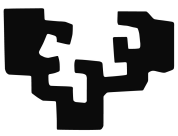
eman ta zabal zazu



Universidad del País Vasco
Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Computación

Proyecto de Fin de Grado

Estudio de Word Embeddings y métodos de generación de Meta Embeddings

---

Autor

Iker García Ferrero

Director

German Rigau i Claramunt

2018

Resumen

Este documento describe el trabajo de final de grado (TFG) del estudiante Iker Garcia. El TFG se enmarca en el área del Procesamiento del Lenguaje Natural (abreviado como PLN, o más comúnmente como NLP por su nombre en inglés, Natural Language Processing). El procesamiento del lenguaje natural es un campo de investigación dentro de las ciencias de la computación, inteligencia artificial y lingüística, que estudia cómo modelar computacionalmente el lenguaje humano. El proyecto ha sido dirigido por German Rigau i Claramunt.

Los word embeddings son representaciones vectoriales de palabras que resultan útiles para una gran variedad de tareas relacionadas con el procesamiento del lenguaje natural (PLN). En este documento realizaremos una evaluación de diferentes métodos de generación de word embeddings y word embeddings pre-entrenados. Aunque el tema que recibirá mayor atención sera el estudio y evaluación de métodos que permitan combinar el conocimiento de diferentes word embeddings para generar representaciones de palabras de mejor calidad, llamaremos a estos nuevos word embeddings, meta embeddings.

Índice general

Índice de figuras

Indice de tablas

1. CAPÍTULO

Introducción

1.1. Contexto

Este trabajo de final de grado (TFG) se enmarca en el área del Procesamiento del Lenguaje Natural (abreviado como PLN, o más comúnmente como NLP por su nombre en inglés, Natural Language Processing). El procesamiento del lenguaje natural es un campo de investigación dentro de las ciencias de la computación, inteligencia artificial y lingüística, que estudia cómo modelar computacionalmente el lenguaje humano. El objetivo de el PLN es que los ordenadores sean capaces de entender, interpretar y manipular el lenguaje humano. Entre los principales desafíos del procesamiento del lenguaje natural se encuentra el reconocimiento del habla, la comprensión del lenguaje natural y la generación del lenguaje. Dentro de estos grandes desafíos se encuentran tareas como la traducción automática o la extracción de información de textos.

La mayoría de sistemas de reglas y sistemas estadísticos de PLN tradicionales, representan las palabras como unidades discretas y atómicas. Este tipo de codificación no aporta al sistema ningún tipo de información sobre como las diferentes palabras pueden estar relacionadas entre ellas. El sistema cuando procese información de animales no podrá saber que un humano se trata de un animal, que a su vez es un mamífero. Representar las palabras como variables únicas y discretas no nos aporta la suficiente información para ser capaces de entrenar modelos capaces de realizar tareas de procesamiento de lenguaje complejas de forma satisfactoria.

Otro enfoque común para tratar de representar el significado de las palabras, es el uso de

taxonomías o redes semánticas como WordNet [39]. WordNet almacena relaciones de hiperonimia del tipo (gato es un animal) y conjuntos de sinónimos. Sin embargo este tipo de estructuras carecen de matices. Por ejemplo, en WordNet si buscamos los sinónimos de la palabra .ªpto", encontramos las palabras .experto", "bueno", .experimentado", çompetentez "habilidoso". Sin embargo, las frases "soy bueno en NLPz "soy experto en NLP"no tienen un significado idéntico. Otro problema importante es que no tenemos forma de medir que por ejemplo, "bueno"puede ser más similar a "habilidoso"que a .experto".

Las palabras pueden detonar diferentes significados en diferentes contextos, esto es conocido como sentidos. Es decir, las palabras pueden ser interpretadas de forma diferente dependiendo del contexto particular en el que se utilizan. La palabra "partido"no significa lo mismo en la frase .El Athletic juega su **partido** mañana"que en la frase .El **partido** decidió votar en contra de la propuesta de ley". La semántica distribucional estudia métodos para cuantificar y categorizar la similitud semántica entre palabras basándose en sus propiedades distribucionales en grandes conjuntos de datos. Por ejemplo, si queremos saber el significado de la palabra "banco", buscaremos cientos de frases donde aparezca, y veremos que generalmente aparecerá junto a palabras como "dinero", "gobierno", "deuda", “regulación", “crisis” ... y trataremos de expresar el significado de la palabra “banco” en función de todas estas palabras. Esto puede ser resumido en la llamada hipótesis de distribución [47] planteada por Zelling Harris. La idea básica de la semántica distribucional se puede resumir en la llamada hipótesis distribucional: «elementos lingüísticos con distribuciones similares tienen significados similares». Es decir, las palabras con distribuciones similares están cerca unas de otras semánticamente. Por ejemplo, si descubrimos que dos palabras W_1 y W_2 tienden a tener propiedades distribucionales similares, como que ambas aparecen generalmente junto a otra entidad W_3 en diferentes textos, entonces podríamos postular que W_1 y W_2 pertenecen a la misma clase lingüística y tienen un significado similar. La idea subyacente de que «una palabra se define por las palabras que la acompañan» fue popularizada por J.R. Firth [40], y es una de las ideas más exitosas en el PLN moderno.

Basándose en la hipótesis de distribución, se han explorado diferentes formas de representar palabras. Uno de los primeros métodos empleados, es representar las palabras con un vector, en el que cada dimensión es una medida de asociación entre palabras y un tipo de información particular, como pueden ser los documentos en los que aparece, las palabras junto a las que aparece... [67]. Esta matriz suele tener un gran número de dimensiones pero con pocos elementos no nulos. Por ello, normalmente luego es reducida usando técnicas de reducción de dimensionalidad como la descomposición en valores sin-

gulares [26] [50]. Estos métodos se conocen como modelos basados en conteo. Pero el verdadero avance en este tipo de técnicas llega con los métodos basados en redes neuronales [23] [72] [29], en especial con el trabajo de Tomas Mikolov [79] (2013). Estos métodos se conocen como modelos predictivos. Los modelos predictivos intentan predecir directamente una palabra a partir de sus vecinos en términos de vectores pequeños y densos que se aprenden durante el entrenamiento. La idea detrás de estos métodos es que si podemos predecir en qué contexto aparece una palabra, entonces significa que entendemos el significado de la palabra en su contexto. Así, las palabras se representan en espacios vectoriales donde palabras semánticamente similares se encontrarán cerca entre ellas. Este tipo de representación recibe el nombre en inglés de "word embeddings".

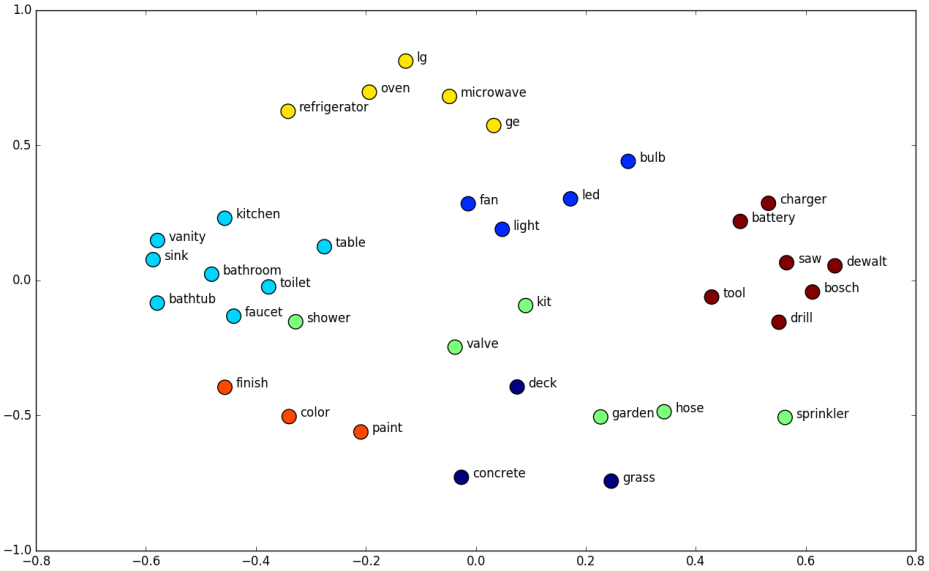


Figura 1.1: Representación en 2D de algunos word embeddings de palabras

La figura 1.1 muestra en un espacio 2D y agrupadas por colores los word embeddings obtenidos para distintas palabras.

Los "word embeddings"han obtenido muy buenos resultados en tareas como el cálculo de la similitud entre palabras o analogías. Gracias a ello se han convertido en la base de sistemas de traducción automática neuronal [16], clasificación de textos [88], etc. Es por esta razón por la que nos interesa ser capaces de obtener los mejores “word embeddings” posibles. Existe una gran variedad de métodos para generar word embeddings y éstos métodos pueden aplicarse a distintos tipos de textos e idiomas. Así, cada conjunto de word embeddings puede contener un conocimiento distinto otros word embeddings. Recien-

temente varios estudios [53] [84] [42] [37] han tratado de combinar el conocimiento de diferentes word embeddings o incluso combinar el conocimiento de word embeddings y bases de conocimiento para generar un nuevo conjunto de embeddings. El nuevo word embedding generado se espera que contenga el conocimiento de todos los word embeddings que se han usado durante su generación, consiguiendo así mejores representaciones de palabras y obtener gracias a ello un mejor rendimiento en todo tipo de tareas. En este aspecto destaca Numberbatch [69]. Numberbatch es un meta embedding que combina la información de diferentes word embeddings y la base de conocimiento ConceptNet [3]. Numberbatch ha superado a otros word embeddings en todo tipo de tareas (ver figura 1.2), mostrando que los meta embeddings cuentan con un gran potencial.

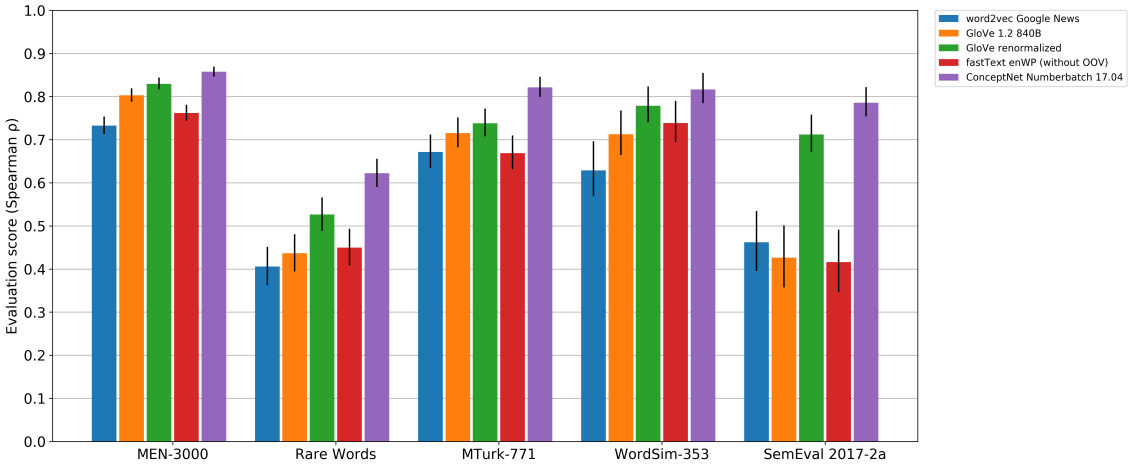


Figura 1.2: Numberbatch comparado con Word2Vec, FasText y Glove, tres de los métodos de generación de word embeddings más populares en diferentes tareas.

Desde que el trabajo de Mikolov los word embeddings han contado con una gran popularidad. Se han publicado una gran cantidad de técnicas de generación de meta embeddings, y junto a ellas word embeddings pre-entrenados que podemos descargar y utilizar. Así, las comparativas del rendimiento de distintos word embeddings han quedado obsoletas rápidamente., Actualmente no disponemos de ningún análisis exhaustivo en el que se haya comprobado el rendimiento de los diferentes word embeddings existentes, ni la mejor forma de combinarlos para crear meta-embeddings. Los estudios publicados no han estudiado los efectos que provocan los diferentes métodos de generación de meta-embeddings sobre un amplio número de word embeddings. Tampoco se ha estudiado en profundidad las diferentes formas de aplicar los métodos de generación de meta-embeddings y las posibles normalizaciones que podemos aplicar para intentar obtener el mejor rendimiento posible. En este trabajo trataremos de dar respuesta a ambas preguntas.

La rotación de word embeddings y alineación de espacios vectoriales es otro de los temas que han recibido una gran atención recientemente. Esto es debido a que los sistemas de traducción automática modernos generalmente usan una red neuronal que es entrenada usando corpus paralelos. Es decir, documentos, frases o palabras idénticos escritos en diferentes idiomas. Estos documentos se obtienen por ejemplo de subtítulos de películas, o de los documentos publicados por el Parlamento Europeo que son traducidos manualmente a todos los idiomas de los países de la Unión Europea. Si bien estos modelos han conseguido obtener buenos resultados, obtener una gran cantidad de corpus paralelos entre todos los pares de idiomas no siempre es una tarea fácil. Es por ello que la alineación de word embeddings ha recibido gran atención. Alinear los espacios vectoriales de word embeddings en diferentes idiomas permite la implementación de sistemas de traducción automática que no cuentan con la necesidad de corpus paralelos para su entrenamiento. Un ejemplo de estos nuevos sistemas son "UNdreaMT" [17] o "UnsupervisedMT" [30]. La alineación de word embeddings nos permite que dados por ejemplo un word embedding entrenado en textos en castellano y otro entrenado en corpus en inglés, para cada palabra del embedding en castellano, la palabra más cercana en el embedding en inglés sea la palabra que cuenta con el mismo significado. Las rotaciones o mapeados de word embeddings han obtenido buenos resultados en el ámbito de la traducción automática, sin embargo no han sido empleados en el ámbito de la generación de meta-embeddings. Creemos que el mapeado de word embeddings puede ayudar a mejorar los métodos actuales de generación de meta-embeddings, y por ello propondremos un nuevo método de generación de meta-embedding que hace uso de estas técnicas.

1.2. Objetivos

El objetivo general de este proyecto es el de evaluar diferentes word embeddings y diferentes técnicas para tratar de mejorar el rendimiento de los word embeddings originales. Para ello hemos escogido una serie de word embedding pre-entrenados mediante diferentes métodos y corpus. Más específicamente buscamos realizar una serie de aportaciones en el ámbito de la evaluación de word embeddings y generación de meta-embeddings.

Por ello, en primer lugar nos planteamos como subobjetivo una exhaustiva revisión bibliográfica sobre el tema de los word embeddings y meta-embeddings. También estudiaremos distintas técnicas de optimización y normalización de embeddings, así como distintas técnicas de combinación de word embeddings. Todas estas técnicas y los word embeddings

resultantes los evaluaremos enpíricamente en un mismo marco experimental controlado usando las mismas métricas.

También estudiaremos un nuevo método de generación de meta-embeddings que hace uso de rotaciones de word embeddings a un mismo espacio vectorial.

Finalmente aplicaremos todo lo aprendido para tratar de generar nuevos y mejores meta-embeddings.

Dentro del los objetivos del proyecto también se incluye el desarrollo de un software que permita evaluar word embeddings y generar meta embeddings. Para el desarrollo de este software se utilizará el lenguaje python (versión 3.x) y entorno de desarrollo Jupyter Notebook. Dicho software estará disponible para su uso en la plataforma GitHub [1] bajo la licencia MIT.

1.3. Estructura de la memoria

El proyecto se divide en diez capítulos. El primero es esta breve introducción. El resto de capítulos se estructuran de la siguiente forma:

- (Capítulo 2) Repasa el estado del arte actual sobre embeddings y meta-embeddings. En él revisaremos las diferentes técnicas de generación de word embeddings. Diferentes aplicaciones actuales de los word embeddings. Las diferentes formas de evaluar el rendimiento de un word embedding. Y las diferentes formas de generar meta-embeddings.

- (Capítulo 3) Estudio exhaustivo del rendimiento de diferentes word embeddings con el objetivo de determinar que word embedding ofrece un mejor rendimiento.

- (Capítulo 4 Estudio de diferentes métodos de normalización de word embeddings. Con ello buscamos poder determinar si es necesario o recomendable aplicar diferentes tipos de normalización a los word embeddings para obtener el mejor rendimiento posible.

- (Capítulo 5). Estudio de diferentes formas simples de combinación de word embeddings. Estas combinaciones no generan nuevos word embeddings, si no que nos

[1] https://github.com/ikergarcia1996/RotEmbeddings

permiten combinar el conocimiento de diferentes word embeddings de forma sencilla para tratar de obtener un mejor rendimiento. Buscamos evaluar el rendimiento de las combinaciones. Además queremos conocer qué tipo de word embeddings producen mejores resultados al combinarlos.

- (Capítulo 6). Presenta diferentes métodos de generación de meta-embeddings. Queremos determinar qué método de generación de meta-embeddings obtiene un mejor rendimiento. Dentro de cada método queremos determinar cuál es la forma de aplicar el método más efectiva, lo que engloba evaluar que tipo de word embeddings tienen a generar mejores meta-embeddings al combinarlos y que tipos de normalizaciones o preprocesamientos son más adecuados para método de generación de meta embeddings.

- (Capítulo 7). Desarrollar y evaluar un nuevo método de generación de meta-embeddings que utiliza el mapeado de word embeddings a un mismo espacio vectorial. Los mapeados de word embeddings a un espacio vectorial no han sido aplicados todavía en el ámbito de la generación de meta-embeddings. Nuestro objetivo es que gracias a este mapeado el método de generación presentado obtenga mejores resultados que los métodos de generación de word embeddings actuales.

- (Capítulo 9) Por último presentaremos las conclusiones del proyecto donde discutiremos las aportaciones realizadas, y el las líneas de investigación futuras que pueden surgir a partir de este proyecto.

Al final de la memoria, se incluye un anexo donde mostraremos algunas tablas con resultados que consideremos relevantes que no han sido mostradas en su totalidad en los diferentes capítulos. Estas tablas muestran una gran cantidad de resultados y son demasiado grandes para mostrarlas en su totalidad en los diferentes capítulos.

2. CAPÍTULO

Estado-del-arte

Este capítulo nos centraremos en el marco teórico del proyecto.

2.1. Word Embeddings

Los word embedding son un enfoque de las semántica de distribución que representa palabras como vectores de número reales. Dicha representación tiene propiedades de de agrupamiento útiles, ya que agrupa palabras que son semánticamente y sintácticamente similares. Por ejemplo esperamos que las palabras “delfín y foca” se encuentren cerca, pero “Paris” y “delfín” no se encuentren cerca ya que no existe una fuerte relación entre ellas. Por lo tanto, las palabras se representan como vectores de valores reales, donde cada valor captura una dimensión del significado la palabra. Esto provoca que palabras semánticamente similares, tengan vectores similares. De forma simplificada, cada dimensión de los vectores representa un significado y el valor numérico en cada dimensión captura la cercanía de la asociación de la palabra a dicho significado. En la Tabla 2.1 se muestra una simplificación de varios vectores de palabras. Su objetivo es cuantificar y categorizar similitudes semánticas entre elementos lingüísticos. Este tipo de representación es densa. Es de esperar sinónimos y palabras intercambiables se encuentren cerca en ese espacio

Los modelos de espacio vectorial han sido usados en semántica distribucional desde la década de los 90. Desde entonces, se han desarrollado diferentes modelos para estimar representaciones continuas de palabras, un ejemplo es el análisis semántico latente (LSA por sus siglas en inglés).

	Rey	Reina	Mujer	Princesa	
Realeza	0.99	0.99	0.02	0.98	...
Masculinidad	0.99	0.05	0.01	0.02	...
Feminidad	0.05	0.93	0.99	0.94	...
Edad	0.7	0.6	0.5	0.1	...
...	...	...	...	...	...

Tabla 2.1: Ejemplo simplificado de vectores de palabras.

El término word embedding fue originalmente concebido por Bengio et al [23] en 2003. Quien entrenó este tipo de vectores en un modelo probabilístico neuronal. Sin embargo, Collobert y Weston fueron posiblemente los primeros en demostrar el poder de los word embeddings en su documento "A unified architecture for natural language processing" [28] publicado en 2008, en el que establecen los word embeddings como una herramienta altamente efectiva en diferentes tipos de tareas, además presentan una arquitectura de red neuronal en la que se basan muchos de los enfoques actuales.

2.1.1. Influencia y Aplicaciones

Los word embeddings fueron popularizados en gran parte gracias a el trabajo de Mikolov et al. [79] quienes en 2013 publicaron word2vec, una herramienta para entrenar y usar word embeddings. Un año después, Pennington et al. [50] introdujeron Glove, una nueva herramienta para la generación de word embeddings. A partir de este momento los word embeddings se han convertido en una de las corrientes principales dentro de el procesamiento del lenguaje natural. Los word embeddings capturan el significado de las palabras, y las traducen a una codificación que puede ser usada como entrada para todo tipo de redes neuronales. Esto ha provocado que su uso se haya extendido rápidamente y actualmente sean una pieza fundamental en la arquitectura de todo tipo de modelos que realizan tareas de PLN. Algunos de las aplicaciones más relevantes son:

- Sistemas de traducción. Generalmente estos sistemas están formados por por una red neuronal que actúa como codificador y una red neuronal que actúa como decodificador. Tanto la entrada como la salida de estas redes neuronales son secuencias de palabras, estas palabras se representan mediante word embeddings. Uno de los ejemplos más famosos de este tipos de sistemas es el traductor de google, este traductor hace uso del modelo seq2seq [25]. Estos sistemas hacen uso de corpus paralelos, es decir, textos idénticos es diferentes idiomas. Sin embargo actualmente el desarrollo de modelos que no necesitan hacen uso de corpus paralelos está

recibiendo una gran atención. Algunos ejemplos de este tipo de modelos son UnsupervisedMT [56] o UNDreaMT[17].

- Análisis de opinión en textos. Con el crecimiento de popularidad de las redes sociales, resulta muy interesante el desarrollo de un sistema capaz de por ejemplo, analizar si las opiniones de sobre un producto son positivas o negativas. Algunos sistemas modernos hacen uso de redes neuronales convolucionales donde la entrada son secuencias de palabras representadas como word embeddings. Un ejemplo de este tipo de sistemas es CharSCNN [33]

- Generación de textos: Mediante el uso de redes neuronales recurrentes es posible generar texto de forma automática. Combinando estos modelos con redes convolucionales es incluso posible crear sistemas que anoten o describan imágenes.

- Chatbox, o sistemas que respondan preguntas de usuarios. Estos sistemas están ganando cada vez más popularidad. Algunos ejemplos son el asistente personal de google que podemos encontrar en una gran cantidad de teléfonos inteligentes o amazon alexa. El modelo seq2seq [25] además de para realizar traducciones de textos, también puede actuar como un chatbox si se realizan pequeños ajustes y es entrenado para ello.

Esto son solo algunas de las aplicaciones de los word embeddings. Puesto que son capaces de codificar el significado de las palabras y las relaciones entre ellas, es posible aplicarlos a todo tipo de tareas.

2.2. Métodos de generación de meta embeddings

Desde que gracias a word2vec [79] en 2013 los word embeddings comenzasen a popularizarse, han surgido una gran cantidad de métodos diferentes para generar word embeddings. En esta sección se expondrán los word embeddings que posteriormente usaremos en las comparativas. Hemos escogido para su análisis los word embeddings más conocidos, además de algunos que nos han resultado interesantes, ya sea por su rendimiento o por ser muy diferentes al resto de métodos de generación de word embeddings. Estos métodos de generación de word embeddings vienen normalmente acompañados por vectores precalculados que serán los que usaremos para la evaluación y también describiremos durante la sección.

- **Word2Vec** [79]: Se trata de un modelo predictivo de generación de word embeddings. Word2Vec implementa dos modelos neuronales: CBOW y Skip-gram. En el primero, dado el contexto de la palabra objetivo, intenta predecirla. En el segundo, dada la palabra intenta predecir el contexto. Las capas internas de la red neuronal codifican la representación de la palabra objetivo, es decir, los word embeddings. En este estudio utilizaremos los vectores entrenados en el dataset Google News [4] con alrededor de 100 mil millones de palabras. El modelo contiene vectores de 300 dimensiones para 3 millones de palabras y frases. Los vectores usados están disponibles en la siguiente dirección: https://code.google.com/archive/p/word2vec/.

- **FastText** [61]: FastText es una extensión del modelo Word2Vec. Cada palabra es tratada como la suma de sus composiciones de caracteres llamados ngrams. El vector para una palabra está compuesto por la suma de sus ngrams. Por ejemplo el vector para la palabra "apple" está compuesto por la suma los vectores para los ngrams "<ap, app, appl, apple, apple>, ppl, pple, pple>, ple, ple>, le>". De esta forma se espera obtener mejores representaciones para palabras "raras", las cuales cuentan con muy pocas apariciones en corpus de textos, y así poder generar vectores para palabras que no se encuentran en el vocabulario de los word embeddings. Hemos utilizado diferentes word embeddings calculados mediante este método. FastText calculados en el corpus common crawl [2] usando 600 mil millones de tokens. Cuentan con representaciones de 300 dimensiones para 2 millones de palabras, nos referiremos a ellos como "FastText CC". FastText calculados usando los corpus wikipedia [10], UMBC [60] y statmt.org [7]. Cuentan con representaciones de 300 dimensiones para 1 millón de palabras. Han sido entrenados usando 16 mil millones de tokens. Nos referiremos a ellos como "FastText wiki" Los vectores usados están disponibles en la siguiente dirección: https://fasttext.cc/docs/en/english-vectors.html.

- **GLOVE**: [50]: GLOVE a diferencia de Word2Vec es un modelo basado en conteo. GloVe genera una matriz de gran tamaño donde se almacena la información de la concurrencia entre palabras y contextos.. Es decir, para cada palabra contamos cuantas veces aparece dicha palabra en algún contexto. El objetivo de entrenamiento de dicha matriz es aprender vectores de forma que el producto escalar entre las palabras sea igual al logaritmo de la probabilidad de co-ocurrencia entre las palabras. El número de contextos es muy alto, por lo tanto se realiza una factorización de dicha matriz para obtener una de menores dimensiones. Obteniendo así un vector que re-

presenta a cada una de las palabras. La ventaja de GLOVE sobre Word2Vec es que es más sencillo paralelizar el entrenamiento, por lo tanto es posible usar más información durante el entrenamiento. Por lo tanto es posible usar una mayor cantidad de datos durante el entrenamiento. Hemos usado varios word embedding entrenados mediante GloVe. Existen dos versiones de GloVe entrenadas en el corpus common crawl [2]. Ambas cuentan con representaciones de 300 dimensiones. En una se han usado 840 mil millones de tokens y cuenta con representación para 2,2 millones de palabras, la cual llamaremos "Glove CC 840B". En la otra se han usado 42 mil millones de tokens y cuenta con representación para 1,9 millones de palabras, la cual llamaremos "Glove CC 42B". También se ha usado GLOVE para generar word embeddings usando los corpus Wikipedia (2014) [10] y Gigaword5 [12], con 6 mil millones de tokens y cuentan con representaciones para 400 mil palabras. Además están disponibles en diferentes versiones según las dimensiones con las que cuenten, existe versiones con 50, 100, 200 y 300 dimensiones, nos referiremos a ellos como "Glove Wiki. dims 50/100/200/300". Por último también se ha usado un corpus formado por 2 millones de tweets extraídos de la red social Twitter [8]. Se han usado 27 mil millones de tokens y cuentan con un vocabulario de 1,2 millones de palabras. Están disponibles en 25, 50, 100 y 200 dimensiones. Nos referiremos a ellos como "Glove twitter. Dims: 25/50/100/200". Los vectores usados están disponibles en la siguiente dirección: https://nlp.stanford.edu/projects/glove/.

- **LEXVEC**: [14]: Es un modelo que busca obtener mejores resultados gracias a la combinación de Glove y Word2Vec. Existen diferentes versiones, ya que el modelo ha sido mejorado con el paso del tiempo. La versión base ha sido entrenada en el corpus common crawl con 58 mil millones de tokens, y cuenta con representaciones de 300 dimensiones para 2 millones de palabras. Nos referiremos a ella como "Lexvec CC Word Vectors". También ha sido entrenado en un corpus formado por el corpus de wikipedia (2015) [10] y NewsCrawl [7]. Se han usado 7 mil millones de tokens y cuenta con representaciones de 300 dimensiones para 369 mil palabras. Nos referiremos a estos últimos como "Lexvec Wiki+NewsCrawls Wordvectors". De ambos word embeddings existe una versión [13] que utiliza vectores de contexto[58]. Los vectores de contexto buscan mejorar el rendimiento de los word embeddings en tareas de analogía. Normalmente en los modelos solo tenemos en cuenta las palabras que se encuentran en el contexto de la palabra objetivo, por ejemplo en la frase "El gran **perro** ladró fuerte" si la palabra objetivo es perro, el contexto estaría formado por "(El, gran, ladró, fuerte)". Los vectores de contexto

tienen además en cuenta la posición relativa que ocupan las palabras alrededor de la palabra objetivo, Por ejemplo en la frase anterior tendrá un contexto (El^{-2}, gran^{-1}, ladró^{1}, fuerte^{2}). Por lo tanto, sabemos que la palabra fuerte^{2} se encuentra dos posiciones a la derecha de la palabra objetivo. Dicha información esperamos que nos ayude a obtener mejores representaciones de palabras. Nos referiremos a estos word embeddings como "Lexvec Wiki+NewsCrawls Wordvectors + ContexVectors" y "Lexvec CC Word Vectors + ContexVectors". La última y más reciente versión [15], incorpora al igual que FastText los ngrams. Esta versión está calculada en el corpus common crawl [2] usando 58 mil millones de tokens. Cuanta con representaciones de 300 dimensiones para 2 millones de palabras. Los vectores usados están disponibles en la siguiente dirección: https://github.com/alexandres/lexvec.

- **PDC y HDC**: [77]. PDC y HDC se tratan de extensiones del modelo Cbow y Skip-gram respectivamente. Buscan capturar relaciones sintagmáticas y paradigmáticas al mismo tiempo durante el entrenamiento. PDC es un modelo donde una palabra objetivo es predicha a partir de su contexto circundante, además del documento en el que aparece. La predicción de la palabra usando su contexto captura la relación paradigmática de las palabras, ya que palabras en contextos similares tenderán a tener representaciones similares. Este modelo también provoca que palabras que tienden a aparecer en un mismo documento tiendan a tener representaciones similares, capturando así las relaciones sintagmáticas. HDC es similar a PDC, pero en este caso se aplica a el modelo Skip-gram. En este caso el documento se utiliza para predecir una palabra objetivo, y a partir de dicha palabra se predice su contexto. Ambos modelos se han entrenado en el corpus de Wikipedia (2010) [10]. Están disponibles en 50, 100 y 300 dimensiones, nos referiremos a ellas como "PDC / HDC Dims: 50/100/300". Los vectores usados están disponibles en la siguiente dirección: http://ofey.me/projects/wordrep/.

- **Context2Vec**: [64]. Context2Vec es una extensión del modelo CBOW de Word2Vec. La diferencia principal entre ambos modelos es que CBOW representa el contexto alrededor de una palabra como la media de los embeddings de las palabras circundantes. Context2Vec propone un enfoque más complejo. El contexto a la izquierda

[0]Conjunto de palabras agrupadas en torno a un núcleo con una misma función sintáctica y de sentido. Ejemplo: Amigo – SN Mi amigo - SN Mi amigo bueno – SN Mi amigo bueno de mi colegio

[0]Un paradigma es una serie de elementos que pueden ocupar una misma situación, teniendo en cuenta que mutuamente pueden sustituirse y que el empleo de uno de ellos excluye el uso de todos los demás del paradigma. Ejemplo: al utilizar la palabra notable, se excluye sobresaliente, aprobado y suspenso, pues los cuatro términos pueden ocupar esa posición

de la palabra objetivo y el contexto a la derecha de la palabra objetivo son introducidos en modelos neuronales independientes (red neuronal recurrente LSTM). Los resultados son luego combinados por una nueva red neuronal (perceptrón multicapa). De esta forma se espera conseguir extraer la información más relevante del contexto de la palabra objetivo. Otra diferencia fundamental es que mientras Cbow toma un número determinado de palabras alrededor de la palabra objetivo, por ejemplo, se toman 2 palabras por la izquierda y dos por la derecha, Context2Vec es capaz de utilizar la frase completa donde se encuentra la palabra objetivo. Existen dos versiones, una calculada en el corpus ukWaC [9], corpus extraído del dominio ".uk´´ a la que nos referiremos como "Context2Vec ukwac" y calculada en el dataset de la competición Microsoft Sentence Completion Challenge [11], nos referiremos a ella como "Context2Vec mscc". Los vectores usados están disponibles en la siguiente dirección: http://u.cs.biu.ac.il/~nlp/resources/downloads/context2vec/.

- **SketchEngine** [6]: Se trata de word embeddings calculados mediante FastText usando el modelo SkipGram y cuentan con 100 dimensiones. Resultan interesantes por que cuentan que con un número de dimensiones menor que los embeddings calculados mediante FastText descritos anteriormente. Tenemos dos versiones, una entrenada a partir de corpus extraído de la web, usando 20 mil millones de tokens, a los que nos referiremos como "SketchEngine Web´´ y una segunda versión calculada a partir del British National Corpus [1] usando 100 millones de tokens, a la que nos referiremos como "SketchEngine "British National". Los vectores usados están disponibles en la siguiente dirección: https://embeddings.sketchengine.co.uk/static/index.html.

- **UKB**: [43].Tambien llamado RWSGwn, sin embargo desde el incio del proyecto no referimos a él como UKB y es el nombre con el que aparecerá en las tablas. Se trata de un modelo que combina random walks sobre WordNet [39] con el modelo skip-gram. Los Random Walk sobre WorNet. Los random walks sobre WordNet generan sentencias del tipo: "amphora wine nebuchadnezzar bear retain long". La frase comienza por amphora, un recipiente que generalmente se llena de vino. Vino es la segunda palabra. Nebuchadnezzar es un tamaño de botella particular. Y las palabras finales están relacionadas con el almacenamiento. Estas sentencias son introducidas al modelo skip-gram para generar word embeddings. Los word embeddings usados cuentan con 300 dimensiones, y nos referiremos a ellos como "UKB´´. Los vectores usados están disponibles en la síguete dirección: http://ixa2.si.ehu.es/ukb/.

- **jointcHYB** [44]: Se trata un método para generar word embedding bilingües, es decir, que incluyen palabras de dos idiomas diferentes. El método extrae información bilingüe de WorNets plurilingües [39] usando Ramdom Walks, y genera un embedding donde ambos idiomas se encuentran en el mismo espacio vectorial usando dicha información además de corpus en ambos idiomas mediante el modelo skip-gram, por lo que se trata de un modelo híbrido. Además se incrementa la equivalencia de las palabras en ambos idiomas mediante una variación del modelo Skip-gram que fuerza a que términos equivalentes en diferentes idiomas se acerquen durante el entrenamiento. Existen diferentes versiones de este modelo, hemos usado la más reciente, llamada ”JOINTChyb´´. Hemos usados los word embedding entrenados en castellano e inglés, nos referiremos a él como jointcHYB EN_ES. Los word embeddings entrenados en italiano e inglés, nos referiremos a ellos como jointcHYB EN_IT. Y los word embeddings entrenados en euskera e inglés, nos referiremos a ellos como jointcHYB EN_EU. Todos ellos cuentan con representaciones de palabras de 300 dimensiones. Los word embeddings usados están disponibles en la siguiente dirección: `http://ixa2.si.ehu.es/ukb/bilingual_embeddings.html`.

- **Numberbatch** [76]: Se trata de un meta embedding, y actualmente su rendimiento es considerado el estado del arte. Numberbatch hace uso de embeddings generados mediante Word2Vec y Glove. Ambos word embeddings son mejorados usando la información de la base de conocimiento ConceptNet [3], esto se realiza mediante una técnica llamada retrofitting que explicaremos y evaluaremos en el Capítulo 6. Después son concatenados y se aplica el método de reducción de dimensionalidad de descomposición en valores singulares. En versiones recientes Numberbatch también puede generar word embeddings plurilingües, para eso se añade también a la combinación un word embedding plurilingüe generado a partir del corpus paralelo OpenSubtitles 2016. Denominaremos a estos word embeddings ”Numberbatch´´. Hemos usado la versión 17.06, la más moderna que puede encontrarse en la siguiente dirección: `https://github.com/commonsense/conceptnet-numberbatch`.

- **Turian y Senna** [28, 29, 27, 83]. Hemos incluido estos word embeddings por razones históricas, ya que fueron los primeros en demostrar el potencial de los word embeddings. Ambos word embeddings cuentan con vectores de 100 dimensiones.

A continuación, incluimos una tabla donde se resumen las principales características de cada word embedding (Tabla 2.2).

Word Embedding	Corpus	Tokens (entre-namien-to)	Numero de pa-labras	Número de dimensiones	Año
Word2Vec	Google News	100B	3M	300	2013
FastText	Common Crawl	600B	2M	300	2018
FastText	Wikipedia (2017), UMBC, statmt	16B	1M	300	2018
GLOVE	Common Crawl	840B	2,2M	300	2014
GLOVE	Wikipedia (2014), Gigawords5	6B	0,4M	300/200/100/50	2014
GLOVE	Twitter	27B	1,2M	200/100/50/25	2014
LexVec	Common Crawl	58B	2M	300	2016
LexVec	Wikipedia (2015) + NewsCrawl	7B	0,37M	300	2016
LexVec + Context Vectors	Common Crawl	58B	2M	300	2016
LexVec + Context Vectors	Wikipedia (2015) + NewsCrawl	7B	0,37M	300	2016
LexVec + Subword	Common Crawl	58B	2M	300	2018
PDC	Wikipedia (2010)	-	0,39M	300/100/50	2015
HDC	Wikipedia (2010)	-	0,39M	300/100/50	2015
Context2Vec	ukWaC	-	0,19M	600	2016
Context2Vec	MSCC	-	0,1M	600	2016
SketchEngine	Web	20B	5,9M	100	2018
SketchEngine	British National Corpus	100M	0,17M	100	2018
UKB	WordNet 3.0	-	0,15M	300	2015
jointcHYB ENES	WordNet 3.0 + Text	-	1,6M	300	2018
jointcHYB ENEU	WordNet 3.0 + Text	-	0,69M	300	2018
jointcHYB ENIT	WordNet 3.0 + Text	-	1,58M	300	2018
NumberBacth 17.06	ConceptNet, Word2Vec, Glove, OpenSubtitles 2016	-	0,4M	300	2018
Turian	-	-	0,25M	100	2010
Senna v3.0	Wikipedia	-	0,13M	50	2011

Tabla 2.2: Comparación de las características principales de cada word embeddings que vamos a evaluar. M = millones, B = mil millones

2.3. Meta Embedding

Como hemos observado en la sección anterior existe una gran variedad de métodos de generación de word embeddings. El término meta embedding fue propuesto por primera vez por Yin y Schütze en 2016 [87], el objetivo es realizar una combinación del la información obtenida por word embeddings generados mediante los diferentes métodos existentes y diferentes fuentes de información durante el entrenamiento. Con esto se busca obtener una mejora en la calidad de los word embeddings.

Se han propuesto diferentes métodos para la generación de word embeddings. Uno de ellos es 1ToN [87]. 1ToN toma K embeddings preentrenados y utiliza una red neuronal lineal junto con K matrices de proyección global, de modo que a través de las matrices, dada una palabra del meta embedding generado podemos recuperar el vector original de cada word embedding. 1ToN+ amplia este método al predecir palabras que no están presentes en la intersección del vocabulario de los word embeddings empleados.

Bollegala et al [24] (2017), proponen un método lineal local no supervisado. Para cada word embedding, para cada cada palabra; se aprende una representación como una combinación lineal de sus vecinos más cercados. La reconstrucciones locales dentro de cada word embedding son luego proyectadas a un meta embedding común. Matemáticamente este método guarda una gran relación con la concatenación de word embeddings.

Sin embargo, uno de los métodos más simples, la concatenación de word embeddings [87], ha mostrado generalmente un buen rendimiento y ha servido para proporcionar una buena referencia de rendimiento para meta embeddings. Sin embargo, debido a que la concatenación genera word embeddings de gran tamaño, también se han experimentado otros métodos como la reducción de dimensionalidad tras la concatenación usando técnicas como la descomposición en valores singulares, o la media de word embeddings [42, 87, 24, 53].

Otro estudio relevante ha sido el realizado por “Goikoetxea et al” [42], en el usando algunos de los métodos que acabamos de citar, tratan de combinar el conocimiento de word embeddings calculados de forma independiente usando corpus de texto y WordNet.

Además de la generación de meta embeddings combinando word embeddings, también se ha estudiado la posibilidad de combinar el conocimiento de word embeddings con el conocimiento que contienen bases de conocimiento o léxicos semánticos como WordNet [39]. “Faruqui et al” [37] (2014), desarrollaron “Retrofitting”, un método que permite ajustar el conocimiento de un word embedding al de una base de conocimiento. El objeti-

vo es mejorar el word embedding al que aplicamos el método al combinar el conocimiento del propio embedding con el de una base de conocimiento.

Como hemos mencionado en la introducción, Numberbatch [69], es quizá uno de los trabajos más relevantes relacionado con la generación de meta embeddings. Aunque como no presentan un nuevo método como tal, combina algunos de los métodos existentes, en concreto el retrofitting, la concatenación y la reducción de dimensionalidad. Los embedding resultantes son considerados el estado del arte actual en diferentes tareas de evaluación.

2.4. Evaluación de word embeddings

Hemos presentado una gran cantidad de modelos tanto para generar word embedding como para generar meta embeddings, por lo tanto surge la necesidad de comparar dichos métodos para comprobar cuales obtienen un mejor resultado en cada tarea. Por lo tanto, a lo largo del tiempo se han ido planteando algunas formas de evaluación del rendimiento de los word embeddings. Vamos a resumir las más importantes a continuación. Los métodos de evaluación de word embeddings se pueden agrupar en dos grandes grupos, los métodos extrínsecos y los métodos intrínsecos.

2.4.1. Métodos extrínsecos de evaluación

Los métodos extrínsecos de evaluación están basados en la habilidad de un word embedding para ser usados como vectores de características de algoritmos supervisados de aprendizaje automático utilizados en diversas tareas de PLN. Se toma el rendimiento de el método supervisado (normalmente medido en un dataset para tareas de PLN) como medida de la calidad del word embedding. Algunas de las tareas más habituales en las que se evalúan los word embeddings son:

1. Extracción de nombre de frases. El objetivo es identificar frases nominales y sus límites dentro de una sentencia. [73, 83, 29].

2. Reconocimiento de nombres de entidades. Identificar nombre de entidades como nombres de organizaciones, personas, marcas... en una sentencia y sus límites. [83, 29]

3. Análisis de sentimiento. Un caso particular de clasificación de textos, donde un fragmento debe ser marcado con una etiqueta binaria reportando si el texto tiene sentimiento positivo o negativo hacia algo. [73]

4. Análisis de sintaxis superficial. Descomponer sentencias en grupos de frases (frases nominales, frases verbales, frases adjetivas...) [29, 21, 55]

5. Etiqueta de rol semántico. identificar roles semánticos para varios predicados dentro de la sentencia. [29, 36, 65]

6. Alcance de negación. Se trata de una tarea de clasificación de texto. Se trata de identificar su una específica acción en una sentencia determina negación o no. [36]

7. Etiquetado de textos. Generalmente el objetivo es marcar un fragmento de texto con una etiqueta indicando su contenido, por ejemplo, categorizar noticias de deportes en función de si son sobre fútbol, baloncesto... [82]

8. Identificación de metáforas. Es otra tarea de clasificación donde se identifica si una frase es una metáfora o es literal. [81]

9. Detección de paráfrasis. Determinar si dados dos fragmentos de texto, son una paráfrasis entre ellos. Puede tratarse de una tarea de clasificación o de medir la similitud entre dos textos. [22, 19].

10. Detección del compromiso textual. Dadas dos frases, determinar si una es continuación de la otra. [22].

    Podemos encontrar algunos datasets para este tipo de tareas en el estudio realizado por ”Amir Bakarov”, “Una encuesta de métodos de evaluación de word embeddings” [18].

