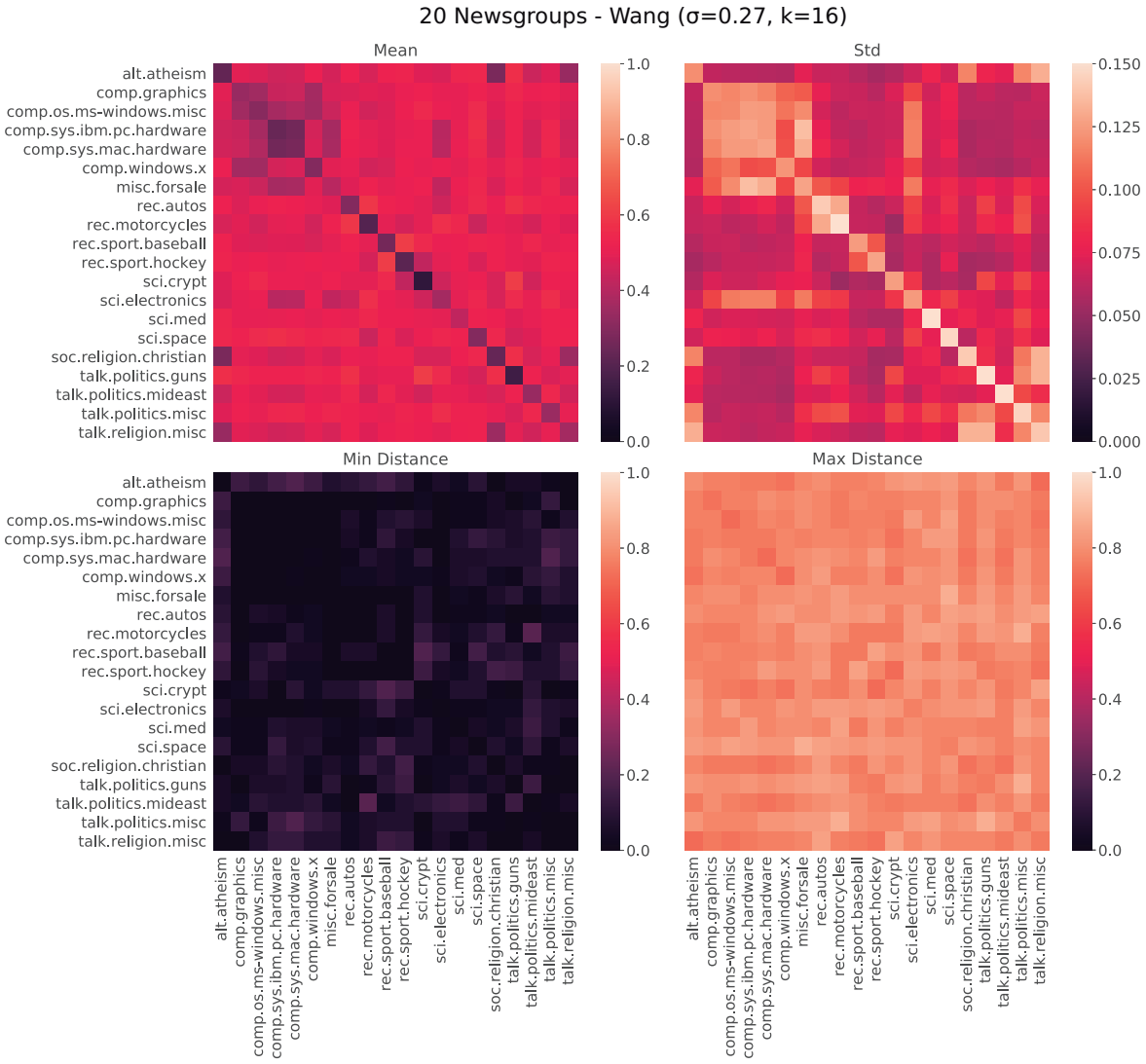


Figura 4.16: Mapas de calor de la distancia de hamming relativa entre clases, generados a partir de los mejores prehashes obtenidos por Wang para 20 Newsgroups.

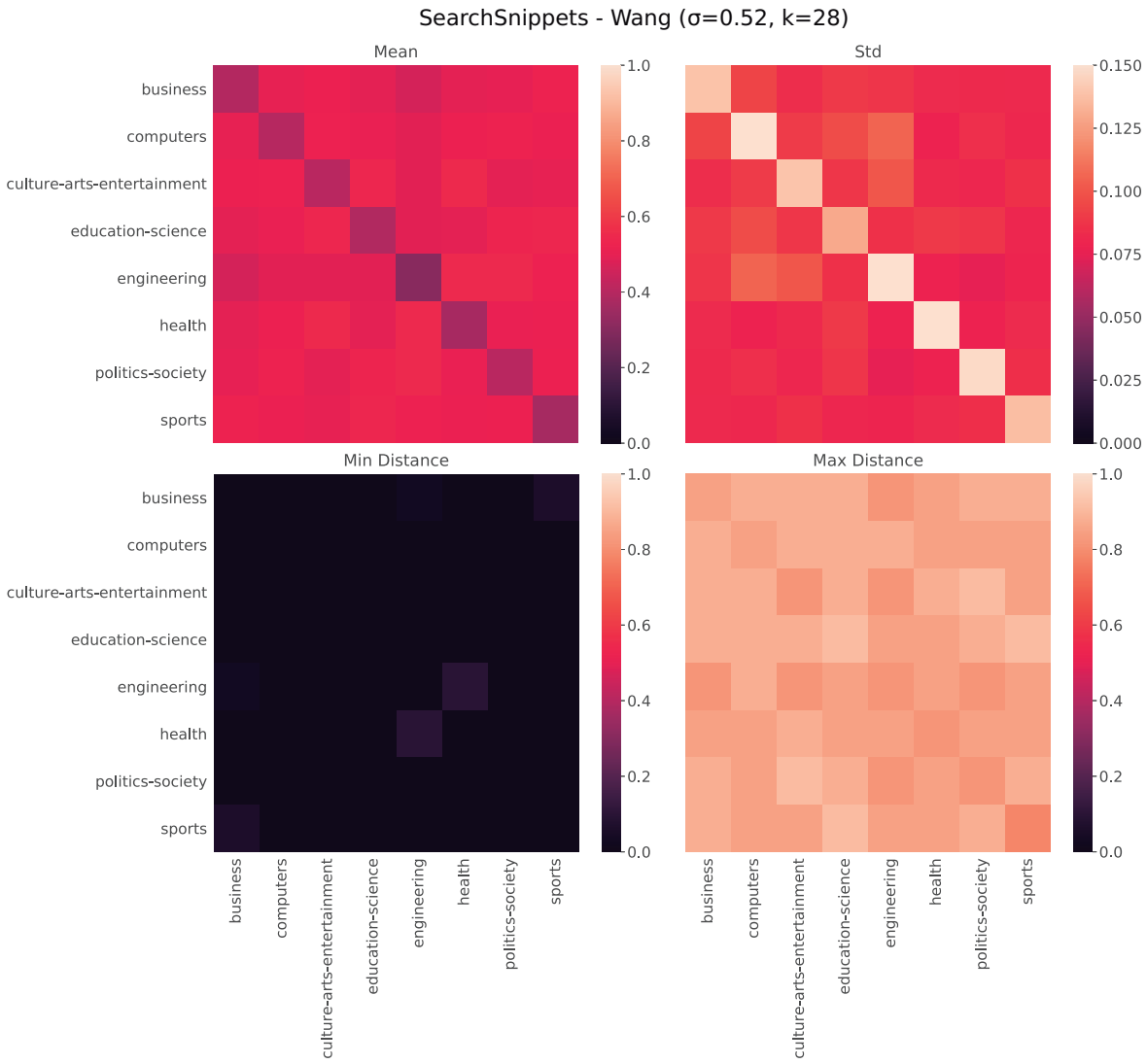


Figura 4.17: Mapas de calor de la distancia de hamming relativa entre clases, generados a partir de los mejores prehashes obtenidos por Wang para SearchSnippets.

Como se aprecia el comportamiento de los *heatmaps* es el esperado. En conclusión, es posible afirmar que **la introducción de una lista de vecinos cercanos en el funcionamiento del método constituye una significativa mejora** respecto al método de Weiss. Esto se debe a la fuerza con que el método penaliza a aquellos elementos que no se encuentran dentro de la vecindad, permitiendo filtrar de mejor manera la información proveniente del espacio tf-idf.

Wang Supervisado

Es el último de los métodos propuestos en [73]. A diferencia de los anteriores, este es un **método supervisado**, por lo que hace uso de las etiquetas de clase disponibles en el *dataset*. Al igual que Wang utiliza una **lista de vecinos más cercanos** y un ***heat kernel*** para construir la matriz $\boldsymbol{W}$, pero además **incluye las constantes** $\boldsymbol{a}$ y $\boldsymbol{b}$ que ocupan el lugar del coeficiente c_{ij} para aumentar o disminuir la influencia de un termino en la matriz dependiendo si es que los elementos x_i y x_j comparten o no la etiqueta de clase.

$$W_{i,j} = \begin{cases} a \cdot \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), & \text{si} \quad [x_i \in \mathbf{N}_k(x_j) \ \vee \ x_j \in \mathbf{N}_k(x_i)] \ \wedge \ t_i = t_j \\ b \cdot \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), & \text{si} \quad [x_i \in \mathbf{N}_k(x_j) \ \vee \ x_j \in \mathbf{N}_k(x_i)] \ \wedge \ t_i \neq t_j \\ 0, & \text{en cualquier otro caso} \end{cases} \tag{4.4}$$

Al igual que en Wang, la matriz $\boldsymbol{W}$ resultante es dispersa. La única innovación que este método introduce con respecto al método anterior es la inclusión de los parámetros a y b por lo que se espera que el comportamiento descrito por el resto de los parámetros en Wang se mantenga, siendo estos nuevos parámetros los únicos dignos de estudio.

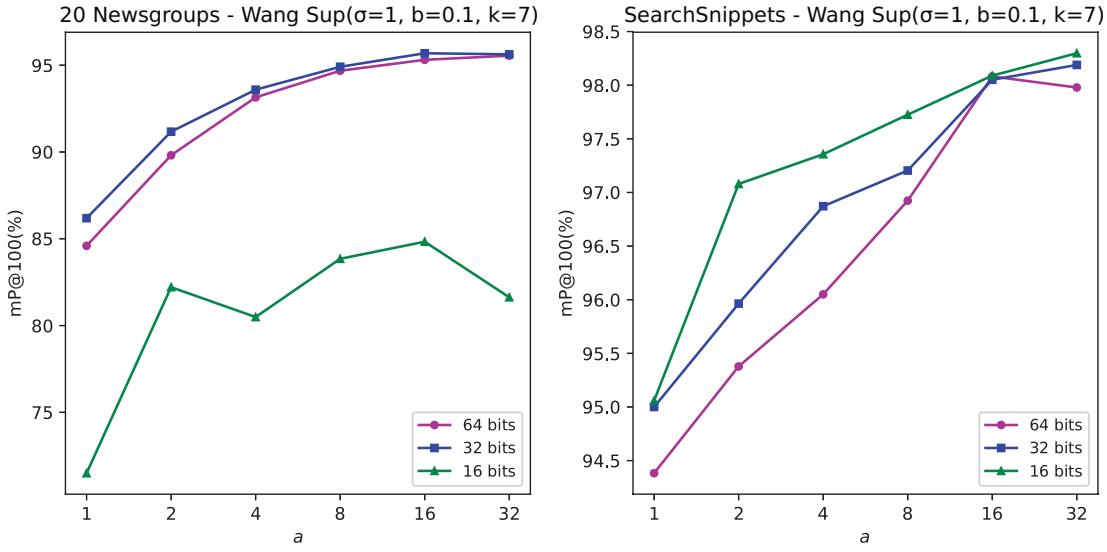


Figura 4.18: Resultados de mP@100 en función del parametro a a 16, 32 y 64 bits. Resultados sobre 20 Newsgroups a la izquierda y Searchsnippets a la derecha.