    Este tipo de evaluación resulta útil si queremos encontrar el mejor word embedding para una tarea en específico. Sin embargo, no resultan buenos métodos de evaluación para evaluar word embeddings que han sido entrenados para servir en diferentes tipos de tareas, ya que que un word embedding obtenga un buen rendimiento en una tarea, no significa que lo ha en las demás. Además, la generación de datasets para este tipo de tareas es muy compleja.

2.4.2. Métodos intrínsecos de evaluación

Los métodos de evaluación intrínseca son experimentos en los que los word embeddings se comparan con los juicios emitidos por humanos sobre relaciones entre palabras. A menudo se utilizan conjuntos de palabras creados manualmente, primero se obtienen las evaluaciones humanas y luego estas se comparan con los word embeddings. La mayoría de métodos de evaluación intrínseca están diseñados para recopilar evaluaciones que son resultado de procesos conscientes en el cerebro humano. Existen una gran cantidad de métodos de evaluación que se engloban dentro de los métodos intrínsecos, por lo que nos centraremos en los más utilizados, los llamados métodos de evaluación intrínseca consciente. Existen diferentes tareas que se engloban dentro de ellos:

1. Analogía de la palabras. Se basa en la idea de que las operaciones aritméticas en un espacio vectorial de palabras podrían ser predichas por los humanos: dado un conjunto de tres palabras, a, a* y b, la tarea es identificar tal palabra b* tal que la relación b: b * sea igual que la relación a: a *. "París es a Francia como Moscú es a **Rusia**". La principal crítica a este método es la falta de una métrica de evaluación precisa.

2. Ajuste temático. El método evalúa la capacidad de un modelo para separar diferentes roles temáticos de los argumentos de un predicado. La idea es encontrar qué tan bien los word embeddings pueden encontrar el sustantivo más semánticamente similar para cierto verbo que se usa en cierto rol. Para los humanos, un cierto verbo podría hacer que una persona espere que un cierto rol deba ser ocupado con un nombre determinado (por ejemplo, para el argumento "voy a cortar" el argumento más esperado en el rol de objeto es "pastel")

3. Detección de sinónimos. El objetivo es evaluar la habilidad de un word embedding para dada una palabra W y una serie de palabras K=a_1, a_2, a_3... encontrar la palabra de K más similar a W.

4. Detección de palabras atípicas. Dada una lista de palabras K=a_1, a_2, a_3... el objetivo es detectar la palabra anómala dentro del grupo. Por ejemplo dadas la palabras "Piña, manzana, cereza, naranja, libro, plátano" la palabra anómala es "libro" ya que no guarda relación por el resto.

5. Similitud semántica entre palabras: Este es el método mas popular de evaluación de word embeddings y es el método que vamos a usar para evaluar los word embed-

dings en los próximos capítulos. El método se basa en la idea de que las distancias entre palabras en un word embedding pueden ser evaluadas mediante juicios heurísticos humanos sobre las distancias reales entre palabras. (Por ejemplo, la distancia entre “copa” y “taza” podría ser definida en un intervalo 0,1 como 0.8 ya que son muy similares pero no son exactamente el mismo objeto). Al evaluador se le da una serie de palabras y se pide que evalúe el grado de similitud para cada par. La distancia entre estos pares es también medida en el word embedding, y ambas medidas se comparan. Cuanto más similares sean, mejor será el word embedding. Este método data se 1965, cuando se realizó el primer experimento con juicios humanos sobre la similitud semántica entre palabras para comprobar la hipótesis de distribución. [71]. Sin embargo, este método también recibe ciertas críticas, ya que existen condiciones, lingüísticas, psicológicas y sociales que pueden afectar a los jueces humanos, incluso el cansancio tras anotar un gran número de pares de palabras puede afectar a la puntuación dada. Otra crítica que recibe este método es que diferentes experimentos, tienden a dar diferentes definiciones de similitud semántica, por ejemplo algunos la describen como hiperonímia/hiponímia (“máquina”, “coche”) y otros como sinonimia (“coche”, “vehículo”). Sin embargo, este método es el más popular para evaluar word embeddings.

Existen diferentes datasets para evaluar la similitud entre palabras. Estos se pueden agrupar en dos grandes grupos, los que miden la similitud semántica entre palabras, y los que miden la relación o asociación semántica. Estos dos conceptos son diferentes. Los dataset que estudian la similitud entre palabras generalmente capturan relaciones de sinonimia entre palabras, en ocasiones también relaciones de hiperonímia e hiponímia. Los dataset que se centran en el estudio de la relación semántica, estudian todo tipo de relaciones entre palabras. Pongamos un ejemplo, el par de palabras “costa” y “litoral” son sinónimos, por lo tanto, tanto en los dataset donde se estudia la similitud semántico como en los que se estudia la relación semántica, aparecerán con una puntuación muy alta. En cambio, las palabras “ropa” y “armario” no guardan ningún tipo de relación de sinonimia o hiperoníamia/hiponímia, por lo tanto, en datasets que estudian la similitud semántica, tendrán una puntuación muy baja. Sin embargo, ambas palabras guardan una gran relación entre sí, ya que la ropa se guarda en armarios, por lo tanto en datasets que analizan la relación semántica tendrán una puntuación muy alta.

A continuación enumeraros y describiremos brevemente los dataset que vamos a utilizar para la evaluación de los word embedding y meta embeddings:

Par	Traducción aproximada en Español	Puntuación en Simlex-999	Puntuación en WordSim-353
coast - shore	Costa - Costa/orilla/playa	9	9.1
clothes - closet	ropa - armario	1.96	8

Tabla 2.3: Diferencia en la puntuación de dos pares de palabras en el dataset Simlex 999 (similitud semántica) y WordSim 353 (relación semántica)

1. SimVerb-3500 [31]. 3500 pares de verbos, juzgados por similitud semántica, en una escala del 0 al 4.

2. MEN [34] (Acrónimo para Marco, Elia, y Nam). 3000 pares de palabras juzgados por relación semántica en una escala del 0 al 50. Este dataset se encuentra dividido en tres conjuntos, uno de ellos con el objetivo ser usado como entrenamiento para diferentes tipos de algoritmos, otro con el objetivo de ser usado como test, y otro que combina ambos. Nos referiremos a ellos como DEV, TEST y ALL respectivamente.

3. RW [59] (Acrónimo para Rare Words), 2034 pares de palabras, con la peculiaridad de que son palabras poco usadas, que cuentan con pocas apariciones en los corpus de entrenamiento. Los pares están juzgados por similitud semántica en una escala del 0 al 10.

4. SimLex-999 [38]. 999 pares de palabras juzgados con un fuerte respecto a la similitud semántica en una escala de 0 a 10.

5. MTurk-771 [46] (Acrónimo para Mechanical Turk), 771 pares de palabras juzgados por relación semántica en una escala del 0 al 5.

6. MTurk-287 [54]. 287 pares juzgados por relación semántica en una escala del 0 al 5.

7. WordSim-353 [57]. 353 pares juzgados por similitud semántica en una escala del 0 al 10. Estén dos divisiones de este dataset. La primera es la original, en el que el dataset se divive en dos set, llamados set1 y set2. Sin embargo muchos investigadores consideran que las instrucciones dadas a los jueces sobre similitud y asociación fueron ambiguas provocando que ambas se mezclen en la evaluación. Por ello existe una segunda división propuesta por “Agirre et al.” [35] donde las 353 palabras se dividen en dos datasets. Uno cuyos pares están centrados en medir la similitud semántica entre palabras y el otro cuyos pares están centrados en medir la relación

semántica entre palabras. En las secciones posteriores nos referiremos como “ALL” al conjunto de palabras formado por las 353 palabras. Nos referiremos como “set1” y “set2” a los dos conjuntos originales. Y nos referiremos como “similarity” y “relatedness” a los conjuntos de pares centrados en el análisis de similitud y relación respectivamente.

8. Verb-142 [20]. 143 pares de verbos juzgados por similitud semántica en una escala del 0 al 4.

9. YP-130 [86] (Acrónimo para Yang and Powers), 130 pares de verbos juzgados por similitud semántica en una escala del 0 al 4.

10. RG-65 [48] (Acrónimo para Rubenstein and Goodenough). 65 pares de palabras juzgados por similaridad semántica en una escala del 0 al 4.

11. MC-30 [62] (Acrónimo para Miller and Charles), 30 palabras, un subconjunto de RG-65 que contiene 10 pares con gran similitud semántica, 10 con una similitud semántica media y 10 con baja similitud.

Para facilitar la distinción de los tipos de dataset, los hemos divido en la Tabla 2.4.

Datasets	
Similitud semántica	Relación Semántica
SimVerb-3500	MEN
RW	MTurk-287
SimLex999	MTurk-771
WS353 similarity	WS353 relatedness
Verb 143	
RG 65	
MC30	

Tabla 2.4: Datasets divididos en similitud semántica y relación semántica

Los dataset contienen los pares de palabras seguidos de la puntuación para cada par, normalmente denominado “Gold Standard”. En ocasiones también se añaden las puntuaciones dadas por cada evaluador y el “Gold Standard” por lo general es la media de las puntuaciones dadas por los evaluadores. La estructura de los dataset es la siguiente (Primeros diez pares de palabras de MTurk-287).

Para cada par de palabras, se calcula la distancia entre las los vectores que representan a ambas palabras en el word embeddings que estamos evaluando. La distancia entre

```
water   shortage    2.714285714
horse   wedding 2.266666667
plays   losses  3.2
classics    advertiser  2.25
latin   credit  2.0625
ship    ballots 2.3125
mistake error   4.352941176
disease plague  4.117647059
sake    shade   2.529411765
....
```

palabras puede medirse de diferentes formas, distancia euclídea, producto escalar... sin embargo la medida más utilizada es la similitud coseno. La similitud coseno es una medida de similitud entre dos vectores (no nulos) en un espacio que posee un producto interior con el que se evalúa el valor del coseno del ángulo comprendido entre ellos. Esta función trigonométrica proporciona un valor igual a 1 si el ángulo comprendido es cero, es decir, ambas palabras son idénticas. Si los vectores fuesen ortogonales el coseno se anularía, lo cual significaría que ambas palabras no cuentan con ninguna relación entre ellas. En caso de que ambos vectores apunten en sentidos opuestos el resultado sería -1, ambas palabras tendrían significados contrarios. El valor de esta métrica se encuentra por lo tanto entre -1 y 1. Esto quiere decir, que para calcular la similitud entre dos palabras, utilizamos el ángulo que forman sus vectores como medida.

$$\text{Similitud coseno } cos(\boldsymbol{x},\boldsymbol{y}) = \frac{\boldsymbol{x}\cdot\boldsymbol{y}}{||\boldsymbol{x}||\cdot||\boldsymbol{y}||}$$

Una vez evaluados todos los pares de palabras dispondremos de dos vectores, uno con los "Gold Standard" proporcionados por el dataset y otro con los resultados del embedding para cada par de palabras, en la Figura 2.1 podemos ver una representación gráfica de los Gold Standard proporcionados por SimLex999 y los resultados obtenidos por FastText CC para cada par de palabras de Simlex999.

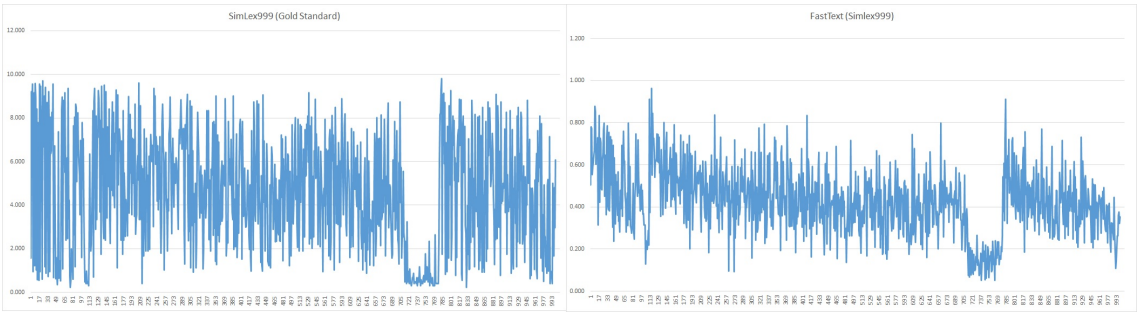


Figura 2.1: Comparativa entre los Gold Standard proporcionados por Simlex999 y las puntuaciones obtenidas por FastText CC para cada par de palabras

El último paso es comparar ambos vectores para determinar como de similares son. Esto se realiza mediante el coeficiente de correlación de spearman [74]. Esta es una medida de correlación entre dos variables aleatorias. Este coeficiente se encuentra en el rango entre -1 y 1, indicando asociaciones negativas o positivas, y 0, significa que no existe correlación. En el caso del ejemplo de la Figura 2.1 , la correlación de spearman entre los gold standard y los resultados de FastText es 0.503. Cuando más se aleje del cero este valor, significa que una mayor correlación y por lo tanto los resultados obtenidos por el word embedding son más similares a los gold standard proporcionados por el dataset.

2.5. Estudios previos realizados.

Por último haremos un pequeño repaso de los estudios realizados en el ámbito de la evaluación de word embeddings y meta embeddings.

Existen algunas evaluaciones de word embeddings pre entrenados previas, como la realizada por Yin y Schütze en 2016 [87] y más recientemente la realizada por "Jastrzebski et al" [49]. Sin embargo, estas evaluaciones no han analizado un pequeño número de word embeddings, embeddings tan populares como FastText no aparecen en las comparativas. Tampoco aparecen word embeddings calculados mediante Ramdom Walks sobre WordNet.

En cuando a la generación de meta embeddings, en la Sección 2.3 hemos mencionado diferentes estudios en este ámbito. En el estudio realizado por Bollegala et al [24], se presenta una comparación de su método contra el presentado por Yin y Schütze [87]. En ella también se incluyen los resultados obtenidos por la concatenación y la reducción de dimensionalidad tras la concatenación. Sin embargo, no se realiza un estudio profundo de todos los métodos aplicados a diferentes word embeddings, se realiza una comparativa combinando cinco word embeddings, siendo la varios de ellos word embeddings antiguos que ofrecen un rendimiento muy inferior a los actuales, y siendo todos ellos word embeddings calculados a partir de textos. Por lo tanto, no podemos saber que tipo de word embeddings es mejor combinar entre ellos, tampoco se profundiza en la mejor forma de aplicar los métodos de generación de meta embeddings analizados (normalizaciones, resolución de ciertas problemáticas...).

En el estudio realizado por "Goikoetxea et al" [42], se muestran buenos resultados al combinar word embeddings producidos a partir de concatenar word embeddings calculados a partir de corpus de textos y a partir de WordNet. Sin embargo, en las pruebas realizadas

no queda claro si tomando varios word embeddings es mejor realizar la concatenación o la media de los mismos. Relacionado con los estudios mencionados en el párrafo anterior, también encontramos una diferencia, no parece que se realice una normalización L_2 de los vectores antes de aplicar la normalización o la media, mientras que en Bollegala et al [24] si se menciona dicha normalización. Carecemos de una comparativa para determinar si es mejor aplicar dicha normalización o no.

En el estudio realizado por "Coates et al" [53], se realiza una pequeña comparativa entre la media y la concatenación de vectores. Aunque se realiza un muy buen análisis matemático de ambos métodos, se realizan pruebas con tan solo tres word embeddings, todos esos calculados a partir de corpus de textos.

Por lo tanto, aunque algunos estudios muestran resultados prometedores, no se ha realizado un análisis en profundidad de diferentes métodos de generación de word embeddings en el que se incluyan diferentes tipos de word embeddings. Es por ello que resulta complicado responder a la pregunta de cual es la mejor forma de combinar un determinado número de word embeddings. Y es la razón por lo que hemos realizado este proyecto.

3. CAPÍTULO

Comparativa de Word Embeddings

En este capítulo se expondrán y analizarán los resultados de la evaluación de los diferentes Word Embeddings de forma individual. Esta sección tiene dos objetivos principales. Por un lado, analizar el rendimiento de los diferentes word embedding en la tarea del cálculo de la similitud entre palabras. De esta forma si deseamos escoger un word embedding para aplicarlo a dicha tarea podremos saber cuales obtendrán un mejor rendimiento. De dicho análisis también esperamos conocer características interesantes de los diferentes métodos y técnicas de generación de word embeddings, como pueden ser como afecta al rendimiento el uso de diferentes corpus. Por otro lado este análisis nos servirá de linea base para los siguientes capítulos. Conocer el rendimiento de forma individual de cada word embedding nos permitirá establecer comparativas para analizar si los word embedding generados en los próximos capítulos obtienen un mejor rendimiento. Además, nos permitirá conocer cuales son los mejores word embeddings y de esta forma utilizarlos para generar los mejores meta embeddings posibles.

3.1. Metodología de evaluación

Comenzaremos exponiendo la metodología de evaluación utilizada para evaluar los word embedding. Como mencionado en la Subsección 2.4.2 vamos a centrarnos en el estudio del rendimiento en la tarea del cálculo de la similitud entre palabras. Vamos a estudiar el rendimiento de diferentes word embeddings, en concreto los nombrados en la Sub-

sección 2.4.2. Evaluaremos dichos word embeddings en los dataset mencionados en la Sección 2.2.

La forma en la que hemos realizado la evaluación ha sido la siguiente: Por cada par de palabras de los dataset se ha calculado la similitud coseno entre los vectores que representan cada palabra en cada word embedding. Solo se proporciona un resultado cuando las dos palabras del par cuentan con representación en el word embedding. En caso contrario, el par de palabras es descartado. Tras esto el resultado final para cada dataset se obtiene calculando el coeficiente de correlación de Spearman, entre los resultados obtenidos y los proporcionados por el dataset para las palabras para las que se ha podido dar respuesta.

Descartar los pares de palabras para los que el word embedding que estamos evaluando no puede dar una respuesta por que no cuenta con representación para una o ambas palabras del par es la práctica más habitual. Sin embargo ¿Es correcto decir que un word embedding que solo puede dar respuesta a dos pares de palabras en un dataset de dos mil palabras es un buen embedding en ese dataset si el coeficiente de correlación de Spearman nos da un resultado alto sólo para esos dos pares de palabras?. Este método es fácilmente manipulable limitando el número de palabras para el que damos respuesta. Podemos dar respuesta solo para los pares para los que word embedding cuenta con un buen rendimiento y de esta forma aumentar la puntuación en un dataset. Además, este método de evaluación nos genera otro problema. Una de las posibles ventajas de los meta-embeddings es generar nuevos embeddings cuyo vocabulario incluye los vocabularios de los word embeddings usados para generarlo. Es decir, el meta embedding generado cuanta con un vocabulario mayor que los word embeddings usados para generarlos individualmente. Sin embargo este método de evaluación no lo tiene en cuenta esta mejora. Por desgracia no existe un método de evaluación consolidado y ampliamente usado que resuelva este problema.

Recientemente algunas de las últimas competiciones y tareas relacionadas con el cómputo de la similitud entre palabras palabras como “SemEval-2017-Task 2” [5] han tomado la decisión de forzar a que todos los sistemas reporten siempre para cada par de palabras un resultado de similitud. La organización de “SemEval-2017-Task 2” recomienda dar como resultado por defecto un punto intermedio de la escala de similitud. Si bien este método puede ser una buena alternativa como último recurso cuando se está utilizando un word embedding en un sistema real, añade una componente de aleatoriedad al sistema de evaluación, ya que si el valor dado por defecto resulta ser un buen resultado para el par de palabras hará que la puntuación final del embedding mejore mientras que si resulta ser un mal resultado hará que la puntuación final se reduzca. Lo mismo ocurre con otras propuestas como usar la media de todos los vectores del embedding como valor

por defecto cuando no tenemos representación de una palabra. Puesto que buscamos un sistema de evaluación robusto, no consideramos que la aleatoriedad sea un buen añadido al sistema.

Otros estudios muestran la puntuación obtenida en un determinado dataset y muestran también el porcentaje de pares de palabras para los que se ha dado respuesta. El problema de esta representación, es que en caso es que resulta difícil saber a simple vista si es mejor obtener una puntuación de 0.8 con una cobertura del 80 % o obtener una puntuación de 0.7 con una cobertura del 90 %. Por ello, inspirándonos en esta forma de mostrar los resultados, nosotros proponemos usar como medida la multiplicación entre el resultado del embedding en un determinado dataset y el porcentaje de pares de palabras para los que el embedding ha sido capaz de dar una respuesta dividido entre cien. Por ejemplo, si el resultado de un determinado embedding en un dataset ha sido 0.875 y ha dado respuesta a 80 de 100 pares de palabras, el resultado final será 0.875 * (80/100), es decir, 0.7. Con esta métrica buscamos penalizar que un word embedding no sea capaz de dar respuesta para un determinado número de pares de palabras, haciendo que la métrica de evaluación tenga también en cuenta este factor. Decimos que un word embedding no puede dar respuesta para un determinado par de palabras cuando no cuenta con representación para una o ambas palabras. Es decir, si podemos dar respuesta para la mitad de pares de palabras la cobertura será de un 50 % independientemente de si el word embedding contaba con representación para una o ninguna palabra de los pares para los que no se ha podido dar respuesta.

Durante las evaluaciones que realizaremos en los diferentes capítulos usaremos las dos métricas. La primera métrica es la usada más comúnmente, descartaremos los pares de palabras para los que no podemos dar respuesta y daremos el resultado obtenido en el dataset sin tener en cuenta esos pares. Esta métrica no tiene en cuenta el número de pares de palabras para los que se ha podido dar respuesta. Cuando consideremos que resulta interesante tener en cuenta el número de pares de palabras para los que el word embedding ha podido dar respuesta, usaremos la métrica que acabaos de exponer, multiplicar el resultado obtenido por el porcentaje de pares de palabras para el que se ha podido dar respuesta. Esta métrica penaliza que un word embedding no sea capaz de dar respuesta para un determinado número de pares de palabras. Sin embargo, teniendo en cuenta que los word embeddings que vamos a usar cuentan con vocabularios muy extensos, el uso de esta métrica no provocará cambios significativos en el rendimiento de los word embeddings y meta embeddings salvo en casos muy concretos, por lo que para evitar redundancia, la métrica más usada será la primera.

Otro problema al que nos enfrentamos durante la evaluación es que teniendo en cuenta todas las subdivisiones de los datasets, en total tenemos 17 datasets. Por lo tanto, por cada word embedding se aportan 17 resultados diferentes, 34 teniendo en cuenta ambas métricas. Esto provoca que sea muy difícil ver a simple vista que word embedding obtiene un mejor resultado. Hemos observado que normalmente cuando un word embedding es mejor que otro tiende a obtener un mejor rendimiento en todos los dataset por igual, por lo tanto analizar el rendimiento dataset por dataset no aporta demasiada información. Debido a esto hemos usaremos dos valores a modo de "resumen" del rendimiento obtenido. El primero es la media, como su nombre indica se trata de la media aritmética de los resultados obtenidos en los diferentes datasets. Puesto que tenemos datasets de muy diferentes tamaños (desde 30 hasta 3500 palabras) también incluiremos un segundo valor, la media ponderada. Realizaremos la media aritmética dando a cada datasets un peso que dependerá del número de palabras con el que cuente. La media ponderada varía ligeramente según la métrica que usemos. Usando la primera métrica cada dataset tendrá un peso en la media igual al número de pares de palabras para los que ha podido dar respuesta el word embedding que estamos evaluando en dicho dataset. En el caso de la segunda métrica cada dataset tendrá un peso igual al total de palabras del dataset.

Media ponderada primera métrica.

$$\frac{\sum_{Dataset_1}^{Dataset_n}(Puntuacion)*(Cobertura)*(Numero\ de\ palabras\ del\ dataset)}{\sum_{Dataset_1}^{Dataset_n}(Cobertura)(Numero\ de\ palabras\ del\ dataset)}$$

Media ponderada segunda métrica.

$$\frac{\sum_{Dataset_1}^{Dataset_n}(Puntuacion)*(Numero\ de\ palabras\ del\ dataset)}{\sum_{Dataset_1}^{Dataset_n}(Numero\ de\ palabras\ del\ dataset)}$$

En el caso de los dataset que cuentan con varias subdivisiones, MEN y WS353, solo se ha tenido en cuenta el dataset que incluye todos los pares de palabras para realizar la media y la media ponderada. Debido a que como hemos explicado en el capítulo anterior cada dataset se centra en evaluar un tipo de relación entre palabras diferentes, generalmente nos fijaremos más en la media que en la media ponderada. La media ponderada puede dar mejor puntuación a los word embeddings que obtengan un mejor rendimiento en los datasets más grandes, centrándose solo en un tipo de relación entre palabras. En los casos en los que se considere que no es interesante mostrar el rendimiento en cada dataset usaremos estos dos valores a modo de "resumen". En caso de que consideremos que resulta interesante mostrar el rendimiento en determinados datasets, mostraremos también dichos resultados. Sin embargo, todos los resultados obtenidos en todos los datasets por todos los word embeddings y meta embeddings que estudiaremos durante este proyecto,

usando ambas métricas y la cobertura de dichos word embeddings y meta embeddings en cada dataset están disponibles para su consulta en la plataforma GitHub [1], consideramos que es mejor su publicación en una plataforma online en vez de como anexo a este documento por que se trata de tablas de gran tamaño, disponer de una versión digital permite aplicar ordenaciones, búsquedas... por lo que facilita s uso y les proporciona mayor utilidad. Además evitamos que se impriman decenas de folios llenos de tablas ahorrando así papel. Por lo tanto, si los lectores desean consultar algún dato no mostrado en este documento, los resultados completos son accesibles para quien quiera consultarlos.

El dataset WS353 tiene una peculiaridad. Mientras que las palabras en el resto de los conjuntos se encuentran siempre en letras minúsculas, en este dataset tenemos palabras como “Jerusalem” o “CD” que incluyen mayúsculas. Gran parte de los word embeddings se han entrenado convirtiendo todas las palabras a letras minúsculas, por lo tanto, no cuentan con representaciones para palabras que contengan mayúsculas. Debido a esto hemos convertido todas las palabras de este conjunto a minúsculas, salvo en el caso de que el word embedding a evaluar incluya en su vocabulario representaciones de palabras en mayúsculas. Esto es debido a que aunque el embedding tenga una representación para la palabra “Jerusalem” y otra para la palabra “jerusalem”. Como generalmente esta palabra aparece escrita con su primera letra en mayúscula, su representación será mucho más precisa al haber dispuesto de un mayor número de muestras durante el entrenamiento. [2]

En el caso de las combinaciones de word embeddings, como podemos tener algunos que cuenten con representación de palabras en mayúsculas y otros que no cuenten con representación de palabras en mayúsculas, hemos tomado la decisión de convertir todas las palabras a minúsculas en todos los casos.

3.2. Evaluación de word embeddings

3.2.1. Ranking de Word Embeddings

En esta sección analizaremos los diferentes word embeddings que hemos escogido para la evaluación. Dividiremos esta sección en varias subsecciones en las que nos centraremos en analizar diferentes aspectos del rendimiento de los word embeddings. Comenzaremos

[1] https://github.com/ikergarcia1996/RotEmbeddings

[2] Los embeddings que incluyen representación de palabras en mayúsculas son: GLOVE entrenado en el corpus Common Crawl con 840 mil millones de tokens. Todos los FastText. Turian. SW2V. Lexvec entrenado en el corpus Common Crawl con información de subpalabras.

realizando un ranking de word embeddings en función de rendimiento. Dicho ranking nos será de gran utilidad para seleccionar los mejores word embeddings para generar meta-embeddings en los siguientes capítulos. En las siguientes subsecciones analizaremos otros aspectos que pueden resultar de interés, como la comparación de word embeddings generados a partir de grafos y corpus de texto, o como afecta el número de dimensiones o el corpus usado al rendimiento final del word embeddings. La metodología seguida para la evaluación de los word embeddings es la que ha sido descrita en la Sección 3.1. Resulta importante resaltar que en estas sección hemos evaluado los embeddings originales, sin aplicar ningún tipo de post-procesamiento o normalización. Salvo en la excepción de Glove, donde los autores recomiendan aplicar una normalización mediante la norma L_2 a sus variables para obtener un buen rendimiento. Como comprobaremos en el Capítulo 4, Glove requiere de esta normalización para obtener un buen rendimiento.

Para realizar este ranking hemos ordenado los word embeddings de mayor a menor en función de la media obtenida en todos los dataset. Primero vamos a realizar un ranking usando la primera métrica descrita en la Sección 3.1, la cual no penaliza que un word embedding no sea capaz de dar respuesta para un determinado número de pares de palabras. Los resultados obtenidos se muestran en la tabla Tabla 3.1.

El mejor word embeddings resulta ser Numberbatch. Numberbatch como hemos explicado anteriormente se trata de un meta-embedding que combina el conocimiento de diferentes word embeddings y ConceptNet. Que obtenga un rendimiento significativamente superior a cualquier otro word embedding, es uno de los motivos principales por lo que en este documento estudiaremos diferentes métodos de generación de meta embeddings. Numberbatch es la prueba de que combinar diferentes word embeddings puede ayudarnos a generar meta embeddings mejores que los word embeddings principales.

Dentro de los word embeddings, los embeddings que obtienen un mejor rendimiento son los entrenados en corpus de grandes dimensiones, como Common Crawl. Algunos ejemplos de estos word embeddings son FastText, Glove, LexVec o word2vec. Dentro de este grupo FastText es el que mejor rendimiento obtiene. Junto a estos word embeddings también se encuentran jointcHYB y UKB, ambos calculados a partir de la información obtenía de Ramdom Walks sobre WorNet. Estos word embeddings muestran un rendimiento parejo al de los embeddings calculados sobre grandes corpus de texto. A medida que reducimos los corpus usados para entrenar los word embeddings y las dimensiones de los vectores podemos ver como el rendimiento se reduce. Los word embeddings que se encuentran en la parte de abajo de la tabla son Turian y Senna. El rendimiento de word embeddings modernos como FastText o Glove es muy superior al de estos word embed-

Embedding	Media	Media Ponderada
Numberbatch	0.743	0.708
FastText CC	0.675	0.620
jointcHYB EN_ES	0.654	0.606
jointcHYB EN_EU	0.644	0.607
Subword LexVec CC	0.630	0.573
FastText Wiki	0.630	0.562
UKB	0.630	0.610
GLOVE CC 840B	0.629	0.571
GLOVE CC 42B	0.628	0.553
jointcHYB EN_IT	0.617	0.565
Word2Vec	0.616	0.555
SketchEngine web	0.610	0.545
LexVec CC WordVectors	0.606	0.543
LexVec CC WordVectors + ContexVectors	0.601	0.534
PDC. Dims: 300	0.599	0.529
HDC. Dims: 300	0.593	0.515
Contex_to_vec ukwac	0.588	0.531
GLOVE WIKI. Dims 300	0.577	0.511
PDC. Dims 100	0.573	0.494
SketchEngine British National	0.563	0.492
LexVec Wiki+NewsCrawls + ContexVectors	0.559	0.499
HDC. Dims 100	0.559	0.465
LexVec Wiki+NewsCrawls WordsVectors	0.554	0.506
GLOVE Wiki. Dims 200	0.551	0.483
Contex_to_vec mscc	0.540	0.462
PDC. Dims 50	0.538	0.456
HDC. Dims 50	0.518	0.433
GLOVE Wiki. Dims 100	0.517	0.452
GLOVE twitter. Dims 200	0.511	0.449
GLOVE Wiki. Dims 50	0.473	0.422
GLOVE twitter. Dims 100	0.458	0.388
Senna	0.414	0.370
GLOVE twitter. Dims 50	0.389	0.332
GLOVE twitter. Dims 25	0.326	0.267
Turian. Dims: 100	0.307	0.248

Tabla 3.1: Word embedding ordenados de mayor a menor por la media obtenida en los diferentes dataset. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

dings, llegando a duplicarlo. Esto demuestra el gran avance que se realizado en este campo en los últimos años.

Vamos a realizar este mismo ranking usando ahora la segunda métrica. La cual penaliza que un word embedding no sea capaz de dar respuesta para un determinado número de pares de palabras. Podemos ver los resultados en la Tabla 3.2.

En la Tabla 3.1 podemos observar como los word embeddings entrenados en grandes corpus de texto apenas ven su rendimiento afectado por la segunda métrica. Contienen representaciones de palabras para casi el 100 % de los pares de palabras (solo en el dataset RareWords no pueden dar respuesta para todos los pares de palabras, y en dicho dataset su cobertura es superior al 95 %). Sin embargo, UKB, que ha sido entrenado con la información extraída de WordNet mediante RamdomWalks ve su rendimiento afectado muy negativamente por esta métrica ya que su vocabulario es muy limitado y no puede dar respuesta para un número significativo de pares de palabras, por ejemplo en el dataset RareWords solo puede dar respuesta para un 69 % de pares de palabras.

Consideramos que el rendimiento de cada word embedding en cada dataset usando ambas métricas y el porcentaje de pares de palabras para los que puede dar respuesta en cada dataset son de gran importancia, tanto para este capítulo como para las comparativas que realizaremos en futuros capítulos. Por ello en el Capítulo 10 hemos incluido las tablas completas que muestran estos resultados.

Embedding	Media	Media Ponderada
Numberbatch	0.737	0.695
FastText CC	0.674	0.618
jointcHYB EN ES	0.651	0.597
jointcHYB EN EU	0.638	0.590
GLOVE CC 840B	0.629	0.569
Subword LexVec CC	0.629	0.570
FastText Wiki	0.628	0.559
GLOVE CC 42B	0.627	0.552
jointcHYB EN IT	0.614	0.553
SketchEngine web	0.610	0.545
LexVec CC WordVectors	0.606	0.541
Word2Vec	0.604	0.541
LexVec CC WordVectors + ContexVectors	0.601	0.533
UKB	0.596	0.568
PDC. Dims 300	0.595	0.521
HDC. Dims 300	0.589	0.507
Contex_to_vec ukwac	0.575	0.504
GLOVE WIKI	0.572	0.500
PDC. Dims 100	0.569	0.487
HDC. Dims 100	0.555	0.457
LexVec Wiki+NewsCrawls + ContexVectors	0.553	0.486
SketchEngine British National	0.551	0.469
Multilingual512	0.550	0.476
LexVec Wiki+NewsCrawls WordsVectors	0.548	0.493
GLOVE Wiki. Dims 200	0.546	0.472
PDC. Dims 50	0.535	0.448
HDC. Dims 50	0.515	0.426
GLOVE Wiki. Dims 100	0.512	0.441
Contex_to_vec mscc	0.505	0.408
GLOVE twitter. Dims 200	0.493	0.411
GLOVE Wiki. Dims: 50	0.468	0.413
GLOVE twitter. Dims 100	0.441	0.355
Senna	0.398	0.339
GLOVE twitter. Dims 50	0.374	0.303
GLOVE twitter. Dims 25	0.312	0.244

Tabla 3.2: Word embedding ordenados de mayor a menor por la media obtenida en los diferentes dataset. Métrica utilizada: Multiplicación de los resultados obtenidos en los diferentes dataset y el porcentaje de palabras a las que el embedding ha dado respuesta en cada dataset.

3.3. Word embeddings entrenados mediante corpus de textos vs Word Embeddings entrando mediante Random Walks sobre WordNet

En la Sección 2.2 hemos mencionado que existen dos grandes grupos de word embeddings. Los que han sido entrenados a partir de corpus de textos, y los que han sido entrenados a partir de la información extraída de RamdomWalks sobre WordNet. Debido a que los datos usados durante el entrenamiento han sido significativamente diferentes resulta interesante analizar en profundidad el rendimiento de algunos de estos word embeddings en diferentes datasets. Vamos a seleccionar para analizar esto dos word embeddings de cada clase, como word embeddings calculados usando corpus de textos usaremos Glove y FastText, y como word embeddings calculados usando RandomWalks sobre wordnet usaremos FastText y Glove. En la Tabla 3.3 podemos ver los resultados en diferentes datasets.

Embedding	Mturk 287	Mturk 771	WS353 relatedness	MEN all	MC 30	YP 130	SimVerb 3500	SimLex999
UKB	0.644	0.679	0.627	0.756	0.873	0.768	0.545	**0.525**
jointcHYB EN_ES	0.625	0.692	0.668	0.780	**0.877**	**0.816**	**0.558**	0.541
FastText CC	**0.726**	**0.762**	0.737	**0.837**	0.852	0.625	0.426	0.503
GLOVE CC	0.721	0.736	**0.738**	0.829	0.788	0.597	0.338	0.453

Tabla 3.3: Resultados obtenidos en diferentes datasets de FastText y Glove (entrenados en corpus de textos) y UKB y jointcHYB (Random Walks sobre WordNet)

Los resultados de la Tabla 3.3 son muy llamativos. Los embeddings calculados sobre textos muestran un mejor rendimiento en datasets como Mturk, MEN o WS353 (relatedness). Mientras que los word embeddings calculados usando Ramdom Walks sobre WordNet obtienen un mejor rendimiento en datasets como Simlex999 o SimVerb. Como hemos explicado en la Subsección 2.4.2 estos datasets son diferentes entre ellos.

MC30, YP 130, SimVerb 3500 y SimLex 999 se centran en el estudio de relaciones de sinonimia, además de relaciones hiperónimas e hipónimas. Por ejemplo en el dataset YP 130 las palabras “juez” y “carcel” reciben tan solo una puntación de 0.333 sobre 4, ya que pese a que están relacionadas, no son sinónimos o hiperónimos. En cambio en el dataset MC30 podemos observar que las palabras “fruta” y “comida” reciben una puntación de 3.08 sobre 4, ya que comida es un hipónimo de fruta. O “coche” y “automobil” reciben una puntuación de 3.92 sobre 4 ya que son sinónimos.

Por otro lado, lado, Mturk 287, Mturk 771, WS353 (relatedness) y Men, miden la relación

que existe entre palabras, por ejemplo, en MTurk 771 las palabras “caramelo” y “dulce” reciben una puntuación de 4.51 sobre 5. Estas palabras no son sinónimos, ni hipónimos, ni hiperónimos. Sin embargo los caramelos generalmente son dulces por lo que las palabras están altamente relacionadas entre ellas.

Este rendimiento puede explicarse de forma sencilla, UKB y jointcHYB (aunque también se usan corpus de texto en su entrenamiento se comporta de forma similar a UKB) han sido entrenados mediante Random Walks sobre WordNet, WordNet se estructura como un grafo donde los nodos (palabras) son conectados mediante relaciones de sinonímia, hiperonímia e hiponímia. Por lo tanto es lógico que embeddings que contienen el conocimiento de WordNet sean mejores en este tipo de tareas. En cambio, en corpus de texto, es muy probable que en repetidas ocasiones encontremos palabras como “caramelo” y “dulce” o “mar” y “pesca” en un mismo contexto, por lo tanto los métodos de entrenamiento de word embeddings que usan corpus de texto aprenderán que estas palabras están relacionadas, y por eso obtienen un mejor rendimiento en datasets que estudian las relaciones entre palabras. Esto resulta muy interesante, ya que tenemos word embeddings que codifican un conocimiento muy diferente debido a como han sido entrenados, lo cual puede resultar muy útil para generar meta embeddings, ya que podemos combinar estos conocimientos. También resulta interesante por que nos muestra que los datos que usamos para el entrenamiento tienen incluso más importante que el método usado. Glove y FastText son dos métodos de generación de meta embeddings completamente diferentes, sin embargo se comportan de forma similar.

3.4. Rendimiento en palabras “Raras”

En la sección anterior hemos analizado el rendimiento de word embeddings calculados haciendo uso de diferentes tipos de datos durante el entrenamiento. Otra diferencia importante entre los diferentes tipos de métodos de generación de word embeddings es la calidad de los vectores generados para palabras que han aparecido pocas veces durante el entrenamiento. Esto resulta interesante por que como hemos mencionado en la Sección 2.2, FastText incluye los llamados “ngrams” pensados para mejorar la calidad de las palabras que aparecen en pocas ocasiones durante el entrenamiento. La versión más moderna de LexVec (Subword LexVec CC) también incluye los “ngrams”, al igual que SketchEngine el cual ha sido entrenado usando FastText. Disponemos de un dataset pensado específicamente para analizar esto RareWords Tabla 3.4, por lo que analizaremos el

rendimiento de los word embeddings en dicho dataset. Los resultados se muestran en la Tabla 3.4.

Embeddings	RW	Cobertura	RW * Cobertura
Numberbatch	**0.626**	0.8982	0.562
FastText CC	0.595	0.9808	**0.584**
Subword LexVec CC	0.539	0.9784	0.528
Word2Vec	0.534	0.8972	0.479
GLOVE CC 840B	0.530	0.9823	0.521
FastText Wiki	0.523	0.9666	0.506
SketchEngine web	0.515	0.9995	0.515
LexVec CC WordsVectors	0.508	0.9867	0.501
GLOVE WIKI 6B	0.459	0.8761	0.402
UKB	0.443	0.6888	0.305
jointcHYB EN_ES	0.378	0.9189	0.347

Tabla 3.4: Rendimiento de diferentes word embeddings en el dataset RareWords

Como podemos observar usando la primera métrica, la cual no penaliza que un word embedding no sea capaz de dar respuesta para un cierto número de pares de palabras, Numberbatch es el embedding que mejor rendimiento obtiene. Numberbatch se trata de un meta embedding que combina el conocimiento de diferentes fuentes de conocimiento, esto le permite disponer de mayor información sobre palabras raras, mostrando una de las ventajas de los meta embeddings. Por debajo, como era de esperar se encuentra FastText con un rendimiento significativamente superior al resto de word embeddings. La versión de Lexvec que incluye los ngrams es el segundo word embedding que mejor rendimiento optiene. Glove entrenado en el corpus common crawl con 840 mil millones de palabras también obtiene un buen rendimiento gracias al corpus de entrenamiento tan grande que se ha usado. Sin embargo, la versión entrenada en wikipedia con un corpus mejor obtiene un rendimiento significativamente inferior al que obtiene FastText entrenado en wikipedia, el cual incluso usando un corpus mucho más pequeño consigue un rendimiento cercano a la versión más grande de Glove, mostrando de nuevo que FastText consigue un muy buen rendimiento en este dataste. Sin embargo, algunos resultados muestran el problema de la métrica que estamos utilizando. Algunos word embeddings responden a un número significativamente menor de palabras que otros, destaca el caso de UKB y jointcHYB, pese a haber sido entrenado con más información, jointcHYB obtiene una puntuación muy inferior. Esto es resultado de que UKB tan solo responde a un 69 % de los pares de palabras del dataset. Por lo tanto, responde a las palabras “menos raras”, las palabras

para las que más apariciones existen y para las que las representaciones de palabras de las que dispone son buenas. Como no tenemos en cuenta el número de pares de palabras respondido, obtiene una buena puntuación.

Si usamos la segunda métrica, los resultados cambian. Los métodos que incluyen ngrams obtienen un mejor rendimiento respecto al resto. La versión de Glove entrenada con 840 mil millones de palabras sigue obteniendo un buen rendimiento gracias al gran corpus en el que se ha entrenado. Sin embargo, Glove entrenado en wikipedia es muy significativamente inferior a FastText entrenado en wikipedia o SketchEngine. Word2Vec también resulta ahora inferior a FastText. Como podemos comprobar, FastText se muestra un muy buen método para el calculo de word embeddings para palabras para las que se disponen de pocas apariciones durante el entrenamiento.

Por otro lado, los word embeddings calculados sobre WordNet no ofrecen un buen rendimiento en este dataset, esto puede ser consecuencia de que estas palabras no se encuentran bien representadas en WordNet o incluso son inexistentes, ya que UKB no puede responder a una gran cantidad de pares de palabras.

3.5. Corpus usado y dimensiones de los vectores

Otro aspecto interesante a analizar es como afecta a los word embeddings el uso de diferentes corpus de entrenamiento, diferentes tamaños de corpus y diferente número de dimensiones. Esto es interesante por que diferentes métodos de generación de word embeddings pueden necesitar más o menos información para alcanzar el mismo rendimiento. Nos centraremos en los métodos para los que disponemos varios word embeddings calculados en diferentes corpus o con diferente número de vectores. Durante esta sección usaremos la primera métrica, queremos centrarnos en el rendimiento y no en el porcentaje de palabras que puede responder cada word embeddings. Además hemos comprobado que las conclusiones a las que llegaremos no cambian usando una métrica u otra, por lo que mostrar ambas sería redundante. Hemos añadido junto al nombre de todos los word embeddings el número de dimensiones con el que cuentan y cuantas palabras contaba el corpus con el que ha sido entrenado, por que ambos son datos muy relevantes en esta sección. Mostraremos la media y la media ponderada, y los resultados en algunos dataset como información adicional.

3.5.1. FastText

Comenzaremos analizando os distintos word embeddings calculados usando FastText de los que disponemos. Estos son FastText calculados en el corpus common crawl y wikipedia, y los entrenados en los corpus de sketch engine. Podemos ver los resultados en la Tabla 3.5. Podemos observar que al pasar del corpus de Wikipedia al corpus Common Crawl y usar un así un corpus casi 40 veces más grande, obtenemos una mejora de rendimiento significativa. En el caso de SketchEngine el rendimiento es inferior, esto se debe a que estos word embeddings cuentan con tan solo 100 dimensiones, por lo que codifican menos información que los anteriores. Esto hace que SketchEngine web, pese a estar entrenado en un corpus ligeramente mayor que FastText wiki obtenga un rendimiento inferior. Con SketEngine también observamos que el uso de un corpus mayor (20 mil millones de tokens vs 100 millones) aumenta el rendimiento de forma significativa.

Embedding	Media	Media Ponderada	SimLex999	MTurk 771	MEN ALL	RG65
FastText CC 600B. Dims: 300	0.675	0.620	0.503	0.762	0.837	0.863
FastText Wiki 16B. Dims. 300	0.630	0.562	0.450	0.710	0.791	0.846
SketchEngine web 20B. Dims: 100	0.610	0.545	0.412	0.714	0.789	0.714
SketchEngine British National 100M. Dims 100	0.563	0.492	0.329	0.661	0.744	0.739

Tabla 3.5: Rendimiento de Embeddings basados en FastText. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

3.5.2. Glove

El segundo word embedding que analizaremos en profundidad será Glove. Para Glove tenemos una gran cantidad de word embeddings calculados usando diferentes datasets. Podemos ver los resultados en la Tabla 3.6.

Ha diferencia de lo que ocurría en FastText, en el caso de Glove al pasar de un corpus de 42 mil millones de tokens a uno de 840 mil millones de tokens, no supone una mejora significativa en el rendimiento, de hecho, en algunos datasets incluso produce el efecto contrario. Dado que el rendimiento obtenido es similar al que obtiene FastText entrenado en un corpus de 16 mil millones de palabras, esto parece indicar que Glove no escala bien a partir de cierto número de palabras, el punto en el que seguir añadiendo textos para el entrenamiento no produce mejora alguna se encuentra mucho antes que en FastText, lo

que comvierte a FastText en un mejor método. Podemos observar como FastText entrenado en el corpus common crawl con 600 mil millones de palabras supera el rendimiento de Glove usando el mismo corpus pero con un total de 840 mil millones de palabras.

Embedding	Media	Media Ponderada	SimLex999	MTurk 771	MEN ALL	RG65
GLOVE CC 840B. Dims: 300	0.629	0.571	0.453	0.736	0.829	0.757
GLOVE CC 42B. Dims 300	0.628	0.553	0.453	0.740	0.814	0.829
GLOVE Wiki 6B. Dims: 300	0.577	0.511	0.408	0.681	0.772	0.779
GLOVE Wiki 6B. Dims: 200	0.551	0.483	0.376	0.656	0.750	0.744
GLOVE Wiki 6B. Dims: 100	0.517	0.452	0.329	0.613	0.718	0.694
GLOVE Wiki 6B. Dims: 50	0.473	0.422	0.292	0.581	0.684	0.603
GLOVE twitter 27B. Dims: 200	0.511	0.449	0.275	0.635	0.743	0.714
GLOVE twitter 27B. Dims: 100	0.458	0.388	0.199	0.603	0.667	0.687
GLOVE twitter 27B. Dims: 50	0.389	0.332	0.156	0.523	0.594	0.608
GLOVE twitter 27B. Dims: 25	0.326	0.267	0.122	0.425	0.487	0.538

Tabla 3.6: Rendimiento de Embeddings basados en GLOVE. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

También podemos observar otros comportamientos interesantes en Glove. Por ejemplo usando el corpus de twitter, se obtiene menos rendimiento que usando el corpus de wikipedia aún contando con casi 5 veces más palabras. Esto indica que un buen corpus tiene una gran relevancia a la hora de entrenar word embeddings. El corpus de Wikipedia está formado por textos formales, mientras que twitter está formado por textos generalmente vulgares. La temática de los textos de wikipedia y los extraídos de twitter seguramente también tenga influencia en el rendimiento que observamos.

Al igual que ocurría con FastText también podemos ver que a medida que reducimos las dimensiones de los vectores, podemos codificar menos información y el rendimiento se reduce. Podemos ver esto representado gráficamente en la Figura 3.1

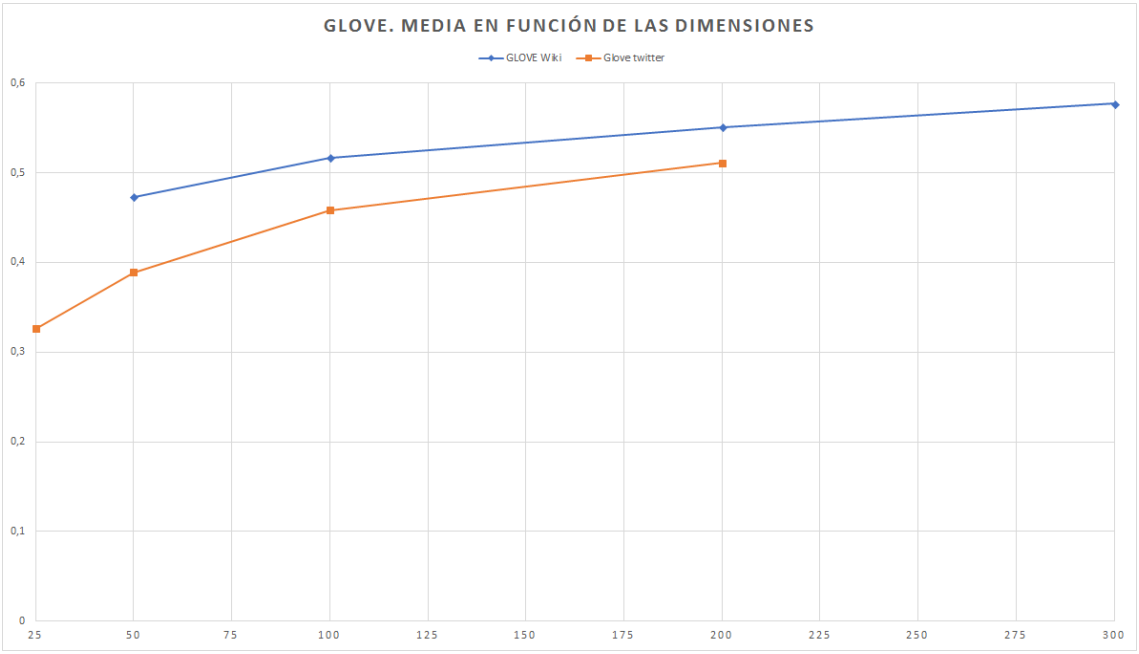


Figura 3.1: Glove. Media obtenida en los datasets en función de las dimensiones de los word embeddings. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta. En Naranja Glove entrenado en el corpus de twitter Azul Glove entrenado en el corpus de wikipedia

3.5.3. LexVec

También disponemos de diferentes versiones de LexVec que podemos comparar. Los resultados se muestran en la Tabla 3.7. En el caso de Glove, de nuevo observamos que el uso de corpus mayores mejora el rendimiento. Sin embargo en este caso resulta más interesante el estudio de las diferentes versiones de LexVec. Podemos ver que los vectores de contexto, que han sido añadidos para mejorar el rendimiento en la tarea de analogía entre palabras, no suponen una mejora en la tarea de la similitud entre palabras. Sin embargo, añadir al igual que FastText información de subpalabras o ngrams, mejora de forma considerable el rendimiento. Esto parece indicar que los ngrams ayudan a mejorar el rendimiento de los métodos de generación de meta embeddings y pueden ser la razón por la que FastText se está posicionando como el mejor método de generación de meta embeddings en este análisis.

Embedding	Media	Media Ponderada	SimLex999	MTurk 771	MEN ALL	RG65
Subword LexVec CC. Dims:300 58B	0.630	0.573	0.477	0.738	0.807	0.744
LexVec CC WordsVectors. Dims:300 58B	0.606	0.543	0.444	0.720	0.795	0.752
LexVec CC WordsVectors + ContexVectors. Dims:300. 58B	0.601	0.534	0.419	0.724	0.809	0.765
LexVec Wiki+NewsCrawls WordsVectors. Dims:300. 7B	0.554	0.506	0.384	0.663	0.751	0.747
LexVec Wiki+NewsCrawls WordsVectors + ContexVectors. Dims:300. 7B	0.559	0.499	0.362	0.666	0.765	0.793

Tabla 3.7: Rendimiento de Embeddings basados en LexVec. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

3.5.4. PDC y HDC

Estudiar el rendimiento de PDC y HDC también es interesante. Estos word embeddings resultan interesantes por que soy muy similares entre ellos, la diferencia principal es que PDC se basa en el modelo CBow y HDC en el modelo Skip gram. Han sido entrenados en el mismo corpus por lo que podemos hacer una comparativa directa entre ambos métodos. Lo que observamos es similar a lo ya estudiado por Mikilov [79]. El modelo cbow obtiene mejor precisión en general, además parece perder menos información a medida que reducimos el número de dimensiones de los word embeddings. El modelo skip-gram funciona mejor con cuando tenemos un corpus pequeño para el entrenamiento, y también representa mejor palabras poco frecuentes, aunque no observamos esto último en el dataset RW, posiblemente por que han sido entrenados con un corpus lo suficientemente grande como para que CBOW supere a Skip-gram. Otra razón que podría influir es que solo responden al 90 % de las palabras del dataset RareWords, por lo que las palabras menos comunes no han sido evaluadas ya que los word embedding no cuentan con representación para ellas.

Embedding	Media	Media Ponderada	SimLex999	MTurk 771	MEN ALL	RG65	RareWords
PDC. Dims: 50	0.538	0.456	0.309	0.614	0.720	0.763	0.447
PDC. Dims: 100	0.573	0.494	0.361	0.663	0.755	0.774	0.475
PDC. Dims 300	0.599	0.529	0.427	0.688	0.773	0.790	0.500
HDC. Dims: 50	0.518	0.433	0.281	0.601	0.708	0.723	0.410
HDC. Dims: 100	0.559	0.465	0.324	0.633	0.738	0.804	0.443
HDC. Dims: 300	0.593	0.515	0.407	0.670	0.760	0.806	0.489

Tabla 3.8: Rendimiento de Embeddings basados en PDC/HDC. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

3.5.5. JointcHYB

Aunque los word embeddings plurilingües no son objeto de estudio en este trabajo, como podemos ver en la Tabla 3.9, resulta llamativo que e rendimiento de JointcHYB varía según el par de idiomas que contiene, incluso aunque todos contengan el inglés. Esto se debe a que la información en WordNet y corpora de textos disponibles no son iguales para todos los idiomas. Si se dispone de menos datos de entrenamiento para un idioma el rendimiento será inferior.

Embedding	Media	Media Ponderada	SimLex999	MTurk 771	MEN ALL	RG65
jointcHYB EN_ES	0.654	0.606	0.541	0.692	0.78	0.863
jointcHYB EN_EU	0.644	0.607	0.541	0.687	0.769	0.836
jointcHYB EN_IT	0.617	0.565	0.467	0.671	0.763	0.821

Tabla 3.9: Rendimiento de Embeddings basados en JointcHYB. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

3.6. Conclusiones

Durante este capítulo hemos realizado un análisis exhaustivo de diferentes word embeddings. Durante el estudio hemos llegado a algunas conclusiones interesantes. En cuanto al rendimiento de los word embeddings, NumberBatch obtiene un rendimiento superior al resto de los word embeddings, esto muestra el potencial de los meta embeddings. Sin contar meta embeddings, FastText parece ser el mejor método para generar word embeddings. En concreto parece ser su forma de tratar las palabras como ngrams lo que le hace obtener un mejor rendimiento que otros métodos. Cuando aplicamos los ngrams a otros modelos como LexVec observamos una mejora en el rendimiento del modelo. Sin embargo, todos los modelos, si son entrenados con corpus lo suficientemente grandes obtienen un buen rendimiento.

También hemos descubierto que los datos usados para el entrenamiento juegan un rol fundamental en el rendimiento del word embeddings. El ejemplo más claro es la diferencia que hemos observado entre los modelos que usan la información extraída de Ramdom Walks sobre WordNet y modelos que usan corpus de textos. Los primeros obtienen un mejor rendimiento en la tarea de la similitud semántica y los segundos obtienen un mejor rendimiento en la relación semántica. También hemos observado que el corpus escogido puede influir en la calidad de los word embeddings, Glove es el caso más claro, donde

podemos observar que usar textos extraídos de wikipedia obtienen un mejor rendimiento que corpus extraídos de twitter.

Por último, usando los resultados obtenidos en esta sección, vamos a seleccionar los mejores word embeddings para usarlos en la generación de meta embeddings en los siguientes capítulos. Estos son:

- FastText calculado en el corpus Common Crawl. A partir de ahora aparecerá nombrado como *FastText* o en caso de que debido a que el texto es demasiado largo sea necesario acortarlo, aparecerá nombrado como *FT*.
- Glove calculado en el corpus Common Crawl con 840 mil millones de tokens y normalizado mediante la normalización L2 por columnas. A partir de ahora aparecerá nombrado como *Glove*
- PDC. 300 dimensiones. A partir de ahora aparecerá nombrado como *PDC*
- JOINTChyb. Escogeremos el embedding bilingüe que combina Inglés y Castellano. A partir de ahora aparecerá nombrado como *JOINTChyb*
- Word2Vec calculado en el corpus de noticias de Google News. A partir de ahora aparecerá nombrado como *W2V*
- UKB calculado sobre WordNet. A partir de ahora aparecerá nombrado como *UKB*
- SketchEngine en el corpus extraído de la web. A partir de ahora aparecerá nombrado como *SKE*
- LexVec calculado en el corpus Common Crawl con información de subpalabras. A partir de ahora aparecerá nombrado como *LexVec*

4. CAPÍTULO

Optimizando Word Embeddings

Nuestro primer intento de mejorar los Word Embeddings analizados en el Capítulo 3, sera la normalización de los mismos. Mediante la normalización buscamos reducir lo que ciertas peculiaridades de los vectores afectan al cálculo del la similitud. Como pueden ser la longitud del vector, las diferencias en el peso de cada característica o columna... Por ello en esta sección estudiaremos el efecto que tienen los métodos de normalización más comunes sobre diferentes word embeddings. El objetivo de este capítulo es determinar para cada word embedding, que método de normalización proporciona mejores resultados en la tarea del cálculo de similitud entre palabras. Es decir, esperamos que mediante la normalización de los vectores de los diferentes word embeddings, consigamos eliminar el posible ruido que puedan contener sus vectores y con ello mejorar el rendimiento de los mismos en el cálculo de la similitud entre palabras.

4.1. Métodos de normalización

En esta sección se describirán los métodos de normalización que se van a aplicar a los word embeddings para su estudio.

- Normalización L_2 de los vectores: La normalización mediante la norma L_2 de los vectores es la más extendida y es altamente recomendada para tareas como la similitud entre palabras y otras tareas que analizan de relación entre palabras. [85]. Esta normalización produce un nuevo vector cuya norma L_2 es igual a uno. Es decir,

produce un nuevo vector cuya dirección es la misma que la del vector original. Y cuya longitud, calculada como la raíz cuadrada de la suma de los cuadrados de cada componente, es igual a uno. Dado un vector "x", la norma L_2 se define matemáticamente de la siguiente forma:

$$|x|_2 = \sqrt{\sum_{k=1}^{n} |x_k|^2}$$

La normalización L_2 provoca que la longitud de los vectores no se tenga en cuenta en el cálculo de la similitud. Mediante este método de normalización conseguimos que la longitud de los vectores no afecte al cálculo de la similitud. Diferentes estudios previos [85] [63], han comprado que en tareas que analizan la relación entre palabras, como el cálculo de la similitud, usar solo la dirección de los vectores e ignorar la longitud produce mejores resultados. Sin embargo, en nuestro caso para el cálculo de la similitud estamos utilizando como medida la similitud coseno. En el cálculo de la similitud coseno la longitud de los vectores es normalizada, por lo tanto la normalización L_2 no tendría efecto alguno. Por ello, para comprobar el efecto de la normalización L_2 vamos a incluir en las pruebas el cálculo de la similitud usando el producto escalar entre los vectores. El producto escalar a diferencia de la similitud coseno toma en cuenta la longitud de los vectores en el cálculo de la similitud entre dos palabras. La similitud coseno es equivalente al producto escalar entre dos vectores que han sido normalizados mediante la norma L_2. Por lo tanto, comparar los resultados obtenidos mediante el producto escalar y la similitud coseno nos permitirá comprobar los efectos de la normalización L_2. Como aclaración para evitar confusiones, todos los resultados de esta sección, salvo los resultados obtenidos mediante el producto escalar se han obtenido mediante la similitud coseno.

Similitud coseno.	Producto escalar.
$cos(\boldsymbol{x},\boldsymbol{y}) = \frac{\boldsymbol{x}\cdot\boldsymbol{y}}{\|\|\boldsymbol{x}\|\|\cdot\|\|\boldsymbol{y}\|\|}$	$\boldsymbol{x}\cdot\boldsymbol{y}$

- L_1 y L_2 por características: En este caso en vez de aplicar la normalización a los vectores o filas de la matriz, se aplica a las características o columnas. Este método de normalización busca aumentar el peso de las características distintivas y reducir el impacto de las características ruidosas. Las características son más distintivas en el cálculo de la similitud coseno cuando tienen unos pocos valores grandes y muchos pequeños. Las normas L_1 y L_2 se definen matemáticamente de la siguiente forma:

Norma L_1.	Norma L_2.
$\|x\|_1 = \sum_{k=1}^{n} \|x_k\|$	$\|x\|_2 = \sqrt{\sum_{k=1}^{n} \|x_k\|^2}$

La normalización L_2 es la más común, y el caso de Glove, los autores recomiendan su uso debido a que se obtienen mejoras de rendimiento significativas [50]. Queremos comprobar si dicha mejora de rendimiento también ocurre cuando aplicamos la normalización L_2 a otros word embeddings. Además, inspirados por "Robert Speer" y "Joshua Chin" [70], que han comprobado que para Glove la normalización mediante la norma L_1 obtiene mejor rendimiento que la normalización L_2, también hemos decidido añadir la normalización L_1 a las comparativas. La normalización L_1 provoca que valores altos tengan un impacto menor en la norma que usando la normalización L_2.

- Centrado de variables: Este método de normalización consiste en por cada variable o columna, calcular su media y restársela. Por lo tanto la media de cada variable tendrá un valor nulo, con ello conseguimos que todas las variables tengan la misma dispersión y media. Además conseguimos que las variables sean independientes de la escala en la que se encuentren los word embeddings. De esta forma evitamos que ciertas variables puedan tener un peso más alto en el cálculo de la similitud por encontrarse en una escala mayor.

- PPA [51]: Este método de normalización es una ampliación del centrado de variables. Se trata de un algoritmo de post-procesado. Word embeddings como Word2Vec, Glove... cuentan con un vector media muy alto. Cuando restamos la media obtenemos alrededor de 8 dimensiones dominantes que influencian las representaciones de palabras en la misma dirección. Eliminar estas direcciones dominantes resulta en mejores representaciones de palabras. Para el cálculo de las dimensiones dominantes se utiliza el algoritmo de Análisis de componentes principales (PCA). PCA utiliza una transformación ortogonal para convertir un set de variables relacionadas en un set de variables no correlacionadas, llamados componentes principales. Esta transformación esta definida de forma que los componentes principales se ordenan en función de la varianza original que describen. PCA generalmente se utiliza para conseguir una reducción de dimensionalidad de un conjunto de datos, pero en nuestro caso, lo usaremos para calcular los componentes principales, es decir, si tenemos una matriz de 300 dimensiones la salida será una nueva matriz de 300 dimensiones. A continuación se muestra el algoritmo en pseudocódigo:

```
Algoritmo de Post-Procesado PPA(X,D)
\\X: Matriz que representa todas las palabras del word embedding
\\D: Número de direcciones dominantes a eliminar (por defecto 7).

    1. Restar la media:
        X = X - Media(X)
    2. Calcular mediante PCA los componentes dominantes:
        u_i = PCA(X), donde i = 1,2,...D
    3. Eliminar los D componentes mayores ∀v in X:
        v = v - ∑_{i=1}^{D} (u_i^T * v) u_i

    SALIDA: Matriz X post-procesada
```

En nuestro caso hemos utilizado el parámetro D = 7, es decir se han restado los 7 componentes principales, ya que es el valor usado por los autores del algoritmo. En la Figura 4.1 podemos observar la diferencia entre la varianza de los 20 componentes principales de GLOVE antes y después de aplicar el algoritmo PPA. Como se puede observar, tras aplicar PPA con D = 7, el embedding no está influenciado por un pequeño número de dimensiones dominantes.

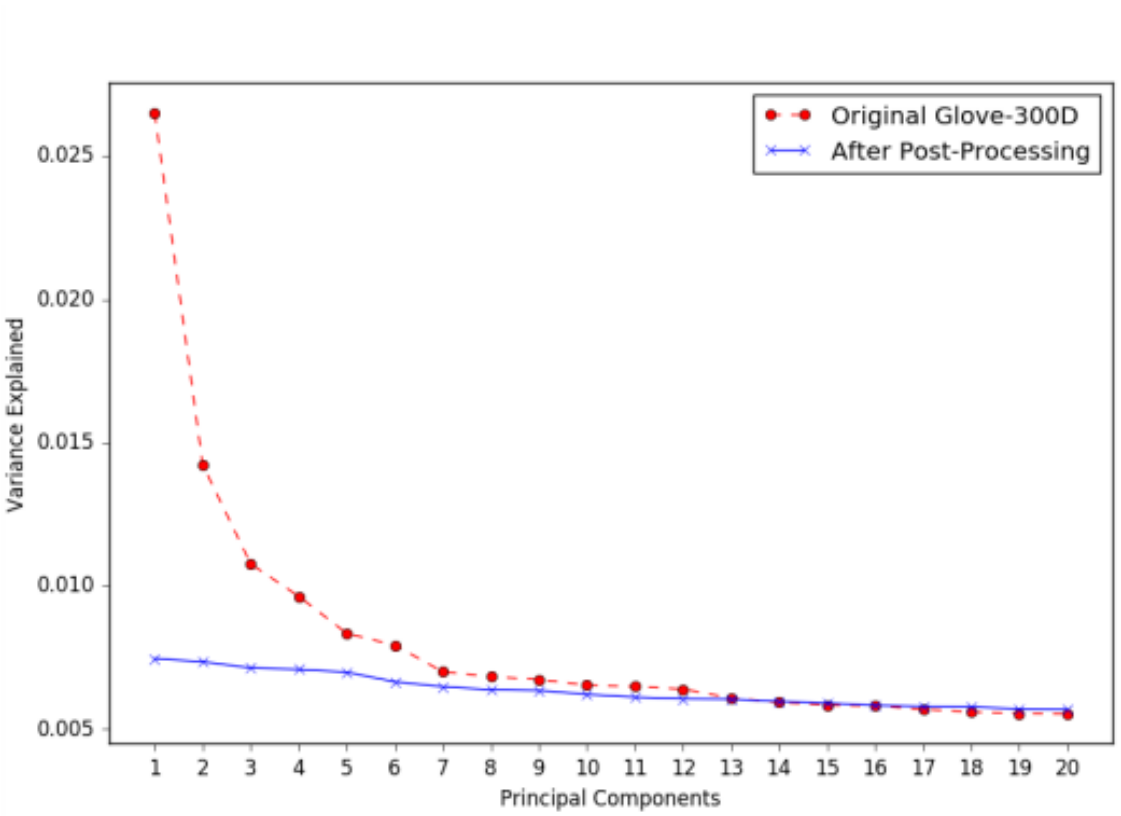


Figura 4.1: En rojo: Varianza de los 20 componentes principales de GLOVE. En Azul varianza de los 20 componentes principales de GlOVE tras aplicar PPA (D=7).

4.2. Resultados obtenidos

En esta sección discutiremos los resultados obtenidos por los diferentes métodos de normalización. Dividiremos la sección en dos partes. Primero analizaremos los resultados de aplicar los diferentes métodos de normalización a cada uno de los word embeddings, de esta forma podremos analizar el efecto que tiene cada método de normalización de forma individual en cada word embedding. En la segunda parte analizaremos si mediante la concatenación de métodos de normalización podemos conseguir un aumento de rendimiento mayor. La metodología para las pruebas se encuentra detallada en la Sección 3.1. Las tablas que se mostraran en esta sección contarán con cuatro columnas. Para facilitar futuras comparaciones se mostrará la media y la media ponderada obtenida en los diferentes dataset, además se mostrará la media y la media ponderada relativa al mismo word embedding sin aplicarle ningún tipo de normalización y usando la similitud coseno como medida de similitud entre palabras. De esta forma podremos ver si el método o la combinación de métodos de normalización aplicados mejora el rendimiento del word embedding original. Los métodos de normalización no afectan al vocabulario disponible, por lo tanto en esta sección no tiene sentido hacer uso de las dos métricas descritas en la Sección 3.1. Haremos uso tan solo de la primera métrica donde se descartan los pares de palabras para los que un determinado word embedding no puede dar respuesta, puesto que no cuenta con representación para una o ambas palabras del par. Por lo tanto esta métrica no tiene en cuenta que un word embedding no sea capaz de dar respuesta para un determinado número de pares de palabras. Nos interesa conocer que efecto tienen los diferentes métodos de normalización sobre los word embeddings, no nos interesa en este caso el número de pares de palabras para los que pueden dar respuesta.

4.2.1. Análisis de los diferentes métodos de normalización

En esta sección analizaremos el efecto que tiene cada uno de los métodos de normalización en los diferentes word embedding. Buscamos determinar para cada word embedding o para cada tipo de word embedding (por ejemplo, word embedding calculados a partir de corpus de texto y word embedding calculados a partir de grafos), que método o métodos de normalización nos proporcionan un aumento de rendimiento mayor. Para ello por cada método de normalización vamos a comprobar el rendimiento obtenido por cada uno de los word embeddings.

Normalización de la longitud de los vectores

Comenzaremos las pruebas analizando el efecto que tiene la normalización mediante la norma L_2 de los vectores. Es decir, comprobaremos si es mejor tener en cuenta la longitud de los vectores o no en la tarea de similitud entre palabras. Como se ha explicado anteriormente para ello comparemos el rendimiento de los word embeddings usando como medida de similitud el producto escalar entre los dos vectores que representan las palabras del par, y la similitud coseno, que es equivalente a realizar el producto escalar entre dos vectores normalizados mediante la norma L_2. En la Tabla 4.1 se muestran los resultados de varios word embeddings usando como medida de similitud el producto escalar. Se muestra la media y la media ponderada obtenida en los diferentes dataset para facilitar futuras comparaciones. Y además, se muestra el rendimiento de forma relativa a usar el mismo word embedding usando como medida de similitud la similitud coseno.

Embedding	Diferencia (Media)	Diferencia (Media ponderada)	Media	Media Ponderada
Word2Vec	-0.027	-0.028	0.589	0.528
UKB	-0.030	-0.020	0.599	0.590
GLOVE	-0.002	-0.001	0.595	0.525
FastText	-0.053	-0.053	0.622	0.568
Lexvec	-0.023	-0.016	0.607	0.557
jointcHYB	-0.012	-0.018	0.642	0.588
SKE	-0.052	-0.054	0.558	0.491

Tabla 4.1: Rendimiento de diferentes word embeddings usando como medida de similitud el Producto escalar de forma relativa a usar como medida la similitud coseno. Métrica utilizada: Ignorar pares para los que no se puede dar respuesta.

Como se puede observar en la Tabla 4.1 usar como medida la similitud coseno, es decir, ignorar la longitud de los vectores, resulta significativamente mejor que usar el producto escalar. Esto indica que para la tarea de la similitud entre palabras, la información relevante es el coseno del ángulo que forman los dos vectores que representan las palabras para las que queremos calcular la similitud. Por lo tanto, para la tarea de similitud entre palabras, es altamente recomendable utilizar la similitud coseno como medida de similitud. Otra alternativa es realizar una normalización mediante la norma L_2 de los vectores, haciendo el producto escalar equivalente a la similitud coseno.

Normalización L_1 y L_2 por características

En esta sección analizaremos el efecto que tiene la normalización L_1 y L_2 sobre diferentes word embeddings. Como se ha explicado anteriormente ambos métodos buscan aumentar el peso de las características más distintivas reduciendo así el impacto de las características ruidosas. Mediante ambos métodos de normalización conseguimos que los vectores estén formados por un número reducido de valores grandes y un gran número de valores pequeños haciendo que las características más distintivas tengan un peso mayor en el cálculo de similitud coseno. La diferencia entre el método de normalización mediante la norma L_1 y L_2 recae en que la normalización L_1 provoca que valores altos tengan un impacto menor en la norma que usando la normalización L_2. En la Tabla 4.2, se muestran los resultados obtenidos usando ambas normalizaciones. Para facilitar futuras comparaciones se muestran la media y la media ponderada obtenidas en los diferentes dataset. Además se muestra el rendimiento de forma relativa al rendimiento de los mismos word embeddings sin aplicarles ningún tipo de normalización.

Como se puede observar en la Tabla 4.2 ambos métodos de normalización aportan una mejora de rendimiento muy significativa cuando los aplicamos a Glove. Para Glove la normalización L_1 resulta ligeramente mejor que la normalización L_2. En cambio Glove es el único word embedding que muestra este comportamiento. El resto de word embeddings apenas ven su rendimiento afectado por la normalización. Por lo tanto, ambos métodos de normalización resultan muy recomendables para el embedding Glove, ya que consiguen una mejora de rendimiento significativa. Pero no resultan útiles para el resto de word embeddings, donde en algunos casos llegamos a perder rendimiento.

Embedding	Preprocessing	Diferencia (Media)	Diferencia (Media ponderada)	Media	Media Ponderada
Word2Vec	L1 por características	0.000	0.000	0.616	0.556
Word2Vec	L2 por características	0.000	0.000	0.616	0.556
UKB	L1 por características	0.000	0.000	0.630	0.610
UKB	L2 por características	0.001	0.000	0.631	0.611
GLOVE	L1 por características	0.034	0.047	0.631	0.573
GLOVE	L2 por características	0.032	0.045	0.629	0.571
FastText	L1 por características	0.008	0.005	0.683	0.625
FastText	L2 por características	0.008	0.005	0.683	0.625
Lexvec	L1 por características	0.001	0.000	0.631	0.572
Lexvec	L2 por características	0.001	0.000	0.630	0.572
jointcHYB	L1 por características	-0.001	0.000	0.653	0.606
jointcHYB	L2 por características	-0.001	0.000	0.652	0.606
SKE	L1 por características	-0.003	0.000	0.607	0.545
SKE	L2 por características	-0.002	0.000	0.607	0.545

Tabla 4.2: Rendimiento de diferentes word embeddings normalizados mediante la normalización L_1 y L_2 por características de forma relativa a los mismos word embeddings sin normalizar. Métrica utilizada: Ignorar pares para los que no se puede dar respuesta.

Centrado de variables

En esta sección analizaremos el rendimiento del método de normalización de centrado de variables. Como se ha explicado anteriormente este método consiste en restar a cada variable o columna su media. Con esto conseguimos que todas las características tengan la misma dispersión y media, haciendo a la variable independiente de la escala en la que se encuentre. En la Tabla 4.3 se muestran los resultados aplicando el centrado de variables a diferentes word embeddings. Para facilitar futuras comparaciones se muestran la media y la media ponderada obtenidas en los diferentes dataset. Además se muestra el rendimiento de forma relativa al rendimiento de los mismos word embeddings sin aplicarles ningún tipo de normalización.

Embedding	Diferencia (Media)	Diferencia (Media ponderada)	Media	Media Ponderada
Word2Vec	-0.001	0.001	0.615	0.557
UKB	0.006	0.010	0.636	0.620
GLOVE	-0.046	-0.058	0.551	0.468
FastText	-0.004	-0.012	0.671	0.608
Lexvec	-0.020	-0.029	0.610	0.544
jointcHYB	0.000	-0.001	0.654	0.605
SKE	-0.002	0.002	0.608	0.547

Tabla 4.3: Rendimiento de diferentes word embeddings normalizados mediante el centrado de variables de forma relativa a los mismos word embeddings sin normalizar. Métrica utilizada: Ignorar pares para los que no se puede dar respuesta.

Como se puede ver en la Tabla 4.3 el centrado de variables no resulta adecuado para la tarea de calcular la similitud entre palabras. Tan solo en el caso de UKB nos proporciona una mejora de rendimiento, la cual no es significativa. Por otro lado en el caso de Lexvec y Glove sufrimos una pérdida de rendimiento significativa. En los word embeddings restantes el efecto de la normalización es insignificante. Por lo tanto, este método de normalización no es un método recomendable al menos para la tarea de similitud entre palabras. El centrado de variables provoca que todas las variables se encuentren en la misma escala, por lo tanto la pérdida de rendimiento que observamos podría ser causada por que estamos dando mayor peso a las variables ruidosas reduciendo el peso de las variables más distintivas. El embeddings que más rendimiento pierde es Glove, el cual era el más beneficiado por las normalizaciones L_1 y L_2. El centrado de variables parece estar teniendo el efecto contrario a dichas normalizaciones. Como mencionan Jiaqi Mu

y Pramod Viswanath [51], el centrado de variables provoca que se obtengan un pequeño número de direcciones dominantes que influencian las representaciones en la misma dirección. Provocando que perdamos información y el cálculo de la similitud entre palabras sea menos preciso.

PPA

Este método se trata de un algoritmo que busca solucionar los problemas del centrado de variables que hemos encontrado en la sección anterior. Cuando restamos la media a cada variable obtenemos un pequeño número de dimensiones dominantes que influencias las representaciones de palabras en una misma dirección. En esta sección probaremos si eliminar estas direcciones dominantes hace que el rendimiento de los word embeddings aumente. El algoritmo usado ha sido descrito anteriormente. En la Tabla 4.4 se muestran los resultados obtenidos aplicando el algoritmo PPA a diferentes word embeddings. Para facilitar futuras comparaciones se muestran la media y la media ponderada obtenidas en los diferentes dataset. Además se muestra el rendimiento de forma relativa al rendimiento de los mismos word embeddings sin aplicarles ningún tipo de normalización.

Embedding	Diferencia (Media)	Diferencia (Media ponderada)	Media	Media Ponderada
Word2Vec	0.002	0.006	0.618	0.562
UKB	-0.022	-0.020	0.608	0.591
GLOVE	0.018	0.029	0.615	0.555
FastText	0.006	0.009	0.681	0.630
Lexvec	0.006	0.004	0.635	0.576
jointcHYB	0.010	0.005	0.663	0.610
SKE	0.012	0.016	0.622	0.561

Tabla 4.4: Rendimiento de diferentes word embeddings normalizados mediante el algoritmo PPA de forma relativa a los mismos word embeddings sin normalizar. Métrica utilizada: Ignorar pares para los que no se puede dar respuesta.

Como se puede observar en la Tabla 4.4, eliminar las direcciones dominantes obtenidas en el centrado de variables supone un aumento de rendimiento significativo respecto a no hacerlo. En la sección anterior veíamos como el centrado de variables no tenía efecto alguno en la mayoría de word embeddings o incluso reducía de forma significativa el rendimiento. Mientras que este método salvo en el caso de UKB consigue aumentar el rendimiento. Glove resulta el principal beneficiado (aunque el beneficio es inferior al

obtenido por las normalizaciones L_1 y L_2). En el resto de word embeddings existe una mejora de rendimiento, aunque no demasiado significativa. UKB, único word embedding cuyo rendimiento mejoraba con el centrado de variables ve su rendimiento reducido significativamente con este método. Por lo tanto, no podemos decir que PPA es un mejor método que el centrado de variables en todos los casos, depende de las características de cada word embedding.

Sumario

Durante esta sección hemos comprobado que efecto tienen diferentes métodos de normalización sobre varios word embeddings. A modo de sumario a continuación se muestra una tabla, donde se ha calculado por cada método de normalización la media de la ganancia o pérdida de rendimiento que han word embeddings. Es decir, para cada método de normalización se mostrará la media de la diferencia de rendimiento entre los word embeddings normalizados y los word embedding sin normalizar.

Normalización	Media	Media Ponderada
Producto escalar	-0.028	-0.027
L1 por características	0.006	0.007
L2 por características	0.005	0.007
Centrado de variables	-0.01	-0.012
PPA	0.005	0.007

Tabla 4.5: Media de la diferencia de rendimiento entre los word embeddings normalizados y los word embeddings originales. Métrica utilizada: Ignorar pares para los que no se puede dar respuesta.

La Tabla 4.5 muestra las conclusiones generales a las que podemos llegar en esta sección. Para la tarea del cálculo de la similitud entre palabras la normalización de la longitud de los vectores, ya sea aplicando una normalización L2 a los vectores o usando como medida la similitud coseno, resulta significativamente mejor que utilizar la longitud de los vectores en el cálculo de la similitud. Por lo tanto, recomendamos el uso de la similitud coseno como medida en el cálculo de la similitud.
Respecto al resto de normalizaciones, solo Glove sufre una mejora de rendimiento significativa al aplicarle una normalización L_1 o L_2 a sus variables o columnas. Glove parece

ser un método que produce un alto número de variables ruidosas, por lo tanto se beneficia de forma importante de este tipo de normalización. En caso de usar Glove recomendamos que sus variables sean mediante una normalización L_2, la más común en los diferentes estudios publicados, o incluso usemos la norma L_1 que muestra un rendimiento ligeramente superior.
En el caso del resto de word embeddings y métodos de normalización no encontramos mejores de rendimiento especialmente significativas. Además, no existe un método de normalización que mejore en todos los casos el rendimiento de los word embeddings. El método a aplicar depende de las características de cada word embedding, por ejemplo, el centrado de variables consigue mejorar el rendimiento de UKB pero no resulta un buen método para el resto de word embeddings, mientras que PPA resulta un buen método de normalización para todos los word embeddings probados excepto para UKB. La Tabla 4.5 también refleja esto mismo. Ningún método de normalización, salvo usar el producto escalar, cuenta con valores altos en la tabla. Esto quiere decir, que existen word embeddings para los que el método obtiene buenos resultados y word embeddings para los que no. Por lo tanto, si queremos tratar de mejorar el rendimiento de nuestros word embeddings tendremos que estudiar las características de los word embeddings o probar diferentes métodos de normalización y analizar el rendimiento obtenido como hemos hecho aquí. Salvo Glove ningún otro word embedding se ha beneficiado de forma significativa de los métodos de normalización, por lo tanto, el tiempo requerido para encontrar una buena normalización para nuestros word embeddings no compensa el aumento de rendimiento que vamos a obtener.

4.2.2. Concatenación de métodos de normalización

En la sección anterior evaluamos el efecto que tenía cada método de normalización sobre diferentes word embeddings. En esta sección analizaremos que ocurre cuando aplicamos una concatenación a diferentes word embeddings una concatenación de diferentes métodos de normalización. Dichas concatenaciones se representarán de la siguiente forma en las tablas:

Método_1 + Método_2 + Método_3 + ...

Esto quiere decir que se ha aplicado el Método_1 al word embedding, al resultado obtenido por dicho método de normalización se le ha aplicado el Método_2, al resultado de dicho método se le ha aplicado el Método_3,... y los resultados mostrados en las tablas

son los resultados obtenidos por el word embedding tras haber aplicado todos los métodos de normalización.

En la sección anterior hemos observado que salvo en el caso de Glove al aplicarle la normalización L_1 o L_2 a sus variables, ningún word embedding muestra una mejora de rendimiento significativa gracias a las normalizaciones. Por lo tanto es de esperar que la concatenación de diferentes métodos de normalización tampoco tenga un efecto demasiado significativo. Como la mayor parte de concatenaciones de métodos de normalización tienen un impacto insignificante o incluso repercuten negativamente en el rendimiento de los word embeddings, por lo tanto, no cuentan con apenas relevancia, en la Tabla 4.6 mostraremos para cada word embedding la mejor combinación de métodos de normalización.

Embedding	Método de normalización	Diferencia (Media)	Diferencia (Media ponderada)	Media	Media Ponderada
Word2Vec	Centrado de variables + PPA	0.002	0.007	0.618	0.563
UKB	L2 por columnas + Centrado de variables	0.006	0.007	0.636	0.618
GLOVE	L2 por características + PPA	0.033	0.051	0.630	0.577
FastText	Centrado de variables + PPA	0.015	0.011	0.690	0.631
Lexvec	L2 vectores + PPA	0.007	0.004	0.637	0.577
jointcHYB	Centrado de variables + PPA	0.011	0.005	0.665	0.611
SKE	L2 vectores + PPA	0.014	0.018	0.624	0.563

Tabla 4.6: Diferencia de rendimiento entre la mejor concatenación de métodos de normalización para cada word embedding y el rendimiento del word embedding principal. Métrica utilizada: Ignorar pares para los que no se puede dar respuesta.

En la Tabla 4.6 podemos observar algo similar a lo que ocurría en la sección anterior.

Tan solo Glove muestra una mejora de rendimiento significativa gracias a los métodos de normalización. Sin embargo, el rendimiento obtenido por la combinación de la normalización L_2 y PPA obtiene un rendimiento muy similar al rendimiento que obtenían las normalizaciones L_1 o L_2. Por lo tanto la concatenación de normalizaciones no supone ninguna mejora. En el caso del resto de word embeddings la mejora obtenida es bastante reducida. En el caso de FastText y SketchEngine obtenemos una pequeña mejora de rendimiento. En el resto de casos la mejora no es significativa.
Con todos los word embeddings la concatenación de métodos de normalización obtiene un rendimiento muy similar a la aplicación de un solo método de normalización. La mejor concatenación se consigue en todos los casos, concatenando el método que había obtenido un mejor rendimiento al aplicarlo de forma independiente con un segundo método, el cual tiene un impacto mínimo en el rendimiento. Por lo tanto la aplicación de más de un método de normalización resulta muy poco atractiva. La mejora de rendimiento que vamos a obtener va a ser insignificante.

A modo de sumario, al igual que en la sección anterior, mostraremos una tabla , donde se ha calculado por cada concatenación de métodos de normalización la media de la ganancia o pérdida de rendimiento que han sufrido los word embeddings. Es decir, para cada método de normalización se mostrará la media de la diferencia de rendimiento entre los word embeddings normalizados y los word embedding sin normalizar.

Como podemos ver en la Tabla 4.7, ocurre lo mismo que en la sección anterior. Ninguna concatenación de métodos de normalización muestra valores altos. Esto quiere decir que no existe una concatenación que afecta de la misma forma a todos los word embeddings. Es decir, una misma concatenación de métodos de normalización puede afectar positivamente al rendimiento de un word embedding y negativamente al rendimiento de otro word embedding. Por lo tanto no existe una combinación que podamos aplicar a todos nuestros word embeddings para mejorar su rendimiento, si no que tendremos que mediante experimentación encontrar la mejor combinación.

Durante esta sección hemos analizado el rendimiento que aporta la aplicación de una concatenación de métodos de normalización a diferentes word embeddings. Ninguna de las concatenaciones muestra una mejora de rendimiento significativa respecto a la aplicación de un solo método de normalización. Esto, sumado a que necesitamos comprobar embedding por embedding cual es la mejor concatenación de normalizaciones, ya que una misma concatenación puede tener efectos diferentes sobre cada word embedding, provoca

Método de normalización	Diferencia (Media)	Diferencia (Media ponderada)
L2 por vectores + Centrado de variables	-0.003	-0.005
L2 vectores + PPA	0.005	0.008
L2 por vectores + L2 por características	0.006	0.007
L2 por columnas + Centrado de variables	0.002	0.001
L2 por características + PPA	0.006	0.010
Centrado de variables + PPA	0.002	0.002
Centrado de variables + L2 por características	-0.005	-0.008
L2 por vectores + L2 por características + PPA	0.005	0.011
PPA + Centrado de variables	0.002	0.002
PPA + L2 por caracteristicas	0.004	0.008
PPA + L2 pro vectores + L2 por caracteristicas	0.005	0.008
Centrado de variables + L2 por vectores + L2 por caracteristicas	-0.004	-0.007
L2 por características +L2 vectores + Centrado de variables	0.006	0.006

Tabla 4.7: Media de la diferencia de rendimiento entre los word embeddings normalizados y los word embeddings originales. Métrica utilizada: Ignorar pares para los que no se puede dar respuesta.

que las concatenaciones de métodos de normalización sean muy poco atractivas. Resulta más interesante centrarse en encontrar un método de normalización que ofrezca buenos resultados para los word embeddings que queremos utilizar. La ganancia de rendimiento que obtendremos concatenando diferentes métodos de normalización va a ser muy reducida o incluso inexistente.