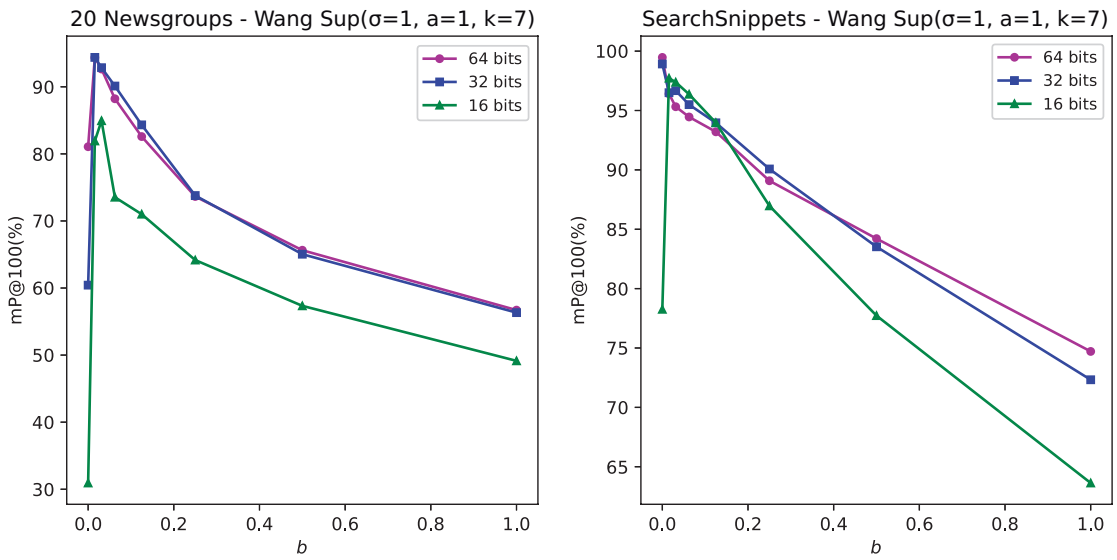


Figura 4.19: Resultados de mP@100 en función del parametro b a 16, 32 y 64 bits. Resultados sobre 20 Newsgroups a la izquierda y Searchsnippets a la derecha.

Como se observa en los gráficos anteriores el método es **muy sensible a la elección de los parámetros *a* y *b***. Cabe destacar como al aumentar el valor de a en ambos casos mejora el rendimiento del método hasta alcanzar una cota superior con $a = 16$ para Newsgroups con cadenas de 32 bits y con $a = 32$ para SearchSnippets con cadenas de 16 bits. Respecto al parámetro b llama la atención el mal desempeño del método cuando se establece su valor en cero, ya que, en teoría, esto descarta aquellas parejas de elementos que no comparten etiquetas. No obstante, en ambos casos el mejor desempeño se alcanza con el valor evaluado más cercano a cero de $b = 0{,}015$. Además se observa que a medida que aumenta el valor de b el desempeño decae.

Es importante aclarar que durante este proceso de sintonización no se realizó la búsqueda fina de parámetros debido a que el desempeño obtenido durante el proceso normal roza el óptimo global.

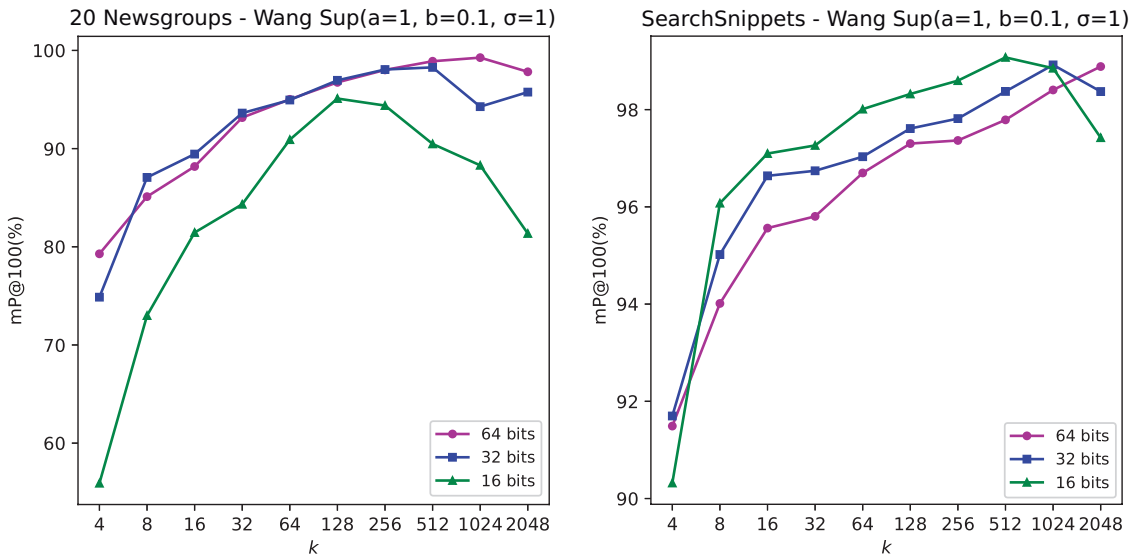


Figura 4.20: Resultados de mP@100 en función del parametro k a 16, 32 y 64 bits. Resultados sobre 20 Newsgroups a la izquierda y Searchsnippets a la derecha.

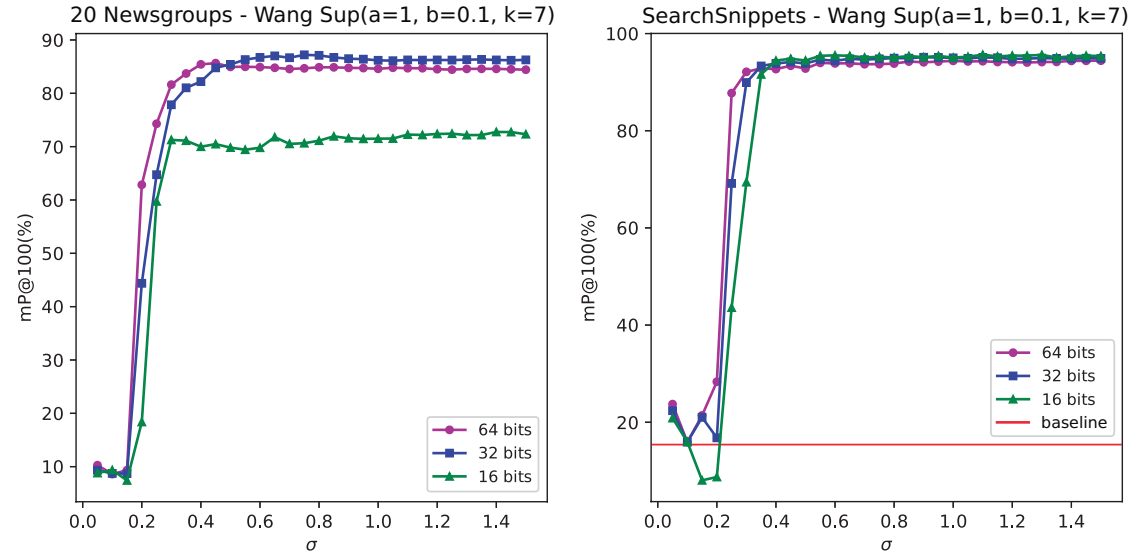


Figura 4.21: Resultados de mP@100 en función del parametro σ a 16, 32 y 64 bits. Resultados sobre 20 Newsgroups a la izquierda y Searchsnippets a la derecha.

En ambos casos se observa un comportamiento muy similar al observado para estos parámetros en Wang. No obstante, la variante supervisada parece ser menos sensible a la elección del parámetro k que su contraparte no supervisada. Dicho lo anterior, el mejor rendimiento respecto a k se obtiene con $k = 1024$ a 64 bits para Newsgroups y $k = 512$ a 16 bits para

SearchSnippets. Con respecto *sigma* los mejores valores encontrados son $\sigma = 0,75$ a 32 bits para Newsgroups y $\sigma = 0{,}6$ a 16 bits para SearchSnippets, aunque en realidad para este *dataset* el rendimiento bordea el 95 % para todas las cadenas con un $\sigma = 0,4$ o superior.

Finalmente, los parámetros seleccionados son $a = 16$, $b = 2^{-6}$, $k = 256$ y $\sigma = 0,75$ para 20 Newsgroups con cadenas de 32 bits, alcanzando un mP@100 del 99,91 %. Por su parte para SearchSnippets la configuración elegida es $a = 32$, $b = 2^{-6}$, $k = 512$ y $\sigma = 0,6$ con cadenas de 16 bits, alcanzado un mP@100 del 99,87 %. Estos resultados se encuentran en la figura A.3 del apartado anexo. En base a esta configuración se generaron los mapas de calor y los diagramas de dispersión que permiten entender mejor como los *prehashes* generados por este método dividen el espacio.

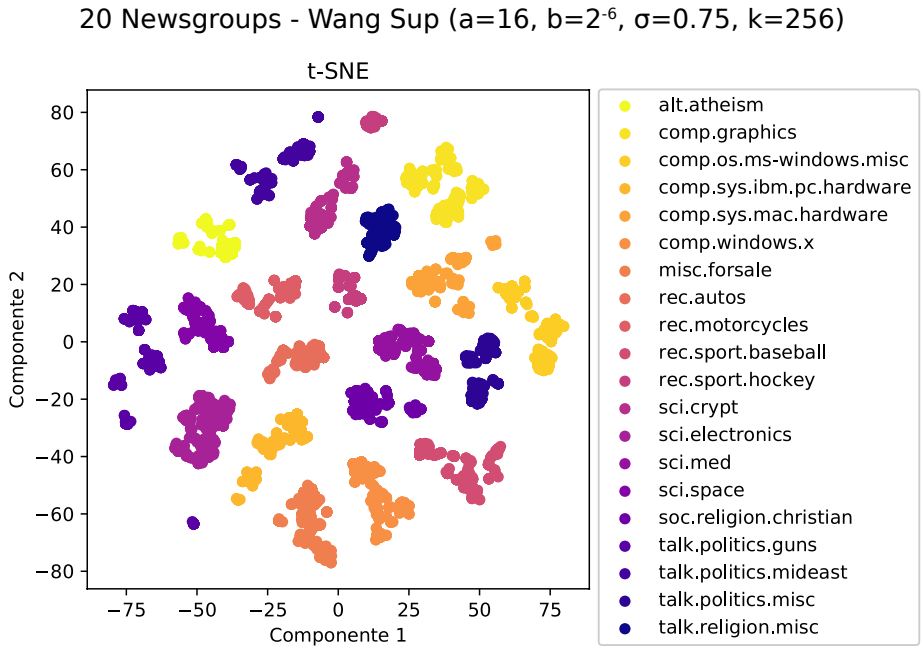


Figura 4.22: Diagrama de dispersión de los mejores prehashes obtenidos por Wang Supervisado para 20 Newsgroups. Reducción mediante t-SNE con distancia de hamming.

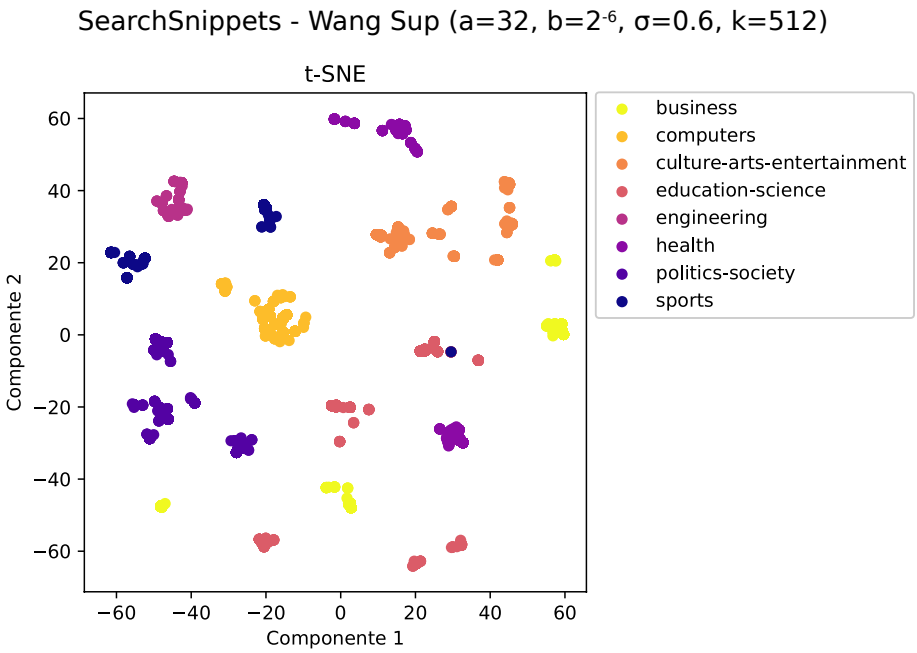


Figura 4.23: Diagrama de dispersión de los mejores prehashes obtenidos por Wang Supervisado para SearchSnippets. Reducción mediante t-SNE con distancia de hamming.

Ambos diagramas presentan una distribución cercana a la ideal. Los documentos de una misma clase se encuentran agrupados en un mismo *cluster*, compacto, bien definido y separado del resto de *clusters*. Dado lo anterior es posible afirmar que los *prehashes* generados por este método son superiores a los producidos por Wang y Weiss, ya que agrupan casi a la perfección documentos similares en el espacio de hamming. Este hecho se ve reflejado en los mapas de calor en donde se observa como la distancia intraclase es significativamente menor que la interclase.