4.3. Conclusiones

Durante este capitulo hemos analizado el rendimiento de distintos métodos de normalización aplicados a diferentes word embeddings. Se han mostrado diferentes tablas para mostrar el rendimiento de cada word embedding, sin embargo, para facilitar la legibilidad de las mismas solo se han mostrado los resultados que consideramos más relevantes y estos se han mostrado parcialmente, es decir, se ha omitido el resultado obtenido en cada uno de los dataset, ya que hemos considerado que en esta sección lo más relevante era el rendimiento general del word embedding y no el rendimiento en un determinado dataset. Sin embargo, en caso de que se quieran consultar, los resultados completos de este análisis se pueden encontrar en la plataforma GitHub [1], y también la tabla Tabla 10.4 en el anexo de este documento. Del análisis realizado durante este capítulo podemos extraer varias conclusiones. La primera y más clara es que para la tarea del cálculo de la similitud entre palabras no tener en cuenta la longitud de los vectores es claramente la mejor opción. Lo realmente importante a la hora de calcular la similitud entre dos palabras es el ángulo que forman los vectores que representan a ambas palabras. Tenemos dos alternativas para evitar que la longitud de los vectores afecte al cálculo, la primera es utilizar como medida de similitud la similitud coseno, y la segunda normalizar mediante la norma L_2 los vectores, haciendo el producto escalar equivalente a la similitud coseno.

En cuanto al resto de métodos de normalización hemos podido observar que en general, los word embeddings no ven su rendimiento mejorado de forma significativa al ser normalizar, e incluso en algunos casos, su rendimiento puede disminuir. El único word embedding que se beneficia de forma importante de las normalizaciones es Glove. Glove es un word embedding que acumula mucho ruido en sus variables. Por lo tanto, tanto la normalización L_1 como al normalización L_2 mejoran el word embedding de forma significativa. En caso de utilizar Glove para tareas donde se busque analizar la relación entre palabras, como el cálculo de la similitud, recomendados que primero se realice una normalización L_1 o L_2 de sus vectores.

[1] https://github.com/ikergarcia1996/RotEmbeddings

En el caso del resto de word embedding, no existe un método de normalización que nos asegure una mejora de rendimiento en todos los casos. Depende de las características del word embedding. Un mismo método de normalización puede afectar de forma positiva al rendimiento de un word embedding y de forma negativa al rendimiento de otro word embedding. Por lo tanto escoger un método de normalización para un determinado word embedding no es una tarea sencilla. Dado que en la mayoría de casos la mejora de rendimiento, si es que existe, va a ser poco significativa, para la tarea del cálculo de la similitud, los métodos de normalización resultan poco atractivos. Las concatenaciones de métodos de normalización resultan aún menos atractivas, ya que escocer una buena concatenación es todavía más complejo y la mejora respecto a aplicar un solo método es mínima.

5. CAPÍTULO

Combinando Word Embeddings

En este capítulo analizaremos dos métodos para la combinación de diferentes word embeddings: la aplicación en cascada, y la media de la similitud. El objetivo de ambos métodos es combinar diferentes fuentes de conocimiento, con ello esperamos conseguir ampliar el vocabulario disponible manteniendo el rendimiento de los word embeddings o incluso mejorándolo. Es importante resaltar que ambos métodos nos permiten combinar el conocimiento de diferentes word embeddings, pero, no generan un nuevo embedding. Es decir, no son métodos de generación de meta-embeddings.

5.1. Aplicación en Cascada de Word Embeddings

En esta sección se describirá el método de aplicación en cascada de word embeddings y se discutirán los resultados obtenidos. Este método se basa en utilizar un word embedding como principal. Nuestro objetivo es ampliar el conocimiento disponible, por lo que para ello utilizaremos una serie de fuentes de conocimiento adicionales, en este caso otros word embeddings. Usaremos estas fuentes de conocimiento en caso del que el word embedding principal no disponga de representación para una o ambas palabras del par de palabras para el que queremos calcular la similitud. Con este método de combinación esperamos conseguir aumentar el vocabulario disponible. De esta forma podremos dar respuesta a un número mayor de pares de palabras. Esperamos poder conseguir esto manteniendo el rendimiento de los word embeddings y esperamos que este método sea especialmente útil en word embeddings que cuentan con un vocabulario poco extenso.

5.1.1. Explicación del método

En esta sección se describirá en profundidad el método de aplicación en cascada. Comenzamos escogiendo un word embedding principal, este será el word embedding que usaremos para calcular la similitud entre pares de palabras, siempre y cuando disponga de representación para ambas palabras. A continuación se escoge uno o varios word embeddings secundarios. El procedimiento a seguir es el siguiente, para cada par de palabras buscamos en el word embedding principal la representación para ambas palabras del par. Si disponemos de representación para ambas palabras, damos como resultado la similitud coseno entre ambas. En caso de que no se disponga de representación para una o ambas palabras, comprobamos si disponemos de representación para ambas en el primer word embedding secundario, si disponemos de representación para ambas palabras damos como resultado la similitud coseno entre ambas. En caso contrario pasamos a utilizar el segundo word embedding secundario. Repetiremos este proceso hasta poder dar una respuesta para el par de palabras o hasta que no dispongamos de más word embeddings, en tal caso ese par de palabras será descartado ya que no es posible dar una respuesta.

Es importante resaltar que siempre buscamos las dos palabras del par en un mismo word embedding. Es decir, si se da el caso de que un word embedding cuenta con representación para solo una de las palabras, no se busca la palabra para la que no contamos con representación en el siguiente word embedding, si no que se buscan ambas. Siempre se calcula la similitud entre pares de palabras extraídas de un mismo word embedding. Las palabras que se encuentran en diferentes word embeddings pueden encontrarse en espacios vectoriales diferentes. Si calculamos la similitud coseno entre representaciones de palabras extraídas de diferentes word embeddings el resultado será un valor aleatorio que dependerá de como sean los espacios vectoriales de cada word embedding. En la siguiente imagen se muestra de forma simplificada esta problemática. Como se puede observar en ambos word embeddings las palabras “Coche” y “Automóvil” cuentan con representaciones similares. En cambio, debido a que ambos word embeddings se encuentran en espacios vectoriales diferentes, si comparamos la palabra “Coche” del primer word embedding y la palabra “Automóvil” del segundo word embedding el resultado que obtenemos es que ambas palabras no son similares.

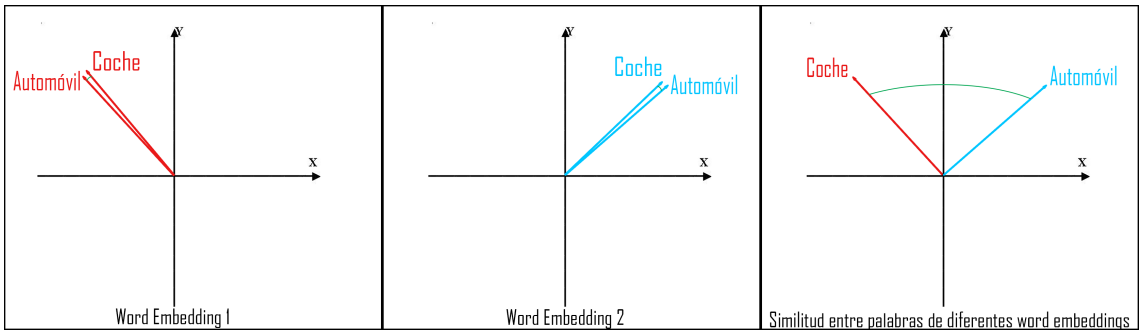


Figura 5.1: Demostración de la problemática al calcular la similitud entre representaciones de palabras extraídas de diferentes word embeddings

A continuación se describe el algoritmo en pseudocódigo:

```
1
2   //La función cascada da como salida la similitud coseno entre ambas palabras en caso de que haya
        podido ser calculada o el valor nulo en caso de que ningún word embedding proporcionado
        cuente con representación para las dos palabras del par.
3
4   función cascada(embedding_principal, embeddings_secundarios, Par):
5   //Entrada:
6       //Embedding_principal: Primer word embedding en el que buscaremos las palabras
7       //Embedding_secundarios: Lista de word embeddings que se usarán en caso de que el principal
        no pueda dar respuesta al par de palabras. Se aplicarán según el orden de la lista, primero
        el situado en la primera posición de la lista, segundo el situado en la segunda...
8       //Par: Par de palabras formado por las dos palabras W_1 y W_2 para la que queremos calcular
        la similitud coseno.
9   //Salida: Similitud coseno entre W_1 y W_2
10
11  //Funciones:
12      //Buscar(P,Emb). Devuelve el vector que representa a la palabra P en el embedding Emb. Si el
        embedding no cuenta con una representación para P, devuelve el valor nulo.
13      //Similitud_Coseno(V1,V2). Devuelve la similitud coseno entre los vectores V1 y V2. En caso
        de que V1 o V2 sean el valor nulo, devuelve el valor nulo.
14
15  similitud = Similitud_Coseno(
16              Buscar(W_1, Embedding_Pricipal),
17              Buscar(W_2, Embedding_Principal))
18  Si similitud es nulo: \\W_1 o W_2 no existen.
19      Para cada embedding de lista embedidng_secundarios:
20          similitud = Similitud_Coseno(
21              Buscar(W_1, embedding),
22              Buscar(W_2, embedding))
23
24          Si similitud no es nulo:
25              Salir del bucle
26
27  Devolver similitud
```

Si queremos obtener el mejor rendimiento posible con este método, debemos aplicar nuestros word embeddings en un orden basado en su rendimiento. Es decir, usaremos como embedding principal nuestro mejor word embedding, como segundo word embedding nuestro segundo mejor embedding... Aquí resulta muy útil el análisis realizado en el Capítulo 3.

5.1.2. Resultados obtenidos

En esta sección discutiremos los resultados obtenidos usando el método de aplicación en cascada. Para decidir el orden de aplicación de los word embeddings hemos ordenado los word embeddings de mejor a peor usando los datos de la Tabla 10.1. Se ha utilizado la media del resultado en los diferentes dataset como métrica para la ordenación. Los embeddings utilizados para estas pruebas y la notación con la que nos referiremos a ellos son los descritos al final del Capítulo 3.

La notación que vamos a utilizar para describir las diferentes pruebas es la siguiente:

embedding_1 ->embedding_2 ->embedding_3 ->...

Esto quiere decir que se va a utilizar el *embedding_1* para la evaluación. En caso de que el embedding no cuente con representación para una o ambas palabras del par a evaluar, se utilizará el *embedding_2* para la evaluación. En caso de que este tampoco cuente con representación para una o ambas palabras, se utilizará el *embedding_3*. Realizaremos este proceso sucesivamente hasta que no dispongamos de más word embeddings, por lo tanto no podremos ofrecer una respuesta para ese par de palabras, o uno de los embeddings cuente con representación para ambas palabras del par.

La metodología para las pruebas está detalla en la Sección 3.1. Las tablas con los resultados que analizaremos en esta sección cuentan con 2 columnas donde se muestra la diferencia entre la media y la media ponderada de la combinación de word embeddings y la media y media ponderada del mejor word embedding que se incluye en la combinación, es decir el que se ha usado como principal. De esta forma podemos ver de forma sencilla si la combinación de word embeddings mejora el rendimiento de los embeddings originales.

Vamos a realizar una primera prueba usando los 6 mejores embeddings que tenemos disponibles. En este caso, FastText entrenado usando el corpus common crawl [2], embed-

Embedding	RW	RW cobertura	RW * Cobertura
FastText->jointcHYB ->UKB ->Subword ->GLOVE ->W2V ->SketchEngine ->PDC	0.582	1	0.582
FastText	0.595	0.981	0.584

Tabla 5.1: Comparación del método de aplicación en cascada usando los 8 mejores embeddings de los que disponemos usando FastText como principal y FastText.)

ding que estamos usando como principal, puede dar respuesta a todos los pares de palabras de todos los dataset, excepto en el caso del dataset Rare Words [59]. Por lo tanto será el único dataset en el que veremos modificaciones en el resultado.

Como podemos ver en la Tabla 5.1, gracias a este método hemos conseguido una cobertura del 100 % en el dataset Rare Words [59]. Sin embargo no hemos conseguido una mejora del rendimiento del embedding. En la comparativa con FastText usando la primera métrica descrita en la Sección 3.1 en la que se descartan los pares de palabras para los que no podemos dar respuesta, el método de aplicación en cascada ofrece un rendimiento significativamente inferior. Esto ocurre por que esta métrica no penaliza el que no se de respuesta para un determinado número de pares de palabras. En el caso del método de aplicación en cascada, estamos usando embeddings que cuentan con un rendimiento inferior cuando no podemos dar respuesta para un par de palabras, por lo tanto estamos penalizando el rendimiento del word embedding principal. Esta es la motivación principal para usar la segunda métrica descrita en la Sección 3.1. En esta métrica se multiplica la puntuación obtenida en el dataset por el porcentaje de pares para los que el word embedding ha podido dar respuesta. De esta forma, se penaliza el no poder dar respuesta para un número determinado de pares de palabras. Con esta segunda métrica el rendimiento del método de aplicación en cascada y el rendimiento de FastText es bastante similar. Por lo tanto hemos conseguido aumentar el vocabulario para el que somos capaz de dar respuesta. Sin embargo, al menos en el caso de este dataset en particular, FastText es capaz de dar respuesta para un número de pares de palabras lo suficientemente alto como para obtener un rendimiento ligeramente superior.

Esta primera prueba la hemos realizado usando como el word embedding principal FastText calculado usando el corpus common crawl [2]. Este es un word embedding de gran tamaño, que cuenta con un vocabulario muy extenso, de hecho es capaz de obtener de dar respuesta para prácticamente todos los pares de palabras de los dataset. De 11.313

pares de palabras, no puede dar respuesta para tan solo 29 palabras. Por lo tanto, usar este método de combinación con FastText no tiene demasiado sentido. Este método resulta más útil en el caso de word embeddings con un vocabulario limitado, como puede ser UKB. Para comprobar que efecto tiene este método sobre UKB, lo hemos seleccionado como embedding principal y como embeddings secundario hemos usado los 5 mejores word embeddings después de UKB. Es decir, nos hemos puesto en la situación de que FastText y JointcHYB no estuviesen disponibles. Si usásemos word embeddings con mejor rendimiento como secundarios no podríamos saber, en caso de obtener una mejora de rendimiento, si esta mejora es debido al método o debido a usar embeddings con mejor rendimiento que el principal como secundarios.

Embedding	MTurk 287	MTurk 771	WS353 all	VERB 143	RW
UKB	0.8014	0.9987	0.9802	0.4167	0.6888
UKB ->Lexvec ->GLOVE->W2V ->SKE ->PDC	1	1	1	1	1

Tabla 5.2: Comparación del porcentaje de pares de palabras a los que pueden responder UKB y el método de aplicación en cascada usando UKB como embedding principal.

En la Tabla 5.2 se muestra una comparativa del porcentaje de palabras para los que puede dar respuesta el word embedding UKB y la combinación usando el método de aplicación en cascada usando UKB como embedding principal. En la tabla no se muestran los dataset para los que UKB es capaz de dar respuesta para el 100 % de pares de palabras. Como se puede observar en la tabla, en el caso de UKB a diferencia de FastText tenemos un número de pares de palabras para los que no podemos dar respuesta significativo. Pero, si usamos el método de aplicación en cascada conseguidnos dar respuesta para todos los pares de palabras.

En la Tabla 5.3 se muestra el rendimiento obtenido por la combinación de word embeddings usando el método de combinación en cascada y UKB como word embedding principal. Con objetivo de facilitar futuras comparativas se muestra el la media y la media ponderada obtenida por la combinación. A continuación se muestra la media y la media ponderada de forma relativa a el rendimiento del embedding UKB. También se muestra la puntuación de forma relativa a UKB en los 3 datasets donde existen más para de palabras para los que UKB no puede dar respuesta. Es decir, los dataset donde vamos a ver una mayor diferencia en la puntuación. En la tabla se muestran dos métricas, que son las descritas en la Sección 3.1. En la primera se ignoran las palabras para las que el word em-

Métrica	Embedding	Media	Media Ponderada	Media (Diferencia)	Media Ponderada (Diferencia)	MTurk 287 (Diferencia)	VERB 143 (Diferencia)	RW (Diferencia)
Ignorar OOV	UKB ->Lexvec ->GLOVE ->W2V ->SKE ->PDC	0.644	0.596	0.014	-0.014	-0.034	0.212	-0.020
Resultado * Cobertura	UKB ->Lexvec ->GLOVE ->W2V ->SKE ->PDC	0.644	0.596	0.048	0.028	0.094	0.301	0.118

Tabla 5.3: Comparación de rendimiento entre la combinación usando el método de aplicación en cascada usando UKB como embedding principal y UKB de forma relativa.

bedding no es capaz de dar respuesta. Por lo tanto no se penaliza que un word embedding no sea capaz de dar respuesta para un determinado número de pares de palabras. Y la segunda métrica es la multiplicación del resultado obtenido den el dataset por el porcentaje de pares de palabras del dataset para los que el word embedding ha podido dar respuesta. Es decir, se penaliza que un word embedding no sea capaz de dar respuesta para todos los pares de palabras del dataset.

En cuanto a los resultados obtenidos. En el caso de la primera métrica observamos algo parecido a lo que ocurría con FastText. En los dataset Mturk-287 [54] y RW [59] hemos aumentado de forma significativa el número de pares para los que podemos dar respuesta. Sin embargo, usando la primera métrica en la que no se penaliza no dar respuesta para todos los pares del dataset perdemos rendimiento. La razón por lo que esto ocurre es por que estamos usando pares de palabras de word embeddings con peor rendimiento cuando UKB no puede dar respuesta. La excepción a esto es el dataset VERB-134 [20]. UKB puede dar respuesta para un número de pares de palabras muy limitado de este dataset, además el rendimiento de UKB en este dataset es bastante malo. Los dataset secundarios cuentan con un mejor rendimiento en este dataset y es por ello por lo que existe una mejora de rendimiento tan significativa. En el caso de nuestra segunda métrica, donde si que se penaliza no ser capaz de de dar respuesta para todos los pares de palabras del dataset los resultados cambian. En este caso si que tenemos una mejora de rendimiento significativa en todos los dataset. Y esto se ve reflejado tanto en la media como en la media ponderada de los resultados en todos los dataset.

A continuación (Tabla 5.4) se muestra una tabla mostrando de forma relativa al mejor embedding de la combinación (el embedding principal), el rendimiento de las diferentes

combinaciones que se han probado. Los resultados completos de las pruebas, donde se muestra el rendimiento en cada dataset se pueden encontrar en la plataforma GitHub [1].

Métrica	Ignorar OOV		Resultado * Cobertura	
Embedding	Media (Diferencia)	Media Ponderada (Diferencia)	Media (Diferencia)	Media Ponderada (Diferencia)
FastText ->jointcHYB ->UKB ->Lexvec ->GLOVE ->W2V ->SKE ->PDC	-0.002	-0.003	-0.001	0
jointcHYB ->UKB ->Lexvec ->GLOVE ->W2V ->SKE ->PDC	-0.001	-0.004	0.002	0.004
UKB ->Lexvec ->GLOVE ->W2V ->SKE ->PDC	0.014	-0.014	0.048	0.028
Lexvec ->GLOVE ->W2V ->SKE ->PDC	-0.002	-0.002	-0.001	0
GLOVE ->W2V ->SKE ->PDC	-0.002	-0.002	-0.001	-0.001
W2V ->SKE ->PDC	-0.009	-0.01	0.003	0.004
SKE ->PDC	0	0	0	0

Tabla 5.4: Comparación de rendimiento relativo entre las combinaciones usando el método de aplicación en cascada y el rendimiento del mejor embedding usado en la combinación (El embedding principal).

5.1.3. Conclusiones

Como se ha mostrado durante la sección, el método de aplicación en cascada se muestra útil para ampliar el vocabulario disponible. Sin embargo, su efecto en word embeddings de gran tamaño como los calculados en el corpus common crawl es muy limitado. Este método solo resulta útil en el caso de word embeddings que cuentan con un vocabulario reducido, como UKB. Aunque este método nos permite aumentar el número de pares de palabras para el que podemos dar respuesta, no permite combinar diferentes fuentes de conocimiento para obtener word embeddings mejores. De hecho, debido a que estamos usando word embeddings con peor rendimiento cuando no podemos dar respuesta para un determinado par de palabras en muchos casos conseguimos el efecto contrario. Por lo tanto, no resulta un método de combinación especialmente interesante. Es por ello por lo que en las próximas secciones estudiaremos diferentes formas de combinar word embeddings

[1] https://github.com/ikergarcia1996/RotEmbeddings

que nos permitan, además de aumentar el vocabulario disponible, usar la información contenida en diferentes fuentes de conocimiento para obtener mejores representaciones de palabras.

5.2. Media de la similitud

En esta sección se describirá el método de combinación de word embeddings mediante la media de la similitud y se discutirán los resultados obtenidos. Este método se basa en calcular la similitud entre dos palabras usando cada fuente de conocimiento disponible independientemente, en este caso word embeddings, para luego combinar los resultados obtenidos, usando la media aritmética. Con esto esperamos lograr dos objetivos, el primero es ampliar el vocabulario disponible, aumentando así número de pares de palabras para los que podemos dar una respuesta. También esperamos que la media aritmética de los resultados proporcionados por diferentes word embeddings sea una medida de similitud mejor que la aportada por cada uno de los word embeddings por separado.

5.2.1. Explicación del método

En esta sección se describirá en profundidad el método de la media de la similitud. Lo que buscamos es combinar las respuestas que cada word embedding da para un determinado par de palabras. Para ello dado un par de palabras, calcularemos por cada word embedding disponible la similitud coseno entre ambas palabras, usando las representaciones para esas palabras contenidas en cada word embedding. Es decir, usando el primer word embedding calcularemos la similitud coseno entre ambas palabras, obteniendo un resultado. Después calcularemos la similitud coseno usando el segundo word embedding, obteniendo otro resultado. Y así sucesivamente con todos los word embeddings disponibles. Una vez tenemos todos los resultados, el resultado final será la media aritmética de los resultados proporcionados por cada word embedding. Adicionalmente a la media aritmética, también hemos decido añadir la posibilidad de realizar una media aritmética ponderada. Es decir, poder dar un peso diferente a cada word embedding. Esto es útil para poder ejemplo dar más peso en el resultado final a los word embedding con mejor rendimiento respecto a los que tienen un rendimiento inferior.
En caso de que uno o varios word embeddings no sean capaces de dar una respuesta para un determinado par de palabras, ya que carecen de representación para una o ambas palabras del par, realizaremos la media tan solo con los resultados disponibles. Es decir,

si disponemos de 4 word embeddings “W1”, “W2”, “W3” y “W4”. Dado un par de palabras, si los embeddings “W1” y “W3” no pueden dar una respuesta para dicho par, el resultado será la media aritmética (ponderada o no) entre la similitud calculada usando “W2” y “W4”. En caso de que ningún word embedding pueda dar respuesta para dicho par diremos que no es posible dar una respuesta. Podemos realizar esto por que estamos realizando la media entre lo similitud calculada por cada word embedding, por lo tanto no estamos combinando directamente los vectores de cada word embedding y no existen problemas debido a combinar información de espacios vectoriales diferentes como ocurría con el método de combinación en cascada descrito en la Sección 5.1. Sin embargo, si que sería posible que la magnitud de los resultados aportados por cada word embedding fuese diferente. Es decir, dado el par de palabras “Automovil” y “Coche”, el par de palabras “Ratón” y “Lápiz”, y los word embeddings “W1” y “W2”. Es posible que usando las representaciones de palabras de “W1” obtengamos una similitud de “0,9” para el par “Automovil” - “Coche”, y una similitud de 0,1 para el par “Ratón” - “Lápiz”. Y en el caso de “W2” podríamos obtener los valores 9 y 1 respectivamente. Con ambos word embeddings hemos obtenido el mismo resultado, salvo que en el caso que los resultados de “W2” son los resultados de “W1” multiplicados por 10, pero en ambos casos, la similitud entre las palabras del primer par es 9 veces superior a la similitud entre las palabras del segundo par. Por lo tanto podemos determinar que las palabras del primer par son más similares entre ellas que las palabras del segundo par. Si realizásemos la media entre ambos resultados obtendríamos los valores “4.95” y “0.55’, por lo tanto se mantiene que la similitud entre las palabras del primer par de palabras es 9 veces superior que la similitud entre las palabras del segundo par. Pero ahora pongámonos en la situación de que “W2” no puede dar respuesta para el primer par de palabras por que no cuenta con representación para dichas palabras. Si siguiésemos la metodología descrita en esta sección para el primer par de palabras calcularíamos la media aritmética entre los resultados obtenidos por los embeddings que pueden dar respuesta para dicho par, en este caso solo “W1” puede dar respuesta por lo tanto daremos como resultado “0.9”. En el caso del segundo par haríamos la media entre los resultados proporcionados por ambos word embeddings, que es “0.55”. Como se puede observar hemos desvirtuado los resultados, hemos pasado de decir que las palabras del primer par son 9 veces más similares entre ellas que las palabras del segundo par a un resultado donde las palabras del primer par no son ni tan siquiera el doble de similares entre ellas que las del segundo par. Tenemos varias formas de afrontar este problema. La primera y la más sencilla es determinar que no es posible dar respuesta para cualquier par de palabras donde algunos de los word embeddings que estamos usando en la combinación no pueda dar respuesta para dicho par. Pero este método tiene dos pro-

blemas fundamentales, el primero es que a medida que añadimos word embeddings a la combinación, en vez de aumentar el vocabulario disponible lo iremos reduciendo, ya que podremos dar respuesta solo para pares de palabras donde ambas palabras se encuentren en la intersección entre el vocabulario de los word embeddings usados en la comparativa. El segundo problema, es que dicha intersección estará formada por las palabras más comunes de las fuentes de conocimiento usadas para entrenar los word embeddings. Si el word embedding "W1" no cuenta con representación para una palabra para la que "W2" si cuenta, es muy probable que dicha palabra sea una palabra poco común y la representación con la que cuenta "W2" no sea precisa. Por lo tanto, eliminando todas las palabras poco comunes y dando solo respuesta para los pares de palabras que incluyen las palabras más comunes provocaremos que los resultados al probar la combinación en los dataset en los que estamos evaluando nuestros word embeddings sean artificialmente más altos. Por lo tanto, esta no es una buena metodología y queda descartada. La segunda forma de solucionar este problema es aplicar algún método de normalización, ya sea a los resultados obtenidos o a los vectores de los word embeddings. En nuestro usaremos para calcular la similitud entre dos palabras la similitud coseno. La similitud coseno es una medida que normaliza la longitud de los vectores. Los resultados que obtengamos irán desde el valor -1, indicando que ambas palabras son totalmente contrarias. Al 1, indicando que son exactamente iguales. El 0 indica que no existe relación entre ambas palabras. Por lo tanto, si usamos como medida de similitud la similitud coseno eliminamos este problema.

$$\text{Similitud coseno: } cos(\boldsymbol{x},\boldsymbol{y}) = \frac{\boldsymbol{x}\cdot\boldsymbol{y}}{||\boldsymbol{x}||\cdot||\boldsymbol{y}||}$$

A continuación se muestran dos gráficas donde se ilustra la diferencia entre el uso de la similitud coseno y el producto escalar como medida de similitud entre palabras. Las gráficas se han realizado de la siguiente forma. Se ha utilizado el dataset YP-130 [86]. Para facilitar la visualización se han ordenado todos los pares de palabras de menor a mayor en función de la similitud proporcionada por el dataset. Usando los embeddings Word2Vec y FastText (entrenado en el corpus common crawl [2]) hemos calculado para cada par de palabras la similitud entre las palabras del par, lo hemos realizado usando como medida la similitud coseno y el producto escalar. En ambas gráficas podemos observar lo descrito anteriormente, usando el producto vectorial como medida ambos word embeddings aportan resultados que se encuentran en escalar diferentes. En cambio, usando la similitud coseno, dado que se normaliza la longitud de los vectores los resultados proporcionados por ambos word embeddings se encuentran en la misma escala.

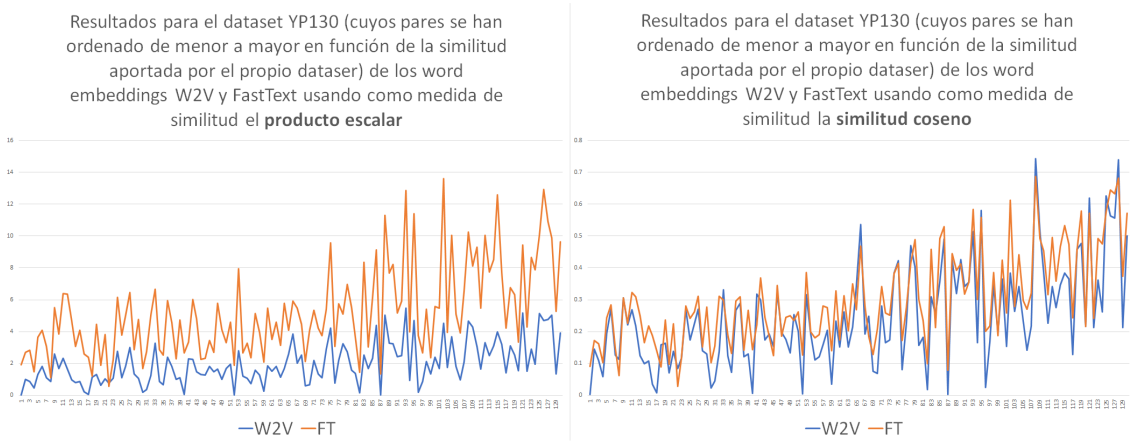


Figura 5.2: Demostración de la diferencia entre calcular la similitud usando el producto escalar, y la similitud coseno, en el dataset YP-130 usando los word embeddings Word2vec y FastText

La tercera opción sería utilizar algún método que nos permita calcular las palabras para las que un word embedding no cuenta con representación, consiguiendo así eliminar la posibilidad de que algún word embedding no pueda dar respuesta para un determinado par de palabras. En próximos capítulos se discutirá un método que dado un embedding, permite aproximar la representación de una palabra para la que el word embedding no cuenta con representación siempre y cuando dispongamos de un word embedding que cuente con representación para dicha palabra. Consiguiendo de esta forma que la intersección del vocabulario de varios word embedding sea igual a la unión del vocabulario de dichos embeddings. Este método también es una forma se solucionar este problema, sin embargo las palabras calculadas son aproximaciones, mientras que si usamos como medida la similitud coseno podemos aplicar el método sin necesidad de utilizar aproximaciones. Por lo tanto, tenemos que tener en cuenta que el método descrito en esta sección solo es aplicable en caso de que usemos como medida de similitud entre palabras la similitud coseno (o en su defecto, usemos algún otro método de normalización que evite el problema descrito).

A continuación se describe el algoritmo para calcular media de la similitud en pseudocódigo:

```
1
2  //La función avg_similarity da como salida la similitud coseno entre ambas palabras en caso de
       que haya podido ser calculada o el valor nulo en caso de que ningún word embedding
       proporcionado cuente con representación para las dos palabras del par.
3
4  función  avg_similarity(Lista_embeddings, Pesos, Par):
5  //Entrada:
6      //Lista_embeddings: Lista que contiene los word embeddings que     queremos combinar usando
       el método de la media de la similitud.
7      //Pesos: Lista que indicará los pesos que queremos dar a cada word embedding en la media
       aritmética de los resultados.
8      //Par: Par de palabras formado por las dos palabras W_1 y W_2 para la que queremos calcular
       la similitud coseno.
9
10 //Salida: Similitud coseno entre W_1 y W_2
11
12 //Funciones:
13     //Buscar(P,Emb). Devuelve el vector que representa a la palabra P en el embedding Emb. Si el
       embedding no cuenta con una representación para P, devuelve el valor nulo.
14     //Similitud_Coseno(V1,V2). Devuelve la similitud coseno entre los vectores V1 y V2. En caso
       de que V1 o V2 sean el valor nulo, devuelve el valor nulo.
15
16 r = 0
17 d = 0
18
19 Para i desde 1 hasta longitud(Lista_embeddings):
20     similitud = Similitud_Coseno(
21                 Buscar(W_1, Lista_embeddings[i]),
22                 Buscar(W_2, Lista_embeddings [i]))
23     Si similitud es no nulo:
24         r = r + (similitud * Pesos[i])
25         d = d + Pesos[i]
26
27 Si d > 0:
28     Devolver r/d
29 En caso contrario:
30     Devolver Nulo
```

5.2.2. Resultados obtenidos

En esta sección discutiremos los resultados obtenidos por el método de la media de las similitudes. La notación que vamos a utilizar para las diferentes pruebas es la siguiente:

$$embedding\_1 + embedding\_2 + embedding\_3 + ...$$

Esto quiere decir que se va calcular la similitud para cada par de palabras haciendo uso del *embedding_1*, el *embedding_2*, el *embedding_3*... y el resultado final será el promedio entre todas las similitudes.
Los pesos usados para la media ponderada se indicarán con la siguiente notación:

$$[1,2,4...]$$

Esto quiere decir que al *embedding_1* se le ha dado un peso *1*, al *embedding_2* un peso *2*, al *embedding_3* un peso *4* y así sucesivamente. En caso de que no se muestren los pesos significa que se ha dado a todos los word embeddings un peso de *1*.

La metodología para las pruebas realizadas es la detallada en la Sección 3.1, Las tablas con los resultados que analizaremos en esta sección cuentan con 4 columnas. En 2 de ellas se muestra la media y la media ponderada obtenida para facilitar futuras comparaciones. Y en las 2 restantes se muestra la media y la media ponderada de forma relativa al mejor word embedding que se ha incluido en la combinación. De esta forma podemos ver de forma sencilla si la combinación de word embeddings mejora el rendimiento de los embeddings originales.

Para analizar los resultados de estas pruebas resulta interesante dividir los resultados en dos grupos para analizarlos. Primero combinaremos word embeddings en parejas de dos, de esta forma podremos estudiar el efecto que tiene la combinación de diferentes tipos de embeddings entre ellos. Y después realizaremos combinaciones en las que usaremos 3 o más word embeddings, de esta forma analizaremos si añadir más conocimiento influye directamente en el rendimiento de la combinación. Una vez hecho este análisis pasaremos a analizar el efecto que tienen añadir pesos a la media aritmética en los resultados.

Combinaciones de Word Embeddings de dos en dos

El objetivo de esta sección es analizar que efectos tiene en el rendimiento la combinación de diferentes word embeddings. Es decir, queremos comprobar si se obtiene mejores re-

sultados combinando ciertos tipos de word embeddings, o el mejor rendimiento se consigue combinando los word embeddings que mejor rendimiento obtienen por separado. Para facilitar este análisis, evaluaremos para cada word embeddings el rendimiento obtenido (media y media ponderada) combinándolo con el resto de word embeddings disponibles. Comenzaremos con nuestro mejor word embedding, FastText, en la Tabla 5.5 se muestra el rendimiento de las combinaciones de FastText con el resto de word embeddings disponibles. Para la realización de esta tabla se ha usado la primera métrica descrita en la Sección 3.1, donde se descartan las palabras para las que no podemos dar respuesta. Por lo tanto, no se penaliza que los word embeddings no sean capaces de dar respuesta para un determinado número de pares de palabras.

Embeddings	Media (Diferencia)	Media Ponderada (Diferencia)	Media	Media Ponderada
GLOVE + FT	-0.011	-0.015	0.664	0.605
W2V + FT	-0.015	-0.020	0.660	0.600
FT + jointcHYB	0.029	0.034	0.704	0.654
FT + UKB	0.036	0.039	**0.711**	**0.659**
FT + LEXVEC	-0.009	-0.011	0.666	0.609
FT + SKE	-0.017	-0.024	0.658	0.596
FT + PDC	-0.009	-0.019	0.666	0.601

Tabla 5.5: Comparación de rendimiento relativo entre las combinaciones de FastText con el resto de word embeddings usando el método de la media de las similitudes y el rendimiento del mejor embedding usado en la combinación. Métrica usada: descartar pares para los que no se puede dar respuesta.

En la Tabla 5.5 podemos observador un comportamiento muy interesante. Cuando combinamos FastText con Glove, Word2Vec, Lexvec, SketchEngine o PDC vemos como perdemos rendimiento respecto al mejor word embedding de la comparativa (en estos casos el propio FastText). En cambio cuando combinamos FastText con los embeddings UKB y jointcHYB obtenemos una mejora de rendimiento bastante significativa. FastText, Glove, Word2Vec, Lexvec y SketchEngine son word embeddings generador a partir de texto. Aunque se han generado usando métodos diferentes, e incluso corpus diferentes todos ellos tienen una base común, toman como entrada un corpus de textos y producen como salida una serie de vectores. UKB y jointcHYB en cambio han sido generados de forma diferente. Estos word embeddings han sido generador a partir de grafos, en ambos casos WordNet [39]. WordNet es una base de datos donde nombres, verbos, adjetivos y adverbios son agrupaos en conjuntos de sinónimos, cada uno expresando un concepto diferente. Estos grupos están interrelacionados por medio de relaciones semántico-conceptuales y

léxicas. Por lo tanto, la entrada que reciben estos métodos de generación de word embeddings no es un corpus de texto, si no que es grafo o conjunto de relaciones. JointcHYB [45] se trata de un sistema híbrido donde también se usan corpus basados en texto, pero incluye también el conocimiento de WordNet. Por lo tanto, estos resultados indican que combinar Embeddings provenientes de fuentes de conocimiento diferentes (texto y grafos), resulta mejor que combinar Word Embeddings calculados a partir de corpus de texto entre ellos.
Resulta especialmente interesante la combinación de FastText con UKB puesto que conseguimos un rendimiento excelente, significativamente superior a cualquier word embedding en solitario que se han analizado en el Capítulo 3.

En la Tabla 5.6 se muestran los resultados obtenidos usando la segunda métrica descrita en la Sección 3.1, donde se multiplica el resultado obtenido por el porcentaje de pares de palabras para los que se ha podido dar respuesta. Como FastText es un embedding con un vocabulario muy amplio, los resultados apenas varían pese a que este método nos permite ampliar el vocabulario disponible.

Embeddings	Media (Diferencia)	Media Ponderada (Diferencia)	Media	Media Ponderada
GLOVE+FT	-0.011	-0.015	0.663	0.603
W2V+FT	-0.015	-0.020	0.659	0.598
FT+jointcHYB	0.030	0.036	0.704	0.654
FT+UKB	0.037	0.040	**0.711**	**0.658**
FT+LEXVEC	-0.008	-0.011	0.666	0.607
FT+SKE	-0.016	-0.023	0.658	0.595
FT+PDC	-0.009	-0.018	0.665	0.600

Tabla 5.6: Comparación de rendimiento relativo entre las combinaciones de FastText con el resto de word embeddings usando el método de la media de las similitudes y el rendimiento del mejor embedding usado en la combinación. Métrica usada: Resultado * Cobertura.

Vistos estos resultados vamos a comprobar si ocurre lo mismo con otros word embeddings o estos resultados son solamente válidos para FastText. Vamos a comprobar que ocurre con Glove. En la Tabla 5.7 se muestran los resultados de las combinaciones de Glove con los diferentes word embeddings disponibles. Se ha usado la primera métrica descrita en la Sección 3.1, en la que se descartan las palabras para las que no podemos dar respuesta. Por lo tanto, no se penaliza que los word embeddings no sean capaces de dar respuesta para un determinado número de pares de palabras.

Embeddings	Media (Diferencia)	Media Ponderada (Diferencia)	Media	Media Ponderada
GLOVE + W2V	0.012	0.009	0.641	0.580
GLOVE + FT	-0.011	-0.015	0.664	0.605
GLOVE + jointcHYB	0.036	0.023	**0.690**	**0.629**
GLOVE + UKB	0.056	0.020	**0.686**	**0.630**
GLOVE + LEXVEC	0.009	0.007	0.639	0.580
GLOVE + SKE	-0.001	-0.004	0.628	0.567
GLOVE + PDC	0.016	0.003	0.645	0.574

Tabla 5.7: Comparación de rendimiento relativo entre las combinaciones de Glove con el resto de word embeddings usando el método de la media de las similitudes y el rendimiento del mejor embedding usado en la combinación. Métrica usada: Descartar palabras para las que no se puede dar respuesta.

Como se puede comprobar en la Tabla 5.7, ocurre algo similar a lo que ocurría con FastText. En este caso algunas combinaciones entre Glove y otros word embeddings entrenados usados corpus de textos si nos proporcionan una mejora de rendimiento respecto al mejor word embedding usado en la combinación. Sin en embargo la mejora en todos los casos es bastante reducida y en algunos casos llegamos incluso a perder rendimiento. En cambio, como ocurría con FastText si combinamos Glove, word embedding entrenado usando un corpus de textos, con UKB o jointcHYB word embeddings obtenidos a partir de grafos obtenemos una mejora significativa. De nuevo, resulta mejor combinar word embeddings generados a partir de texto con word embeddings generados a partir de grafos.

En la Tabla 5.8 se muestran los mismos resultados, pero usando la segunda métrica descrita en la Sección 3.1, la cual penaliza que un word embedding no sea capaz de dar respuesta para un determinado número de pares de palabras. Como ocurre con FastText aunque este método de combinación nos permite ampliar el número de pares de palabras para los que damos respuesta, debido a que Glove es un word embedding que cuenta con un vocabulario muy amplio apenas existen diferencias en los resultados finales.

Embeddings	Media (Diferencia)	Media Ponderada (Diferencia)	Media	Media Ponderada
GLOVE+W2V	0.011	0.009	0.640	0.578
GLOVE+FT	-0.011	-0.015	0.663	0.603
GLOVE+jointcHYB	0.039	0.032	**0.690**	**0.629**
GLOVE+UKB	0.057	0.060	**0.686**	**0.629**
GLOVE+LEXVEC	0.009	0.009	0.638	0.579
GLOVE+SKE	-0.001	-0.002	0.628	0.567
GLOVE+PDC	0.016	0.004	0.645	0.573

Tabla 5.8: Comparación de rendimiento relativo entre las combinaciones de Glove con el resto de word embeddings usando el método de la media de las similitudes y el rendimiento del mejor embedding usado en la combinación. Métrica usada: Resultado * Cobertura.

Para terminar este primer análisis vamos a comprobar que ocurre con el resto de word embeddings. En la Tabla 5.9 se muestran los resultados de las combinaciones entre los word embeddings disponibles que no han sido mostradas en la Tabla 5.5 y la Tabla 5.7. En los resultados mostrados en la Tabla 5.9 ha usado la primera métrica descrita en la Sección 3.1, en la que se descartan las palabras para las que no podemos dar respuesta. Por lo tanto, no se penaliza que los word embeddings no sean capaces de dar respuesta para un determinado número de pares de palabras. Como se puede comprobar si mantienen las conclusiones a las que hemos llegado anteriormente. Las mejoras de rendimiento más significativas se obtienen cuando combinamos word embeddings obtenidos a partir de corpus de texto y grafos. Las combinaciones entre word embeddings obtenidos a partir de corpus de texto, y como se puede comprobar en la Tabla 5.9 las combinaciones obtenidas a partir de unir word embeddings obtenidos a partir de grafos obtienen una mejora de rendimiento inferior respecto al mejor word embedding de la combinación y en algunos casos incluso podemos observar perdidas de rendimiento.