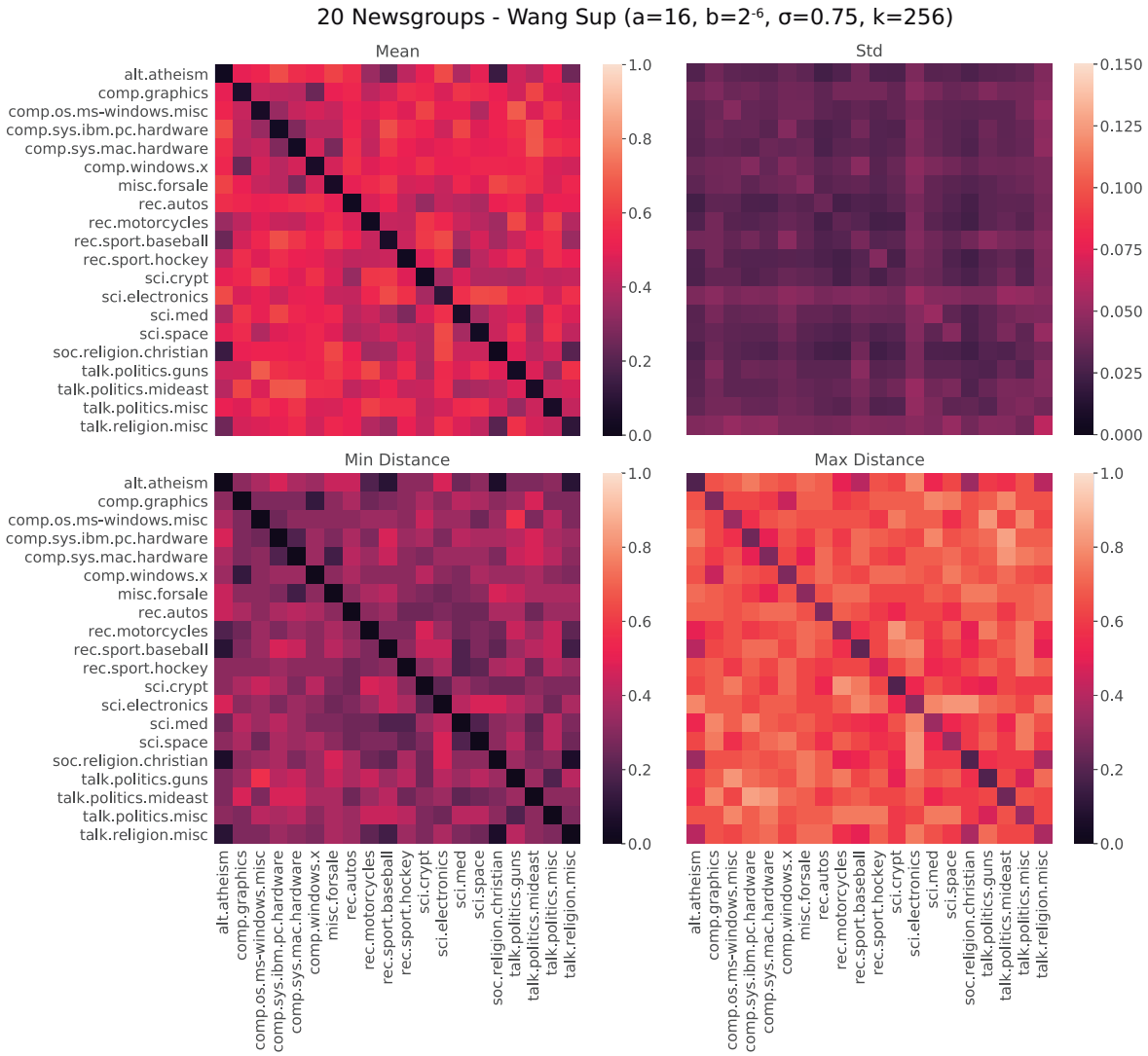


Figura 4.24: Mapas de calor de la distancia de hamming relativa entre clases, generados a partir de los mejores prehashes obtenidos por Wang Supervisado para 20 Newsgroups.

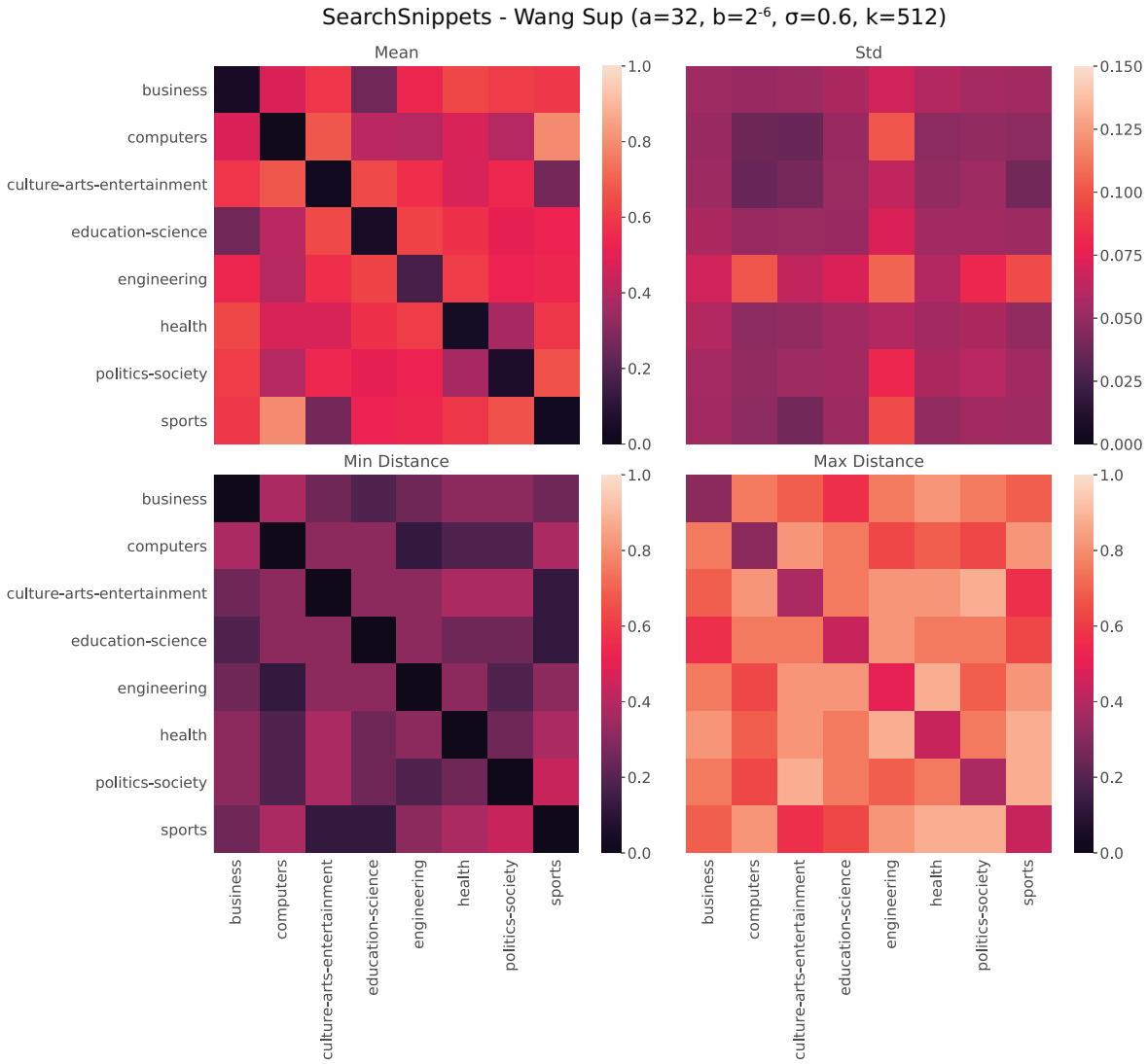


Figura 4.25: Mapas de calor de la distancia de hamming relativa entre clases, generados a partir de los mejores prehashes obtenidos por Wang Supervisado para SearchSnippets.

En ambos casos se observa una marcada diagonal principal lo que es equivalente a distancias intraclase menores. Llama la atención, sobre todo, la matriz de distancia mínima ya que es el único caso presentando hasta ahora en donde las colisiones entre clases solo dominan la diagonal de la matriz.

Dicho lo anterior se procede a describir la fase de entrenamiento en donde se utilizan los *prehashes* generados en esta sección para entrenar la red descrita en la sección 3.2.

4.3. Entrenamiento

En este apartado se presenta la **implementación real del modelo propuesto en la sección 3.2**. El primer paso consiste en construir la representación necesaria para poder alimentar a la red neuronal. Como se mencionó anteriormente se optó por utilizar un *word embedding* preentrenado basado en GloVe [58] entrenado sobre 6 mil millones de tokens de la Wikipedia 2014 y Gigaword 5. Este *embedding* está compuesto por 400.000 palabras sin preprocesar representadas como vectores reales de 50 dimensiones. No obstante, 20 Newsgroups se encuentra preprocesado mediante *stemming* por lo que la cobertura otorgada por GloVe para este *dataset* es limitada. Para solventar este problema se decidió aplicar *stemming* sobre el *embedding*, utilizando la función `PorterStemmer` de la librería `NLTK`, lo que incrementa considerablemente la cobertura. A continuación, se presenta una tabla con las coberturas alcanzadas para el conjunto de entrenamiento de ambos *dataset*.

Cuadro 4.2: Cobertura del *embedding* sobre los conjuntos de entrenamiento de ambos *datasets*. $|V|$ es el tamaño del vocabulario sin las normalizaciones introducidas por `TfidfVectorizer` y $|T|$ es el número de *tokens*.

Dataset	**GloVe**		**GloVe + Porter**	
	$\|V\|$	$\|T\|$	$\|V\|$	$\|T\|$
20 Newsgroups	20793/41877 (50 %)	796845/984277 (81 %)	27743/41877 (66 %)	937987/984277 (95 %)
SearchSnippets	18164/25143 (72 %)	170990/181889 (94 %)	8957/25143 (36 %)	86168/181889 (47 %)

Como puede apreciarse aplicar *stemming* sobre GloVe tiene efectos dispares. Para 20 Newsgroups representa una mejora de 16 puntos en la cobertura del vocabulario y 14 puntos en la cobertura de los términos. No obstante, para SearchSnippets representa una pérdida del 50 % en cobertura, respecto a la versión sin modificar. Este comportamiento es el esperado ya que 20 Newsgroups esta preprocesado, a diferencia de SearchSnippets. Por consiguiente, se utilizará la variante con el algoritmo de Porter para tratar los documentos de 20 Newsgroups y la versión normal para tratar los de SearchSnippets. Para entender mejor como las palabras

se encuentran distribuidas en los *datasets* se presentan los siguientes gráficos de frecuencia.

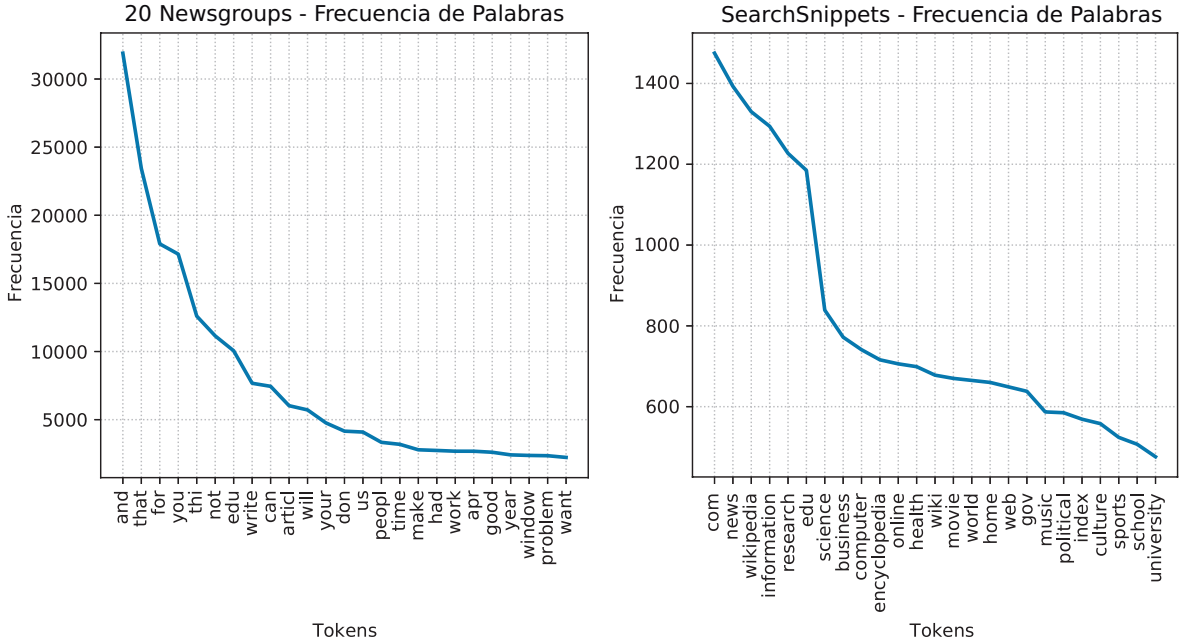


Figura 4.26: Frecuencia del *top* 25 de palabras de los conjuntos de entrenamiento para ambos *datasets*. 20 Newsgroups a la izquierda y SearchSnippets a la derecha.

A partir de los gráficos anteriores es posible extraer las siguientes observaciones. En primer lugar, se advierte que ambas curvas describen una **ley de potencia**, lo cual concuerda con lo planteado por la **ley de Zipf** [27]. Por su parte, las 25 palabras más utilizadas en 20 Newsgroups corresponden en su mayoría a conectores, lo cual es normal si se considera que las *stop words* no fueron removidas. Asimismo este fenómeno no ocurre en SearchSnippets ya que las *stop words* fueron eliminadas de antemano. No obstante, las palabras más utilizadas en este caso corresponden a términos descriptivos de las categorías, como por ejemplo *science*, *health* y *business*.

Dado lo anterior también es importante conocer la frecuencia de los términos que no están cubiertos por el *embedding*. Para esto se construyó un segundo gráfico de frecuencia, esta vez, para las palabras perdidas.

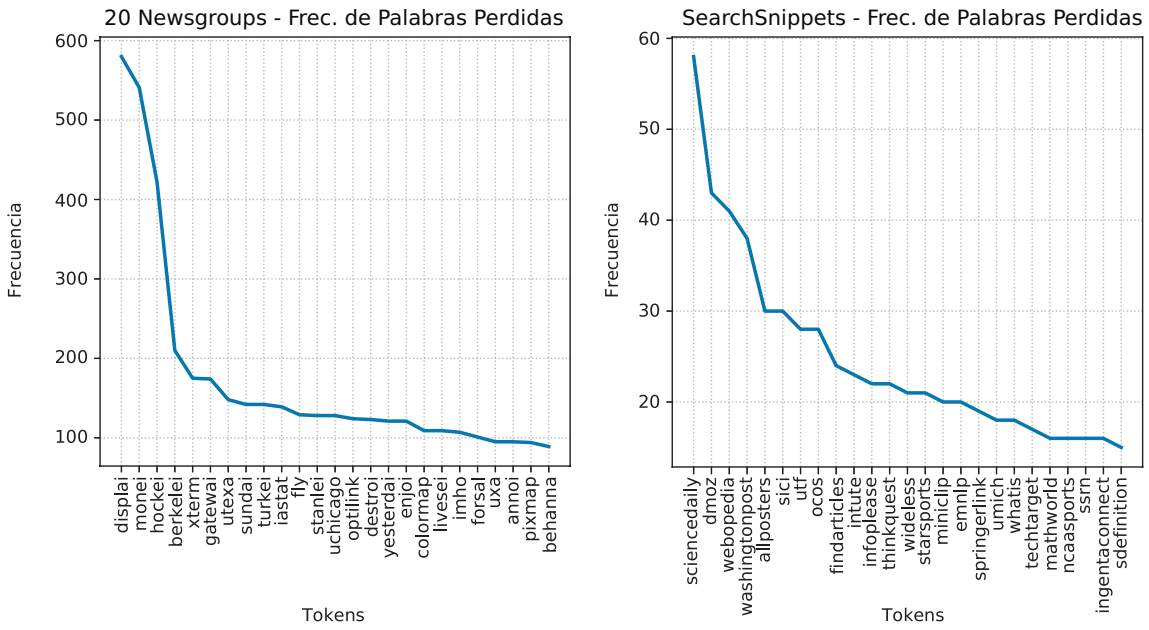


Figura 4.27: Frecuencia del *top* 25 de palabras perdidas por el *embedding* de los conjuntos de entrenamiento para ambos *datasets*. 20 Newsgroups a la izquierda y SearchSnippets a la derecha.

En ambos casos se observa que las palabras no cubiertas por el *embedding* también tienden a ser términos descriptivos de las categorías, así como nombres propios. Aunque su frecuencia sea significativamente menor, todavía constituyen poco más de un cuarto del vocabulario de cada *dataset*. Dado lo anterior, es importante abordar el problema de las palabras no cubiertas por el *embedding*. Una solución habitual es utilizar vectores aleatorios, tal y como se hace en [73]. No obstante, si se utilizan vectores completamente aleatorios se corre el riesgo de quedar en una zona remota del espacio de representación, modificando de manera significativa la distribución del *embedding*. Para mitigar este riesgo, **se propone seleccionar un valor aleatorio para cada componente del vector, proveniente de la distribución de probabilidad observada para esa dimensión a lo largo del *embedding***.

Por lo tanto, para generar el vector aleatorio es necesario conocer la distribución de probabilidad subyacente para cada dimensión del *embedding*.



Figura 4.28: Distribución de probabilidad observada para cada componente del *embedding*. La curva roja representa una distribución normal a partir del μ y σ obtenidos para de cada componente.

En base a la figura anterior es posible afirmar que **todas las dimensiones del *embedding* se distribuyen normalmente**, exceptuado tal vez las componentes 26 y 31, aunque para estas, la curva normal sigue siendo una aproximación aceptable. Por lo tanto, las componentes del vector generado para cubrir las palabras perdidas en GloVe, se elegirán a partir de un valor aleatorio proveniente de la distribución normal para esa componente. A continuación, se presenta una implementación autocontenida de esta funcionalidad.

```
import numpy as np
import pandas as pd
import csv

def generate_random_vector():

    features = pd.read_table("../Glove/glove.6B.50d.txt", sep="␣", index_col=0,
                                header=None, quoting=csv.QUOTE_NONE)
    features_means = features.describe().loc['mean'].values
    features_std = features.describe().loc['std'].values

    generated_vector = []

    for i in range(len(features_means)):
        normal_dist = np.random.normal(features_means[i], features_std[i], 10000)
        selected_element = np.random.choice(normal_dist)
        generated_vector.append(selected_element)

    return np.asarray(generated_vector)
```

Código 4.1: Generador de vectores aleatorios. El código presentado corresponde a una unidad funcional y difiere de la utilizada en la implementación final.

Una vez obtenida la representación matricial de los documentos es necesario construir un **cubo de datos para entrenar la red**. El problema radica en que los documentos poseen diferentes extensiones por lo que es necesario estandarizar sus longitudes. Para realizar esto se llena el espacio vacío de los documentos utilizando una palabra de relleno específica, hasta que todos los documentos tienen la extensión del documento de mayor longitud. A este proceso de normalización se le conoce como ***padding***. En la implementación realizada

en esta memoria se utilizo como marca de *padding* de manera arbitraria el número de 100 ya que este se encuentra fuera de los valores que puede tomar una dimensión del *embedding*. Dicho lo anterior, para rellenar los espacios vacíos se utilizan vectores de 50 dimensiones cuyas componentes tienen valor 100.

Como último paso antes de entrenar la red es necesario aplicar una representación sobre los *prehashes* y su respectivo *shift*. Para realizar esto se utiliza una técnica conocida como ***one-hot encoding*** que consiste en generar un vector de longitud igual a todos los posibles valores que puede tomar la variable. Luego se marcan todos los elementos del vector como cero con la excepción de la posición que representa el valor que tiene que representar el vector, que se marca con un uno. En la implementación realizada en esta memoria se utilizó un vector de dos dimensiones en donde la primera posición representa un 0 y la segunda posición representa un 1. Para representar el carácter de *shift* ambas componentes se marcan con un 0. A continuación se presenta una pequeña ilustración a modo de ejemplo.

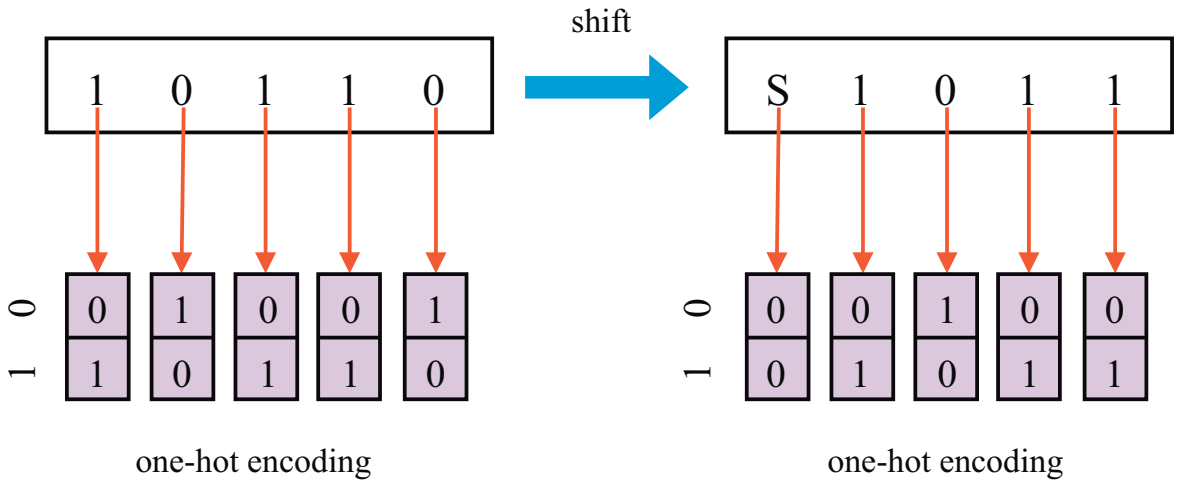


Figura 4.29: Ejemplo de *shift* y *one-hot encoding* sobre un *prehash*. La *S* representa el caracter de *shift*.

Completadas las tareas de preprocesamiento, se procede a implementar y entrenar la red neuronal. El primer componente de la arquitectura propuesta es el ***encoder***. Este consiste en una capa **LSTM** compuesta por **256 unidades** (neuronas) con **función de activación** *tanh*. Lo importante de esta red es su estado interno al terminar de procesar la secuencia de vectores que compone un documento. Este estado interno, también conocido como *vector de contexto*, se utiliza a su vez para inicializar el *decoder*. A continuación, se presenta el código de la implementación real de este componente.

```
from keras.models import Model
from keras.layers import Input
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Masking
from keras.callbacks import EarlyStopping, ModelCheckpoint
import numpy as np

num_features = glove_em.shape[2] # 50
hash_features = encoded_hash_codes.shape[2] # 2
latent_dim = 256
len_hash = encoded_hash_codes.shape[1] # 16, 32 or 64
num_samples = glove_em.shape[0] # 20Newsgroups: 8354, SearchSnippets: 8048
max_lenght = glove_em.shape[1] # 20Newsgroups: 300, SearchSnippets: 50

# Training Model

# Encoder Definition
encoder_inputs = Input(shape=(None, num_features))
encoder_inputs_mask = Masking(mask_value = 100.0)(encoder_inputs)
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs_mask)

encoder_states = [state_h, state_c]
```

Código 4.2: Implementación del *Encoder*.

Del código anterior cabe destacar los siguientes puntos:

- Las dimensiones de la capa de entrada se define como `(None, num_features)` ya que en su fase de predicción la red puede **procesar documentos de diferente extensión**.
- La capa `Masking` permite a la red **ignorar los vectores de *padding*** agregados a los documentos.
- En la declaración de la capa LSTM se marca el argumento `return_state=True` para

que la red a parte de su *output* retorne la memoria de sus unidades (*cell state*) y sus estados ocultos (*hidden state*) finales. Estos atributos que en conjunto denominamos como "estado de la red" se almacena para inicializar el *decoder*.

Construido el *encoder*, queda construir la segunda componente de la arquitectura, el ***decoder***. El objetivo del *decoder* es reconstruir el prehash original a partir de su *shift* y los estados finales generados por el *encoder*.

```
25 # Decoder Definition
26 decoder_inputs = Input(shape=(None, hash_features))
27 decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
28 decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)
29 decoder_dense = Dense(hash_features, activation='softmax')
30 decoder_outputs = decoder_dense(decoder_outputs)
```

Código 4.3: Implementación del *Decoder*.

Sobre el *decoder* cabe recalcar:

- En la declaración de la capa LSTM se incluye el parámetro `return_sequences=True` para que la red retorne su **estado oculto en cada paso**.
- Al especificar el input del `decoder_lstm` se establece mediante el uso del parámetro `initial_state`, como estado **inicial de la red**, el **estado final del *encoder***.
- Finalmente se utiliza una capa de salida densa con función de activación $\boldsymbol{softmax}$ para conseguir la "probabilidad" de que el bit predicho sea 0 o 1.

Finalmente, el *encoder* y el *decoder* se fusionan en un **único modelo** el cual será entrenado para realizar la tarea.

```
31 model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
32 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
```

Código 4.4: Generación del modelo de entrenamiento.