Embeddings	Media (Diferencia)	Media Ponderada (Diferencia)	Media	Media Ponderada
W2V + jointcHYB	0.028	0.019	0.682	0.625
W2V + UKB	0.053	0.015	0.683	0.625
W2V + LEXVEC	0.011	0.007	0.641	0.580
W2V + SKE	0.017	0.010	0.633	0.565
W2V + PDC	0.014	0.010	0.630	0.565
jointcHYB + UKB	0.001	-0.001	0.655	0.609
jointcHYB + LEXVEC	0.035	0.029	0.689	0.635
jointcHYB + SKE	0.028	0.016	0.682	0.622
jointcHYB + PDC	0.021	0.008	0.675	0.614
UKB + LEXVEC	0.060	0.027	0.690	0.637
UKB + SKE	0.057	0.018	0.687	0.628
UKB + PDC	0.043	0.008	0.673	0.618
LEXVEC + SKE	0.002	-0.041	0.632	0.569
LEXVEC + PDC	0.017	-0.033	0.647	0.577
SKE + PDC	0.027	0.015	0.637	0.560

Tabla 5.9: Comparación de rendimiento relativo entre las combinaciones de Word Embeddings con el resto de word embeddings usando el método de la media de las similitudes y el rendimiento del mejor embedding usado en la combinación. Métrica usada: descartar pares para los que no se puede dar respuesta.

Con el objetivo de facilitar futuras comparaciones en la Tabla 5.10 se muestran los resultados usando la segunda métrica descrita en la Sección 3.1. Esta metodología penaliza que un determinado word embedding o combinación no sea capaz de dar respuesta para un determinado número de pares de palabras. La mayoría de resultados de la tabla Tabla 5.10 no varían de forma significativa respecto a los de la tabla Tabla 5.9, ya que los word embeddings usados en las combinaciones cuentan con un vocabulario muy amplio. Sin embargo si que se hay cambios significativos en las combinaciones que incluyen al word embedding UKB, ya que UKB cuenta con un vocabulario bastante limitado. El método de combinación de la media de las similitudes nos permite ampliar el vocabulario disponible y gracias a ello los resultados de las combinaciones de UKB con otros word embeddings utilizando esta segunda métrica suponen una mejora aún mayor respecto al rendimiento de UKB en solitario. Por lo tanto, este método se muestra también útil para ampliar el número de pares de palabras para los que podemos dar respuesta.

Embeddings	Media (Diferencia)	Media Ponderada (Diferencia)	Media	Media Ponderada
W2V + jointcHYB	0.030	0.026	0.681	0.623
W2V + UKB	0.075	0.053	0.679	0.621
W2V + LEXVEC	0.011	0.008	0.640	0.578
W2V + SKE	0.023	0.020	0.633	0.565
W2V + PDC	0.024	0.020	0.628	0.561
jointcHYB + UKB	0.039	0.032	0.653	0.600
jointcHYB + LEX-VEC	0.038	0.038	0.689	0.635
jointcHYB + SKE	0.031	0.025	0.682	0.622
jointcHYB + PDC	0.023	0.014	0.674	0.611
UKB + LEXVEC	0.061	0.066	0.690	0.636
UKB + SKE	0.177	0.060	0.687	0.628
UKB + PDC	0.076	0.047	0.672	0.615
LEXVEC + SKE	0.003	-0.001	0.632	0.569
LEXVEC + PDC	0.018	0.006	0.647	0.576
SKE + PDC	0.027	0.015	0.637	0.560

Tabla 5.10: Comparación de rendimiento relativo entre las combinaciones de Word Embeddings con el resto de word embeddings usando el método de la media de las similitudes y el rendimiento del mejor embedding usado en la combinación. Métrica usada: Resultado * Cobertura.

En esta sección hemos comprado dos cosas. La primera es que el método de combinación mediante la media de las similitudes resulta útil para mejorar el rendimiento de los word embeddings por separado. Sin embargo, no conseguimos mejorar siempre los resultados escojamos los word embeddings que escojamos. Las pruebas han mostrado que combinar word embeddings obtenidos a partir de corpus de textos con otros word embeddings obtenidos también a partir de corpus de texto, al igual que combinar word embeddings provenientes de grafos junto a otros word embeddings provenientes de grafos, no resulta en una mejora de rendimiento significativa. En algunos casos incluso se producen pérdidas de rendimiento. En cambio, si combinamos un word embedding calculado a partir de un corpus de textos con un word embedding proveniente de grafos podemos conseguir mejoras de rendimiento muy significativas. Destacan las combinaciones de FastText con UKB y FastText con jointcHYB (Tabla 5.5 ya que obtienen rendimientos muy buenos, superiores a cualquier word embedding en solitario analizado en el Capítulo 3. Este método también ha demostrado ser una forma eficaz de ampliar el número de pares de palabras para los que podemos dar respuesta. Por lo que aunque en algunas combinaciones no tengamos mejoras de rendimiento significativas, la combinación puede ser interesante debido a que aumentamos el vocabulario disponible.

Combinaciones de tres o más Word Embeddings

En esta sección analizaremos que ocurre cuando combinamos más de dos word embeddings. El objetivo de esta sección es comprobar si añadir más conocimiento a la combinación, aumenta de forma directa el rendimiento de los word embeddings. Para ello hemos realizado varias pruebas combinando diferentes word embeddings. Dichas pruebas se exponen la Tabla 5.11. Esta pruebas se han realizado usando la primera métrica descrita en la Sección 3.1. Sin embargo puesto que la mayoría de las combinaciones son capaces de dar respuesta para prácticamente el 100 % de los pares de palabras de los dataset o incluso el 100 %, no se incluirá una tabla con la segunda métrica la cual penaliza que un word embedding o combinación no sea capaz de dar respuesta para un determinado número de palabras, ya que los resultados de ambas tablas serían prácticamente idénticos.

En la Tabla 5.11 podemos ver combinaciones desde 3 hasta 8 word embeddings. Podemos observar como las concluimos a las que llegábamos en las combinaciones de word embeddings de dos en dos se mantienen. Las combinaciones que incluyen tan solo word embeddings calculados a partir de corpus de texto resultan en un rendimiento inferior al rendimiento del mejor word embedding de la combinación. Las mejores combinaciones

Embeddings	Media (Diferencia)	Media Ponderada (Diferencia)	Media	Media Ponderada
GLOVE + W2V + LEXVEC + SKE + PDC	-0.024	-0.037	0.651	0.583
GLOVE + W2V + FT + LEXVEC + SKE	-0.019	-0.027	0.656	0.593
GLOVE + W2V + FT + SKE + PDC	-0.015	-0.027	0.66	0.593
FT + jointcHYB + UKB + LEXVEC + GLOVE + W2V + SKE + PDC	0.021	0.009	0.696	0.629
GLOVE + FT + jointcHYB	0.024	0.021	0.699	0.641
FT + UKB + PDC	0.024	0.022	0.699	0.642
GLOVE + FT + UKB	0.025	0.022	0.7	0.642
W2V + FT + UKB	0.026	0.02	0.701	0.64
FT + jointcHYB + LEXVEC	0.026	0.026	0.701	0.646
FT + UKB + SKE	0.027	0.021	0.702	0.641
FT + jointcHYB + UKB	0.029	0.032	0.704	0.652
FT + UKB + LEXVEC	0.029	0.027	0.704	0.647
W2V + FT + jointcHYB + UKB + LEXVEC	0.032	0.027	0.707	0.647
FT + jointcHYB + UKB + SKE	0.033	0.031	0.708	0.651
FT + jointcHYB + UKB + LEXVEC	0.034	0.034	0.709	0.654
GLOVE + FT + jointcHYB + UKB	0.034	0.030	0.709	0.650

Tabla 5.11: Comparación de rendimiento relativo entre las combinaciones de Word Embeddings con el resto de word embeddings usando el método de la media de las similitudes y el rendimiento del mejor embedding usado en la combinación. Métrica usada: Resultado * Cobertura.

resultan de combinar dos word embeddings obtenidos a partir de texto con dos word embeddings calculados a partir de grafos. También es interesante comprobar como añadir más word embeddings no tiene por que traducirse en un mejor rendimiento. La combinación de todos los word embeddings disponibles (GLOVE, W2V, FT, LEXVEC, SKE, jointcHYB, UKB y PDC) obtiene un rendimiento inferior a la combinación que solo incluye a FT, jointcHYB, UKB y Lexvec. De hecho, ninguna de las combinaciones de más de 3 word embeddings que se han probado ha obtenido un rendimiento superior a la combinación de UKB y FastText que podemos encontrar en la Tabla 5.5. No se han probado todas las combinaciones posibles, ya que llevaría un tiempo excesivo, pero si que se han realizado pruebas exhaustivas probando todas las combinaciones posibles de hasta 4 word embeddings [2] y las mejores se encuentran en la Tabla 5.11, por lo que dando a todos los word embedding un peso de 1, la mejor combinación ha resultado ser de dos word embeddings. Estos resultados nos muestran que combinar un número mayor de word embeddings no tiene por que significar que la combinación obtendrá un mejor rendimiento respecto a combinar un número inferior de word embeddings. El factor más relevante en el rendimiento de la combinación es el combinar word embeddings calculados a partir de corpus de textos con word embeddings calculados a partir de grafos.

Efecto de los pesos en la combinación

Hasta ahora, todas las pruebas se han realizando dando un peso de 1 a todos los word embeddings de la combinación. Es decir, todos aportan por igual al resultado final. En esta sección modificaremos los pesos para dar a ciertos word embeddings más relevancia que a otros a la hora de realizar la media aritmética entre los resultados de todos los word embeddings. Es decir, realizaremos una media aritmética ponderada. Objetivo de realizar una media ponderada es dar un mayor peso a los word embeddings que cuentan con un mejor rendimiento en la tarea de similitud entre palabras. De esta forma esperamos conseguir aumentar el rendimiento final de la combinación. Para comprobar los efectos los pesos en la media de las similitudes hemos realizado varias pruebas, cuyos resultados se exponen en la Tabla 5.12. En dicha tabla se muestra la media y la media ponderada de los resultados obtenidos en los diferentes dataset. Además se muestra la media y la media ponderada de forma relativa a la misma combinación de word embeddings dando a todos los word embeddings un peso de 1.

Como se puede comprobar en la Tabla 5.12, asignar pesos a los diferentes word embed-

[2] Los resultados completos donde se incluyen todas las combinaciones probadas pueden consultarse en la siguiente dirección: http://bit.ly/2JF92x2

Embeddings	Pesos	Media (Diferencia)	Media Ponderada (Diferencia)	Media	Media Ponderada
FT + UKB	[2,1]	-0.002	-0.005	0.709	0.654
FT + UKB	[1,2]	-0.006	-0.007	0.705	0.652
UKB + PDC	[2,1]	0.003	0.007	0.676	0.625
UKB + PDC	[1,2]	-0.017	-0.021	0.656	0.597
GLOVE + UKB	[2,1]	-0.012	-0.015	0.674	0.615
GLOVE + UKB	[1,2]	0.001	0.004	0.687	0.634
FT + UKB + SKE	[2 3 1]	0.013	0.015	**0.715**	0.656
FT + UKB + SKE	[3 2 1]	0.007	0.010	0.709	0.651
FT + UKB + SKE	[2 2 1]	0.010	0.012	0.712	0.653
GLOVE + FT + jointcHYB + UKB	[1 4 2 3]	0.004	0.007	0.713	0.657
FT + jointcHYB + UKB + SKE	[4 2 3 1]	0.006	0.007	0.714	**0.658**
FT + jointcHYB + UKB + LEX-VEC	[3 1 4 2]	0.004	0.003	0.713	0.657
FT + jointcHYB + UKB + LEX-VEC	[2 1 2 1]	0.004	0.004	0.713	**0.658**
FT + jointcHYB + UKB + LEX-VEC + GLOVE + W2V + SKE + PDC	[4 4 4 4 1 1 1 1]	0.013	0.019	0.709	0.648
FT + jointcHYB + UKB + LEX-VEC + GLOVE + W2V + SKE + PDC	[2 2 1 1 1 1 1 1]	0.006	0.009	0.702	0.638
FT + jointcHYB + UKB + LEX-VEC + GLOVE + W2V + SKE + PDC	[8 7 6 5 4 3 2 1]	0.011	0.016	0.707	0.645

Tabla 5.12: Comparación de rendimiento relativo entre las combinaciones de Word Embeddings usando el método de la media de las similitudes y el rendimiento de la misma comparación dando a todos los embeddings un peso de uno. Métrica usada: descartar pares para los que no se puede dar respuesta.

dings, incluso dando mayor peso a los mejores word embeddings de la combinación, no nos asegura un mejor rendimiento. Un buen ejemplo de esto es la combinación de FastText con UKB o la combinación de FastText con UKB y SketchEngine. Sin embargo, si que resulta evidente que podemos lograr pequeñas mejoras de rendimiento gracias a la asignación de pesos. Destaca especialmente la combinación de fastText (peso 2), UKB (peso 3) y SketchEngine (peso 1), que obtiene el rendimiento más alto de las combinaciones probadas en esta sección. Otras combinaciones también se ven beneficiadas en mayor o menor medida de una buena asignación de pesos. Aunque los resultados obtenidos son buenos, la asignación de pesos tiene un problema importante, y es la elección de los pesos. Una buena asignación puede reportarnos una mejora de rendimiento, sin embargo una mala asignación puede provocar una reducción en el rendimiento de la combinación. Si queremos obtener el máximo rendimiento posible de la combinación, necesitamos implementar un algoritmo de optimización de pesos que calcule los mejores pesos para cada word embedding de la combinación. Sin embargo, sin asignación de pesos conseguimos una media en todos los dataset de 0.711 combinando UKB con FastText. No hemos probado todas las combinaciones posibles, ni todos los pesos posibles (solo se han asignado números naturales como pesos desde el 1 hasta el número de embeddings de la combinación), pero se ha realizado una cantidad importante de pruebas incluyendo todas las posibles combinaciones de hasta 4 word embeddings, y la mejor combinación obtenida ha sido la de fastText (peso 2), UKB (peso 3) y SketchEngine (peso 1) que ha obtenido una media de 0.715 en todos los dataset. Se ha conseguido una mejora de rendimiento mediante la asignación de pesos, sin embargo dicha mejora es mínima. En el caso de la comparación de las combinaciones asignando pesos a cada word embedding respecto a la misma combinación dando a todos los word embedding un peso de 1, tampoco se aprecian mejoras de rendimiento demasiado significativas. Por lo tanto, hay que considerar si realmente merece la pena invertir el tiempo requerido para la implementación de un algoritmo de optimización que nos permita calcular unos buenos pesos para cada word embedding de la combinación.

La asignación de pesos se muestra como un método que puede aportarnos una pequeña mejora de rendimiento. Sin embargo la tarea de seleccionar que peso damos a cada word embedding no es sencilla, ya que dar mayor peso a los mejores word embeddings de la combinación no siempre va a aportarnos una mejora de rendimiento significativa. De hecho, en determinados casos incluso podemos reducir el rendimiento de la combinación dando a todos los word embeddings el mismo peso.

Por último para dar por terminada esta sección, creemos que resulta interesante mostrar el

rendimiento de la combinación que mejor rendimiento ha obtenido (FastText + UKB dando igual peso a ambos) y de los word embeddings usados en la combinación en diferentes datasets. Estos resultados se muestran en la Tabla 5.13, donde se usa la primera métrica, en la que no se penaliza no poder dar respuesta para un determinado número de pares de palabras y la Tabla 5.14 donde se usa la segunda métrica con la que si se penaliza. Con esto queremos mostrar que le aumento de rendimiento no se debe a que tan solo mejore el rendimiento en unos pocos datasets y esto afecte a la media, si no que se consigue una mejora en prácticamente todos. Incluso en datasets como Verb 143 donde UKB no obtiene buen rendimiento, la combinación del conocimiento de ambos embeddings resulta beneficiosa.

Embedding	Media	Media Pon-dera-da	Sim Lex 999	MTurk 287	MTurk 771	MEN ALL	WS 353 all	WS 353 rela-ted-ness	WS 353 simi-larity	RG65	VERB 143	MC 30	YP 130	Sim Verb 3500	RW
FastText CC	0.675	0.620	0.503	**0.726**	0.762	0.837	0.791	0.737	0.840	0.863	0.447	0.852	0.625	0.426	**0.595**
UKB	0.630	0.610	0.525	0.644	0.679	0.756	0.720	0.627	0.785	0.823	0.153	0.873	0.768	**0.545**	0.443
FT+UKB	**0.711**	**0.659**	**0.563**	0.717	**0.779**	**0.847**	**0.810**	**0.761**	**0.846**	**0.888**	**0.484**	**0.878**	**0.773**	0.544	0.539

Tabla 5.13: Comparativa entre de la combinación de FT+UKB con el método de la media de las similitudes y los word embeddings usados en la combinación. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

Embedding	Media	Media Pon-dera-da	Sim Lex 999	MTurk 287	MTurk 771	MEN ALL	WS 353 all	WS 353 rela-ted-ness	WS 353 simi-larity	RG65	VERB 143	MC 30	YP 130	Sim Verb 3500	RW
FastText CC	0.674	0.618	0.503	**0.726**	0.762	0.837	0.791	0.737	0.840	0.863	0.447	0.852	0.625	0.426	**0.584**
UKB	0.596	0.568	0.525	0.516	0.678	0.756	0.705	0.612	0.769	0.823	0.064	0.873	0.768	**0.545**	0.305
FT+UKB	**0.711**	**0.658**	**0.563**	0.717	**0.779**	**0.847**	**0.810**	**0.761**	**0.846**	**0.888**	**0.484**	**0.878**	**0.773**	0.544	0.537

Tabla 5.14: Comparativa entre de la combinación de FT+UKB con el método de la media de las similitudes y los word embeddings usados en la combinación. Métrica utilizada: Resultado obtenido en cada datataset multiplicado por la cobertura.

5.2.3. Conclusiones

En esta sección se han mostrado tablas con los resultados que se han considerado más relevantes en cada momento. Sin embargo se han realizado más pruebas de las expuestas, los resultados completos de las pruebas, donde se muestra el rendimiento en cada dataset se pueden encontrar en la plataforma GitHub [3]. Como se ha mostrado durante esta sección el método de la media de las similitudes es capaz de obtener mejoras de rendimiento significativas. Varias de las combinaciones probadas durante esta sección han mostrado un rendimiento significativamente superior al rendimiento de los word embedding en solitario analizado en el Capítulo 3. Además este método también nos permite ampliar el número de pares de palabras para los que podemos dar respuesta. Sin embargo para aplicar este método de combinación con éxito tenemos que tener algunos detalles importantes en cuenta. El estudio que hemos llevado a cabo durante este capítulo ha mostrado que no necesariamente conseguiremos un mejor rendimiento combinando el mayor número de word embeddings posibles. Las combinaciones que mejor rendimiento nos han aportado han sido las que combinan word embeddings calculados a partir de corpus de textos con word embeddings calculados a partir de grafos. Word embedding calculados usando corpus similares contendrán conocimiento similar, por lo tanto combinarlos no hace que aumentemos de forma significativa el conocimiento disponible. En cambio, combinar word embeddings calculados a partir de datos diferentes como los corpus de texto y los Ramdom Walks sobre WordNet, puesto que son word embeddings que codifican un conocimiento diferente, hace que la combinación nos permita aumentar el conocimiento disponible y con ello el rendimiento.

También hemos podido comprobar que realizar una media aritmética ponderada de los resultados, es decir, dar un peso mayor a determinados word embeddings sobre el resto, puede aportarnos una ligera mejora de rendimiento. Sin embargo, la elección del peso que damos a cada word embedding no es una elección sencilla ya que no siempre dar mayor peso a los word embedding que cuentan con un mejor rendimiento nos dará los mejores resultados. Por lo tanto, es necesaria la implementación de un algoritmo de optimización que nos permita escoger los pesos óptimos para cada combinación. Dado que dando pesos a cada word embedding no vamos a conseguir una mejora de rendimiento significativa, no merece la pena invertir el tiempo necesario que conlleva una buena elección de pesos.

[3] https://github.com/ikergarcia1996/RotEmbeddings

5.3. Conclusiones: Combinaciones de Word Embeddings

Durante este capítulo hemos estudiado dos formas de combinar word embeddings. El método de aplicación en cascada y el método de la media de las similitudes. El método de la media de las similitudes ha resultado ser un método de combinación mucho más atractivo que el método de aplicación en cascada, puesto que nos aporta dos beneficios: Mejorar el rendimiento de los word embeddings y ampliar el número de pares de palabras para los que podemos dar respuesta. El método de aplicación en cascada solo puede aumentar el número de pares de palabras para los que podemos dar respuesta. Sin embargo, el método de la media de las similitudes no necesariamente va a mejorar el rendimiento de cualquier combinación de word embeddings que realicemos. Por lo que el método de aplicación en cascada podría resultar útil para aumentar el número de pares de palabras para los que podemos dar respuesta cuando el método de la media de las similitudes no sea capaz de realizar una buena combinación para los word embeddings que queremos combinar.
Ambos métodos nos permiten combinar el conocimiento proveniente de varios word embeddings. Sin embargo, no son métodos de generación de meta embeddings, es decir, estos métodos no generan un nuevo word embedding. Esto puede suponer un problemas. Primero por que ambos métodos requieren tener varios word embeddings cargados de forma simultanea en memoria. Debido al tamaño de los word embeddings esto supone un coste en memoria muy importante (Como ejemplo, Glove calculado en el corpus common crawl con 840 mil millones de tokens tiene un peso aproximado de 5,5GB). Como alternativa, es posible cargar en memoria los word embeddings (o los vectores necesarios) en el momento en el que sean necesarios y luego borrarlos. Sin embargo, la carga de un word embeddings es un proceso lento, por lo tanto provocaría que calcular la similitud para múltiples pares de palabras con las combinaciones de word embeddings sea extremadamente lento. Adicionalmente dificulta el uso de los word embeddings en otros ámbitos. Por ejemplo, si queremos utilizar nuestros word embeddings en una red neuronal como seq2seq [25]. Utilizar una combinación de word embeddings en vez de un solo word embedding puede complicar de forma importante la arquitectura de la red neuronal. Es por estos motivos por los que en los siguientes capítulos exploraremos diferentes métodos de generación de meta-embeddings. Es decir, métodos que permiten combinar el conocimiento de diferentes fuentes de conocimiento para generar un nuevo word embedding, que esperamos que, como hemos conseguido con el método de la media de las similitudes, sea mejor que las diferentes bases de conocimiento por separado.

6. CAPÍTULO

Meta Embeddings

En este capítulo analizaremos el rendimiento de diferentes métodos de generación de meta embeddings. En concreto analizaremos la concatenación de word embeddings, la reducción de dimensionalidad como extensión de la concatenación, la media de word embeddings y el "retrofitting" o readaptación de word embeddings a léxicos semánticos. Nuestro objetivo es combinar diferentes fuentes de conocimiento para generar nuevos word embeddings. Esperamos que estos word embeddings obtengan un mejor rendimiento en la tarea del cálculo de la similitud entre palabras que las diferentes fuentes de conocimiento usadas de forma individual.

6.1. Concatenación

El primer método de generación de meta embeddings que vamos a estudiar es la concatenación de word embeddings. La concatenación es el método de generación de meta embeddings más popular, debido a su simplicidad y ya que generalmente obtiene buenos resultados. Mediante este método podemos obtener un nuevo word embedding que contenga el conocimiento de diferentes word embeddings. Esperamos que este nuevo word embedding obtenga un mejor rendimiento que los diferentes word embeddings usados para generarlo individualmente.

6.1.1. Explicación del método

En esta sección explicaremos en detalle la concatenación de word embeddings y como la hemos aplicado a nuestros word embeddings. El método es sencillo, suponiendo que queremos concatenar dos word embeddings $E1$ y $E2$, generaremos el nuevo word embedding $CONC$ de la siguiente forma. Para cada palabra w perteneciente a la intersección de los vocabularios de ambos word embeddings $CONC_w$ se obtiene mediante la concatenación de el vector que representa a dicha palabra en $E1$ con el vector que representa a dicha palabra en $E2$.

$$\forall w \in E1 \cap E2 : CONC_w = E1_w \oplus E2_w$$

La concatenación genera un nuevo embedding con un número de dimensiones igual a la suma de las dimensiones de los word embeddings que se han concatenado. Por lo tanto obtenemos un nuevo espacio vectorial que incluye el conocimiento de todos los word embeddings concatenados.

Vamos a realizar un análisis matemático de la concatenación. Podemos expresar la concatenación de dos vectores v_1 y v_d2 de dimensiones d_1 y d_2, como la suma de v_1 tras añadirle d_2 por la izquierda y v_2 tras añadirle d_1 ceros por la derecha.

$$[1,2,3] \oplus [4,5,6] = [0,0,0,1,2,3] + [4,5,6,0,0,0] = [1,2,3,4,5,6]$$

Supongamos que tenemos dos word embeddings que queremos concatenar, E_1 y E_2, cuyas dimensiones son d_{E_1} y d_{E_2} respectivamente. Añadimos a los vectores de E_1 d_{E_2} ceros por la derecha y a los de E_2 d_{E_2} por la izquierda. Los vectores de ambos word embeddings ahora cuentan con un número de dimensiones $d_{E_1} + d_{E_2}$. El embedding para cualquier palabra $u \in S_1 \cap S_2$ será entonces $u_{E_1}^{ceros} + u_{E_2}^{ceros}$. Resaltar que el producto escalar de los vectores $u_{E_1}^{ceros} y u_{E_2}^{ceros}$ da como resultado cero, por lo tanto son ortogonales. Denotaremos la distancia euclídea entre dos palabras u y v como D_{E_1} y D_{E_2}. Resaltar que para cualquier vector $u \in \mathbb{R}^n$ añadir ceros no afecta a su norma L_2, por lo tanto:

$$D_{E_1} = \|u_{E_1} - v_{E_1}\|_2 = \|u_{E_1}^{ceros} - u_{E_1}^{ceros}\|_2$$
$$D_{E_2} = \|u_{E_2} - v_{E_2}\|_2 = \|u_{E_2}^{ceros} - u_{E_2}^{ceros}\|_2$$

La distancia euclídea entre u y v tras la concatenación será:

$$
\begin{aligned}
CONC &= \|(u_{E_1}^{ceros} + u_{E_2}^{ceros}) - (v_{E_1}^{ceros} + v_{E_2}^{ceros})\|_2 \\
&= \|(u_{E_1}^{ceros} - v_{E_1}^{ceros}) - (v_{E_2}^{ceros} - u_{E_2}^{ceros})\|_2 \\
&= \sqrt{(D_{E_1})^2 + (D_{E_2})^2 - 2D_{E_1}D_{E_2}cos(\Theta)} \\
&\text{Como los vectores son ortogonales:} \\
&= \sqrt{(D_{E_1})^2 + (D_{E_2})^2}
\end{aligned}
$$

Por lo tanto, para cualquier par de palabras pertenecientes al meta embedding resultante de la concatenación, la distancia euclídea entre dichas palabras es igual a la raíz cuadrada de la suma de las distancias euclídeas de dichas palabras en los word embedding [53]. Hemos realizado el cálculo para la concatenación de dos word embeddings, sin embargo, dicho cálculo es fácilmente generalizable a cualquier número de word embeddings. Esto prueba que la concatenación genera un meta embedding que combina el conocimiento de diferentes word embeddings.

La problemática de la escala de los word embeddings

Cuando realizamos la concatenación hay que tener en cuenta la escala de ambos vectores. Un embedding que se encuentra en una escala mayor que otro, es decir, que cuyos vectores tengan una longitud mucho mayor,al ser concatenado con otro con una longitud menor, puede provocar que en el cálculo de la similitud coseno tenga un peso mayor. Es decir, un word embedding que se encuentre en una escala más alta que el resto puede influenciar el cálculo de la similitud provocando que perdamos la información del resto de word embeddings concatenados. Podemos solucionar esto normalizando los vectores mediante la norma L_2, de esta forma todos los vectores tendrán una longitud igual a uno, y todos los word embeddings tendrán el mismo peso en la concatenación.

$$
\begin{aligned}
&\text{Norma } L_2. \\
&|x|_2 = \sqrt{\textstyle\sum_{k=1}^{n} |x_k|^2}
\end{aligned}
$$

Para demostrar esta problemática pongamos un ejemplo llevado al extremo. Imaginemos que disponemos de dos word embeddings que queremos concatenar y vamos a calcular la similitud mediante la similitud coseno entre las palabras A y B. En la Tabla 6.1 se muestran los vectores que representan a A y B en cada word embedding. Como se puede observar la similitud entre ambas palabras en cada word embedding es contradictoria. El primer word embedding da como resultado 1, es decir, ambas palabras son idénticas, mientras que el segundo word embedding obtiene como resultado que no existe relación

alguna entre ellas. Aunque hay una diferencia importante entre ambos word embeddings, los vectores del primer word embedding se encuentran en una escala mucho mayor, es decir, cuentan con una longitud mucho mayor. Si concatenamos ambos word embeddings nos encontramos con que el primer word embedding al contar con valores mucho más altos influencia el calculo de la similitud coseno. El resultado que obtenemos es de nuevo 1, lo que quiere decir que la concatenación no ha tenido efecto alguno ya que hemos perdido la información que el segundo word embedding nos aporta.

Palabras:	Embedding 1	Embedding 2	Concatenación	L2 + Concatenación
A	[100,100]	[1,0]	[100,100,1,0]	[0.70710678, 0.70710678, 1, 0]
B	[100,100]	[0,1]	[100,100,0,1]	[0.70710678, 0.70710678, 0, 1]
Similitud Coseno	1	0	0.99995	0.5

Tabla 6.1: Ejemplo de la problemática al concatenar embeddings en diferentes escalas.

Podemos comprobar esto matemáticamente, supongamos que tenemos dos word embedding, el word embedding A y el word embedding B, y A se encuentra en una escala 10 veces superior a B. Queremos calcular la similitud coseno entre la palabra W_1 y W_2 y para ello vamos a concatener el word embedding A y el word embedding B. Representaremos el vector que representa a W_1 en el embedding A como $[a_1^{w1}, a_2^{w1}, ...]$ y el vector que representa a W_2 como $[a_1^{w2}, a_2^{w2}, ...]$, haremos lo mismo con los vectores del embedding B pero usaremos la notación $[b_1^{w1}, b_2^{w1}, ...]$. Si concatenamos ambos word embedding y realizamos el calculo de la similitud coseno ocurriría lo siguiente:

$$cos(W_1, W_2) = \frac{[10a_1^{w1}, 10a_2^{w1}, b_1^{w1}, b_2^{w1}] \cdot [10a_1^{w2}, 10a_2^{w2}, b_1^{w2}, b_2^{w2}}{|||[10a_1^{w1}, 10a_2^{w1}, b_1^{w1}, b_2^{w1}]|| * ||[10a_1^{w2}, 10a_2^{w2}, b_1^{w2}, b_2^{w2}]||} =$$
$$= \frac{100 * (a_1^{w1} * a_1^{w2} + a_2^{w1} * a_2^{w2}) + 1 * (b_1^{w1} * b_1^{w2} + b_2^{w1} * b_2^{w2})}{|||[10a_1^{w1}, 10a_2^{w1}, b_1^{w1}, b_2^{w1}]|| * ||[10a_1^{w2}, 10a_2^{w2}, b_1^{w2}, b_2^{w2}]||} =$$
$$= \frac{100 * (w1_A \cdot w2_A) + (w1_B \cdot w2_B)}{|||[10a_1^{w1}, 10a_2^{w1}, b_1^{w1}, b_2^{w1}]|| * ||[10a_1^{w2}, 10a_2^{w2}, b_1^{w2}, b_2^{w2}]||}$$

Como podemos observar lo que ocurre es que la similitud coseno se convierte en el producto escalar de los vectores que representan a W_1 y W_2 en el embedding A multiplicado por cien más el producto el producto escalar de los vectores que representan a W_1 y W_2 en el embedding B. Esto quiere decir que el embedding A tiene un peso mucho mayor en el calculo de la similitud coseno que en el embedding B, por lo tanto estamos perdiendo el conocimiento del embedding B.

En cambio, si realizamos una normalización L_2 antes de la concatenación, conseguimos que la longitud de todos los vectores sea igual a 1. De esta forma ambos word embeddings

contribuirán con el mismo peso a la concatenación. Como se puede observar en la tabla al aplicar la normalización L_2 el resultado que obtenemos al calcular la similitud coseno es 0.5, la media entre las similitudes calculadas con cada word embedding individualmente. Este detalle es muy importante, si normalizamos los vectores mediante la norma L_2 todos los vectores contribuyen de igual forma en la concatenación, lo cual significa que el resultado que obtendremos es el mismo que el de calcular de forma individual la similitud coseno entre las palabras del par de palabras para el que queremos calcular la similitud y después realizar la media aritmética entre los resultados obtenidos. Es decir, la concatenación de vectores normalizados mediante la norma L_2 es equivalente al método de la media de las similitudes estudiado en el Capítulo 5. Dicho método tenía el problema de que no generaba un nuevo word embedding lo cual podría dificultar aplicar el método en determinadas tareas, la concatenación resuelve ese problema, ya que si genera un nuevo word embedding.

Podemos comprobar esto matemáticamente de la misma forma que en el caso anterior. Supongamos que tenemos los mismos word embeddings y palabras, pero ahora hemos normalizado todos los vectores de ambos word embeddings mediante la normalización L_2, es decir, la longitud de todos los vectores es uno. Si concatenamos dos vectores cuya longitud es uno, la longitud del nuevo vector será igual a $\sqrt{2}$:

$|||[a_1^{w1}, a_2^{w1}, b_1^{w1}, b_2^{w1}]|| = \sqrt{(a_1^{w1})^2 + (a_2^{w1})^2 + (b_1^{w1} + b_2^{w1})^2}$ Sabemos que el vector $[a_1^{w1}, a_2^{w1}]$ y $[b_1^{w1}, b_2^{w1}]$ tienen una longitud igual a uno. Por lo tanto $\sqrt{(a_1^{w1})^2 + (a_2^{w1})^2} = (a_1^{w1})^2 + (a_2^{w1})^2 = 1$ Sabiendo esto $\sqrt{[(a_1^{w1})^2 + (a_2^{w1})^2] + [(b_1^{w1} + b_2^{w1})^2]} = \sqrt{[1] + [1]} = \sqrt{2}$
Una vez conocemos la longitud del vector que resulta al concatenar dos vectores cuya longitud es uno:

$$cos(W_1, W_2) = \frac{[a_1^{w1}, a_2^{w1}, b_1^{w1}, b_2^{w1}] \cdot [a_1^{w2}, a_2^{w2}, b_1^{w2}, b_2^{w2}}{|||[a_1^{w1}, a_2^{w1}, b_1^{w1}, b_2^{w1}]|| * |||[a_1^{w2}, a_2^{w2}, b_1^{w2}, b_2^{w2}]||}$$
$$= \frac{(a_1^{w1} * a_1^{w2} + a_2^{w1} * a_2^{w2}) + (b_1^{w1} * b_1^{w2} + b_2^{w1} * b_2^{w2})}{\sqrt{2} * \sqrt{2}} =$$
$$= \frac{(w1_A \cdot w2_A) + (w1_B \cdot w2_B)}{2}$$

Por lo tanto, la similitud coseno se dos vectores normalizados mediante la normalización L_2 se puede expresar como la suma del producto escalar de los vectores que representan a W_1 y W_2 en el embedding A y los vectores que representan a W_1 y W_2 en el embedding B dividido entre dos, es decir, la media de los productos escalares. Pero sabemos que el producto vectorial de dos vectores normalizados mediante la norma L_2 es equivalente a la similitud coseno, como todos los vectores están normalizados mediante dicha norma, estamos realizando la media de las similitud coseno de la misma forma que hacíamos en

la Sección 5.2. Ambos word embeddings tienen ahora el mismo peso en el calculo de la similitud coseno.

La normalización L_2 también nos proporciona otra ventaja, y es la de poder concatenar word embeddings con un número diferente de dimensiones. Si concatenamos un embedding cuyos vectores cuentan con 300 dimensiones, y un embedding cuyos vectores cuenten con 100 dimensiones, el embedding de 300 dimensiones tenderá a dominar al embedding de 100 dimensiones en la concatenación, por lo que tendrá un peso mayor en el cálculo de la similitud coseno. Mediante la normalización L_2 ambos word embedding, aún teniendo un número diferente de dimensiones tendrán el mismo peso en la concatenación.

Mediante la normalización L_2 podemos conseguir que todos los word embedding tengan el mismo peso en la concatenación, pero ¿Queremos que sea así en todos los casos? En el Capítulo 3 hemos analizado el rendimiento de una gran cantidad de word embeddings y sabemos que algunos word embeddings obtienen un mejor rendimiento en la tarea de similitud entre palabras que otros. Por dar un mayor peso a los word embeddings que sabemos que cuentan con un mejor rendimiento puede resultar en mejores meta embeddings. Podemos conseguir esto haciendo el proceso inverso a la normalización, multiplicando los vectores de un word embedding por un determinado valor conseguimos aumentar su longitud y de esta forma tendrán un mayor peso en la concatenación. Resaltar que multiplicar todos los vectores de un word embedding por cualquier valor no modifica los resultados que obtendremos al calcular la similitud coseno entre cualquier par de palabras del word embedding, por lo tanto modificamos la longitud de los vectores pero no perdemos ni alteramos la información que contiene el word embedding, solo lo escalamos. Por lo tanto en esta sección analizaremos el rendimiento de la concatenación de word embeddings sin ningún tipo de preprocesamiento, y a continuación estudiaremos si la normalización L_2 y la asignación de pesos (tanto habiendo realizado la normalización L_2 como no habiéndola realizado) pueden ayudarnos a obtener mejores meta embeddings. De esta forma podremos comprobar como afecta la longitud de los vectores a la generación de meta embeddings y si es mejor mantenerla o no tenerla en cuenta usando la normalización L_2.

La problemática de combinar diferentes espacios vectoriales

Al concatenar word embeddings existe un último problema a resolver. En el Capítulo 5, realizamos diferentes combinaciones entre word embeddings. Para dichas combinaciones usábamos el resultado de la similitud para un determinado par de palabras obtenido por

diferentes word embeddings. Es decir, realizábamos una combinación del resultado final, eso nos permitía evitar la problemática de que cada word embedding se encontrase en un espacio vectorial diferente. En el caso de este capítulo no ocurre lo mismo, tanto para la concatenación como para la media de word embeddings lo que combinamos son los vectores de diferentes word embeddings. Esto afecta a las palabras fuera de la intersección del vocabulario de los word embeddings que queremos concatenar. Es decir, palabras para los que al menos un word embedding cuenta con representación, pero para la que uno o varios word embeddings no. Tenemos varias alternativas para tratar estas palabras. La primera es no incluir estas palabras en el nuevo word embedding generado, es decir, descartar todas las palabras que no se encuentren en la intersección entre los vocabularios que queremos combinar. Sin embargo, como ya se ha expuesto en capítulos anteriores esta opción tiene varios problemas que la descartan. La primera es que a medida que añadimos word embeddings a la combinación perdemos vocabulario en el embedding final. La segunda es que si un word embedding no tiene representación para una determinada palabra, es probable que se trate de una palabra poco usada, por lo tanto el resto de word embeddings, aunque cuenten con representación para ella dicha representación habrá sido obtenida a partir de un número de apariciones reducido por lo que la representación de dicha palabra no será precisa. Esto significa que quedándonos solo con las palabras pertenecientes a la intersección entre los vocabularios de los word embedding que queremos combinar, estamos quedándonos con las palabras más comunes y para los que los word embeddings cuentan con mejores representaciones. Por lo tanto, podemos caer en un aumento del rendimiento de los word embedding artificial, no podríamos saber si los meta embedding generados obtienen un buen rendimiento debido a haber eliminado las palabras menos comunes o debido al método con el que hemos generado dicho meta embedding.

Si queremos que dichas palabras se encuentre en el meta embedding generado, tenemos que asignarles un vector para poder realizar la concatenación. Una solución simple y bastante popular, es usar un vector predefinido que asignaremos a las palabras para los que un word embedding no tenga representación. Se suele usar un vector formado por ceros o el vector obtenido al realizar la media de todos los vectores del word embedding. Pero este método puede alterar de forma negativa el conocimiento del nuevo word embedding generado. Pongamos un ejemplo para explicar esto, imaginemos que tenemos tres word embeddings, y queremos concatenarlos para calcular la similitud entre las palabras A y B usando la similitud coseno. Como se puede observar en la Tabla 6.2. Los vectores de los tres word embedding son muy similares, de hecho el primer y el tercer word embedding obtienen un resultado casi idéntico al calcular la similitud coseno entre A y B. Por otro

lado el segundo word embedding no cuenta con representación para la palabra *B*, aunque su representación para *A* es muy similar a la de los otros dos word embedding. Concatenamos los tres word embeddings y en el caso de el segundo embedding decidimos asignarle un vector con ceros como representación de la palabra *B* para poder realizar la concatenación. El resultado que obtenemos varia de forma significativa respecto los obtenido individualmente por el primer y el tercer word embedding pese a que todos los vectores son muy similares. Esto ha ocurrido por los ceros que hemos añadido, estos ceros han provocado que la palabra *B* se encuentre en un espacio vectorial diferente a la palabra *A*. Por lo tanto, escoger un vector predefino a las palabras fuera de la intersección del vocabulario de los word embedding que queremos concatenar provoca que dichas palabras sean situadas en espacios vectoriales diferentes a las palabras que se encuentran dentro de la intersección. Cualquier cálculo de similitud que involucre a dichas palabras dará como resultado un valor aleatorio que dependerá del espacio vectorial en el que se encuentren, ya que mediante este método estamos modificando tanto la distancia como el ángulo que forman las palabras.

Palabras	Embedding 1	Embedding 2	Embedding 3	Concatenación
A	[1, 0.5]	[1, 0.55]	[1.1, 0.55]	[1, 0.5, 1.0, 0.55, 1.1, 0.55]
B	[1, 0.6]	-	[0.9, 0.6]	[1, 0.6, 0, 0, 0.9, 0.6]
Similitud Coseno	0.997	-	0.992	0.817

Tabla 6.2: Ejemplo de la problemática al asignar vectores predefinidos al concatenar word embeddings.

Queda claro que necesitamos un método mejor para asignar un vector a las palabras para las que un embedding no cuente con representación. Por ello hemos implementado un algoritmo que permite aproximar el vector para cualquier palabra perteneciente a la unión del vocabulario de los word embedding que queremos concatenar. El método está inspirado por “Robert Speer y Joshua Chin” [70]. Dada una palabra W que no se encuentra en la intersección de los vocabularios de los word embeddings, para el embedding que no cuenta con representación de W, W se calcula como la media aritmética de las K palabras más cercanas (usando la similitud coseno como medida) pertenecientes a la intersección de los vocabularios. En nuestro caso, hemos decidido usar las 10 palabras más cercanas (K=10). Es decir, usaremos un embedding que cuente con representación para W para calcular las diez palabras más cercanas. Generaremos W como la media de los vectores del embedding que no cuenta con representación para W de las diez palabras más cerca-

nas. Dado que las diez palabras deben contar con representación en el embedding para el que queremos aproximar W, las diez palabras deben pertenecer al vocabulario común entre los word embeddings que queremos concatenar, es decir, a la intersección de los vocabularios. Si queremos concatenar más de 2 word embeddings las diez palabras podrían pertenecer a la intersección entre el embedding para el que queremos generar la palabra y el embedding que usaremos para calcular las palabras más cercanas, pero para hacer la implementación del algoritmo lo más eficiente posible hemos decidido que pertenezcan a la intersección de los vocabularios de todos los word embeddings que queremos concatenar. De esta forma no necesitamos calcular la intersección de los vocabularios cada vez que queremos generar una palabra, si no que podemos calcular el vocabulario común entre todos los embedding que queremos concatenar al comienzo de la concatenación. También nos permite generar desde el comienzo una matriz para cada word embedding formada por los vectores de las palabras pertenecientes a la intersección de los vocabularios y de esta forma no tenemos que hacer búsquedas de palabras en los word embeddings a la hora de calcular las diez palabras más cercanas.