Como puede apreciarse, el *input* de este modelo está compuesto por el *embedding* de un documento para el *encoder* y el *shift* de su respectivo *prehash* para el *decoder*. Siendo el *output* del modelo la salida retornada por la capa $softmax$ del *decoder*. La función de pérdida para el modelo es la ***Categorical Cross-Entropy***, ampliamente utilizada en problemas de clasificación, la cual mide la diferencia entre la distribución de probabilidad real observada y la distribución predicha por la red. La métrica de desempeño utilizada es el ***Accuracy*** (exactitud) que corresponde al ratio entre el número de predicciones correctas y el total de predicciones realizadas.

Para concluir lo referente a la implementación del modelo, se presenta a continuación un diagrama de la arquitectura implementada.

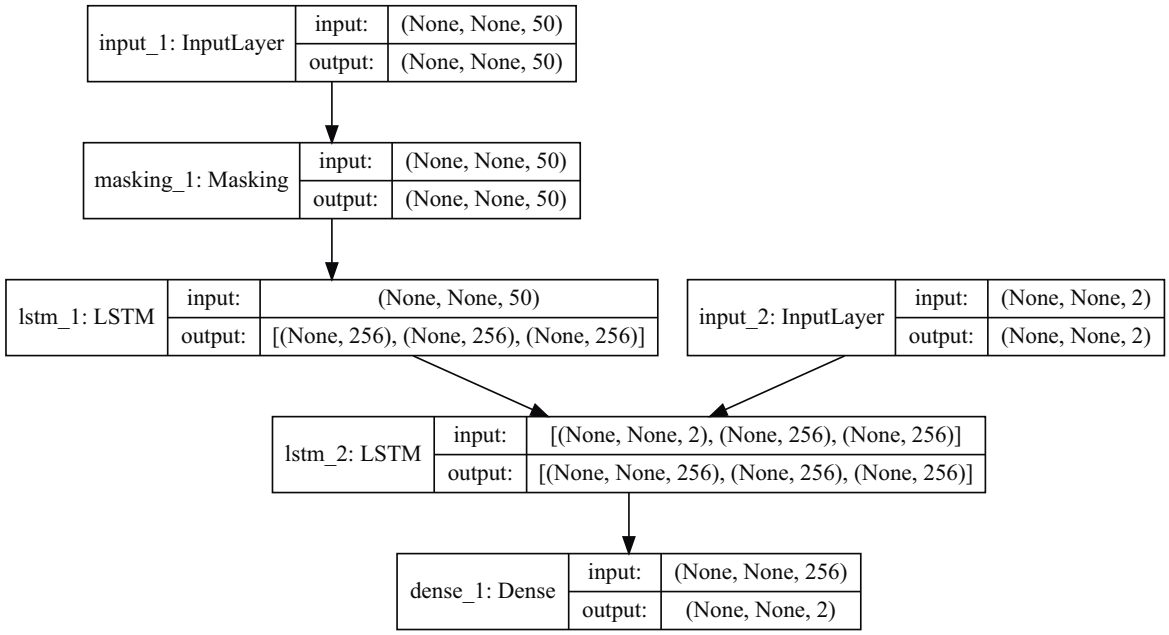


Figura 4.30: Diagrama del modelo de entrenamiento. Generado automáticamente por la función `model_to_dot` de la librería `keras`.

Para entrenar la red y evitar los problemas relacionados al *overfitting* se utilizó ***Early Stopping*** con una tolerancia de 5 *epochs* sobre la pérdida del conjunto de validación (*validation loss*). Lo anterior implica que el modelo dejara de entrenarse luego de aumentar el *validation loss* durante 5 *epochs* consecutivos. Para generar el conjunto de validación se realizó un *split* aleatorio del 20 % sobre el conjunto de entrenamiento.

```
callbacks = [EarlyStopping(monitor='val_loss', patience = 5, verbose = 1)]

history = model.fit(
    [glove_em, encoded_shited_hash_codes],
    encoded_hash_codes,
    epochs = 200,
    validation_data = ([val_glove_em, val_encoded_shited_hash_codes],
                       val_encoded_hash_codes),
    callbacks = callbacks
)
```

Código 4.5: *Early Stopping* y Entrenamiento del modelo.

Finalmente se procedió a entrenar la red bajo un máximo de 200 *epochs* con el *Early Stopping* anteriormente descrito. A continuación, se presentan los resultados del entrenamiento mediante los *gráficos de Accuracy y Loss* del modelo.

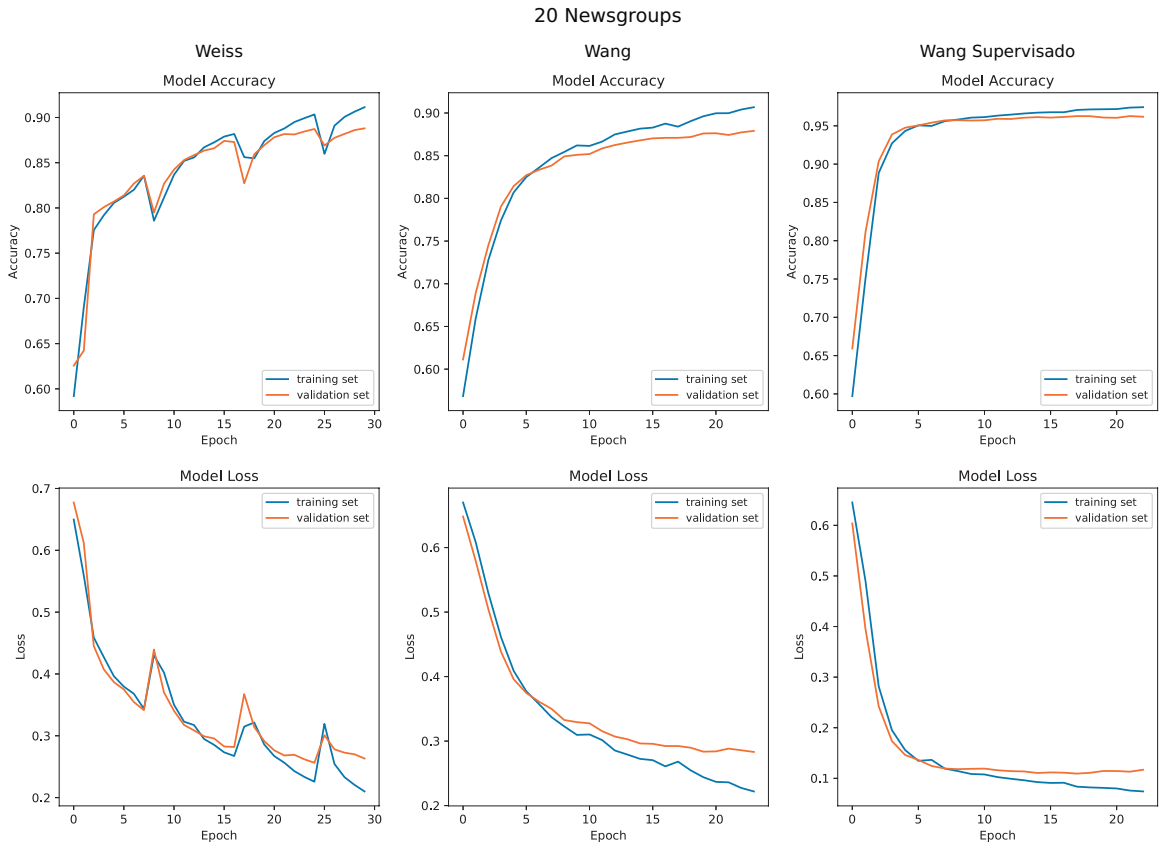


Figura 4.31: Gráficos de *Accuracy* y *Loss* del entrenamiento del modelo para 20 Newsgroups.

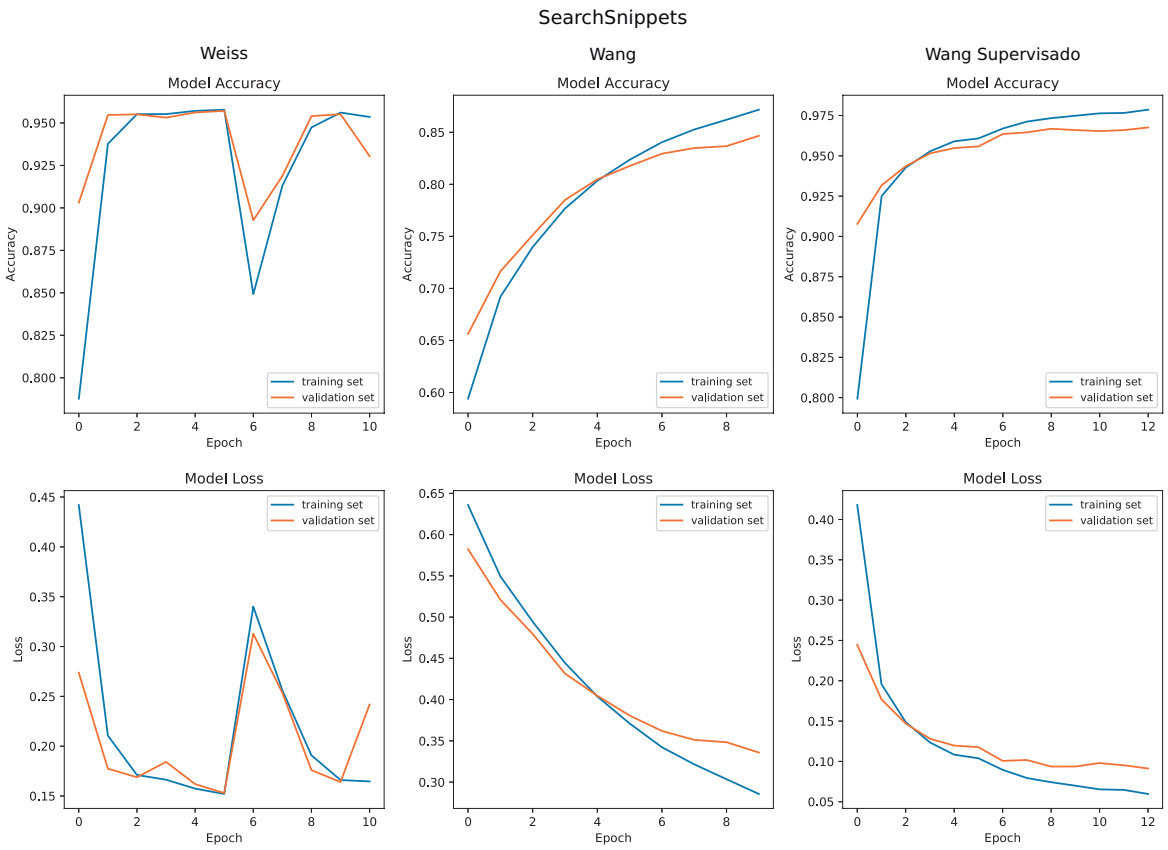


Figura 4.32: Gráficos de *Accuracy* y *Loss* del entrenamiento del modelo para SearchSnippets.

Los gráficos anteriores muestran el comportamiento clásico del entrenamiento de un modelo. En ambos casos se observa como las curvas de *training* y de *validation* se comportan de igual manera hasta llegar a un punto de quiebre en donde las curvas de ambos conjuntos se separan mostrando siempre un mejor desempeño la curva de *train*. Esto se debe al fenómeno de **sobreajuste** estudiado en la sección 1.1.3 y calza perfectamente con lo mostrado por la figura 1.2. Sin embargo, no es posible afirmar que el sobreajuste representa un problema para el modelo entrenado, ya que en este caso **la brecha de generalización observada es muy pequeña**. También cabe destacar que en ambos *datasets* los mejores resultados se obtienen con los *prehashes* generados por Wang Supervisado, luego por Wang y finalmente por Weiss. Lo anterior indica que los resultados del entrenamiento están intrínsecamente ligados a la calidad de los *prehashes*, tal y como era esperado.

Por último, cabe señalar lo rápido del entrenamiento del modelo, llevándose a cabo en aproximadamente 30 *epochs* para Newsgroups y en apenas 11 *epochs* para SearchSnippets.

4.4. Predicción y Evaluación

En esta sección se **presentan y evalúan las predicciones echas por el modelo**. El primer paso para realizar las predicciones consiste en modificar el modelo entrenado para generar una secuencia de manera recursiva. Este nuevo **modelo de inferencia** utiliza los pesos aprendidos por el modelo de entrenamiento, por lo que no requiere ser entrenado. El ***encoder* de inferencia** es simplemente un modelo que contiene el *encoder* de entrenamiento.

```
46 #Inference Model
47 #Encoder Definition
48 encoder_model = Model(encoder_inputs, encoder_states)
```

Código 4.6: Implementación del *Encoder* de inferencia.

Por su parte el ***decoder* de inferencia** sufre modificaciones más relevantes. Ahora constituye un modelo completamente separado del *encoder*, interactuando con el solamente dentro de la función `predict_hash_code` que definiremos más adelante. Este modelo se utiliza de manera **recursiva** retornando un *bit* del *hash* por cada invocación que se le haga. A continuación, se presenta el código de la implementación del *decoder* de inferencia.

```
49 #Decoder Definition
50 decoder_inputs = Input(shape=(None, hash_features))
51 decoder_state_input_h = Input(shape=(latent_dim,))
52 decoder_state_input_c = Input(shape=(latent_dim,))
53
54 decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
55 decoder_outputs, state_h, state_c = decoder_lstm(decoder_inputs,
56                                                  initial_state=decoder_states_inputs)
57 decoder_states = [state_h, state_c]
58 decoder_outputs = decoder_dense(decoder_outputs)
59 decoder_model = Model([decoder_inputs] + decoder_states_inputs,
60                       [decoder_outputs] + decoder_states)
```

Código 4.7: Implementación del *Decoder* de inferencia.

Ya que el *encoder* y el *decoder* son modelos completamente separados es necesario proveer explícitamente al *decoder* de los estados de finales del *encoder* por lo que se definen dos nuevas capas de entrada, `decoder_state_input_h` y `decoder_state_input_c`. Ya que estas capas forman parte del *input* del modelo, es necesario proveer de estados a la red en cada iteración. Por lo tanto, en cada interacción posterior a la inicial a la red se le provee con el *bit* y los estados obtenidos en la iteración anterior.

Para ilustrar de mejor manera el modelo de inferencia se presenta el siguiente diagrama.

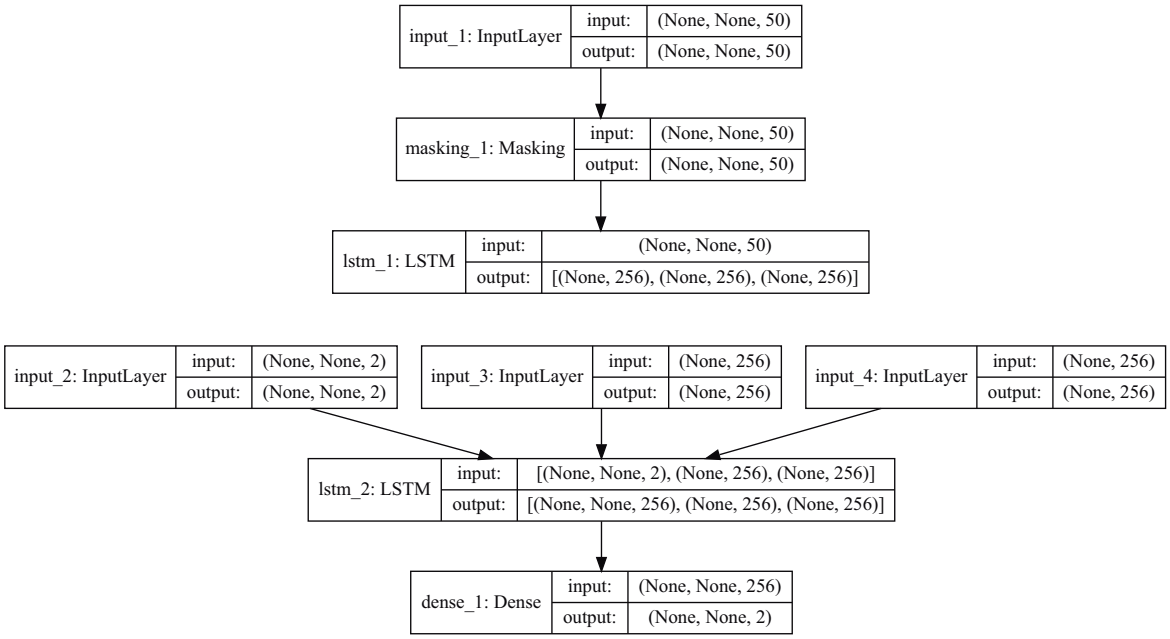


Figura 4.33: Diagrama del modelo de inferencia. Generado automáticamente por la función `model_to_dot` de la librería `keras`.

Finalmente, para obtener el *hash* de un documento utilizando el modelo de inferencia anteriormente descrito, se implementa la función **`predict_hash_code`**, la cual recibe el *embedding* de un documento y retorna su *hash*. Como puede apreciarse en el código tras el salto, la longitud del *hash* esta definida por la variable *len_hash*, siendo un proceso iterativo en el que en cada iteración se selecciona el valor del *bit*, en base a la probabilidad retornada por el modelo.

```
60 def predict_hash_code(input_seq):
61
62     # Encode the input as state vectors.
63     states_value = encoder_model.predict(input_seq)
64
65     # Generate an empty target sequence.
66     target_seq = np.zeros((1, 1, hash_features))
67     hash_code = []
68
69     prepare = prepare_input()
70
71     for _ in range(len_hash):
72
73         output_tokens, h, c = decoder_model.predict([target_seq] + states_value)
74
75         decoded_hash = prepare.decode_hash(output_tokens[0])
76
77         hash_code.append(decoded_hash[0]);
78
79         selected_bit = np.argmax(output_tokens[0])
80         target_seq = np.zeros((1, 1, hash_features))
81         target_seq[0, 0, selected_bit] = 1
82         states_value = [h, c]
83
84     return np.asarray(hash_code)
```

Código 4.8: Implementación de la función de predicción.

Implementada la función de predicción se procede a construir el ***ground truth*** correspondiente para cada *dataset*. Para obtener el *ground truth* primero se genera un *hash* para cada elemento del conjunto de entrenamiento utilizando la función de predicción, luego se almacena el *hash* generado con su respectiva etiqueta de categoría.

Una vez construido el *ground truth* se procede a realizar una búsqueda por similitud utilizando como consulta los *hashes* generados por la función de predicción para los documento del *test set*. El desempeño final de la técnica propuesta se evalúa utilizando ***mean precision*** (mP)

sobre el top 50, 100, 150 y 200 de documentos recuperados. A continuación, se presenta una tabla con los resultados obtenidos por la arquitectura *encoder-decoder* (ED) en conjunto con las técnicas de preentrenamiento utilizadas, para ambos *datasets*.

Cuadro 4.3: Tabla de resultados obtenidos en términos mP@k(%) de por la arquitectura *encoder-decoder* (ED) bajo Weiss, Wang y Wang Supervisado. El largo de los *hashes* varía dependiendo del *dataset* y de la técnica de preentrenamiento utilizada. Las longitudes utilizadas para Weiss, Wang y Wang Supervisado son 64, 64 y 32 *bits* para 20 Newsgroups y 64, 32 y 16 *bits* para SearchSnippets respectivamente.

Método	**20 Newsgroups**			
	mP@50(%)	mP@100(%)	mP@150(%)	mP@200(%)
ED - Weiss	31,48	29,83	28,37	27,42
ED - Wang	47,84	47,41	46,77	45,95
ED - Wang S	64,92	65,09	64,71	64,37

Método	**SearchSnippets**			
	mP@50(%)	mP@100(%)	mP@150(%)	mP@200(%)
ED - Weiss	22,58	22,50	22,39	22,16
ED - Wang	76,48	75,61	75,11	73,83
ED - Wang S	80,62	80,64	80,71	80,58

Como puede apreciarse en el cuadro anterior, el rendimiento sobre SearchSnippets es superior al alcanzado sobre 20 Newsgroup. No obstante, **los resultados sobre ambos *datasets* presentan un comportamiento similar**. En los dos casos la red entrenada utilizando los *prehashes* generados por Wang Supervisado obtiene los mejores resultados, seguida por la red entrenada con Wang y finalizando con la entrenada con Weiss. Lo anterior se condice con lo observado en la sección de preentrenamiento, ya que la calidad de los *prehashes* generados sigue este mismo orden.

En términos globales se aprecia que en todos los casos estudiados **el rendimiento del ED supera con creces la línea base** marcada por el recuperador aleatorio definido en la figura

4.1. Sin embargo, para obtener una mejor idea del desempeño real de este método es necesario compararlo con otras técnicas de *hashing* sobre los mismos *datasets*. Para realizar esto se incorporaron los resultados obtenidos en la tabla 4.3 a la tabla comparativa expuesta en [73]. Las tablas resultantes se presentan a continuación divididas por *dataset*.

Cuadro 4.4: Tabla comparativa de resultados para 20 Newsgroups. Las siglas describen los siguientes métodos: *Encoder-Decoder* (ED), *Text Hashing with Convolutional neural networks* (THC), Nuestra implementación de THC (CNN), *Self Taught Hashing* (STH), *Iterative Quantization* (ITQ), *Laplacian Co-Hashing* (LCH), *Spectral Hashing* (SpH), *Two Step Hashing* (TSH). Todos los resultados con excepción de ED se obtuvieron empleando cadenas de 64 bits.

-	**Método**	**20 Newsgroups**			
		mP@50(%)	mP@100(%)	mP@150(%)	mP@200(%)
No Supervisado	ED-Weiss	31,48	29,83	28,37	27,42
	ED-Wang	47,84	47,41	46,77	45,95
	THC-I (Wang)	65,83	63,73	61,82	60,03
	CNN-Wang	58,95	57,26	56,05	55,09
	STH-RBF	63,82	61,82	59,88	57,95
	STH	63,63	61,16	58,98	56,88
	ITQ	48,31	46,55	44,87	43,17
	LCH	32,75	30,41	28,89	27,69
	SpH	32,81	27,84	24,85	22,78
Supervisado	ED-Wang S	64,92	65,09	64,71	64,37
	THC-III (Wang)	66,21	64,02	62,11	60,31
	THC-IV (Wang S)	79,53	78,13	76,80	75,96
	CNN-Wang S	77,43	77,39	77,30	77,20
	TSH-KSH	75,83	75,74	75,57	75,24
	FastHash	73,76	73,54	73,34	73,17
	STHs	77,78	76,80	75,26	73,23

En 20 Newsgroups se observa que para el caso **no supervisado el método propuesto es**

superado ampliamente por técnicas como como THC y STH. No obstante, la variante con Wang se encuentra a la par con ITQ y supera a otras técnicas como LCH y SpH. Respecto a esta última es interesante notar la poca diferencia con respecto a la variante con Weiss, ya que este método de preentrenamiento proviene precisamente de *Spectral Hashing* [71]. Para el caso supervisado ED se encuentra a la par con THC-III pero sigue por debajo del resto de métodos. En CNN el rendimiento es inferior a THC, por entre 5 y 2 puntos porcentuales.

Cuadro 4.5: Tabla comparativa de resultados para SearchSnippets. Las siglas describen los siguientes métodos: *Encoder-Decoder* (ED), *Text Hashing with Convolutional neural networks* (THC), Nuestra implementación de THC (CNN), *Self Taught Hashing* (STH), *Iterative Quantization* (ITQ), *Laplacian Co-Hashing* (LCH), *Spectral Hashing* (SpH), *Two Step Hashing* (TSH). Todos los resultados con excepción de ED y CNN se obtuvieron empleando cadenas de 64 bits.

-	**Método**	**SearchSnippets**			
		mP@50(%)	mP@100(%)	mP@150(%)	mP@200(%)
No Supervisado	ED-Weiss	22,58	22,50	22,39	22,16
	ED-Wang	76,48	75,61	75,11	73,83
	THC-I (Wang)	78,07	77,46	76,80	76,08
	CNN-Wang	70,45	69,10	68,10	67,03
	STH-RBF	68,94	67,51	66,38	65,18
	STH	65,44	63,70	62,45	61,24
	ITQ	35,03	33,24	31,73	30,45
	LCH	54,20	52,36	51,09	50,05
	SpH	32,74	29,47	27,28	25,72
Supervisado	ED-Wang S	80,62	80,64	80,71	80,58
	THC-III (Wang)	79,32	78,67	78,00	77,27
	THC-IV (Wang S)	80,56	79,90	79,21	78,50
	CNN-Wang S	78,92	78,41	77,92	77,67
	TSH-KSH	71,09	70,87	70,76	70,34
	FastHash	61,23	60,97	60,52	60,22
	STHs	69,31	68,39	67,27	66,05

En SearchSnippets los resultados son mucho más prometedores. En el caso no supervisado se aprecia que ED-Wang se posiciona muy cercano a THC-I casi liderando la tabla. Muy por el contrario, el rendimiento de Weiss es el peor de esta sección quedando muy por debajo de SpH. En el **caso supervisado ED-Wang S presenta el mejor desempeño** de entre todos los métodos señalados, siendo el más cercano THC-IV. Nuevamente CNN demuestra un desempeño inferior a THC. Para el caso no supervisado, la diferencia se sitúa en torno a los 8 puntos porcentuales, mientras que en el caso supervisado la diferencia nuevamente de 2 puntos porcentuales.