A continuación se detalla el algoritmo en pseudocódigo

```
1
2  \\Entrada:
3      //word: Palabra para la que queremos generar un vector aproximado. Debe pertenecer a la unión
         de los vocabularios de los embeddings que queremos concatenar. Es decir, al menos uno de
       los embeddings que queremos concatenar debe contar con una representación para dicha palabra
       .
4      //embedding: Embedding para el que queremos generar la palabra
5      //embeddings_list: Lista de embeddings que queremos concatenar
6      overlapping_vocabulary: Lista de palabras que contiene la intersección de los vocabularios de
         los word embeddings que vamos a concatenar.
7  \\Salida: Un vector aproximado para Word en embedding.
8
9  \\Funciones:
10     //Buscar(P,Emb). Devuelve el vector que representa a la palabra P en el embedding Emb. Si el
         embedding no cuenta con una representación para P, devuelve el valor nulo.
11     //Similitud_Coseno(V1,V2). Devuelve la similitud coseno entre los vectores V1 y V2. En caso
         de que V1 o V2 sean el valor nulo, devuelve el valor nulo.
12     //Insertar_Ordenado(L, W, V): Inserta de mayor a menor W en la lista L en función del valor V
       .
13     //Palabras_cercanas(V,E): Dado un vector V devuelve una lista las palabras del embedding E
       ordenadas de mayor a menor en función de la similitud coseno.
14 función generate_word(word, embedding, embeddings_list, overlapping_vocabulary):
15
16
17 vector = Nulo
18 emb_n = Nulo
19 \\Buscamos el primer word emebdding que cuente con representación para word.
20 Para x desde 1 hasta longitud(embeddings_list):
21     vector = Buscar(word, embeddings_list[x])
22     emb_n = x
23     Si vector no es nulo:
24         Salir del bucle
25
26
27 top = Palabras_cercanas(vector,embeddings_list[emb_n])
28
29 \\Buscamos los vectores para las 10 palabras más cercanas en el embedding para el que queremos
       generar la palabra. El resultado será la media aritmética de dichos vectores.
30 m = [0,0,0...,0]
31 Para x desde 1 hasta 10:
32     m = m + Buscar(top[x], embedding)
33
34 Devolver m/10
35
36
37 //Nota: Para facilitar la legibilidad del algoritmo se han omitido las comprobaciones de datos y
       errores.
```

Este método nos permitirá generar representaciones aproximadas para cualquier palabra perteneciente a la unión de los vocabularios de los embedding a concatenar. Esperamos que dichas aproximaciones sean una mejora significativa respecto a usar vectores predefinidos para las palabras fuera de la intersección de los vocabularios de los word embeddings que queremos concatenar. Existen otras soluciones para generar vectores. Por ejemplo "1ton+" [84] que hace uso de redes neuronales, o el uso de bases de conocimiento como ConceptNet [75], usada para generar el word embedding Numberbatch [69]. Sin embargo, puesto que en el siguiente capítulo propondremos un método de generación de meta-embeddings que soluciona esta problemática, y debido a que esta forma de aproximación obtiene un buen rendimiento no entraremos en estudiar los diferentes métodos de generación de vectores para palabras para las que un word embedding no cuenta con representación.

Para comprobar la eficacia de este método hemos realizado un prueba. Hemos escogido tres palabras, "dog" (perro), "car" (coche) y "love" (amor). Hemos eliminado estas palabras del word embeddings Glove. A continuación hemos usado el algoritmo de generación de palabras junto al word embedding FastText, para generar aproximaciones para las palabras que hemos eliminado de Glove. Para comprobar como de buenas son estas aproximaciones hemos calculado la similitud coseno entre las palabras generadas y la palabra original. Los resultados se muestran en la Tabla 6.3. Como se puede observar, el resultado no es perfecto, pero tenemos una similitud entre palabras muy alta. Por lo tanto, esperamos que estas aproximaciones nos sirvan para generar meta embeddings mejores a otras alternativas como la asignación de un vector predefinido.

Palabra	10 palabras más cercanas (usando FastText)	Similitud coseno (generada-original)
dog	dogs, puppy, Dog, pet, cat, canine, puppies, terrier, pup, Dogs	0.929
car	cars, Car, vehicle, automobile, Cars, truck, vehicles, auto, driving, dealership	0.881
love	loved, loving, adore, LOVE, loves, Love, luv, hope, looove, loooove	0.900

Tabla 6.3: Similitud coseno entre las palabras generadas mediante el algoritmo de aproximación y las palabras originales del word embeddings.

Algoritmo de concatenación

Finalmente mostraremos el algoritmo usado para realizar la concatenación de word embeddings.

```
\\Entrada:
    //lista_embeddings: Lista que contiene los word embeddings que queremos concatenar
    //lista_pesos: El peso que queremos dar a cada uno de los word embeddings en la concatenación
     , por defecto será [1,1,1,1....,1], es decir, daremos a todos los word embeddings el mismo
     peso.
    //normalizar: True si queremos aplicar una normalización L2 a los vectores que vamos a
     concatenar. False en caso contrario.
\\Salida:
    //Word embedding cuyo vocabulario es la unión de los vocabularios de los embeddings que hemos
      concatenado y cuyos vectores son la concatenación de los vectores que representan a cada
     palabra de cada word embedding original.

\\Funciones:
    //Buscar(P,Emb). Devuelve el vector que representa a la palabra P en el embedding Emb. Si el
     embedding no cuenta con una representación para P, devuelve el valor nulo.
    //intersección(EMBS): Recibe como entrada una lista de embeddings. Devuelve una lista de
     palabras que representa la intersección de los vocabularios de la lista de embeddings.
    //unión(EMBS): Recibe como entrada una lista de embeddings. Devuelve una lista de palabras
     que representa la unión de los vocabularios de la lista de embeddings
    //generate_word(word, embedding, embeddings_list, overlapping_vocabulary): Función descrita
     en  la sección anterior.
    //normalizar_L2(v): Recibe como entrada un vector v y devuelve el vector v normalizado
     mediante la norma L2
    //concatenar(V1,V2): Recibe como entrada dos vectores, devuelve como salida V2 concatenado a
     V1.
    //añadir(L,E): Recibe como entrada la lista L y el elemento E. Devuelve como salida L con el
     elemento E añadido al final de la lista.


función concatenar_embeddings(lista_embeddings, lista_pesos, normalizar):
    overlapping_vocabulary = intersección(lista_embeddings)
    vocab = unión(lista_embeddings)

    vectores = []

    Para cada palabra w de vocab:
        vector = []
        Para e desde 1 hasta longitud(lista_embeddings):
            v = Buscar(w, lista_embeddings[e])
            Si v es nulo:
                v = generate_word(w,lista_embeddings[e],lista_embeddings,overlapping_vocabulary)

            Si normalizar:
                v = normalizar_L2(v)

            vector = concatenar(vector, v * lista_pesos[e])
        vectores = añadir(vectores,vector)

    Devolver vocab, vectores

//Nota: Para facilitar la legibilidad del algoritmo se han omitido las comprobaciones de datos y
     errores.
```

6.1.2. Resultados obtenidos

Esta sección analizaremos los resultados obtenidos por las diferentes concatenaciones de word embeddings. La notación que vamos a utilizar para las diferentes pruebas es la siguiente:

$$\text{embedding\_1} + \text{embedding\_2} + \text{embedding\_3} + ...$$

Esto quiere decir que se ha concatenado el embedding_1 con el embedding_2 y el embedding_3 y se muestra el rendimiento obtenido por la concatenación. Los pesos que se han dado a cada embedding en la concatenación se representan de la siguiente forma:

$$[1,2,4...]$$

Esto quiere decir que al *embedding_1* se le ha dado un peso *1*, al *embedding_2* un peso *2*, al *embedding_3* un peso *4* y así sucesivamente. En caso de que no se muestren los pesos significa que se ha dado a todos los word embeddings un peso de *1*.

La metodología para las pruebas realizadas es la detallada en la Sección 3.1, Las tablas con los resultados que analizaremos en esta sección cuentan con 4 columnas. En 2 de ellas se muestra la media y la media ponderada obtenida para facilitar futuras comparaciones. Y en las 2 restantes se muestra la media y la media ponderada de forma relativa al mejor word embedding que se ha incluido en la concatenación. De esta forma podemos ver de forma sencilla si la concatenación de word embeddings mejora el rendimiento de los embeddings originales.

Como hemos mencionado anteriormente, dividiremos el análisis en varias partes. Primero comenzaremos estudiando si el método de generación de palabras presentado resulta útil o no. A continuación analizaremos el rendimiento de concatenaciones de dos word embeddings para estudiar que word embeddings consiguen un mayor rendimiento al ser concatenados. Por último analizaremos que ocurre cuando concatenamos tres o más word embeddings y el efecto de la normalización L_2 y la asignación de pesos.

Rendimiento del método de generación de palabras

En esta sección analizaremos el rendimiento del método de generación de palabras. Para ello compararemos el rendimiento de diferentes concatenaciones de word embeddings

asignando un vector predefinido formado por ceros cuando uno de los word embeddings no disponga de representación para alguna palabra para la que otro word embedding si que cuenta con representación. Y el rendimiento de aproximar una representación para dicha palabra usando el método descrito en la sección anteriormente. Para realizar esta comparativa vamos a ha analizar el rendimiento de diferentes concatenaciones en datasets para los que los word embeddings que vamos a concatenar no cuentan con representación para todas las palabras. De esta forma podremos comprobar la diferencia de rendimiento entre asignar vectores predefinidos formados por ceros y nuestro algoritmo de aproximación de palabras.

En la Tabla 6.4 se muestra el rendimiento de diferentes concatenaciones de word embeddings en los datasets RareWords (RW) [59] y MTurk 287 [54]. Dichos dataset han sido elegidos por que por que tienen número importante de palabras para los que algunos word embeddings no pueden dar respuesta ya que no cuentan con representación para una o ambas palabras del par. Como usando ambos métodos el vocabulario final de la concatenación es el mismo no nos interesa que los pares para los que no podemos dar respuesta influyan en los resultados, a si que la métrica usada ha sido la primera descrita en la Sección 3.1. Descartar los pares de palabras para los que no se puede dar respuesta. En la tabla se muestran los resultados obtenidos en ambos dataset y a continuación, para facilitar la comparativa, se incluye una columna donde se muestra la diferencia de rendimiento entre el método de generación de palabras y el método de asignación de ceros. Como apoyo ha esta tabla, en la Tabla 10.3 (anexo) se muestra el porcentaje de pares de palabras para los que cada word embedding puede dar respuesta.

Como se puede comprobar en la Tabla 6.4, cuando juntamos un word embedding que cuenta con un vocabulario pequeño como UKB con uno o varios que cuentan con vocabularios muy extensos como FastText o Glove, el método de generación de palabras supone una mejora muy importante respecto a asignar un vector predefinido. Cuando asignamos un vector predefinido provocamos que dicha palabra termine en un espacio vectorial diferente al resto, por lo tanto cualquier cálculo que use esa palabra queda desvirtuado. El método de aproximación de palabras aunque genere representaciones aproximadas, esas representaciones son significativamente mejores que las generadas asignando vectores predefinidos. En el caso de que concatenemos word embeddings con vocabularios muy extensos entre ellos, como FastText con Glove, va a haber un número reducido de palabras fuera de la intersección entre el vocabulario de los word embeddings. Por lo tanto el método que elijamos apenas tendrá impacto en los resultados obtenidos en los datasets, ya que habrá muy pocos casos en los que tengamos que asignar un vector predefinido a

Embeddings	RW (ceros)	RW (Generar)	Diferencia	MTurk 287 (Ceros)	MTurk 287 (Generar)	Diferencia
FT+UKB	0.471	0.565	0.094	0.680	0.724	0.044
UKB+lexvec	0.328	0.484	0.156	0.565	0.616	0.051
FT+GLOVE	0.591	0.595	0.004	0.729	0.726	-0.003
FT+lexvec	0.594	0.594	0.000	0.733	0.733	0.000
UKB + LEXVEC + W2V	0.394	0.515	0.121	0.626	0.701	0.075
FT + JointcHYB + UKB + LEXVEC + GLOVE + W2V + SKE + PDC	0.464	0.498	0.034	0.708	0.714	0.006

Tabla 6.4: Comparación del rendimiento en diferentes dataset entre el método de generación de palabras y la asignación de un vector predefinido (ceros) para las palabras para los que un word embedding de la concatenación no cuenta con representación pero al menos un word embedding si cuenta con representación para dicha palabra. Métrica usada: descartar pares para los que no se puede dar respuesta.

una palabra o aproximar una palabra.

Los resultados obtenidos muestran que el método de generación de palabras es significativamente mejor que la asignación de vectores predefinidos. Por lo tanto durante toda la sección usaremos este método cuando durante la concatenación de word embeddings al menos un word embedding cuente con representación para una palabra pero un o varios word embeddings no cuenten con representación para la palabra.

Rendimiento de la concatenación de dos Word Embeddings

El objetivo de esta sección es analizar que efectos tiene en el rendimiento la concatenación de diferentes word embeddings. Queremos comprobar si se obtienen mejores resultados combinando cierto tipo de embeddings entre si, como ocurría en el Capítulo 5, o el mejor rendimiento se obtiene combinando los word embeddings que mejor rendimiento obtiene por separado. Para esto, evaluaremos el rendimiento obtenido (media y media ponderada) de cada word embedding concatenándolo con el resto de word embeddings disponibles.

Comenzaremos con el embedding Glove. En la Tabla 6.5 se muestra el rendimiento de las concatenaciones de Glove con el resto de word embeddings. Para realizar esta tabla se ha usado la primera métrica descrita en la Sección 3.1, donde se descartan las palabras para las que no podemos dar respuesta. Por lo tanto, no se penaliza que los word embeddings no sean capaces de dar respuesta para un determinado número de pares de palabras.

Embeddings	Media (Diferencia)	Media Ponderada (Diferecia)	Media	Media Ponderada
GLOVE + W2V	-0.017	-0.018	0.612	0.553
GLOVE + FT	0.000	0.000	0.675	0.620
GLOVE + jointcHYB	-0.001	-0.005	0.653	0.601
GLOVE + UKB	0.002	-0.013	0.632	0.597
GLOVE + LEXVEC	-0.002	-0.001	0.628	0.572
GLOVE + SKE	-0.019	-0.026	0.610	0.545
GLOVE + PDC	-0.031	-0.045	0.598	0.526

Tabla 6.5: Comparación del rendimiento relativo entre las concatenaciones de Glove con el resto de word embeddings y el rendimiento del mejor word embedding de la concatenación. Métrica usada: descartar pares para los que no se puede dar respuesta.

En la Tabla 6.5 podemos observar un comportamiento muy interesante. Ninguna de las concatenaciones obtiene una mejora de rendimiento respecto al mejor word embedding de la concatenación. Sin embargo si nos fijamos en los resultados obtenidos podemos observar que lo que está ocurriendo es que Glove está siendo dominado por los word embeddings que estamos concatenando. El rendimiento de la concatenación es muy similar al rendimiento del word embedding que estamos concatenando a Glove. Por ejemplo, FatText individualmente obtenía una media de "0.675" en los diferentes dataset y una media ponderada de "0.620" exactamente los resultados que obtiene la concatenación de Glove y FastText. LexVec obtenía una media de "0.630" y media ponderada de "0.573" resultados casi idénticos a los obtenidos por la concatenación de Glove y LexVec. Lo mismo ocurre con el resto de concatenaciones. Glove se encuentra en una escala mucho menor que el resto de word embeddings por lo tanto al realizar la concatenación estamos perdiendo todo el conocimiento que contiene Glove.

Una solución ha este problema es la asignación de pesos, podemos escalar GLOVE multiplicando todos sus vectores por un valor. Como se puede observar en la Tabla 6.6 si asignamos un peso muy alto a GLOVE podemos evitar que sea dominado por los word embedding con los que lo estamos concatenando. Y en este caso obtenemos una mejora de rendimiento significativa, similar a la que obteníamos en la Sección 5.2 con el método

de la media de las similitudes.

Embeddings	Pesos	Media (Diferencia)	Media Ponderada (Diferecia)	Media	Media Ponderada
GLOVE + jointcHYB	[1000,1]	0.039	0.033	0.693	0.639
GLOVE + UKB	[400,1]	0.067	0.038	0.697	0.648

Tabla 6.6: Comparación del rendimiento relativo entre las concatenaciones de Glove con UKB y jointcHYB realizando una asignación de pesos y el rendimiento del mejor word embedding de la concatenación. Métrica usada: descartar pares para los que no se puede dar respuesta.

Vamos a analizar ahora que ocurre con el embedding Word2Vec, vamos a concatenarlo con el resto de word embeddings (exepto Glove ya que los resultados ya se encuentran en la Tabla 6.5). Los resultados se encuentran en la Tabla 6.7.

Embeddings	Pesos	Media (Diferencia)	Media Ponderada (Diferecia)	Media	Media Ponderada
W2V + FT	[1,1]	-0.006	-0.009	0.669	0.611
W2V + jointcHYB	[1,1]	0.011	0.009	0.665	0.615
W2V + jointcHYB	[2,1]	0.028	0.025	0.682	0.631
W2V + UKB	[1,1]	0.053	0.029	0.683	0.639
W2V + LEXVEC	[1,1]	-0.001	-0.004	0.629	0.569
W2V + LEXVEC	[1,2]	0.011	0.007	0.641	0.580
W2V + SKE	[1,1]	0.017	0.010	0.633	0.565
W2V + PDC	[1,1]	-0.004	-0.014	0.612	0.541
W2V + PDC	[2,1]	0.012	0.005	0.628	0.560

Tabla 6.7: Comparación del rendimiento relativo entre las concatenaciones de Word2Vec con el resto de word embeddings y el rendimiento del mejor word embedding de la concatenación. Métrica usada: descartar pares para los que no se puede dar respuesta.

En la Tabla 6.7 podemos ver como cuando concatenamos Word2Vec y UKB obtenemos una mejora de rendimiento significativa, al igual que ocurría cuando los combinábamos con con el método de la media de las similitudes (Tabla 5.9). De hecho la puntación obtenida es similar. Word2Vec y UKB se encuentran en una escala similar, por lo tanto

concatenándolos directamente obtenemos buenos resultados. Sin embargo esto no ocurre con todos los Word Embeddings. Al concatenar Word2Vec y jointcHYB, obtenemos una mejora de rendimiento considerablemente inferior a la que conseguíamos con el método de la media de las similitudes. Lo que está ocurriendo es que jointcHYB está dominando a Word2Vec haciendo que se pierda su información. Si multiplicamos todos los vectores de Word2Vec por dos, es decir, le damos el doble de peso que a jointcHYB conseguimos evitar que jointcHYB domine a Word2Vec y obtenemos una mejora de rendimiento mayor. De nuevo los resultados obtenidos por la concatenación de Word2Vec y jointcHYB dando el doble de peso a Word2Vec obtiene un rendimiento similar al que conseguíamos con el método de la media de las similitudes (Tabla 5.9). Ocurre lo mismo con la concatenación de Word2Vec con Lexvec y la concatenación con Word2Vec y PDC.

Vamos a comprobar con otros word embeddings si esto mismo se repite. En la Tabla 6.8 se muestran los resultados de diferentes concatenaciones de word embeddings. Esta vez tenemos dos columnas extra, en la primera se muestra la diferencia entre la media y la media ponderada obtenida por la concatenación y el rendimiento del mejor word embedding de la concatenación. Y en la segunda, para facilitar la comparativa, se muestra la diferencia de rendimiento entre la concatenación y los mismos word embeddings combinados mediante el método de la media de las similitudes de la sección Sección 5.2 (Se ha tomado siempre el rendimiento de la combinación dando a todos los word embeddings el mismo peso). En la tabla podemos observar algo similar a lo que ocurría con Glove. Tenemos casos donde la concatenación de dos word embeddings produce un nuevo word embedding con un rendimiento inferior al mejor word embedding empleado en la concatenación, esto ocurre por que uno de los word embeddings está dominando al otro al estar en una escala mayor. Escalando uno de los word embeddings conseguimos evitar esto. Los resultados que obtenemos escalando el word embedding que está siendo dominado son muy similares a los obtenidos mediante el método de la media de las similitudes de la Sección 5.2.

Lo que estamos observando, es que concatenar directamente dos word embeddings en general no produce buenos resultados, ya que en muchos casos un word embedding domina al otro al estar en una escala mayor. Por lo tanto, estamos perdiendo el conocimiento de uno de los word embeddings de la concatenación. Modificando el peso que damos a cada word embedding de la concatenación, es decir, escalando uno de los word embeddings, podemos evitar que esto ocurra, y en general obtenemos un mejor rendimiento. Lo que conseguimos al escalar uno de los word embeddings son resultados bastante similares a los que obteníamos con el método de la media de las similitudes (Sección 5.2). En di-

Embeddings	Pesos	Respecto a mejor word embedding		Respecto a media de las similitudes		Media	Media Ponderada
		Media (Diferencia)	Media Ponderada (Diferencia)	Media (Diferencia)	Media Ponderada (Diferencia)		
FT + jointcHYB	[1,1]	0.003	0.009	-0.026	-0.025	0.678	0.629
FT + jointcHYB	[2,1]	0.029	0.037	0.000	0.003	0.704	0.657
FT + UKB	[1,1]	0.037	0.046	0.001	0.007	0.712	0.666
FT + UKB	[2,1]	0.033	0.035	-0.003	-0.004	0.708	0.655
jointcHYB + LEXVEC	[1,1]	0.003	0.001	-0.032	-0.028	0.657	0.607
jointcHYB + LEXVEC	[1,2]	0.012	0.011	-0.023	-0.018	0.666	0.617
jointcHYB + LEXVEC	[1,8]	0.035	0.032	0.000	0.003	0.689	0.638
jointcHYB + SKE	[1,1]	0.010	0.005	-0.018	-0.011	0.664	0.611
jointcHYB + SKE	[1,2]	0.023	0.021	-0.005	0.005	0.677	0.627
jointcHYB + PDC	[1,1]	0.024	0.014	0.003	0.006	0.678	0.619
jointcHYB + PDC	[2,1]	0.011	0.005	-0.01	-0.003	0.665	0.611
UKB + LEXVEC	[1,1]	0.027	0.010	-0.033	-0.017	0.657	0.620
UKB + LEXVEC	[1,2]	0.059	0.036	-0.001	0.009	0.689	0.646
UKB + PDC	[1,1]	0.021	-0.022	-0.022	-0.03	0.651	0.588
UKB + PDC	[2,1]	0.051	0.017	0.008	0.009	0.681	0.627
SKE + PDC	[1,1]	0.005	-0.005	-0.022	-0.02	0.615	0.540
SKE + PDC	[2,1]	0.024	0.010	-0.003	-0.005	0.634	0.555

Tabla 6.8: Comparación del rendimiento de diferentes concatenaciones de word embeddings respecto a el mejor word embedding de la cancatenación y el método de la media de las similitudes. Métrica usada: descartar pares para los que no se puede dar respuesta.

cho método para cada par de palabras, se calculaba la similitud coseno usando diferentes word embeddings, y a continuación se realizaba la media aritmética de las similitudes calculadas usando cada word embedding, por defecto todos los word embeddings tenían el mismo peso en la media.

Para comprobar las diferencias en la escala de cada word embedding hemos realizado una prueba. Cada cada word embedding hemos calculado la media de la longitud de todos los vectores, siendo la longitud su norma L_2. En la Tabla 6.9 se puede observar que hay diferencias importantes entre word embeddings, siendo GLOVE (habiendo normalizado sus columnas mediante la norma L_2) el embedding que más difiere del resto. También podemos observar que los pesos que mejor rendimiento nos proporcionaban en las concatenaciones, coinciden con la diferencia en la escala entre embeddings. Por ejemplo, al concatenar Word2Vec y jointcHYB obteníamos un mejor rendimiento dando el doble de peso a jointcHYB, en la Tabla 6.9 podemos observar como la media de la longitud de los vectores de jointcHYB es dos veces mayor que la de Word2Vec.

EMBEDDING	LONGITUD
GLOVE	0.011
W2V	2.040
FT	5.958
jointcHYB	4.491
UKB	4.926
LEXVEC	2.652
SKE	3.718
PDC	9.379

Tabla 6.9: Media de la longitud de todos los vectores de cada word embedding, siendo la longitud su norma L_2

Esto es un claro indicativo de que la mejor forma de realizar la concatenación de word embeddings es dando a todos los word embedding de la concatenación el mismo peso. La forma más sencilla y precisa de realizar esto es mediante la normalización L_2 de los vectores antes de la concatenación. De esta forma conseguimos que todos los vectores tengan la misma longitud, uno, y gracias a ello el conocimiento de todos los word embeddings va a contribuir de la misma forma en el word embedding resultante. No vamos a encontrarnos con una situación en la que un word embedding domine a otro. Como hemos mostrado anteriormente, la normalización L_2 de los vectores provoca que la concatenación sea equivalente a la media de las similitudes expuesta en la Sección 5.2. Por lo tanto, no tiene sentido repetir lo expuesto en dicha sección, rendimiento y conclusiones son las mismas. Sin embargo vamos a mostrar algunas comparativas, para comprobar la diferen-

Embeddings	Concatenación	Concatenación + Ajuste de pesos	L_2+ Concatenación	Media de las similitudes
GLOVE + JointcHYB	0.653	0.670	**0.691**	0.690
FT + UKB	**0.712**	0.703	0.710	0.711
JointcHYB + W2V	0.665	0.684	**0.685**	0.682
W2V + UKB	0.683	0.670	**0.687**	0.683
JointcHYB + LEXVEC	0.657	0.663	**0.691**	0.689
UKB + LEXVEC	0.657	0.686	**0.693**	0.690
FT + LEXVEC	**0.671**	0.661	0.666	0.666

Tabla 6.10: Comparativa de la media obtenida en los datasets por diferentes métodos de combinación de word embeddings. Métrica utilizada: Ignorar pares de palabras para los que no se puede dar respuesta.

cia de rendimiento entre los diferentes métodos con los que estamos experimentado. En la Tabla 6.10 se muestran 4 columnas. En la primera se muestra el rendimiento (media de la puntuación obtenida en los dataset) de la concatenación sin realizar ninguna normalización. En la segunda columnas hemos dado pesos a los word embeddings usando los datos de la Tabla 6.9. El word embedding cuya media de la longitud de los vectores es mayor ha recibido peso uno, y el segundo embedding ha recibido como peso la división entre la media de la longitud de los vectores del embedding mayor dividida entre la suya. De esta forma esperamos escalar el embedding menor a la misma escala que el embedding mayor. En la tercera columna se muestra el rendimiento obtenido por la concatenación habiendo aplicado la normalización L_2 a los vectores de los word embedding. Y en la cuarta columna se muestra el rendimiento obtenido aplicando el método de la media de las similitudes.

Como se puede observar, tanto la media de las similitudes como la normalización L_2 de los vectores antes de la concatenación obtienen mejor rendimiento que la concatenación, incluso aunque tratemos de ajustar los pesos para compensar la diferencia en la escala de los word embedding. En esta tabla se han añadido algunos resultados que consideramos interesantes, si se quiere comprobar la diferencia de rendimiento de más concatenaciones basta con comparar los resultados de obtenidos en esta sección con los obtenidos en la

Sección 5.2. Los únicos casos donde la concatenación sin aplicar normalizaciones consigue un mejor rendimiento es cuando combinamos dos word embedding calculados a partir de corpus de textos o dos word embeddings calculados a partir de corpus de grafos. En estos casos como vimos en la Sección 5.2 no se suele conseguir un aumento de rendimiento, por lo tanto dar más peso al mejor embedding provoca que el rendimiento del embedding final esté más cerca del mejor embedding y la pérdida de rendimiento sea mejor, como es el caso de la concatenación de FT y LEXVEC que vemos la Tabla 6.10, pero las combinaciones de embeddings donde perdemos rendimiento no nos interesan. Buscamos conseguir mejores word embeddings.

Aunque hemos asegurado que la concatenación de word embeddings normalizados mediante la norma L_2 es equivalente a el método de la media de las similitudes en la Tabla 6.10 podemos observar algunas pequeñas variaciones en el rendimiento. Esto se debe a como tratamos las palabras fuera de la intersección del vocabulario de los word embedding que estamos combinando. En el caso de la media de la similitudes hacíamos la media con los resultados aportados por los word embeddings que podían dar respuesta para el par de palabras que queremos evaluar, es decir, si tenemos cuatro word embeddings pero solo tres pueden dar respuesta para un determinado par de palabras, haremos la media aritmética entre los resultados calculados con los tres word embeddings que pueden dar respuesta. En el caso de la concatenación, cuando un embedding no puede dar respuesta para un determinado par de palabras por que no cuenta representación para una o ambas utilizamos un word embedding que si cuente con representación para generar un vector aproximado. Para terminar nuestro estudio de la concatenación de word embeddings vamos hacer una comparativa entre ambos métodos. En el caso de la media de las similitudes no estamos introduciendo ningún error al no realizar aproximaciones, por lo tanto nos dará un muy buen indicativo de lo bueno que es nuestro algoritmo de aproximación. Para esta comparativa vamos a utilizar combinaciones de word embeddings donde al menos uno de los word embedding no tenga un vocabulario muy extenso (como por ejemplo UKB), ya que será en los casos donde la diferencia entre la intersección y la unión de los vocabularios de los word embedding a combinar será mayor, y por lo tanto las diferencias entre ambas formas de tratar las palabras para las que un word embedding no cuenta con representación y el otro si serán mayores.

Como se puede observar en la Tabla 6.11 el método de aproximación de palabras da muy buenos resultados, incluso superiores a descartar en la media de las similitudes los embedding que no pueden dar respuesta para el par de palabras que queremos evaluar. Aunque las palabras generadas no sean representaciones exactas resultan ser mejores que no to-

	Media de las similitudes		L2 + Concatenación	
Embedding	MTurk-287	RW	MTurk-287	RW
GLOVE + UKB	0.715	0.495	**0.734**	**0.542**
W2V + UKB	0.681	0.493	**0.712**	**0.534**
FT + UKB	**0.717**	0.539	0.716	**0.572**
UKB + LEX-VEC	0.721	0.513	**0.732**	**0.548**
W2V + jointcHYB	0.695	0.473	**0.701**	**0.485**
UKB + SKE	**0.730**	**0.527**	0.721	0.503

Tabla 6.11: Comparativa entre el método de la media de las similitudes y la concatenación de embedding normalizados. Métrica usada: Descartar pares para los que no se puede dar respuesta.

mar en cuenta el embedding cuando no pueda dar respuesta para un par de palabras para las que otro embedding si puede dar respuesta. Por lo tanto el algoritmo de generación de palabras resulta ser un método que consigue un muy buen rendimiento a la hora de combinar word embeddings. Sin embargo este método tiene un problema, y es que resulta computacionalmente muy demandante, tenemos que buscar las diez palabras más cercanas en una matriz que puede contener millones de palabras. Nuestra implementación es capaz de calcular varios vectores aproximados por segundo. Pero si queremos concatenar dos embedding donde la diferencia entre la unión y la intersección de vocabularios es muy grande, como puede ser el caso de concatenar UKB (vocabulario poco extenso) con FastText (vocabulario muy extenso) generar el embedding concatenado completo con el equipo usado para estas pruebas puede llevar hasta 40 horas, ya que hay que generar muchos miles de vectores aproximados. Aunque si no es necesario calcular un meta embedding completo, por que por ejemplo solo nos interesan las palabras de un dataset, nuestra implementación puede generar el meta embedding parcial relativamente rápido, por ejemplo en nuestro caso el tiempo para generar un meta embedding que contenga solo las palabras usadas en los dataset que estamos usando para la evaluación tarda en general menos de 5 minutos, esto nos ha permitido realizar todas las pruebas expuestas en esta sección. Nuestra implementación utiliza tantos núcleos como núcleos tenga disponible la CPU en la que se ejecuta, por lo tanto es posible mejorar estos tiempos utilizando un equipo con más núcleos. Adicionalmente también son necesarios al menos 20GB de memoria ya que nuestro algoritmo usa una gran cantidad de memoria (se almacenan 2 matrices por cada word embedding que queremos concatenar) para tratar de ser lo más rápido posible.

Para dar por terminada esta sección creemos que es interesante mostrar la mejor concate-

nación obtenida, es decir, FT+UKB, con los mismos word embeddings aplicando la media de las similitudes y el rendimiento de los word embeddings individualmente en diferentes datasets. Estos resultados se muestran en la Tabla 6.12, donde se usa la primera métrica, en la que no se penaliza no poder dar respuesta para un determinado número de pares de palabras y la Tabla 6.13 donde se usa la segunda métrica con la que si se penaliza

Embedding	Media	Media Pon-dera-da	Sim Lex 999	MTurk 287	MTurk 771	MEN ALL	WS 353 all	WS 353 rela-ted-ness	WS 353 simi-larity	RG65	VERB 143	MC 30	YP 130	Sim Verb 3500	RW
FastText CC	0.675	0.620	0.503	**0.726**	0.762	0.837	0.791	0.737	0.840	0.863	0.447	0.852	0.625	0.426	**0.595**
UKB	0.630	0.610	0.525	0.644	0.679	0.756	0.720	0.627	0.785	0.823	0.153	0.873	0.768	**0.545**	0.443
FT+UKB (AVG Simil)	**0.711**	0.659	**0.563**	0.717	**0.779**	**0.847**	0.810	0.761	0.846	**0.888**	**0.484**	**0.878**	**0.773**	0.544	0.539
FT+UKB (L2 Concat)	0.710	**0.664**	**0.563**	0.716	0.778	**0.847**	**0.812**	**0.766**	**0.847**	**0.888**	0.441	**0.878**	**0.773**	0.544	0.572

Tabla 6.12: Comparativa entre de la combinación de FT+UKB con el método de la media de las similitudes (FT+UKB (AVG Simil)), FT+UKB aplicando la normalización L_2 seguida de la concatenación (FT+UKB (L2 Concat)) y los word embeddings usados en la combinación. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

Embedding	Media	Media Pon-dera-da	Sim Lex 999	MTurk 287	MTurk 771	MEN ALL	WS 353 all	WS 353 rela-ted-ness	WS 353 simi-larity	RG65	VERB 143	MC 30	YP 130	Sim Verb 3500	RW
FastText CC	0.674	0.618	0.503	**0.726**	0.762	0.837	0.791	0.737	0.840	0.863	0.447	0.852	0.625	0.426	**0.584**
UKB	0.596	0.568	0.525	0.516	0.678	0.756	0.705	0.612	0.769	0.823	0.064	0.873	0.768	**0.545**	0.305
FT+UKB (AVG Simil)	**0.711**	0.658	**0.563**	0.717	**0.779**	**0.847**	0.810	0.761	0.846	0.888	**0.484**	**0.878**	**0.773**	0.544	0.537
FT+UKB (L2 Concat)	0.710	**0.664**	**0.563**	0.716	0.778	**0.847**	**0.812**	**0.766**	**0.847**	**0.888**	0.441	**0.878**	**0.773**	0.544	0.572

Tabla 6.13: Comparativa entre de la combinación de FT+UKB con el método de la media de las similitudes (FT+UKB (AVG Simil)), FT+UKB aplicando la normalización L_2 seguida de la concatenación (FT+UKB (L2 Concat)) y los word embeddings usados en la combinación. Métrica utilizada: Resultado obtenido en cada datataset multiplicado por la cobertura.

Conclusiones

Durante esta sección hemos estudiado el rendimiento de diferentes métodos para la concatenación de word embeddings. La concatenación de word embeddings ha resultado ser un método que nos permite generar meta embedding cuyo rendimiento es superior de forma significativa al rendimiento de los word embeddings usados en la concatenación individualmente. El rendimiento que podemos obtener es el mismo o incluso ligeramente superior al método de la media de las similitudes expuesto en el capítulo anterior. Sin embargo cuenta con una ventaja importante, la concatenación genera un nuevo word embedding, que podremos usar en todo tipo de aplicaciones. Sin embargo para aplicar de forma efectiva este método tenemos que tener varios factores en cuenta.

El primero es que concatenar word embeddings que se encuentran en escalas diferentes puede provocar que un embedding domine al resto, haciendo que su información se pierde y reduciendo cualquier aumento de rendimiento que podamos obtener. Por lo tanto es necesario realizar un escalado para que todos los word embedding se encuentren en una misma escala. La forma más efectiva de hacer esto es normalizando mediante la norma L_2 la longitud de los vectores de los word embedding antes de realizar la concatenación. Esto provoca que todos los word embedding tengan exactamente el mismo peso en la concatenación. La normalización L_2 ha sido el método que mejor rendimiento ha obtenido.

A la hora de concatenar word embeddings hay que tratar las palabras para las que un word embedding cuenta con representación pero uno o varios no cuentan con representación. La asignación de un vector predefinido, método que se ha utilizado en algunos estudios previos no resulta ser una buena elección, ya que provoca que algunas palabras acaben en espacios vectoriales diferentes al resto de palabras, desvirtuando haciendo que cualquier operación que utilice esas palabras. Una mejor alternativa es buscar un método que nos permita generar aproximaciones de palabras. En esta sección hemos introducido un método de aproximación de palabras que obtiene un buen rendimiento en la concatenación de word embeddings. Nuestro algoritmo permite generar vectores para cualquier palabra que se encuentre en la unión de los vocabularios de los word embedding que se quieren concatenar.

Por último, puesto que la concatenación de vectores normalizados mediante la norma L_2 es equivalente a la media de las similitudes descrita en la Sección 5.2, todas las conclusiones a las que llegamos en dicha sección son aplicables a la concatenación. Las conclusión más importante a la que llegamos es que no se obtiene un mejor rendimiento por añadir el máximo número posible de word embeddings a la concatenación, ni concatenando

los embedding que mejor rendimiento obtengan. El mejor rendimiento se obtiene cuando combinamos word embeddings lo más diferentes posibles, en nuestro caso word embeddings calculados a partir de corpus de texto con word embeddings calculados a partir de grafos.

Aunque los resultados obtenidos en esta sección son muy positivos, la concatenación de word embeddings cuenta con un problema. Los meta embedding generados tienen tantas dimensiones como la suma de las dimensiones de los word embedding usados en la concatenación. Esto quiere decir que el tamaño de los meta embedding generados puede llegar a ser muy grande. La concatenación de dos word embedding puede llegar a producir un meta embedding de alrededor de 30GB. Concatenar más de 3 word embedding provoca que el word embedding generado sea prácticamente intratable debido a su gran tamaño. Un embedding de 1600 dimensiones hace que su uso sea muy difícil o incluso imposible. Por ello en las dos próximas secciones estudiaremos dos métodos con los que esperamos conseguir un rendimiento similar al obtenido en esta sección, pero generando word embeddings con un número de dimensiones inferior.

6.2. Reducción de dimensionalidad

Como hemos visto en la sección anterior, la concatenación produce buenos resultados, pero tiene un problema importante. El meta embedding generado tiene tantas dimensiones como la suma de las dimensiones de los word embeddings concatenados. Esto produce matrices de gran tamaño con las que resulta difícil trabajar. Sin embargo, gran parte de la información al concatenar word embeddings es redundante. Esto nos permite aplicar algoritmos de reducción de dimensionalidad que eliminan las posibles redundancias que pueden surgir al concatenar word embeddings y reducen las dimensiones de los word embedding, consiguiendo así word embedding de menor tamaño con los que resulta más sencillo trabajar. Puesto que estos métodos buscan reducir la información redundante, esperamos conseguir reducir la dimensionalidad de los meta embedding generados a partir de la concatenación con una pérdida de rendimiento baja.

6.2.1. Métodos de reducción de dimensionalidad:

Existen diferentes algoritmos de reducción de dimensionalidad. Hemos decidido estudiar tres de ellos. Los dos algoritmos más usados para reducir la dimensionalidad de word em-

beddings son PCA y SVD truncado, por lo tanto vamos a estudiar cual de ellos consigue un mejor resultado. Además, vamos a estudiar un tercer algoritmo que es una extensión de PCA, llamado DRA [51]. A continuación detallaremos cada uno de los algoritmos.

- PCA: El análisis de compotentes principales, abreviado como PCA, es una técnica que enfatiza la variación y busca patrones fuertes en un conjunto de datos. PCA realiza una transformación lineal ortogonal para convertir un set de variables correlacionadas en un set de variables linealmente independientes, llamados componentes principales. El primer componente principal representa la mayor variabilidad posible en en los datos, y cada componente subsiguiente representa la mayor variabilidad posible restante. PCA no es método que nos permita decidir que variables eliminar, lo que nos haría perder información, si no que construye un nuevo set de variables compuesto por los componentes principales. Por lo tanto estamos generando un nuevo word embedding que esperamos que sea un "resumen" lo más preciso posible del word embedding original. Existen diferentes formas de implementar PCA, la que hemos usada es la implementada en la librería scikit-learn [66]. Dicha implementación utiliza la descomposición de valores singulares o SVD para proyectar los datos a un espacio dimensional menor.

- SVD truncado o LSA: Este método es muy similar a PCA. La diferencia principal radica en que este método no realiza un centrado de los datos antes de calcular la descomposición de valores singulares. Al igual que con PCA, se ha usado la implementación de scikit-learn[66]. En caso de que aplicásemos este método a un word embedding que ha sido normalizado mediante el método de centrado de variables (explicado en el Capítulo 4) sería equivalente a PCA.

- DRA [51]: DRA es una extensión de PCA. En el Capítulo 4 presentamos un algoritmo de post-procesado de word embeddings llamado PPA. Dicho algoritmo se basa en la idea de al aplicar el método de centrado de variables a un word embedding, se crean un pequeño número de dimensiones dominantes que influencian la dirección de las palabras en una misma dirección, haciéndonos perder información. Eliminar estas direcciones genera representaciones de palabras más fuertes. Puesto que PCA implica realizar un centrado de variables, este método de reducción de dimensionalidad genera un nuevo word embedding donde un pequeño número de dimensiones dominan al resto. Por ello combinando PCA con PPA esperamos conseguir mejores word embeddings que usando PCA en solitario. DRA se compone de 3 pasos. Primero aplicamos el algoritmo de post-procesado PPA, por lo que realizamos un

centrado de variables y eliminamos las direcciones dominantes resultantes. Con este paso esperamos que el algoritmo PCA obtenga un mejor rendimiento. Tras esto aplicamos el método de reducción de dimensionalidad PCA. Con ello obtenemos un nuevo word embedding con un número de dimensiones menor. Sin embargo, es de esperar que este nuevo word embedding cuente de nuevo con un pequeño número de dimensiones dominantes, por lo que aplicamos de nuevo el algoritmo PPA para eliminarlas. Esperamos que el word embedding generado obtenga un mejor rendimiento que si aplicamos solo PCA. Como se recomienda por los autores de este algoritmo, y como hemos realizado en el Capítulo 4, usaremos PPA con el parámetro D = 7, es decir, eliminaremos las 7 direcciones dominantes.

```
\\Entrada:
    //X: Matriz que representa las palabras del word embedding
    //N: Nuevo número de dimensiones.
    //D: Número de direcciones dominantes a eliminar.

\\Salida:
    //Matriz X reducida a N dimensiones

    función DRA(X,N,D):
        //1: Aplicar el algoritmo de post-procesado:
        X = PPA(X,D).
        //2: Transformar X usando PCA:
        X = PCA(X,N)
        //3: Aplicar el algoritmo de post procesado:
        X = PPA(X,D)

        devolver X
```

En la Figura 6.1 podemos observar la varianza de los 20 componentes principales de GLOVE tras aplicar PPA+PCA reduciendo Glove a 150 dimensiones, y tras aplicar PPA+PCA (150 dimensiones) + PPA. Como se puede comprobar al aplicar PPA (D=7) al final del proceso conseguimos que el word embedding esté menos influido por un pequeño número de variables dominantes.