A partir de las tablas anteriores es posible extraer algunas conclusiones y realizar ciertas hipótesis. En primer lugar, es evidente que **el rendimiento observado para SearchSnippets excede significativamente al de 20 Newsgroups**. Hay muchos factores que pueden contribuir a explicar este fenómeno, uno de ellos es la cobertura del *embedding*. Como quedo patente en la tabla 4.2 la cobertura sobre el vocabulario de SearchSnippets superaba a la de 20 Newsgroups por 22 puntos porcentuales. Al aplicar *stemming* para el caso de 20 Newsgroups se logró disminuir esta diferencia a 6 puntos, aunque sacrificando en parte la consistencia del *embedding* debido a la colisión de términos. Este problema de cobertura también nos hace sospechar de los resultados publicados en [73] ya que nuestra implementación obtuvo resultados inferiores, incluso amortiguando el problema mediante el uso del *stemming*.

Para evaluar el impacto de la cobertura se realizaron de nuevo los experimentos utilizando un *embedding* completamente aleatorio. Además, para el caso de 20 Newsgroups se utilizó GloVe sin *stemming* para evaluar el *trade-off* entre cobertura y consistencia. A continuación, se presentan las tablas con los resultados de estos experimentos.

Cuadro 4.6: Tabla comparativa de resultados en función del *embedding* utilizado sobre SearchSnippets. Se compara el mejor desempeño alcanzado por GloVe con el desempeño logrado utilizando un *embedding* completamente aleatorio (*random*).

Embedding	**Método**	**SearchSnippets**			
		mP@50(%)	mP@100(%)	mP@150(%)	mP@200(%)
GloVe	ED - Weiss	22,58	22,50	22,39	22,16
	ED - Wang	76,48	75,61	75,11	72,83
	ED - Wang S	80,62	80,64	80,71	80,58
Random	ED - Weiss	6,71	8,02	8,29	8,52
	ED - Wang	39,36	38,91	38,40	37,86
	ED - Wang S	37,93	37,92	37,85	37,85

Cuadro 4.7: Tabla comparativa de resultados en función del *embedding* utilizado sobre 20 Newsgroups. Se compara el mejor desempeño alcanzado por GloVe + Porter con el desempeño logrado por GloVe y un *embedding* completamente aleatorio (*random*).

Embedding	**Método**	**20 Newsgroups**			
		mP@50(%)	mP@100(%)	mP@150(%)	mP@200(%)
GloVe +Porter	ED - Weiss	31,48	29,83	28,37	27,42
	ED - Wang	47,84	47,41	46,77	45,95
	ED - Wang S	64,92	65,09	64,71	64,37
Glove	ED - Weiss	21,00	19,85	18,85	18,20
	ED - Wang	43, 25	42,28	42,05	41,28
	ED - Wang S	59,04	58,85	58,27	56,28
Random	ED - Weiss	5,65	5,73	5,62	5,69
	ED - Wang	29,91	29,75	29,34	28,96
	ED - Wang S	43,14	42,45	41,62	40,44

A partir de las tablas anteriores es acertado concluir que **la cobertura es un factor determinante a la hora de obtener un buen desempeño por parte del método**. En 20 Newsgroups

se aprecia una diferencia aproximada de 20 puntos porcentuales a todos los niveles entre GloVe + Porter y el *embedding* aleatorio. Respecto a la diferencia entre en GloVe y su versión con Porter, se aprecia que la decisión de aumentar la cobertura en detrimento de la consistencia del *embedding* se traduce en un incremento de rendimiento que oscila entre 5 y 10 puntos porcentuales, dependiendo del método de preentrenamiento utilizado. En SearchSnippets la diferencia entre GloVe y el *embedding* aleatorio es todavía más dramática generando una disminución en el desempeño de entre 30 y 40 puntos porcentuales dependiendo del método de preentrenamiento.

En definitiva, **la arquitectura propuesta en esta memoria presenta un desempeño que supera el estado del arte sobre SearchSnippets y un desempeño dentro del promedio para 20 Newsgroups**. Parte de esta diferencia puede ser explicada por la cobertura entregada por GloVe sobre ambos *datasets*, se profundizará más sobre este tema en el apartado de conclusiones.

Conclusiones

En este capítulo se presentan las conclusiones y reflexiones finales sobre el trabajo realizado, así como un conjunto de observaciones y propuestas sobre como ahondar en los diferentes aspectos que conforman la arquitectura expuesta en esta memoria.

Conclusiones Generales

Como se mencionó en el capítulo 3, la propuesta de esta memoria consiste en emplear un arquitectura de red recurrente, generalmente utilizada en tareas de traducción automática, para afrontar el problema de *hashing* para búsqueda por similitud. Los resultados obtenidos por la implementación de esta arquitectura señalan un desempeño notable del método para la recuperación de documentos de pequeña extensión, como los de SearchSnippets, y un desempeño razonable en conjuntos conformados por documentos de mayor extensión, como 20 Newsgroups. Este resultado parece indicar una verdad subyacente al concepto de similitud entre textos escritos, y es que, **entre mayor es la extensión de un texto, menor relevancia tiene el orden de las palabras a la hora de determinar su significado**.

La afirmación vertida en el párrafo anterior es razonable si se considera que la cantidad de tópicos tratados en un documento está estrechamente ligado a su extensión. De la misma forma, en un texto breve, la escasez de términos limita la cantidad de temas que pueden ser tratados, por lo que el orden de las palabras cobra una mayor relevancia a la hora de descifrar su significado. Para tratar este punto es necesario extender los experimentos a más *datasets* y evaluar si esta tendencia persiste, no obstante, eso ya forma parte del trabajo futuro.

Otro de los factores relevantes a la hora de evaluar el éxito del método es la cobertura proveída por el *embedding* al *dataset*. Como fue estudiado al final de la sección 4.4, **una baja cobertura se traduce en un empeoramiento significativo en el desempeño de la técnica**. Lo anterior se debe a que sin una cobertura adecuada la red no posee información sobre la similitud semántica entre términos en el espacio de representación, lo cual es vital para su correcto funcionamiento. Para profundizar todavía más en el impacto que genera el *embedding* utilizado sobre la técnica propuesta, es recomendable el uso de otros *embeddings* preentrenados, así como evaluar los costos y beneficios de entrenar un *embedding* específico para cada *dataset*.

Sobre las diferentes variantes de la técnica propuesta, cabe destacar la **prominente superioridad de la vertiente supervisada en comparación a las otras**. Esta es una tendencia que parece repetirse a lo largo de las técnicas de *hashing* estudiadas, como reflejan las tablas 4.4 y 4.5. Esto se debe a que el uso de etiquetas otorga conocimiento adicional al modelo sobre la distribución deseada de los elementos, lo que enriquece la información disponible en comparación a utilizar solamente los ejemplos. Un aspecto negativo del uso de etiquetas reside en los problemas que acarrea un mal etiquetado de los elementos. Por ejemplo, en los *datasets* utilizados en esta memoria se descubrió que un pequeño porcentaje de los ejemplos de entrenamiento poseían etiquetas diferentes, pese a ser léxicamente idénticos. Esta clase de errores pueden ocasionar serios problemas a algunas técnicas de aprendizaje supervisado como por ejemplo las máquinas de soporte vectorial (SVM). No obstante, la técnica propuesta en esta memoria demostró ser lo suficientemente robusta como para afrontar esta clase de problemas.

En el campo del aprendizaje no supervisado, **es preciso subrayar el enorme impacto del uso de la lista de vecinos cercanos en el desempeño del método**, como se refleja en la tabla 4.3. Esta innovación introducida en THC [73] supera por un amplio margen a la versión original propuesta en SpH [71]. Esto se debe a que al reducir el número de términos en la matriz $\boldsymbol{W}$ se incrementa la diferencia entre elementos disímiles lo que aumenta la efectividad del método de preentrenamiento.

En conclusión, el uso de una arquitectura tipo *encoder-decoder* basada en redes neuronales recurrentes ha demostrado ser un enfoque viable, y hasta prometedor bajo ciertas condiciones, como técnica de hashing para búsqueda por similitud.

Trabajo Futuro

Después de analizar los resultados obtenidos por el método propuesto, se identificaron diferentes aspectos en los cuales existe un margen de mejora.

En primer lugar, es necesario continuar profundizando en el impacto del *embedding* utilizado, evaluando la conveniencia de **incorporar el aprendizaje del *embedding* directamente en la arquitectura de la red**.

También es importante **extender el conjunto de *datasets* utilizados para evaluar el método** y corroborar si es que efectivamente es el largo de los documentos el factor determinante en el desempeño de la técnica, o si por el contrario, son otros los factores más relevantes como la presencia de *stop words* u otras técnicas de preprocesado, la cobertura del *embedding*, entre otros.

Asimismo, existen numerosas mejoras que se pueden implementar a nivel de la arquitectura de la red. Por ejemplo, sería conveniente **incorporar las etiquetas del *dataset* al proceso de entrenamiento de la variante supervisada**, modificando la red para que prediga la categoría de un documento al mismo tiempo que genera su *hash*. Aún mas relevante sería la **incorporación de módulos atencionales (*Attention Mechanisms*)** a la red, lo que permitiría mejorar el desempeño en documentos extensos al permitir a la red focalizar su atención en diferentes áreas del texto. También existen otros mecanismos que podrían mejorar el desempeño del modelo, como sería el **incorporar las *positional features*** utilizadas en TCH [73] o, siguiendo el enfoque seq2seq **modificar el modelo completo para convertirlo en un *Transfomer***, el cual constituye una de las arquitecturas de red más prometedoras en tareas de procesamiento de secuencias ligadas al lenguaje natural.

Capítulo A

Anexo

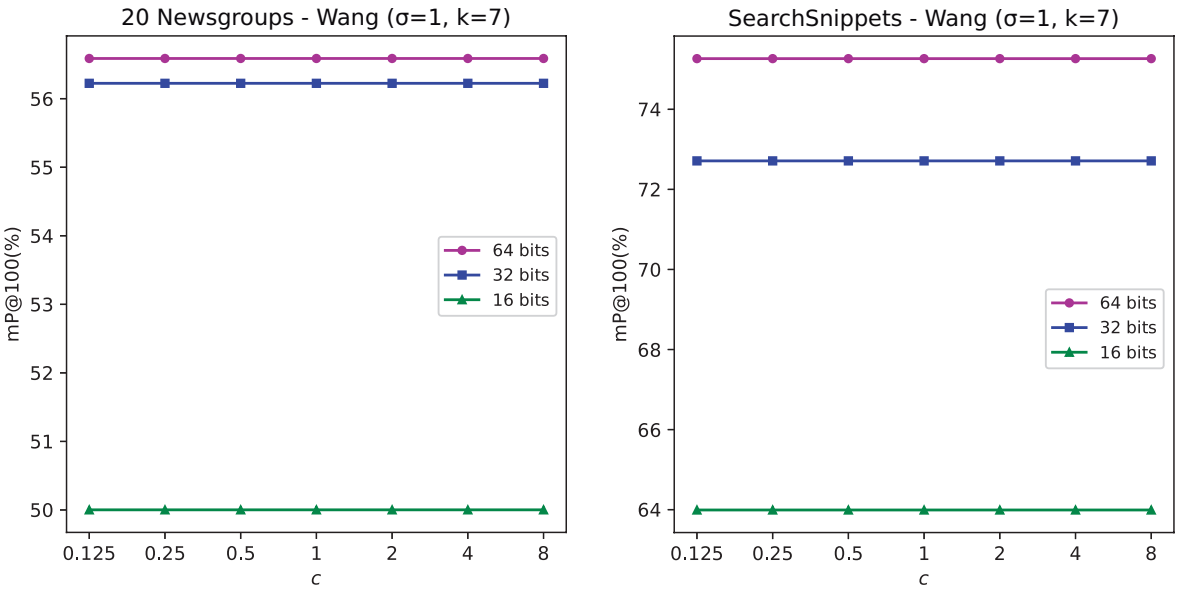


Figura A.1: Resultados de mP@100 en función del parámetro c_{ij} a 16, 32 y 64 bits. Resultados sobre 20 Newsgroups a la izquierda y SearchSnippets a la derecha.

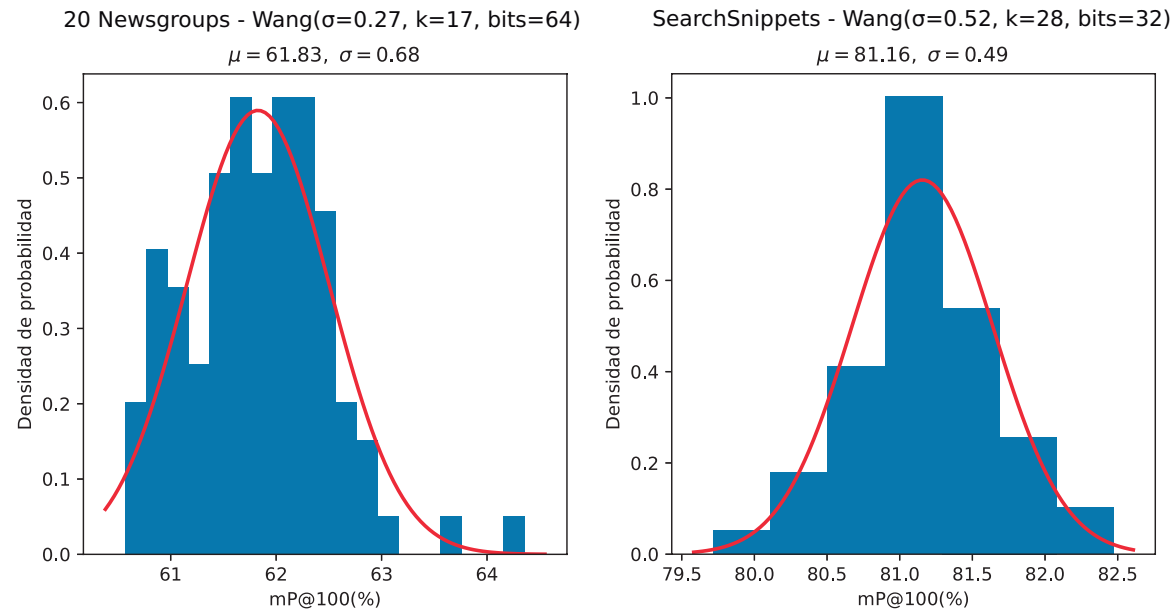


Figura A.2: Distribución de los resultados de mP@100(%) para Wang utilizando la mejor configuración de parámetros para ambos *datasets*. Tamaño de la muestra 100 particiones aleatorias al 20 %.

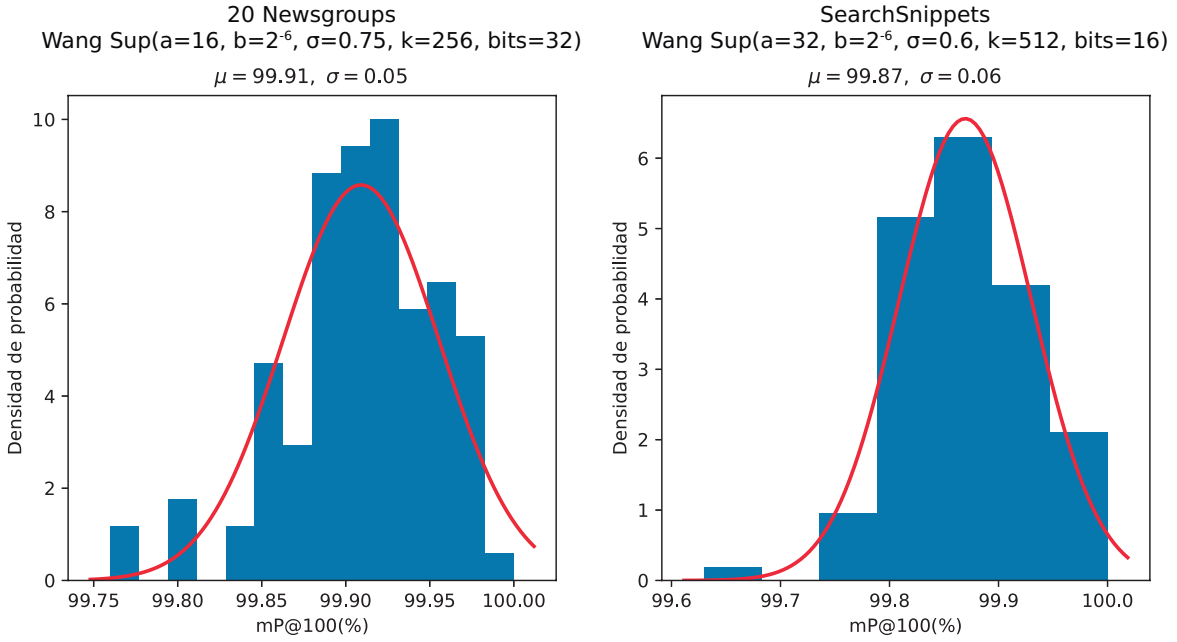


Figura A.3: Distribución de los resultados de mP@100(%) para Wang Supervisado utilizando la mejor configuración de parámetros para ambos *datasets*. Tamaño de la muestra 100 particiones aleatorias al 20 %.

Bibliografía

[1] A brief history of word embeddings (and some clarifications). https://www.gavagai.se/blog/2015/09/30/a-brief-history-of-word-embeddings/. Accessed: 25-07-2018.

[2] The digitization of the worldfrom edge to core. https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf. Accessed: 30-05-2019.

[3] File:neuron-no labels2.png. https://en.wikipedia.org/wiki/File:Neuron-no_labels2.png. Accessed: 29-07-2018.

[4] Generalización: Riesgos del sobreajuste. https://developers.google.com/machine-learning/crash-course/generalization/peril-of-overfitting?hl=es-419. Accessed: 07-07-2018.

[5] Glosario sobre aprendizaje automático. https://developers.google.com/machine-learning/crash-course/glossary. Accessed: 07-07-2018.

[6] Ibm big data success stories. https://public.dhe.ibm.com/software/data/sw-library/big-data/ibm-big-data-success.pdf. Accessed: 13-06-2019.

[7] An intuitive understanding of word embeddings: From count vectors to word2vec. https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/. Accessed: 28-07-2018.

[8] Machine learning algorithm != learning machine. https://www.ibm.com/developerworks/community/blogs/jfp/entry/Machine_Learning_Learning_Machine?lang=en. Accessed: 13-07-2018.

[9] Machine learning, what it is and why it matters. https://www.sas.com/en_us/insights/analytics/machine-learning.html. Accessed: 05-07-2018.

[10] Machine learning workflow. https://cloud.google.com/ml-engine/docs/tensorflow/ml-solutions-overview?hl=es-419. Accessed: 12-07-2018.

[11] Natural language processing, what it is and why it matters. https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html. Accessed: 23-07-2018.

[12] Overfitting and underfitting with machine learning algorithms. https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/. Accessed: 08-07-2018.

[13] Supervised vs. unsupervised machine learning. https://www.datascience.com/blog/supervised-and-unsupervised-machine-learning-algorithms. Accessed: 06-07-2018.

[14] Tfidftransformer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html#sklearn.feature_extraction.text.TfidfTransformer. Accessed: 20-04-2019.

[15] Underfitting and overfitting in machine learning. https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/. Accessed: 08-07-2018.

[16] Underfitting vs. overfitting. http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html. Accessed: 08-07-2018.

[17] What is machine learning? https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Is_Machine_Learning?lang=en_us. Accessed: 05-07-2018.

[18] Mohammad Reza Abbasifard, Bijan Ghahremani, and Hassan Naderi. A survey on nearest neighbor search methods. *International Journal of Computer Applications*, 95(25), 2014.

[19] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, Oct 2006.

[20] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[21] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[22] Jason Brownlee. Gentle introduction to statistical language modeling and neural language models. https://machinelearningmastery.com/

statistical-language-modeling-and-neural-language-models/. Accessed: 24-07-2018.

[23] Jason Brownlee. What are word embeddings for text? https://machinelearningmastery.com/what-are-word-embeddings/. Accessed: 08-07-2018.

[24] Ana Cardoso-Cachopo. Improving Methods for Single-label Text Categorization. PhD Thesis, Instituto Superior Tecnico, Universidade Tecnica de Lisboa, 2007.

[25] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*, chapter 1, pages 1–8. The MIT Press, 1st edition, 2010.

[26] D Manning Christopher, Raghavan Prabhakar, and Schutza Hinrich. *Introduction to information retrieval*, volume 151. 2008.

[27] D Manning Christopher, Raghavan Prabhakar, and Schutza Hinrich. *Introduction to information retrieval*, volume 151. 2008.

[28] James H. Martin Daniel Jurafsky. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Third Edition Draft.

[29] James H. Martin Daniel Jurafsky. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Third Edition Draft.

[30] James H. Martin Daniel Jurafsky. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Third Edition Draft.

[31] Pedro M Domingos. A few useful things to know about machine learning. *Commun. acm*, 55(10):78–87, 2012.

[32] Gérard Dreyfus. *Neural networks: methodology and applications*, chapter 1, pages 1–76. Springer Science & Business Media, 2005.

[33] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2475–2483, June 2015.

[34] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 35(12):2916–2929, 2013.

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 5, pages 96–162. MIT Press, 2016. http://www.deeplearningbook.org.

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 7, pages 236–237. MIT Press, 2016. http://www.deeplearningbook.org.

[37] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 6, pages 163–220. MIT Press, 2016. http://www.deeplearningbook.org.

[38] Yun Gu, Chao Ma, and Jie Yang. Supervised recurrent hashing for large scale video retrieval. In *Proceedings of the 24th ACM International Conference on Multimedia*, MM '16, pages 272–276, New York, NY, USA, 2016. ACM.

[39] Vikas Gupta. Understanding feedforward neural networks. https://www.learnopencv.com/understanding-feedforward-neural-networks/. Accessed: 21-07-2018.

[40] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[41] Mohamad H Hassoun et al. *Fundamentals of Artificial Neural Networks*, chapter 2, pages 49–50. A Bradford book. MIT Press, 1995.

[42] Geoffrey E Hinton et al. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA, 1986.

[43] Simeon Kostadinov. Understanding encoder-decoder sequence to sequence model. https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346. Accessed: 12-06-2019.

[44] David Kriesel. *A Brief Introduction to Neural Networks*, chapter 2, pages 13–31. 2007.

[45] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(6):1092–1104, June 2012.

[46] Yang Li and Tao Yang. *Word Embedding for Understanding Natural Language: A Survey*, chapter 4, pages 83–104. Springer International Publishing, Cham, 2018.

[47] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton Van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1971–1978, June 2014.

[48] Guosheng Lin, Chunhua Shen, David Suter, and Anton van den Hengel. A general two-step approach to learning-based hashing. *CoRR*, abs/1309.1853, 2013.

[49] Venice Erin Liong, Jiwen Lu, Yap-Peng Tan, and Jie Zhou. Deep video hashing. *IEEE Transactions on Multimedia*, 19(6):1209–1219, June 2017.

[50] Xiaoqiang Lu, Yaxiong Chen, and Xuelong Li. Hierarchical recurrent neural hashing for image retrieval with hierarchical convolutional features. *IEEE Transactions on Image Processing*, 27(1):106–120, Jan 2018.

[51] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[52] Jonathan Masci, Michael M. Bronstein, Alexander M. Bronstein, and Jürgen Schmidhuber. Multimodal similarity-preserving hashing. *CoRR*, abs/1207.1522, 2012.

[53] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[54] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[55] Michael Negnevitsky. *Artificial intelligence: a guide to intelligent systems*. Pearson Education, 2005.

[56] University of Toronto. Neural networks for machine learning. `https://www.coursera.org/learn/neural-networks/home/welcome`. Accessed: 19-07-2018.

[57] Galina Olejnik. Word embeddings: exploration, explanation, and exploitation (with code in python). `https://towardsdatascience.com/word-embeddings-exploration-explanation-and-exploitation-with-co/de-in-python-5dac99d5d795`. Accessed: 09-07-2018.

[58] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[59] Xuan-Hieu Phan, Le-Minh Nguyen, and Susumu Horiguchi. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 91–100, New York, NY, USA, 2008. ACM.

[60] Nihar Ranjan, Kaushal Mundada, Kunal Phaltane, and Saim Ahmad. A survey on techniques in nlp.

[61] Jason Rennie's. 20 newsgroups. `http://qwone.com/~jason/20Newsgroups/`. Accessed: 03-04-2019.

[62] Stefano Lodi Claudio Sartori Ricardo Ñanculef, Carlos Valle. Learning complementary hash tables by boosting a consensus hamming metric. unpublished, N.D.

[63] Sebastian Ruder. An overview of word embeddings and their connection to distributional semantic models. http://blog.aylien.com/overview-word-embeddings-history-word2vec-cbow-glove/#reference6. Accessed: 29-07-2018.

[64] Sebastian Ruder. An overview of word embeddings and their connection to distributional semantic models. http://blog.aylien.com/overview-word-embeddings-history-word2vec-cbow-glove/#reference6. Accessed: 05-08-2018.

[65] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969 – 978, 2009. Special Section on Graphical Models and Information Retrieval.

[66] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.

[67] Manjeet Singh. Word embedding. https://medium.com/data-science-group-iitr/word-embedding-2d05d270b285. Accessed: 26-07-2018.

[68] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[69] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014.

[70] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2393–2406, Dec 2012.

[71] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in neural information processing systems*, pages 1753–1760, 2009.

[72] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1):37 – 52, 1987. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

[73] Jiaming Xu, Peng Wang, Guanhua Tian, Bo Xu, Jun Zhao, Fangyuan Wang, and Hongwei Hao. Convolutional neural networks for text hashing. In *IJCAI*, pages 1369–1375, 2015.

[74] Dell Zhang, Deng Cai, and et al. Extensions to self-taught hashing: Kernelisation and supervision, 2010.

[75] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. Self-taught hashing for fast similarity search. *CoRR*, abs/1004.5370, 2010.