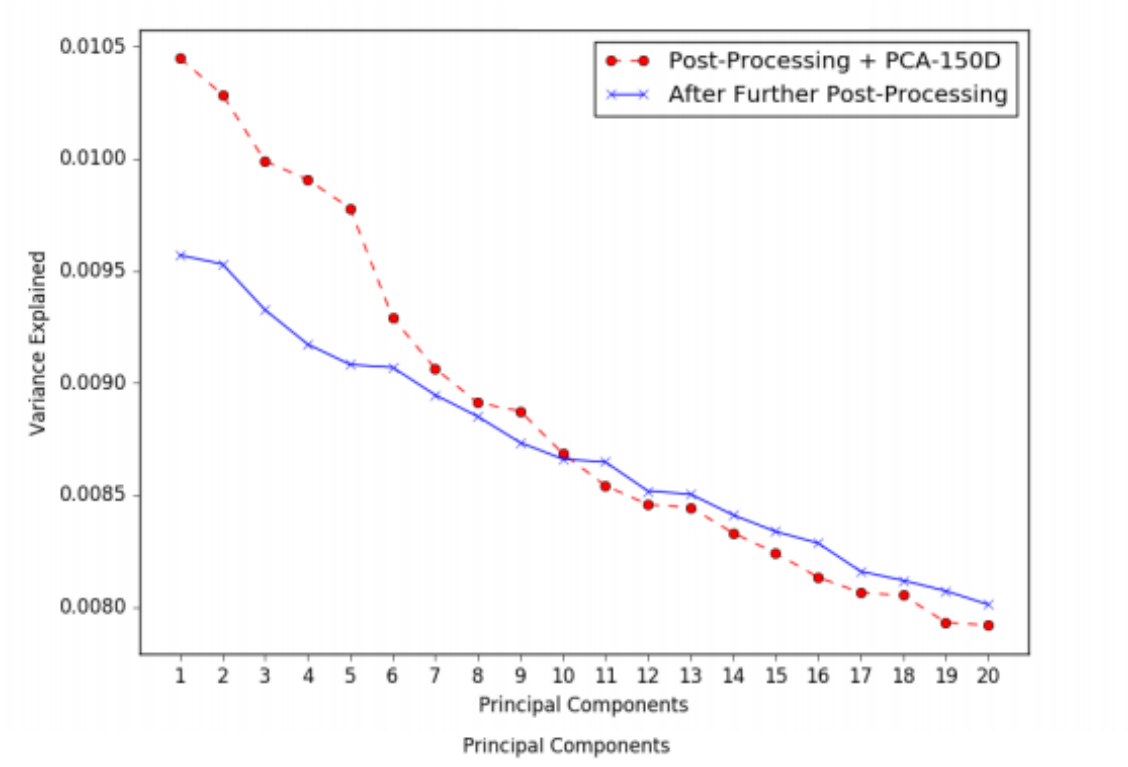


Figura 6.1: En rojo: Varianza de los 20 componentes principales de GLOVE tras aplicar PPA y PCA (150 dimensiones). En Azul varianza de los 20 componentes principales de GlOVE tras aplicar PPA+PCA(150 dimensiones) + PPA (D=7).

6.2.2. Resultados obtenidos

En esta sección analizaremos los resultados obtenidos por los diferentes métodos de reducción de dimensionalidad. Hemos escogido algunas de las concatenaciones que mejor rendimiento han obtenido en la sección anterior. En todas ellas hemos realizado una normalización L_2 a los vectores antes de realizar la concatenación. Debido al alto coste computacional de los algoritmos de reducción de dimensionalidad cuando los aplicamos a matrices de gran tamaño como los word embeddings que estamos usando, el número de pruebas que hemos podido realizar ha sido limitado. Además nos hemos visto obligados a no usar el método de aproximación de palabras descrito en la sección anterior. Para aplicar la reducción de dimensionalidad tenemos que aplicar el algoritmo a un meta embedding completo, es decir, un meta embedding formado por la unión de los vocabularios de los word embedding que estamos concatenando. En caso contrario, si lo aplicamos un meta embedding parcial formado por solo las palabras contenidas en los dataset que estamos usando para la evaluación, no obtendríamos resultados válidos, ya que los resultados de

aplicar la reducción de dimensionalidad a un word embedding completo pueden diferir de los resultados al aplicar el algoritmo a un pequeño subconjunto de palabras del mismo word embedding. En general, obtendríamos resultados más altos en los dataset que si aplicamos el algoritmo al word embedding completo). Pero como se expone en la sección anterior, el método de aproximación de palabras pruebe tardar hasta 40 horas en generar un meta embedding completo. Por lo tanto, generar varios meta embeddings aplicando el algoritmo de aproximación de palabras y diferentes algoritmos de reducción de dimensionalidad, conlleva mucho más tiempo del que tenemos disponible para la realización de este proyecto, no disponemos de la potencia computacional necesaria para realizarlo en un tiempo razonable. Para poder realizar estas pruebas hemos tenido que simplificar la concatenación eliminando el algoritmo de aproximación de palabras. En su lugar cuando durante la concatenación un word embedding dispone de representación para una palabra, pero otro word embedding no dispone de ella, asignamos un vector predefinido formado por ceros. En la sección anterior hemos comprobado que no es un buen método, pero nos permite reducir el tiempo de computo de la concatenación de word embeddings de horas a unos pocos minutos, haciendo posibles las comparativas que veremos en esta sección. Para intentar que esta elección afecta lo mínimo posible a los resultados analizaremos la concatenación y posterior reducción de dimensionalidad de embeddings que cuentan con un vocabulario muy extenso, como FastText, GLOVE o jointcHYB. Estos word embedding cuentan con representación para prácticamente todas las palabras de los dataset, por lo tanto el impacto de asignar un vector predefinido respecto a generar una aproximación será mínimo en las puntuaciones obtenidas en los dataset, permitiéndonos así realizar un estudio objetivo de los resultados. Debido a que concatenar jointcHYB con word embeddings calculados a partir de corpus de texto ha proporcionado buenos resultados, vamos a realizar pruebas concatenando jointcHYB y otros word embeddings con vocabularios extensos. Por lo tanto, salvo en el dataset RareWord (salvo en el caso de Word2vec) los word embeddings contarán con representación para todas las palabras de los dataset, por lo que la asignación de vectores predefinidos no tendrá ningún impacto en las puntuaciones de dichos dataset.

Hemos reducido las concatenaciones de word embeddings a 300 dimensiones, ya que son las dimensiones con las que cuentan todos los word embeddings que estamos utilizando (excepto SketchEngine que cuenta con 100 dimensiones) y es el número de dimensiones más comúnmente utilizado, ya que proporciona un buen rendimiento manteniendo una matriz de un tamaño razonable. La metodología empleada para las pruebas es la detallada enla Sección 3.1. Aunque en este caso nos interesa comparar el rendimiento de los

word embeddings antes y después de aplicar el algoritmo de reducción de dimensionalidad. Dichos algoritmos no afectan al vocabulario de los word embedding por lo tanto, usaremos solo la primera métrica descrita en la Sección 3.1, en la que se descartan los pares de palabras para los que no se puede dar respuesta, por lo que no se penaliza que un word embedding no sea capaz de dar respuesta para un determinado número de pares de palabras. En la Tabla 6.14 podemos observar el rendimiento obtenido por las diferentes concatenaciones tras aplicar diferentes métodos de reducción de dimensionalidad. Para facilitar la comparativa se han añadido varias columnas. En la primera se muestra el rendimiento respecto al mejor word embedding de la concatenación, de esta forma podemos comprobar si el meta embedding resultante ha mejorado el rendimiento de los word embeddings originales. En la segunda se muestra el rendimiento respecto a la concatenación sin aplicar ningún método de reducción de dimensionalidad. Y por último se muestra la media y la media ponderada obtenida por el meta embedding.

Como podemos observar en la Tabla 6.14 todos los métodos de reducción de dimensionalidad tienen un impacto negativo en el rendimiento. Reducir los word embeddings de 600 a 300 dimensiones nos hace perder información y tiene un impacto negativo en el rendimiento. En el caso de FastText y LexVec concatenados con jointcHYB, pese a que perdemos rendimiento respecto a la concatenación, la reducción de dimensionalidad nos aporta mejor rendimiento que el del mejor word embedding de la concatenación individualmente. Por lo tanto obtenemos un meta embedding con mejor rendimiento que los embeddings usados para generarlo, y con un número de dimensiones menor que la concatenación, lo que facilita su uso en todo tipo de tareas. Sin embargo, este no es el caso de Glove y W2V concatenados con jointcHYB, donde perdemos rendimiento respecto al mejor word embedding usado en la concatenación. Por lo tanto, no estamos consiguiendo mejorar el rendimiento. Aunque, tenemos que tener en cuenta que este método genera un meta embedding cuyo vocabulario es la unión de los vocabularios usados, por lo tanto, aunque no consigamos una mejora de rendimiento, si como en este caso, la pérdida de rendimiento es reducida el método puede ser interesante. En la Tabla 6.14 estamos mostrando los resultados con una métrica que no penaliza no dar respuesta para un determinado número de pares de palabras. Aunque, debido a que estamos usando word embeddings con vocabularios muy extensos, si usamos la segunda métrica, donde si se penaliza no poder dar respuesta para un determinado número de pares de palabras los resultados apenas varían como se puede comprobar en la Tabla 6.15.

Embeddings	Método	Respecto a mejor word embedding		Respecto a concatenación		Media	Media Ponderada
		Media (Diferencia)	Media Ponderada (Diferencia)	Media (Diferencia)	Media Ponderada (Diferencia)		
GLOVE + jointcHYB	DRA	-0.003	-0.010	-0.040	-0.036	0.651	0.596
	PCA	-0.009	-0.030	-0.046	-0.056	0.645	0.576
	tSVD	-0.002	-0.022	-0.039	-0.048	0.652	0.584
FT + jointcHYB	DRA	0.023	0.022	-0.007	-0.014	0.698	0.642
	PCA	0.008	-0.003	-0.022	-0.039	0.683	0.617
	tSVD	0.008	0.000	-0.022	-0.036	0.683	0.620
W2V + jointcHYB	DRA	-0.005	-0.020	-0.036	-0.042	0.649	0.586
	PCA	-0.016	-0.041	-0.047	-0.063	0.638	0.565
	tSVD	-0.020	-0.041	-0.051	-0.063	0.634	0.565
LEXVEC + jointcHYB	DRA	0.019	0.009	-0.018	-0.022	0.673	0.615
	PCA	0.002	-0.010	-0.035	-0.041	0.656	0.596
	tSVD	0.004	-0.005	-0.033	-0.036	0.658	0.601

Tabla 6.14: Comparativa relativa del rendimiento de diferentes concatenaciones tras aplicar diferentes métodos de reducción de dimensionalidad respecto al mejor word embedding de la concatenación y la concatenación sin aplicar la reducción de dimensionalidad. Métrica usada: Descartar pares para los que no se puede dar respuesta.

Embeddings	Método	Respecto a mejor word embedding		Respecto a concatenación		Media	Media Ponderada
		Media (Diferencia)	Media Ponderada (Diferecia)	Media (Diferencia)	Media Ponderada (Diferecia)		
GLOVE + jointcHYB	DRA	0.000	-0.002	-0.040	-0.037	0.651	0.595
W2V + jointcHYB	DRA	-0.002	-0.013	-0.036	-0.042	0.649	0.584

Tabla 6.15: Comparativa relativa del rendimiento de diferentes concatenaciones tras aplicar diferentes métodos de reducción de dimensionalidad respecto al mejor word embedding de la concatenación y la concatenación sin aplicar la reducción de dimensionalidad. Métrica usada: Resultado multiplicado por cobertura.

Analizando dataset por dataset los resultados, podemos comprobar algunos detalles interesantes. En la Tabla 6.16 se muestran los resultados de algunos de los dataset que nos han parecido más relevantes de forma relativa a los resultados obtenidos por la concatenación sin aplicar la reducción de dimensionalidad. En dicha tabla podemos observar que la pérdida de rendimiento en algunos dataset, como SimLex 999, YP 130 o SimVerb 3500 es muy fuerte. Sin embargo, en otros datasets la pérdida de rendimiento es menor e incluso existe mejora de rendimiento. Estos 3 dataset tienen en común que buscan capturar la sinonimia entre palabras, y no si las palabras están relacionadas de alguna otra manera entre ellas. Por lo que esto indica que la reducción de dimensionalidad afecta negativamente a la sinonimia. En cambio, datasets como Mturk o Men los cuales miden la relación que existe entre las palabras, en estos dataset la pérdida de rendimiento es inferior e incluso podemos llegar a obtener una mejora del rendimiento. Otro dataset interesante es RareWord. Dicho dataset está formado por palabras muy poco comúnes, por lo que es el dataset donde existen más palabras para las que los word embeddings no cuentan con representación. Dado que estamos utilizando el método de asignar un vector formado por ceros cuando un word embedding cuanta con representación para una palabra y otro word embedding no cuenta con representación para esa palabra durante la concatenación, es de esperar que sea el dataset donde obtengamos peores resultados en relación a la concatenación, donde estamos usando un método de aproximación de palabras en vez de asignar un vector predefinido. Sin embargo, no es lo que ocurre. DRA obtiene incluso mejor rendimiento que la concatenación sin aplicar métodos de reducción de dimensionalidad. Atribuimos

este resultado a que DRA implica usar el método de normalización PPA explicado en el Capítulo 4. Dicho método de normalización en general mejora el resultado de los word embeddings en el dataset RareWord, como se puede comprobar en la Tabla 6.17. Sin embargo, salvo en el caso de la concatenación de Lexvec y jointcHYB, el método de reducción de dimensionalidad SVD truncado, también obtiene buenos resultados teniendo en cuenta que estamos asignando vectores predefinidos. Por lo tanto, los algoritmos de reducción de dimensionalidad son capaces de "compensar" el error introducido al asignar vectores predefinidos.

Embeddings	Método	SimLex 999	MTurk 287	MTurk 771	MEN ALL	YP 130	SimVerb 3500	RW
GLOVE + jointcHYB	DRA	-0.046	-0.005	-0.016	-0.010	-0.108	-0.101	0.031
	PCA	-0.069	0.005	-0.016	-0.014	-0.109	-0.119	-0.029
	tSVD	-0.063	0.009	-0.015	-0.012	-0.101	-0.116	-0.002
FT + jointcHYB	DRA	-0.031	0.024	0.014	0.001	-0.065	-0.074	0.061
	PCA	-0.043	0.012	0.001	-0.006	-0.094	-0.099	-0.007
	tSVD	-0.050	0.013	0.003	-0.007	-0.090	-0.101	0.013
W2V + jointcHYB	DRA	-0.059	0.016	-0.008	-0.011	-0.154	-0.117	0.029
	PCA	-0.073	0.001	-0.022	-0.024	-0.141	-0.136	-0.019
	tSVD	-0.078	0.008	-0.024	-0.028	-0.162	-0.139	-0.005
LEXVEC + jointcHYB	DRA	-0.030	0.032	0.000	-0.003	-0.080	-0.069	0.012
	PCA	-0.041	0.005	-0.012	-0.011	-0.109	-0.084	-0.034
	tSVD	-0.039	0.010	-0.003	-0.008	-0.082	-0.081	-0.021

Tabla 6.16: Comparativa relativa del rendimiento de diferentes concatenaciones tras aplicar diferentes métodos de reducción de dimensionalidad respecto y la concatenación sin aplicar la reducción de dimensionalidad. Métrica usada: Descartar pares de palabras para los que no se puede dar respuesta.

Embedding	RW	RW (PPA)
FastText	0.595	0.612
LexVec	0.539	0.548
Word2Vec	0.534	0.535
jointcHYB	0.378	0.397

Tabla 6.17: Comparativa entre el rendimiento de diferentes word embeddings en el dataset RareWords sin aplicar ningún método de normalización y aplicando el método de normalización PPA. Métrica usada: Descartar pares de palabras para los que no se puede dar respuesta.

En cuanto a la comparativa de los diferentes métodos de reducción de dimensionalidad, la SVD truncado obtiene en general mejores resultados que PCA. Lo que indica que el centrado de variables que realiza PCA no tiene un efecto positivo en el rendimiento. Sin embargo, combinar el algoritmo de post procesado PPA con la reducción de dimensionalidad PCA obtiene un rendimiento significativamente superior al de SVD truncado, siendo así el método de reducción de dimensionalidad que mejor rendimiento obtiene. Esto demuestra que PCA produce un pequeño número de dimensiones dominantes, eliminar dichas dimensiones dominantes permite obtener mejores representaciones de palabras.

6.2.3. Conclusiones

En esta sección hemos estudiado el efecto que tiene la reducción de dimensionalidad en diferentes meta embeddings generados a partir de la concatenación. La concatenación de word embeddings produce word embeddings con un número de dimensiones muy alto, lo cual dificulta su uso. La reducción de dimensionalidad genera word embeddings con un número de dimensiones inferior, facilitando así el uso de los word embedidngs generados en todo tipo de tareas. Sin embargo, como hemos comprobado, la reducción de dimensionalidad conlleva una pérdida de información que provoca que el rendimiento del meta embedding generado sea inferior al rendimiento de la concatenación. Dicha pérdida de rendimiento varía según los word embeddings usados en la concatenación. En ocasiones, pese a que existe una pérdida de rendimiento respecto a la concatenación el meta embedding generado mantiene un rendimiento superior a los word embeddings usados en la concatenación individualmente. Por lo tanto la reducción de dimensionalidad tiene un efecto positivo. Sin embargo este no es siempre el caso, en ocasiones el rendimiento puede ser inferior al del mejor word embedding usado la concatenación.

Resulta interesante resaltar que la reducción de dimensionalidad tiene un impacto especialmente negativo en la tarea del cálculo de la sinonimia entre palabras, mientras que su impacto es inferior en la puntuación en los dataset donde se mide la relación entre palabras. Dependiendo de la tarea en la que queramos usar los meta emebddings generados tener en cuenta este comportamiento es importante.

Uno de los principales problemas de la reducción de dimensionalidad es la capacidad de computo y memoria requeridas para aplicar estos métodos. Hemos comprobado que aplicar un algoritmo de reducción de dimensionalidad a word embeddings de 1200 dimensiones llega a requerir hasta 90GB de memoria y el tiempo requerido para aplicar aplicar los algoritmos es muy alto. Ejecutados en un procesador de 6 núcleos reducir la

dimensionalidad de un word embedding de 600 dimensiones son necesarias varias horas. En el caso de un word embedding de 1200 dimensiones tratamos de aplicar el algoritmo DRA pero tras más de 12 horas detuvimos la ejecución al ver que no finalizaba. Aplicar el método de reducción de dimensionalidad a concatenaciones de varios word emebddings puede resultar imposible para la mayoría de computadoras.

Debido a la alta capacidad de cómputo y memoria necesarios para la ejecución de estos algoritmos, y una pérdida de rendimiento significativa respecto a la conctenación de word emebddings, la reducción de dimensionalidad no resulta demasiado atractiva. Es por ello que en la siguiente sección estudiaremos una forma diferente de obtener meta embeddings con un número de dimensiones reducido.

6.3. Media de word embeddings

En este capítulo hemos estudiado el método de generación de meta embeddings mediante la concatenación de word embeddings. Dicho método nos ha proporcionado buenos resultados, sin embargo los meta embeddings generados cuentan con un número de dimensiones muy alto, tan alto como la suma de las dimensiones de los word embeddings usados en la concatenación. Un número muy alto de dimensiones dificulta el uso de los meta embeddings. Nuestro primer intento de resolver esta problemática ha sido aplicar métodos de reducción de dimensionalidad, sin embargo, la potencia de cómputo y memoria requeridos para aplicar estos métodos, combinados con una pérfida de rendimiento importante respecto a la concatenación hacen que estos métodos no sean especialmente atractivos. Debido a esto, en esta sección vamos a estudiar un enfoque diferente, la media de word embeddings. Con este método esperamos generar meta embeddings con una dimensionalidad inferior a los generados por la concatenación manteniendo el rendimiento obtenido por dicho método.

6.3.1. Explicación del método

En esta sección explicaremos en detalle el método de la media de word embeddings, y describiremos como se ha aplicado a nuestros word embeddings. El método es simple, suponiendo que queremos realizar la media entre los embeddings $E1$ y $E2$, generamos el nuevo meta embedding AVG de la siguiente forma. Para cada palabra w perteneciente a la intersección de los vocabularios de ambos word embedidngs AVG_w se obtiene mediante

la media aritmética del el vector que representa a w en $E1$ y el vector que representa a w en $E2$.

$$\forall w \in E1 \cap E2 : AVG_w = \frac{E1_w + E2_w}{2}$$

Aunque intuitivamente podamos pensar que la media de word embeddings que se encuentran en espacios vectoriales diferentes carece de sentido, la media guarda una estrecha relación con la concatenación. Vamos a demostrar esta relación matemáticamente de la misma forma que con la concatenación. Usaremos una nomenclatura similar a la usada en la Sección 6.1. Imaginemos que tenemos por word embeddings E_1 y E_2, por simplicidad asumiremos que ambos cuentan con un número de dimensiones común d. Queremos calcular la distancia euclídea entre las palabras u y v. La distancia euclídea en el meta embedding AVG generado realizando la media entre E_1 y E_2, sería:

$$\begin{gathered} AVG = \\ = \| \frac{(u_{E_1} + u_{E_2})}{2} - \frac{(v_{E_1} + v_{E_2})}{2} \|_2 \\ = \frac{1}{2} \| (u_{E_1} - v_{E_1}) - (v_{E_2} - u_{E_2}) \|_2 \\ \approx \sqrt{(D_{E_1})^2 + (D_{E_2})^2 - 2 D_{E_1} D_{E_2} cos(\Theta)} \end{gathered}$$

En este caso, a diferencia de la concatenación, nuestros word embedding no son ortogonales, por lo tanto tenemos un término dependiente del ángulo entre $u_{E_1} - v_{E_1}$ y $u_{E_2} - v_{E_2}$. Sin embargo, "Cai et al" [80] demostraron que si, X es un conjunto de puntos aleatorio pertenecientes a $\mathbb{R}^n$ con cardinalidad $|X|$, entonces el límite de la distribución de ángulos, cuando $|X| \to \infty$, entre pares de elementos de X es una distribución gaussiana con media $\frac{\pi}{2}$. Además. "Cai et al.´´ demostraron que la varianza de esta distribución se reduce a medida que aumenta la dimensionalidad.

Los word embeddings generalmente cuentan con cientos de miles o incluso millones de vectores que cuentan con un alto número de dimensiones. Asumiendo que el vector diferencia entre dos palabras cualquiera de un word embedding es lo suficientemente aleatorio, podemos aproximar la distribución descrita por "Cai et al". En dicho caso, supondría que los vectores $u_{E_1} - v_{E_1}$ y $u_{E_2} - v_{E_2}$ son ortogonales, lo cual provocaría que la formula anterior se convirtiese en:

$$AVG = \sqrt{(D_{E_1})^2 + (D_{E_2})^2} \approx CONC$$

Con esto hemos probado que, si los word embeddings pueden verse como aproximadamente ortogonales, la media contendrá aproximadamente la misma información que la

concatenación, con la ventaja de no aumentar la dimensionalidad del meta embedding generado a medida que combinamos más word embeddings [53].

Problemáticas en la media de word embeddings

La media de word embeddings cuenta con las mismas problemáticas a tener en cuenta que la concatenación. En primer lugar tenemos que tener en cuenta que la escala de los word embeddings puede provocar que un word embedding domine al resto, provocando que se pierda su información. Si multiplicamos todos los vectores de un word embedding por un determinado valor, aumentando así su escala, dicho word embedding contará con un mayor peso en la media. Sin embargo, en este caso la solución no es tan sencilla como normalizar los vectores, ya que este factor se ve afectado por el ángulo que formen los vectores de los diferentes word embeddings. Por lo tanto vamos a estudiar el rendimiento de la media sin aplicar ningún tipo de normalización y de la media habiendo normalizado los vectores mediante la norma L_2. Al igual que en la concatenación y la media de las similitudes también probaremos que ocurre al asignar pesos a determinados word embeddings, es decir, escalarlos para cuenten con mayor peso en la media. Lo que buscamos con esto es otorgar mayor peso a los word embeddings que cuentan con mejor rendimiento para intentar obtener mejores meta embedidngs.

Además de la problemática de las escalas de los word embeddings existe otra compartida con la concatenación. Y son las palabras fuera de la intersección del vocabulario de los word embeddings. Es decir, palabras para los que un word embeddings cuente con representación pero uno o más embeddings no cuenten con representación. En este caso a diferencia de en el método de la media de las similitudes (Sección 5.2) necesitamos que todos los word embeddings cuenten con representación para todas las palabras. Imaginemos que queremos realizar la media entre los word embeddings E1 y E2. La media generará un meta embedding que cuyos vectores se encuentran en un espacio vectorial diferente al de los word embeddings originales. Por lo tanto si dada la palabra w para la que E1 cuenta con representación, pero E2 no, si tomamos directamente el vector de E1 dicho vector se encontrará en el espacio vectorial de E1, mientras que el resto de vectores del meta embedding estarán situados en un espacio vectorial diferente. Esto provoca que cualquier cálculo que implique a la palabra w dará un resultado que dependerá de como sean los espacios vectoriales y no de la relación que existe en w y el resto de palabras, haciendo dicho cálculo inútil. Sin embargo, en la concatenación de word embeddings ya nos enfrentamos a esta misma problemática, por lo que implementamos un método de

aproximación de palabras que como pudimos comprobar proporciona buenos resultados. Dicho método se describe en la Subsubsección 6.1.1. Este método nos permite aproximar vectores para cualquier palabra que se encuentre en la unión de los vocabularios de todos los word embeddings usados en la media. Lo usaremos para aproximar palabras cuando un word embedding no cuente con representación para una palabra para la que otro word embedding si cuenta con representación.

Algoritmo de media entre word embeddings

Una vez detallado el método de la media de word embeddings, a continuación describimos en pseudocódigo el algoritmo que hemos implementado para esta tarea:

```
\\Entrada:
    //lista_embeddings: Lista que contiene los word embeddings para los que queremos aplicar la media.
    //lista_pesos: El peso que queremos dar a cada uno de los word embeddings en la media, por defecto será [1,1,1,1....,1], es decir, daremos a todos los word embeddings el mismo peso.
    //normalizar: True si queremos aplicar una normalización L2 a los vectores para los que vamos a calcular la media. False en caso contrario.
\\Salida:
    //Word embedding cuyo vocabulario es la unión de los vocabularios de los embeddings en lista_embeddings y cuyos vectores son la media aritmética de los vectores que representan a cada palabra de cada word embedding original.

\\Funciones:
    //Buscar(P,Emb). Devuelve el vector que representa a la palabra P en el embedding Emb. Si el embedding no cuenta con una representación para P, devuelve el valor nulo.
    //intersección(EMBS): Recibe como entrada una lista de embeddings. Devuelve una lista de palabras que representa la intersección de los vocabularios de la lista de embeddings.
    //unión(EMBS): Recibe como entrada una lista de embeddings. Devuelve una lista de palabras que representa la unión de los vocabularios de la lista de embeddings
    //generate_word(word, embedding, embeddings_list, overlapping_vocabulary): Función descrita en  la sección 7.1.1.
    //normalizar_L2(v): Recibe como entrada un vector v y devuelve el vector v normalizado mediante la norma L2
    //añadir(L,E): Recibe como entrada la lista L y el elemento E. Devuelve como salida L con el elemento E añadido al final de la lista.


función media_embeddings(lista_embeddings, lista_pesos, normalizar):
    overlapping_vocabulary = intersección(lista_embeddings)
    vocab = unión(lista_embeddings)

    vectores = []

    Para cada palabra w de vocab:
        vector = []
        div = 0
        Para e desde 1 hasta longitud(lista_embeddings):
            v = Buscar(w, lista_embeddings[e])
            Si v es nulo:
                v = generate_word(w,lista_embeddings[e],lista_embeddings,overlapping_vocabulary)

            Si normalizar:
                v = normalizar_L2(v)

            vector = vector + (v * lista_pesos[e])
            v = v + lista_pesos[e]

        Si div > 0:
            vectores = añadir(vectores,(vector/div))

    Devolver vocab, vectores

//Nota: Para facilitar la legibilidad del algoritmo se han omitido las comprobaciones de datos y errores.
```

6.3.2. Resultados obtenidos

En esta sección analizaremos los resultados obtenidos por las diferentes medias de word embeddings. La notación que vamos a utilizar para las diferentes pruebas es la siguiente:

$$\text{embedding\_1} + \text{embedding\_2} + \text{embedding\_3} + ...$$

Esto quiere decir que se ha realizado la media aritmética de los word embeddings embedding_1, embedding_2 y embedding_3 y se muestra el rendimiento obtenido por el meta embedding generado. Los pesos que se han dado a cada embedding en media se representan de la siguiente forma:

[1,2,4...]

Esto quiere decir que al embedding_1 se le ha dado un peso 1, al embedding_2 se le ha dado un peso 2, al embedding_3 un peso 4... y así sucesivamente. En caso de que no se muestren los pesos significa que se ha dado a todos los word embeddings peso 1.

La metodología para las pruebas realizadas es la detallada en la Sección 3.1. Las tablas con los resultados que analizaremos en esta sección cuentan con 6 columnas. En las 2 primeras se muestra la media y la media ponderada de forma relativa al mejor word embedding usado en la media. Esto nos permitirá saber si el meta embedding generado obtiene un mejor rendimiento que los word embedding usados. En las 2 columnas siguiente se muestra el rendimiento obtenido respecto a la concatenación, por lo tanto podremos conocer como de eficaz es este método de generación de meta embeddings. Y por último para facilitar futuras comparativas mostraremos la media y la media ponderada obtenidas en los diferentes dataset. Vamos a dividir esta comparativa en varias partes. Primeros analizaremos el rendimiento del algoritmo sin aplicar ningún tipo de normalización a los word embedding generados. A continuación probaremos a normalizar los vectores antes de realizar la media. Y por último compararemos la media de word embeddings con la reducción de dimensionalidad.

Media de word embeddings sin aplicar ninguna normalización

En esta sección analizaremos el rendimiento de la media de word embedding, realizando dicha media sin aplicar ningún método de normalización a los vectores. En la Tabla 6.18

se muestran los resultados obtenidos. Para facilitar la comparativa se muestra la diferencia entre la media y la media ponderada obtenidos y la media y la media ponderada obtenidos por la concatenación de los mismos word embeddings (sin aplicar ningún algoritmo de reducción de dimensionalidad ni normalizar sus vectores). También se muestra la diferencia de rendimiento respecto al mejor word embedding de la concatenación para comprobar si el meta embedding generado nos aporta una mejora de rendimiento.

Como se puede observar en la Tabla 6.18, ocurre algo muy similar a lo que ocurría con la concatenación de word embeddings. La escala de los word embeddings afecta a la media, provocando que embeddings que unos word embeddings dominen a otros. Al igual que en la concatenación destaca el caso de GLOVE que se ve completamente dominado por el resto de word embeddings, haciendo que su información se pierda. Podemos ver esto ya que la puntuación obtenida es la misma que obtienen individualmente los embeddings con los que estamos combinando Glove. Podemos comprobar esto también en el caso de la media entre Word2Vec y jointcHYB, UKB y LexVec y PDC, FastText y jointcHYB. En estos casos la asignación de pesos nos permite obtener mejores resultados. En ambos casos si comprobamos la Tabla 6.9, podemos ver que esto ocurre por que el embedding al que estamos asignando un mayor peso o escalando se encuentra en una escala inferior que el otro word embeddings. Por lo tanto, al igual que en la concatenación este es un claro indicativo de que es necesario aplicar un método de normalización para que todos los word embeddings tengan el mismo peso en la media, evitando así que unos dominen a otros.

Embeddings	Pesos	Respecto a mejor word embedding		Respecto a concatenación		Media	Media Ponderada
		Media (Diferencia)	Media Ponderada (Diferencia)	Media (Diferencia)	Media Ponderada (Diferencia)		
GLOVE + W2V	[1,1]	-0.017	-0.018	0.000	0.000	0.612	0.553
GLOVE + FT	[1,1]	0.000	0.000	0.000	0.000	0.675	0.620
GLOVE + jointcHYB	[1,1]	-0.001	-0.005	0.000	0.000	0.653	0.601
GLOVE + UKB	[1,1]	0.002	-0.013	0.000	0.000	0.632	0.597
W2V + jointcHYB	[1,1]	0.005	0.002	-0.006	-0.007	0.659	0.608
W2V + jointcHYB	[2,1]	0.012	0.007	-0.016	-0.018	0.666	0.613
W2V + UKB	[1,1]	0.026	0.006	-0.027	-0.023	0.656	0.616
FT + jointcHYB	[1,1]	-0.010	-0.006	-0.013	-0.015	0.665	0.614
FT + jointcHYB	[2,1]	0.004	0.010	-0.025	-0.027	0.679	0.630
FT + UKB	[1,1]	0.014	0.020	-0.023	-0.026	0.689	0.640
UKB + LEXVEC	[1,1]	0.013	0.002	-0.014	-0.008	0.643	0.612
UKB + LEXVEC	[1,2]	0.032	0.015	-0.027	-0.021	0.662	0.625

Tabla 6.18: Comparativa relativa del rendimiento de realizar la media de diferentes word embeddings respecto al mejor word embedding usado en la media y la concatenación de word embeddings. En ambos casos sin aplicar ninguna normalización a los word embeddings. Métrica usada: Descartar pares para los que no se puede dar respuesta.

Media de word embeddings aplicando la normalización L_2

En la sección anterior hemos comprobado que es necesario normalizar los word embeddings antes de realizar la media. Si no lo hacemos el word embedding que se encuentre en una escala mayor tenderá a dominar al resto haciendo que se pierda su información que el embedding resultante obtenga un peor rendimiento. Hemos estudiado los efectos de dos normalizaciones sobre la media, la normalización L_1 de los vectores, que asegura que la suma de todos los componentes del vector sea igual a uno. Y la normalización L_2, la cual asegura que raíz cuadrada de la suma de los cuadrados de cada componente sea igual a uno. Hemos descubierto que el ambas normas obtienen el mismo rendimiento final como se puede comprobar en la Tabla 6.19. Por lo tanto, puesto que es la misma normalización que hemos usado en la concatenación, analizaremos el rendimiento de la media de word embeddings aplicando la normalización L_2 a sus vectores antes de realizar la media. Con esto esperamos conseguir que todos los embeddings cuenten con una escala similar, evitando que un word embedding pueda dominar a otro y de esta forma obtengamos un mejor rendimiento.

Embedding	Pesos	Normalización L1		Normalización L2	
		Media	Media Ponderada	Media	Media Ponderada
GLOVE + W2V	[1,1]	0.626	0.562	0.628	0.565
FT + UKB	[1,1]	0.689	0.638	0.689	0.638
UKB + LEXVEC	[1,1]	0.665	0.621	0.665	0.621
W2V + jointcHYB	[1,1]	0.661	0.605	0.660	0.605

Tabla 6.19: Comparativa relativa del rendimiento de realizar la media de diferentes word embeddings aplicando previamente la normalización L_1 o L_2 a sus vectores. Métrica usada: Descartar pares para los que no se puede dar respuesta.

Nos interesa comparar el rendimiento de la media de word embeddings con el rendimiento obtenido por la concatenación. Por lo tanto en la Tabla 6.20, vamos a comparar el rendimiento obtenido por la media de word embeddings habiendo aplicado la normalización L_2, con la concatenación de los mismos word embeddings habiendo realizado también la normalización L_2 antes de la concatenación. También mostraremos la diferencia de rendimiento respecto al mejor word embeddings usado en la media para comprobar si hemos conseguido generar un meta embedding mejor que los word embeddings utilizados.

Embeddings	Respecto a mejor word embedding		Respecto a concatenación		Media	Media Ponderada
	Media	Media Ponderada	Media	Media Ponderada		
GLOVE + jointcHYB	0.020	0.009	-0.017	-0.017	0.674	0.615
GLOVE + UKB	0.045	0.009	-0.021	-0.020	0.675	0.619
W2V + jointcHYB	0.006	-0.001	-0.025	-0.023	0.660	0.605
W2V + UKB	0.026	-0.001	-0.031	-0.024	0.656	0.609
FT + jointcHYB	0.006	0.008	-0.024	-0.028	0.681	0.628
FT + UKB	0.014	0.018	-0.021	-0.026	0.689	0.638
jointcHYB + LEXVEC	0.005	0.011	-0.032	-0.020	0.659	0.617
UKB + LEXVEC	0.035	0.011	-0.028	-0.022	0.665	0.621
GLOVE + jointcHYB + UKB	0.019	0.008	-0.026	-0.029	0.673	0.618
FT + jointcHYB + LEXVEC	-0.002	0.000	-0.029	-0.027	0.673	0.620
GLOVE + jointcHYB + UKB + PDC	0.004	-0.004	-0.042	-0.036	0.658	0.606
GLOVE + FT + jointcHYB + UKB	0.001	-0.001	-0.037	-0.038	0.676	0.619
GLOVE + FT + jointcHYB + UKB + LEXVEC	-0.004	-0.007	-0.037	-0.038	0.671	0.613
W2V + FT + jointcHYB + UKB + LEXVEC	-0.011	-0.005	-0.045	-0.037	0.664	0.615
GLOVE + W2V + FT + jointcHYB + UKB	-0.005	-0.009	-0.041	-0.039	0.670	0.611
W2V + FT + jointcHYB + UKB + LEXVEC + PDC	-0.017	-0.015	-0.048	-0.040	0.658	0.605
GLOVE + W2V + FT + jointcHYB + UKB + LEXVEC + PDC	-0.028	-0.022	-0.057	-0.041	0.647	0.598

Tabla 6.20: Comparativa relativa del rendimiento de realizar la media de diferentes word embeddings respecto al mejor word embedding usado en la media y la concatenación de word embeddings. En ambos casos aplicando la normalización L_2 antes de la combinación de los word embeddings. Métrica usada: Descartar pares para los que no se puede dar respuesta.

Como podemos observar en la Tabla 6.20, al igual que con la reducción de dimensionalidad, la media de word embedding supone una pérdida de información respecto a la concatenación, haciendo que exista una pérdida de rendimiento significativa. Sin embargo, pese a ello, la media de dos word embeddings produce meta embeddings mejores que los embeddings usados individualmente. Aunque el rendimiento obtenido sea inferior, los meta embeddings generados cuentan con menos dimensiones que los generados mediante la concatenación, haciendo que su uso en todo tipo de aplicaciones sea más sencillo.

Cuando aumentamos el número de word embeddings usados para la media, este método no proporciona buenos resultados. A medida que añadimos word embeddings el rendimiento se reduce. Comportamiento que no hemos observado ni en la concatenación, ni en la media de las similitudes. Esto puede explicarse por que estamos realizando la media entre word embeddings que se encuentran en espacios vectoriales diferentes. Esto puede provocar que existan vectores que se contrarresten entre ellos debido a el ángulo que exista entre un espacio vectorial y otro. A medida que añadimos word embeddings en diferentes espacios vectoriales magnificamos este efecto. A medida que añadimos word embeddings en diferentes espacios vectoriales, la longitud de todos los vectores tendrá a 0. Por lo tanto, esto nos indica que realizar la media de word embeddings en diferentes espacios vectoriales no es un método adecuado. Si bien en general podemos obtener un buen rendimiento realizando la media de dos word embeddings. A medida que añadimos word embeddings a dicha media provocamos una pérdida de información cada vez mayor.

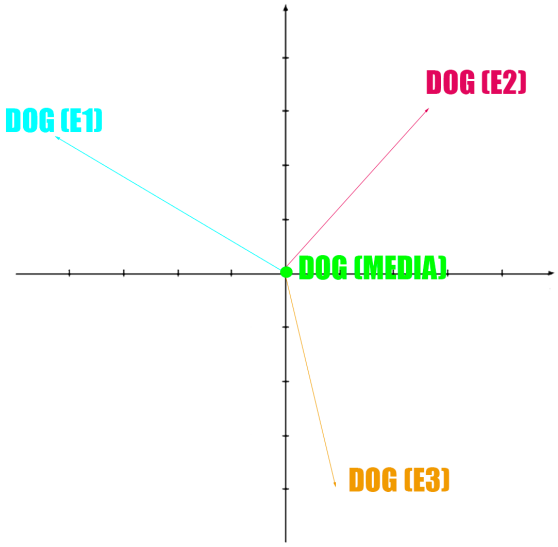


Figura 6.2: Demostración simplificada de como word embeddings en diferentes espacios vectoriales puede contrarestarse entre ellos provocando que se pierda información

Media de word embeddings vs reducción de dimensionalidad

Si bien hemos mostrado que la media no resulta adecuada en el caso de queramos combinar un gran número de word embeddings, resulta interesante realizar una comparativa entre realizar la media de dos word embeddings y concatenar dos vectores y luego aplicar un método de reducción de dimensionalidad. En el caso de la reducción de dimensionalidad tomaremos los resultados obtenidos por el método DRA (aplicado a la concatenación de dos word embeddings normalizados mediante la norma L_2), ya que es el método que mejores resultados obtiene. En el caso de la media tomaremos los resultados de la media habiendo normalizado los vectores mediante la norma L_2. Para facilitar la comparativa mostraremos los resultados obtenidos por la concatenación de word embeddings normalizados mediante la norma L_2, y en el caso de la media y la reducción de dimensionalidad se mostrará el rendimiento relativo a dicho resultado. Podemos comprobar los resultados obtenidos en la Tabla 6.21.

	L2 + Concatenación		Reducción de dimensionalidad (DRA)		L2 + Media de Word embeddings	
	Media	Media Ponderada	Media (Diferencia)	Media Ponderada (Diferencia)	Media (Diferencia)	Media Ponderada (Diferencia)
GLOVE + jointcHYB	0.691	0.632	-0.040	-0.036	**-0.017**	**-0.017**
FT + jointcHYB	0.705	0.656	**-0.007**	**-0.014**	-0.024	-0.028
W2V + jointcHYB	0.685	0.628	-0.036	-0.042	**-0.025**	**-0.023**
LEXVEC + jointcHYB	0.691	0.637	**-0.018**	-0.022	-0.032	**-0.020**

Tabla 6.21: Comparativa relativa del rendimiento entre la concatenación, la concatenación + reducción de dimensionalidad, y la media de word embeddings. Métrica usada: Descartar pares para los que no se puede dar respuesta.

En la Tabla 6.21 podemos observar que en algunos casos la reducción de dimensionalidad obtiene un mejor rendimiento, y en otros casos es la media la que mejor rendimiento obtiene. Aunque en ambos casos existe pérdida de rendimiento respecto a la concatenación. Sin embargo, para poder realizar una comparativa completa entre ambos métodos hay que tener en cuenta otro factor importante. En el caso de la reducción de dimensionalidad observamos que producían una gran pérdida de rendimiento en datasets que analizan la sinonimia entre palabras. En cambio, en datasets que analizaban la relación entre palabras no existía una pérdida de rendimiento significativa respecto a la concatenación, e inclu-

so en algunos casos existe una pequeña mejora de rendimiento. Podemos consultar estos resultados en la Tabla 6.16. En el caso de la media de word embeddings no observamos el mismo comportamiento, en la media de word embeddings observamos una pérdida de rendimiento similar en todos los dataset, como puede comprobarse en la tabla Tabla 6.22. No existen datasets que se vean afectados de forma más significativa que el resto. Por lo tanto la media de word embeddings resulta ser un método más homogéneo.

Embeddings	SimLex 999	MTurk 287	MTurk 771	MEN ALL	WS353 ALL	YP 130	SimVerb 3500	RW
GLOVE + jointcHYB	-0.022	0.001	-0.014	-0.018	-0.031	-0.031	-0.021	-0.006
FT + jointcHYB	-0.017	-0.032	-0.037	-0.031	-0.022	-0.019	-0.027	-0.023
W2V + jointcHYB	-0.022	-0.041	-0.028	-0.027	-0.023	-0.034	-0.027	-0.006
LEXVEC + jointcHYB	-0.032	-0.025	-0.029	-0.025	-0.023	-0.027	-0.017	-0.008

Tabla 6.22: Comparativa relativa del rendimiento de diferentes medias de word embeddings normalizados mediante la norma L_2 y la concatenación en diferentes dataset. Métrica usada: Descartar pares de palabras para los que no se puede dar respuesta.

Ahora que tenemos todos los datos podemos realizar una comparativa entre la media y la reducción de dimensionalidad. En cuanto al rendimiento no hay claro ganar, en unas combinaciones un método obtiene un mejor rendimiento y en otras el método es el que obtiene un mejor rendimiento. Aunque es importante resultar que la reducción de dimensionalidad tiene un impacto muy negativo en la sinónima entre palabras, mientras que apenas se pierde información sobre la relación entre palabras. En cambio la media produce una pérdida de rendimiento más homogénea entre datasets. Este comportamiento puede determinar la elección dependiendo que uso vayamos a dar a los meta embeddings generados. La media cuenta con una ventaja respecto a la reducción de dimensionalidad, cuanta con un coste computacional mucho menor. La media tiene el mismo coste que realizar la concatenación. En cambio la reducción de dimensionalidad requiere de realizar la concatenación y aplicar el método de reducción, método que requiere de una gran capacidad de cómputo y memoria. Por lo tanto, la media resulta un método mucho más atractivo desde este punto de vista.

6.3.3. Conclusiones

En esta sección hemos analizado el método de la media de word embeddings. Este método permite obtener meta embeddings que almacenan una información similar a la concatenación, con la ventaja de los word embeddings generados cuentan con un número de dimensiones inferior. Esto hace que su aplicación para todo tipo de tareas sea más sencillo. Hemos podido comprobar que al igual que en la concatenación, es necesario normalizar los vectores antes de realizar la media. En caso contrario los word embeddings que se encuentren en una escala mayor tenderán a dominar al resto, provocando que su conocimiento se pierda. Respecto a la reducción de dimensionalidad, la media consigue un rendimiento similar, siendo mejor para la combinación de algunos word embeddings e inferior para otros. Sin embargo, la media produce una perdida de rendimiento homogénea en los diferentes dataset, mientras que la reducción de dimensionalidad afecta de forma muy negativa a la sinonimia entre palabras y apenas afecta a la relación entre palabras.

Sin embargo nos hemos encontrado con un problema fundamental, la media no ofrece buenos resultados al combinar un número alto de word embeddings. Sin embargo, creemos que este es un problema que se puede solventar mediante la alineación de los espacios vectoriales en los que se encuentra cada word embeddings. Esto nos permitiría resolver el problema de la pérdida de rendimiento al combinar más de dos word embeddings, y nos permitiría obtener mejores resultados con este método. Es por ello que estudiaremos esta posibilidad en el siguiente capítulo.

6.4. Retrofitting

Hasta ahora hemos estudiado diferentes formas de combinar fuentes de conocimiento. Sin embargo, todas las fuentes de conocimiento usadas eran word embeddings. Hemos utilizado diferentes tipos de word embeddings, como embedding calculados a partir de corpus de textos, embeddings calculados a partir de grafos e incluso embeddings híbridos que combinan ambos métodos. Sin embargo, también es posible generar meta embeddings combinando diferentes tipos de fuentes de conocimiento, como combinar un word embedding con bases de conocimiento léxico semánticas como WorNet [39]. El método que vamos a utilizar para realizar esto se conoce como retrofitting [37]. El retrofitting es un algoritmo de post-procesado de word embeddings que permite incorporar el conocimiento de léxicos semánticos. La versión que hemos utilizado para este estudio es la versión original publicada por “Faruqui et al” [37]. Recientemente se han publicado versiones

alternativas, que buscan mejorar la implementación original, ya sea utilizando diferentes tipos de bases de conocimiento o introduciendo modificaciones en el algoritmo. Algunos ejemplos son "Expanded retrofitting" [68] o "Extrofitting" [52]. En este estudio buscamos determinar si el retrofitting puede ayudarnos a generar meta embeddings mejores que los anteriores. El objetivo no es determinar cual de todas las versiones del retrofitting obtiene un mejor rendimiento, por lo tanto no hemos realizado una comparativa de todas las versiones que se han publicado, hemos decidido limitarnos a la versión original.

6.4.1. Explicación del retrofitting

El retrofitting es un proceso que ajusta una matriz de word embeddings usando un grafo de conocimiento. El retrofitting infiere nuevos vectores q_i con el objetivo de estar cerca de los valores originales $\hat{q}_i$, y también cerca de sus vecinos en el grafo. Imaginemos el vocabulario V = (w_1,...w_n), y la ontología Ω que codifica relaciones semánticas entre las palabras en V. Representamos Ω como un grafo no direccionado (V,E) con un vertice por cada palabra y aristas (w_i, w_j) $\in$ E $\sqsubseteq$ V $\times$ V que indican relaciones semánticas de interés. La matriz $\hat{Q}$ será una colección de representaciones $\hat{q}_i \in \mathbb{R}^d$, para cada $w_i \in$ V donde d es la longitud de los vectores. El objetivo es aprender la matriz Q = (q_1,...q_n) tal que las columnas están cerca (usando como métrica la distancia euclídea) a sus equivalentes en $\hat{Q}$ y sus vértices adyacentes en Ω. El objetivo de minimización es por lo tanto

$$\Psi(Q) = \sum_{i=1}^{n} [\alpha_i \|q_i - \hat{q}_i\|^2 + \sum_{(i,j)\in E} \beta_{ij} \|q_i - q_j\|^2]$$

donde α y β son valores que controlan la fuerza relativa de las asociaciones. El retrofitting es un método de post-procesado, es decir, primero entrenamos los word embeddings de forma independiente a la información en los léxicos semánticos y luego los adaptamos. Para ello podemos seguir el siguiente método iterativo, inicializamos los vectores en Q con sus valores en $\hat{Q}$. Actualizamos dichos vectores mediante el siguiente método iterativo:

$$q_i = \frac{\sum_{j:(i,j)\in E} \beta_{ij} q_j + \alpha_i \hat{q}_i}{\sum_{j:(i,j)\in E} \beta_{ij} + \alpha_i}$$

6.4.2. Bases de datos léxicas

Para realizar el retrofitting de word embeddings necesitamos una base de datos léxico semántica. Hemos utilizado dos bases de datos, WortNet [39] y PPDB [41]. En ambos casos hemos usado los archivos proporcionados por los autores del retrofitting.

- WordNet [39]: Es un gran léxico semántico construido por humanos de palabras en inglés. Agrupa las palabras en grupos de sinónimos llamados synsets, proporciona pequeñas definiciones generales y registra las diferentes relaciones semánticas entre synsets. WordNet se estructura en un grafo, por lo que resulta especialmente adecuado para el retrofitting. Relaciones explícitamente conceptos con relaciones semánticas como hiperónimos e hipónimos. Vamos a usar dos versiones de WordNet, la primera conecta cada palabra con sinónimos, y la segunda (a la que nos referiremos como WordNet+) conecta cada palabra con sinónimos, hipónimos [1] e hiperónimos [2].
- PPDB [41]: Se trata de un léxico semántico formado por más de 220 millones de pares de paráfrasis [3] en inglés. El principio en el que se basa esta base de datos es que si dos palabras en un lenguaje, se alinean en textos paralelos a la la misma palabra en otro lenguaje, deben ser sinónimos. Por ejemplo si las palabras “monarca” y “rey” se traducen como la misma palabra en otro idioma, es razonable pensar que tienen el mismo significado. PPDB se proporciona en diferentes tamaños, desde S hasta XXXL, la usada en nuestras pruebas es la versión XL.

6.4.3. Resultados obtenidos:

En esta sección analizaremos los resultados obtenidos tras aplicar el retrofitting a los word embeddings que estamos utilizando. Para realizar el retrofitting hemos utilizado el código proporcionado por ”Faruqui et al´´ [37], aunque hemos incluido una pequeña modificación. El código convierte todas las palabras a minúsculas, y en caso de que se encuentren duplicados el vector que se conserva en el embedding final es el último que se

[1] Hipónimo: Palabra cuyo significado incluye el de otra. Gorrión es hipónimo de pájaro.

[2] Hiperónimo: Palabra cuyo significado está incluido en el de otras. Pájaro es hiperónimo de jilguero y de gorrión.

[3] La paráfrasis consiste en decir, con palabras más sencillas y con menos tecnicismos, las ideas propias obtenidas de un texto predeterminado. Por ejemplo: Si tenemos la frase “Dos hombres esperaban ya en la puerta a Pedro Páramo” podríamos construir la paráfrasis siguiente “**Un par de** hombres **aguardaban** ya en la **entrada** a Pedro Páramo”.

ha leído. En el caso de word embeddings de gran tamaño calculados en el corpus Common Crawl, como Glove o FastText, tenemos representaciones de palabras en mayúsculas, por ejemplo podemos tener las palabras ”dog´´, “Dog´´ y ”DOG´´. Si transformamos todas las palabras a minúsculas, las tres palabras se convertirán en la misma, y nos quedaremos que la última que leamos. Dado que en los word embeddings las palabras generalmente se ordenan por número de apariciones durante el entrenamiento, nos quedaremos con el vector que es una peor representación de la palabra. Lo que resulta un meta embedding considerablemente peor que el original. Por lo tanto hemos modificado el código para no transformar todas las palabras a minúsculas. Sin embargo, puesto que las palabras de los léxicos semánticos se encuentran en minúsculas es posible que al ajustar los word embeddings las representaciones de palabras en mayúsculas pierdan precisión.

En cuanto a los parámetros, hemos realizado en todos los casos 10 iteraciones ya que es lo recomendado por los autores. En el caso de los parámetros α_i y β_{ij} hemos usado los mismos valores usados en la publicación original. Se quiere dar mayor peso a las aristas α_i que conectan los vectores adaptados (q) a los vectores de palabras originales (q̂) que a las artísticas que conectan vectores adaptados ente ellos β_{ij}. Por lo que todas las aristas α_i toman el valor 1 y β_{ij} toman el valor $grado(i)^{-1}$ donde i es el nodo al que se está aplicando la actualización.

La metodología utilizada para las pruebas es la detallada en la Sección 3.1. Como el retrofitting no afecta al vocabulario disponible, no nos interesa tener en cuenta el número de pares de palabras para los que se puede dar respuesta. Por ello en esta sección usaremos tan solo la primera métrica detallada en la Sección 3.1. Ignoraremos los pares de palabras para los que un word embedding no puede dar respuesta, por lo tanto no se penaliza que un word embedding no pueda dar respuesta para un número determinado de pares de palabras.

Usaremos la siguiente nomenclatura durante las pruebas:

Embedding_Léxico

Donde *embedding* representa el nombre del word embedding al que se le ha aplicado retrofitting y *Léxico* indica que léxico semántico se ha usado en las pruebas, PPDB o WordNet (abreviado como WN, WN+ si se incluyen hiperónimos e hipónimos).

En la Tabla 6.23 se muestran los resultados obtenidos por diferentes word embeddings tras aplicar retrofitting usando diferentes léxicos semánticos. Para facilitar futuras comparativas se muestra la media y la media ponderada obtenidas en los diferentes dataset.

También se muestra la diferencia entre el rendimiento obtenido por el embedding tras el retrofitting con el rendimiento original del mismo word embedding.

Embeddings	Media (Diferencia)	Media Ponderada (Diferecia)	Media	Media Ponderada
FT_ppdb	0.024	0.033	0.699	0.653
FT_WN	-0.019	0.007	0.656	0.627
FT_WN+	-0.014	-0.006	0.661	0.614
GLOVE_ppdb	0.042	0.046	0.671	0.617
GLOVE_WN	0.014	0.033	0.643	0.604
GLOVE_WN+	0.017	0.018	0.646	0.589
jointcHYB_ppdb	0.028	0.042	0.682	0.648
jointcHYB_WN	-0.010	-0.001	0.644	0.605
jointcHYB_WN+	0.000	0.001	0.654	0.607
LEXVEC_ppdb	0.037	0.016	0.667	0.622
LEXVEC_WN	0.003	-0.006	0.633	0.600
LEXVEC_WN+	0.009	-0.018	0.639	0.588
PDC_ppdb	0.040	0.051	0.639	0.580
PDC_WN	-0.024	0.006	0.575	0.535
PDC_WN+	-0.019	-0.006	0.580	0.523
UKB_ppdb	0.050	0.055	0.680	0.665
UKB_WN	-0.014	0.002	0.616	0.612
UKB_WN+	-0.003	0.007	0.627	0.617
W2V_ppdb	0.045	0.051	0.661	0.606
W2V_WN	0.002	0.028	0.618	0.583
W2V_WN+	0.011	0.015	0.627	0.570
SKE_ppdb	0.045	0.100	0.655	0.592
SKE_WN	-0.008	0.069	0.602	0.561
SKE_WN+	0.006	0.063	0.616	0.555

Tabla 6.23: Resultados obtenidos al aplicar retrofitting a diferentes word embeddings. Métrica utilizada: Ignorar pares de palabras para los que no se puede dar respuesta.

Como se puede observar en la Tabla 6.23 el retrofitting en general consigue aumentar el rendimiento de los word embedding. En cuanto a las dos versiones de WordNet, la que mejores resultados parece obtener es la versión que incluye hiperónimos e hipónimos (WordNet con solo sinónimos obtiene mejor resultado en algunos dataset con un gran número de palabras, en especial en RareWord haciendo que en ocasiones obtenga una mejor media ponderada). Esto ocurre por que usando sólo sinónimos no podemos capturar de

forma adecuada todas las relaciones entre palabras. Por ejemplo, la palabra “átomo” está relacionada con la palabra “hidrógeno”, pero usando solo los sinónimos de WordNet no podemos ”acercar´´ estas palabras mediante el retrofitting, de hecho existe la posibilidad de que aumentemos la distancia entre ellas debido a otras relaciones. En cambio, si incluimos hipónimos e hiperónimos si que existe una relación entre dichas palabras, puesto que átomo es un hiperónimo de hidrógeno, e hidrógeno un hipónimo de átomo.

Sin embargo, PPDB parece capturar mucho mejor las relaciones entre palabras. PDDB es una base de datos de paráfrasis, por ejemplo ”El **templo** había sido destruido´´, es una paráfrasis de ”La **iglesia** estaba ya vacía´´. Esta forma de capturar relaciones entre palabras parece funcionar mucho mejor con la técnica del retrofitting. Mientras WortNet consigue mejoras no demasiado significativas e incluso reduce el rendimiento en algunos casos. El retrofitting usando PPDB mejora de forma muy significativa el rendimiento de todos los word embeddings.

Para encontrar una explicación a por que PPDB obtiene un rendimiento significativamente mejor que WordNet hemos analizado los resultados obtenidos dataset por datastet. Como se ve en la Tabla 6.24, hemos encontrado que los embeddings tras aplicar retrofitting con WorNet obtienen buenos resultados en dataset donde se mide la sinonimia entre palabras. En especial consiguen mejores resultados en RG65 y SimVerb 3500. Ambos son datasets que se entran en analizar la sinonimia entre pares de palabras, por ejemplo en el dataset RG65 las palabras “pájaro” y “bosque” reciben una puntuación de 1.24 sobre 4, ya que aunque están relacionadas, no son sinónimos. En cambio en datasets donde si que se mide la relación entre palabras, como MTurk 771 o WS353 relatedness, los embeddings a los que se ha aplicado retrofitting con PPDB obtienen mejores resultados. Esto indica que WorNet es un léxico semántico muy centrado en la sinonimia, y provoca que al usarlo en el retrofitting los sinónimos reduzcan su distancia entre ellos, lo que puede afectar de forma negativa a otro tipo de relaciones entre palabras. En cambio PPDB es mejor capturando las relaciones entre palabras fuera de la sinonimia. Aunque teniendo en cuenta todos los datasets usar el retrofitting con PPDB nos aporte mejores resultados, realmente decir que PPDB es mejor que WorNet en para adaptar vectores no es totalmente cierto, ya que cada léxico semántico consigue un efecto diferente sobre los word embeddings. En caso de que los sinónimos sean importantes en la tarea en la que vamos a aplicar nuestros WordEmbeddings WordNet puede ser una mejor opción.

Embedding	MTurk771	WS353 related-ness	WS353 similarity	RG 65	SimVerb 3500
FT_ppdb	**0.784**	**0.723**	**0.843**	0.856	0.484
FT_WN	0.702	0.670	0.805	0.815	**0.495**
FT_WN+	0.754	0.669	0.835	**0.886**	0.492
GLOVE_ppdb	**0.772**	**0.729**	**0.834**	0.809	0.420
GLOVE_WN	0.726	0.693	0.810	0.778	**0.433**
GLOVE_WN+	0.753	0.693	**0.834**	**0.828**	0.417
jointcHYB_ppdb	**0.742**	**0.658**	0.827	0.847	**0.595**
jointcHYB_WN	0.646	0.650	0.816	0.840	0.580
jointcHYB_WN+	0.683	0.652	**0.833**	**0.863**	0.590
W2V_ppdb	**0.714**	**0.638**	**0.808**	0.812	0.443
W2V_WN	0.657	0.586	0.767	0.766	**0.452**
W2V_WN+	0.698	0.582	0.795	**0.839**	0.438

Tabla 6.24: Resultados obtenidos en diferentes dataset al aplicar retrofitting a diferentes word embeddings. Métrica utilizada: Ignorar pares de palabras para los que no se puede dar respuesta.

Hemos decido comprobar si podemos conseguir aún mejores resultados combinando PPDB y WordNet. Para ello hemos concatenado la información de PPDB (los archivos usados para el retrofitting) primero con la versión de WorNet que incluye tan solo sinónimos y después con la versión de WordNet formada por sinónimos, hiperónimos e hipónimos. Tras comprobar que WordNet parece obtener mejores resultados en cuanto a la sinonimia entre palabras y PPDB en relación entre palabras, buscamos comprobar si combinar PPDB con WordNet nos permite obtener mejores resultados.

	PPDB		PPDB + WN		PPDB + WN+	
Embedding	Media	Media Pondera-da	Media	Media Ponde-rada	Media	Media Pondera-da
FastText	**0.699**	**0.653**	0.665	0.630	0.671	0.617
GLOVE	**0.671**	**0.617**	0.649	0.606	0.652	0.592
W2V	**0.661**	**0.606**	0.626	0.585	0.637	0.573
UKB	**0.680**	**0.665**	0.616	0.612	0.627	0.617

Tabla 6.25: Resultados obtenidos al aplicar retrofitting a diferentes word embeddings combinando léxico semánticos. Métrica utilizada: Ignorar pares de palabras para los que no se puede dar respuesta.

Sin embargo, como se puede observar en la Tabla 6.25, concatenar léxicos semánticos no nos ha servido para obtener mejores resultados. De hecho los resultados indican que

WordNet está dominando a PPDB, esto se debe a que en PPDB también se encuentran muchos de los sinónimos de WordNet, por lo que estamos duplicándolos, haciendo que los sinónimos tengan todavía más peso en el retrofitting y descartando el conocimiento adicional de PPDB.

6.4.4. Conclusiones

En esta sección hemos estudiado la combinación del conocimiento de word embeddings con léxicos semánticos. Hemos comprobado el retrofitting puede ayudarnos a mejorar de forma significativa el rendimiento de los word embeddings. Cuando usamos PPDB para el retrofitting, en general, conseguimos muy buenos resultados. WordNet en cambio, se centra demasiado en los sinónimos, por lo que aunque consigue buenos resultados en datasets que evalúan la sinonimia entre palabras, en general teniendo en cuenta todos los dataset los resultados no son tan buenos como cuando usamos PPDB. En algunos casos podemos llegar a reducir el rendimiento de nuestros word embeddings.

En cuanto a la comparativa con el resto de métodos mostrados en esta sección, el retrofitting no es el método que mejores resultados obtiene. A diferencia del resto de métodos tampoco nos permite aumentar el vocabulario disponible. Sin embargo cuenta dos dos ventajas, la primera es que el tiempo para generar los meta embeddings es significativamente inferior a la del resto de métodos propuestos en estas sección. El algoritmo es muy rápido. La segunda ventaja, es que usar este método no nos impide el uso de otros métodos de generación de meta embeddings, es decir, podemos aplicar el retrofitting a dos word embeddings y después concatenarlos o hacer la media, consiguiendo así, como veremos en futuros capítulos word embeddings aún mejores que aplicando solo uno de los métodos.

6.5. Conclusiones generales:

En este capítulo hemos estudiado diferentes métodos de generación de meta embeddings. La concatenación de word embeddings, la reducción de dimensionalidad, la media de word embeddings y el retrofitting. El método que nos ha permitido generar los mejores meta embeddings ha sido la concatención. Para aplicar este método efectivamente hemos propuesto un algoritmo de aproximación de palabras. Este método nos permite solventar el problema que surge cuando existe un word embedding que cuenta con representación

para una palabra, pero existe otro que no cuenta con representación para dicha palabra. Además hemos comprobado la importancia de la normalización de los word embeddings para evitar que word emebddings que se encuentran en una escala superior dominen al resto, provocando que su conocimiento se pierda. Hemos demostrado también que la concatenación de word embeddings normalizados mediante la norma L_2 es equivalente a la media de las similitudes expuesta en el Capítulo 5. Por lo que todas las conclusiones a las que llegamos en dicho capítulo son aplicables a la concatenación. Destaca sobre todo que la forma de conseguir el mejor rendimiento posible es mediante la combinación de word embeddings lo más diferentes posibles. En nuestro caso, word embeddings obtenidos a partir de corpus de texto con word embeddings calculado a partir de grafos, en concreto WordNet. Combinar una gran cantidad de word embeddings similares, aunque estos cuenten con muy buen rendimiento no hará que consigamos un mejor rendimiento.

Sin embargo la concatenación cuanta con un problema importante. Los meta embeddings generados cuentan con tantas dimensiones como la suma de las dimensiones de los word embeddings concatenados. Esto provoca que los word embeddings generados sean demasiado grades para trabajar de forma eficiente con ellos. Para solucionar esto hemos estudiado la reducción de dimensionalidad y la media de word embeddings. Entre ambos métodos no hay un claro ganador que debamos aplicar en todas las situaciones. Sin embargo ninguno ha resultado ser un método especialmente interesante. La reducción de dimensionalidad provoca una pérdida de información importante sobre la sinonimia entre palabras. Además la capacidad de computo y memoria necesarios para aplicar estos algoritmos es muy alta. Esto dificulta mucho aplicar estos métodos a word embeddings de grandes dimensiones, generados a partir de la concatenación de un número alto de word embeddings. Por su parte la media es menos costosa, pero ofrece un mal rendimiento cuando la aplicamos a más de dos word embeddings. Por lo tanto, en el siguiente capítulo propondremos un nuevo método que busca solventar los problemas de los métodos estudiados en este capítulo.

Por últimos hemos estudiado el retroffiting. Hemos descubierto que combinar el conocimiento de word embeddings con la base de datos PPDB nos permite aumentar el rendimiento de los word embeddings. Los meta embeddings generados no son tan buenos como los que hemos generado mediante la concatenación, pero aplicar el retrofitting no impide aplicar otros métodos métodos de generación de meta embeddings posteriormente. Es decir, podemos aplicar el retroffiting a varios word embeddings, y a continuación concatenarlos. Por lo tanto el retrofitting resulta ser un método muy interesante. Estudiaremos esta posibilidad en futuros capítulos.

7. CAPÍTULO

Rotaciones de Word Embeddings para optimizar los métodos de creación de Meta-Embeddings

En el capítulo autorefchap:metaembeddings hemos mostrado diferentes métodos de generación de meta embeddings. Sin embargo todos ellos mostraban desventajas importantes. Los principales problemas que hemos encontrado están relacionados con el hecho de que cada word embedding se encuentra en un espacio vectorial diferente. Esto afecta especialmente a la media de word embeddings, método de generación de meta embeddings que en la teoría resulta muy interesante, pero como hemos podido comprobar, en la práctica, rta los resultados esperados.

A nuestro juicio, el método de generación de meta embeddings ideal debe cumplir una serie de requisitos:

- El primero es ser capaz de combinar diferentes fuentes de conocimiento sin que durante el proceso se produzca una pérdida de información significativa. Esto lo hemos logrado con la concatenación y la media de las similitudes. Ambos métodos pueden producir meta embeddings donde la información de todos los word embeddings se preserve y contribuya por igual al resultado final.

- El segundo es ser capaz de generar meta embeddings con un número de dimensiones reducido. Un meta embedding de 1800 dimensiones, aunque obtenga un rendimiento muy alto, no es de gran utilidad. Su almacenamiento resulta complicado, realizar operaciones con él requiere de una gran potencia de computo y memoria...

El número de dimensiones más común actualmente son 300, buscamos que nuestros word embeddings cuenten con un número de dimensiones similar.

- Por último, el método debe ser capaz de combinar los vocabularios de diferentes word embeddings de forma efectiva, generando un meta embedding cuyo vocabulario sea más amplio que el de los word embeddings utilizados para generarlo individualmente. Aspiramos que el meta embedding generado cuente con un vocabulario igual a la unión de los vocabularios de los word embeddings usados para generarlo. En este apartado destaca la problemática a la que nos hemos enfrentado tanto en la concatenación como en la media de word embeddings. Si buscamos que el vocabulario del meta embedding generado sea igual a la unión de los vocabularios de los word embeddings empleados para generarlo, el método debe ser capaz de resolver de forma eficaz la situación en la que un word embedding cuente con representación para una palabra, pero uno o varios word embeddings no cuenten con representación para dicha palabra. Es decir, debe ser capaz de tratar de forma adecuada las palabras fuera de la intersección de los vocabularios de los word embeddings que estamos combinando.

Hasta ahora ninguno de los métodos estudiados cumple estos tres requisitos. Es por ello que hemos desarrollado un nuevo método de generación de meta embeddings con el que esperamos cumplirlos. Nuestro método es una extensión de la media de word embeddings. Como hemos comprobado en el Capítulo 6, la media de word embeddings depende en gran medida de el ángulo que formen los word embeddings utilizados. Esta diferencia en los espacios vectoriales de los word embeddings también provoca que la media no sea capaz de combinar de forma efectiva más de dos word embeddings. Sin embargo, solucionar esta problemática es posible. Si rotamos los word embeddings para que todos ellos se alineen en un mismo espacio vectorial, esperamos resolver los problemas que hemos encontrado en la media de word embeddings. Resolver estos problemas nos permitiría cumplir los requisitos con los que deseamos que cuente un método de generación de word embeddings.

7.1. Rotación de Word Embeddings: VecMap

La rotación de word embeddings y alineación de espacios vectoriales han recibido una gran atención recientemente. Sin embargo, los estudios se han centrado en alinear word

embeddings en diferentes idiomas a un espacio vectorial común. El objetivo es el siguiente. Imaginemos que disponemos del word embedding A el cual ha sido generado a partir de corpus de textos en inglés. Y también disponemos del embedding B que ha sido entrenado en corpus de texto en castellano. Buscamos rotar ambos word embeddings de forma que para cada palabra w de A, la palabra más cercana en el embedding B de w, sea la palabra que tiene el mismo significado que w en castellano. Es decir, si tomamos el vector que representa a la palabra "love" y calculamos la distancia a cada una de las palabras de B, esperamos que la palabra más cercana sea "amor". Los sistemas de traducción automática modernos como seq2seq [78] , generalmente están formados por una red neuronal que es entrenada usando corpus paralelos. Es decir, documentos, frases o palabras idénticos escritos en diferentes idiomas. Si bien estos modelos han conseguido obtener buenos resultados, obtener una gran cantidad de corpus paralelos entre diferentes idiomas no siempre es una tarea fácil. Es por ello que la alineación de word embeddings ha recibido gran atención. Alinear los espacios vectoriales de word embeddings en diferentes idiomas permite la implementación de sistemas de traducción automática que no cuentan con la necesidad de corpus paralelos para su entrenamiento. Un ejemplo de estos nuevos sistemas es "UNdreaMT" [17] un modelo neuronal de traducción automática no supervisado que no necesita corpus paralelos para su entrenamiento.

Nosotros buscamos aplicar estas técnicas de rotación de word embeddings a la generación de meta embeddings. En nuestro caso el objetivo es que dados por ejemplo Glove y FasText, rotemos ambos word embeddings de forma que dada una palabra, por ejemplo w de Glove, la palabra más cercana a w en FastText sea la palabra que representa a w en FastText, es decir, la misma palabra. De esta forma, Glove y FastText se encontrarán en un mismo espacio vectorial, y como explicaremos más adelante, esto proporciona ventajas muy importantes a la hora de combinar ambos word embeddings para generar un meta embeddings.

La aplicación "UNdreaMT" [17] mencionada anteriormente, cuenta con una pieza fundamental para su funcionamiento: VecMap. VecMap es un software desarrollado por "Artetxe et al" que es capaz de aprender mapeados de incrustación de palabras en varios idiomas. Es decir, es capaz de alinear los espacios vectoriales de word embeddings en diferentes idiomas. VecMap es incluso capaz de aprender mapeados multilingües de forma no supervisada. Sin embargo, queremos rotar word embeddings que se encuentran en un mismo idioma, por lo tanto, generar un diccionario es extremadamente sencillo, ya que todas las palabras se encuentran en un mismo idioma.

Puesto que tenemos a nuestra disposición esta herramienta, no hemos implementado nues-

tro propio software de mapeado de word embeddings, usaremos VecMap.

A continuación describiremos en profundidad el funcionamiento de VecMap. VecMap realiza el mapeado de word embeddings siguiendo una serie de pasos:

1. Normalización de word embeddings: El paso inicial es la normalización de word embeddings. Este paso es opcional, sin embargo puede mejorar de forma significativa el resultado final. Este paso puede involucrar dos métodos de normalización. El primero es la normalización de la longitud de los vectores mediante la norma L_2 haciendo que todos los word embeddings tengan una longitud euclídea de 1. Esto asegura que todos los vectores tengan el mismo peso durante el cálculo del mapeado. El segundo es el centrado de variables, haciendo que cada variable tenga una media igual a 0 y misma dispersión. Los autores recomiendan aplicar ambos métodos de normalización, ya que ayuda a preservar el conocimiento de los word embeddings durante el mapeado, por lo que aplicaremos ambos.

2. Whitening (opcional): Este paso aplica una transformación esférica a los vectores, lo que hace que las diferentes componentes tengan una varianza de 1 y no estén correlacionados entre sí, convirtiendo sus matrices de covarianza en la matriz identidad. Probaremos si aplicar o no esta transformación nos permite obtener mejores word embeddings.

3. Mapeado Ortogonal. Esta es la fase en el que los diferentes word embeddings son mapeados al mismo espacio vectorial. Existen diferentes métodos de mapeado de word embedding, sin embargo el mapeado ortogonal resulta el más atractivo para nuestra tarea. El mapeado ortogonal asegura que el producto vectorial entre cualquier palabra de cada word embedding se mantendrá tras el mapeado. Es decir, nos permite mantener intacto el conocimiento que contiene cada word embedding. Otros métodos de mapeado provocan que la información que contienen los word embeddings se degrade, reduciendo el rendimiento en tareas como la similitud entre palabras, tarea en la que nos estamos centrando. Imaginemos que disponemos de los word embeddings X y Z, de forma que X_i se corresponde con la i-ésima palabra del word embeddings fuente y Z_j se corresponde con la j-ésima palabra del lenguaje objetivo (al que será mapeado el word embedding fuente). El diccionario se representa mediante una matriz binaria D, donde $D_{ij} = 1$ si la i-ésima palabra del embedding fuente está alineada con la j-ésima palabra del lenguaje objetivo. El objetivo es encontrar la matriz de mapeado óptima W^* tal que la suma de los cuadra-

dos de las distancias euclídeas entre el word embedding fuente mapeado $X_{i*}W$ y el word embedding objetivo Z_{j*} para las entradas del diccionario D_{ij} es minimizado:

$$W^* = \arg\min_W \sum_i \sum_j D_{ij} \|X_{i*}W - Z_{j*}\|^2$$

Como hemos mencionado anteriormente buscamos que el mapeado sea ortogonal para preservar el conocimiento de los word embeddings. Por lo tanto se fuerza que W sea una matriz ortogonal ($WW^T = W^T$" $= I$). Bajo la condición de que la matriz sea ortogonal, minimizar el cuadrado de la distancia euclídea, es equivalente a maximizar el producto escalar, por lo tanto el objetivo de minimización anterior se convierte en:

$$W^* = \arg\max_W \mathrm{Tr}(XWZ^T D^T)$$

Donde $\mathrm{Tr}(\cdot)$ denota la suma de todos los elementos en la diagonal principal. Si tomamos la descomposición en valores singulares (SVD) de $X^T DZ$ como $X^T DZ = U\Sigma V^T$, la solución óptima al problema será $W* = UV^T$. Puesto que D es una matriz en la que la mayor parte de sus elementos son cero, es posible esto se puede realizar en tiempo lineal resto al número de entradas en el diccionario.

4. : Re-weighting (opcional): Este paso modifica los pesos de cada variable en función de su correlación cruzada, haciendo que se aumente la relevancia de las variables que mejor coinciden en los diferentes word embeddings. Solo se aplicará si se ha aplicado el segundo paso. Puede ser aplicado al word embedding fuente u objetivo. En nuestro caso probaremos si obtener mejores resultados aplicando o no este paso.

5. De-whitening (opcional): Este paso restaura la varianza original en todas las direcciones, solo se aplica si se ha aplicado el primer paso. Probaremos si obtenemos mejores resultados aplicándolo o no.

6. Reducción de dimensionalidad (opcional): Este paso no resulta interesante para la tarea a la que vamos a aplicar VecMap, por lo que lo omitiremos.

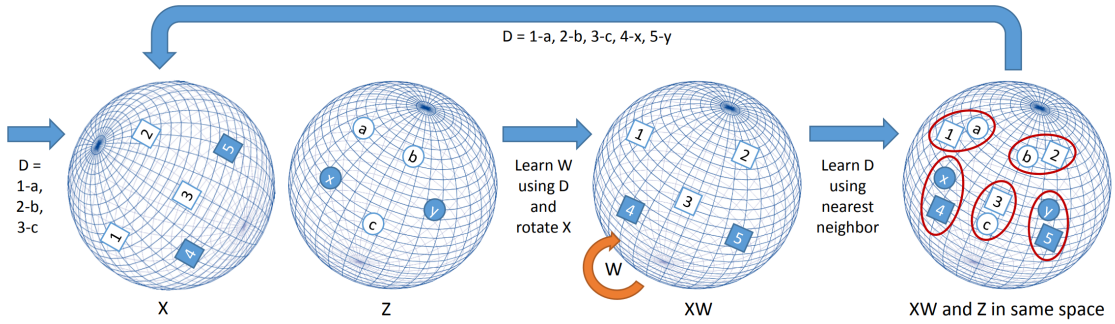


Figura 7.1: Ilustración del mapeado de word embeddings que realiza VecMap.

7.2. Método de generación de Word Embeddings: Media de word embeddings alineados

Una vez explicados los mapeados de word embeddings, detallaremos nuestro método de generación de meta embeddings. Nuestro método se trata de una extensión de la media de word embeddings, sin embargo cuenta con una diferencia fundamental. Previamente al cálculo de la media, alineamos los word embeddings que vamos a utilizar a un mismo espacio vectorial. Esto nos proporciona diversas ventajas:

- La media deja de ser dependiente del ángulo que forman los word embeddings. De esta forma la media pasa a ser un método casi equivalente a la concatenación, por los que los meta embeddings generados serán muy similares a los generados por la concatenación de word embeddings. Son casi equivalente, ya que los word embeddings no son idénticos unos a otros, por lo que el mapeado puede ser perfecto y existirá un pequeño ángulo entre los vectores. Sin embargo esta diferencia entre word embeddings es lo que hace que queramos combinar su conocimiento para generar mejores word embeddings. Este método por lo tanto, no provocará la pérdida de información que observábamos la sección Sección 6.3, al aplicar el método de la media de word embeddings. Y respecto a la concatenación este método cuenta con una ventaja muy importante, no aumenta la dimensionalidad del meta embedding generado a media que añadimos más word embeddings a la combinación. Por lo que los meta embeddings generados por este método cuentan con un número de dimensiones que facilita su aplicación en todo tipo de tareas.

- Elimina la posibilidad de que diferentes word embeddings se “contraresten” entre ellos. En la Sección 6.3 pudimos observar que la medida perdía rendimiento

a media que añadíamos word embeddings. Si realizásemos la media de infinitos word embeddings en diferentes espacios vectoriales la longitud de todos los vectores generados tendería a cero. Dicho comportamiento no ocurre si todos los word embeddings se encuentran en un mismo espacio vectorial, lo que elimina la posibilidad de que dos vectores se contraresten entre ellos produciendo una pérdida de información.

- Trata de forma adecuada la situación en la que un word embedding cuente con representación para una palabra para la que otro word embedding no cuenta. Imaginemos que disponemos del embedding A y el embedding B, y queremos calcular la media para los vectores que representan a w en A y B. En caso de que A cuente con representación para w, pero B no cuente con representación para w, podemos tomar directamente la representación de w en A. Esto es debido a que ambos word embeddings se encuentran en el mismo espacio vectorial, por lo que los que los vectores que representan a las mismas palabras en A y B cuentan con una similitud coseno muy alta entre ellas. Esto quiere decir que si B constase con representación para w, dicha representación debería ser similar a la de w en A, por lo tanto la media entre ambas representaciones debe ser al mismo tiempo similar a la representación de w en A y w en B. Esto hace que la representación de w en A sea una buena aproximación de la media de los vectores para w en A y B. Sin embargo hay que tener en cuenta que los word embeddings pueden estar en diferentes escalas. La similitud coseno no tiene en cuenta la escala de los vectores, pero si vamos a aplicar los word embeddings a otro tipo de tarea donde si que se tome en cuenta, para que se cumpla lo que acabamos de exponer será necesario normalizar los word embeddings. Además, si queremos obtener todavía mejores aproximaciones, podemos usar también el método de aproximación de palabras descrito en la Subsubsección 6.1.1.

Implementación del método

En esta sección detallaremos como se ha implementado el método. El método costa de varias fases:

1. Normalización de los word embeddings: Los autores de VecMap resaltan la importancia de normalizar los vectores antes de realizar el mapeado. En concreto se recomienda realizar un centrado de variables y una normalización L_2 de los vectores. Por lo tanto, la primera fase del método es la normalización de los vectores. Para

ello, aunque hemos implementado ambos métodos de normalización y los hemos usado en capítulos anteriores, hemos hecho uso de las funciones de normalización que aporta VecMap. Hemos usado el siguiente comando para la normalización de los vectores:

```
python3 normalize_embeddings.py unit center -i EMBEDDING.txt -o RESULT.txt
```

2. Mapeado de los word embeddings a un mismo espacio vectorial: En este caso hemos realizado dos pruebas. La primera es aplicar solo una rotación ortogonal que realice un mapeado de un word embedding al espacio vectorial de otro. Hemos probado a mapear los word embeddings al espacio vectorial de FastText y al espacio vectorial de jointcHYB. Podemos mapear multiples word embddings al espacio vectorial de un word embdding, de esta forma podemos combinar más de dos word embddings. Para ello hemos usado el siguiente comando:

```
python3 map_embeddings.py --orthogonal SRC_EMBEDDINGS.NORMALIZED.TXT  TRG_EMBEDDINGS.
 NORMALIZED.TXT   -d TRAIN_DICTIONARY.TXT
```

   También hemos probado a realizar un mapeado usando todos los pasos opcionales de VecMap en el que ambos word embeddings son mapeados y sus variables se ajustan para tratar de que el mapeado sea lo más exacto posible.

```
python3 map_embeddings.py --whiten --src_reweight 0.5 --trg_reweight  0.5 --
 src_dewhiten src --trg_dewhiten trg  SRC_EMBEDDINGS.NORMALIZED.TXT TRG_EMBEDDINGS.
 NORMALIZED.TXT  SRC_EMBEDDINGS.MAPPED.TXT TRG_EMBEDDINGS.MAPPED.TXT -d
 TRAIN_DICTIONARY.TXT
```

   Para realizar los mapeados es necesario un diccionario, VecMap incluye la posibilidad de inferirlo, pero dada la sencillez de generar un diccionario en nuestro caso no tiene sentido usar esa posibilidad. Hemos generado los diccionarios como la intersección de los vocabularios de los embeddings que estamos mapeando. Es decir, nuestro diccionario será de la forma:

```
dog dog
cat cat
car car
love love
human human
... ...
```

3. Media de los word embeddings: La media de los word embeddings la hemos realizado de la misma forma que en la Sección 6.3. Con la diferencia de que hemos usado word embeddings mapeados a un mismo espacio vectorial. Hemos realizado diversas pruebas: Normalizar o no los vectores antes de realizar la media, y usar o no el algoritmo de generación de palabras.

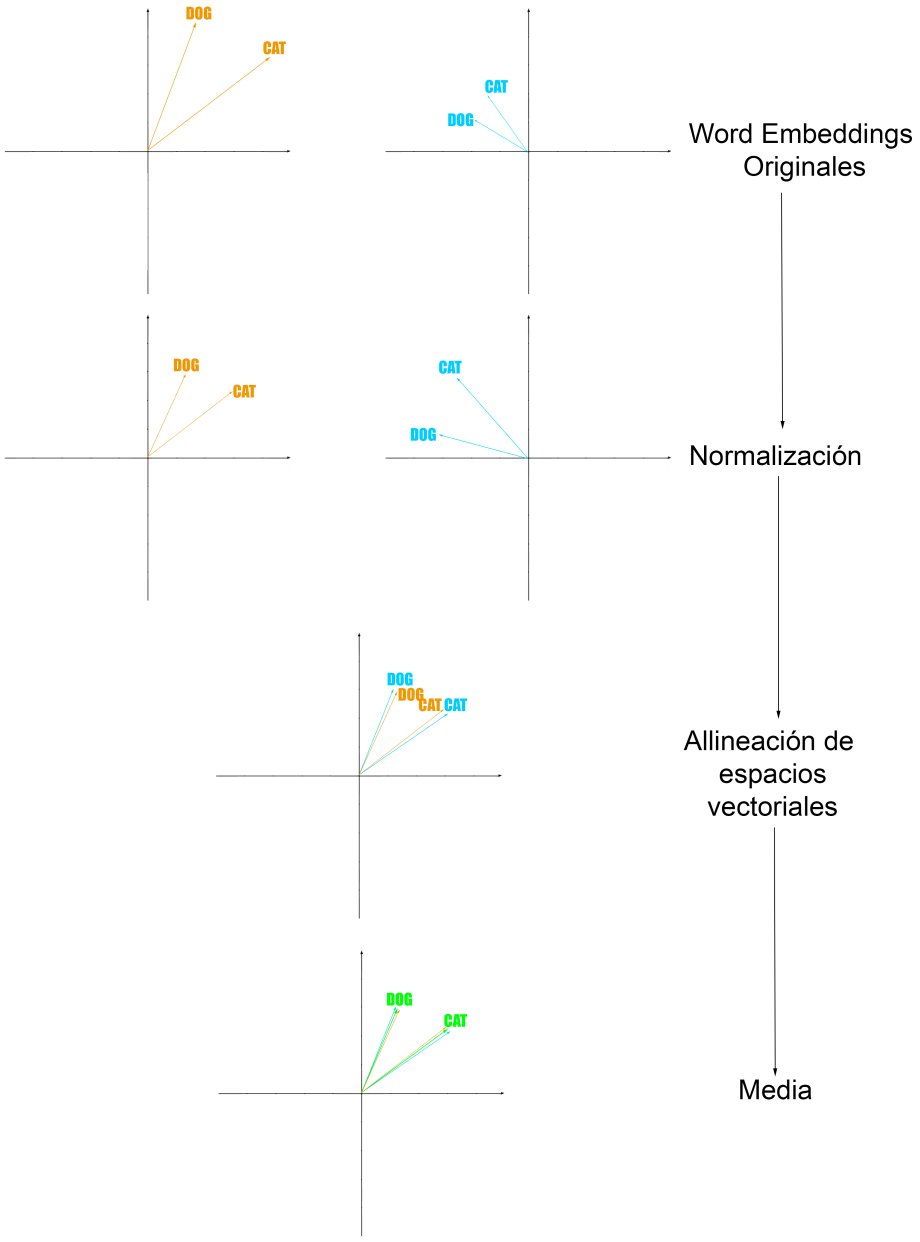


Figura 7.2: Explicación simplificada del método de generación de meta embeddings

7.3. Estudiando y optimizando el método

El objetivo de esta sección es estudiar las diferentes variaciones del algoritmo de generación de meta embeddings para decidir cual es la que mejor rendimiento proporciona. Estudiaremos el efecto de los diferentes métodos de mapeado, las diferentes formas de realizar la media de word embeddings, y el mapeado a diferentes espacios vectoriales. Con esto esperamos determinar que proceso debemos seguir para obtener los mejores word embeddings con este método. También esperamos poder estudiar diferentes características con las que cuenta este método y las ventajas que ofrece respecto a otros métodos. Por lo que dividiremos la sección en varias subsecciones en las que estudiaremos aspectos diferentes del método. La metodología utilizada para las pruebas realizadas es la detallada en la Sección 3.1.

Primero hemos realizado una prueba para comprobar el efecto el mapeado de word embeddings. Para ello, hemos realizado una rotación ortogonal en la que hemos mapeado Glove a FastText. A continuación hemos extraído tres palabras de Glove "dog", "car" y "love". Hemos probado a extraer el vector que representa a las tres palabras en Glove y hemos buscado las 10 palabras más cercanas, usando la similitud coseno, a esas representaciones en FastText. Además hemos calculado la similitud coseno entre la representación de Glove y FastText para las tres palabras. Hemos realizado esto tanto con Glove mapeado a FastText, es decir, estando Glove y FastText en un mismo espacio vectorial. Y con las versiones de Glove y FastText originales, es decir, situadas en espacio vectoriales diferentes.

Palabra	Similitud coseno GLOVE-FT	Similitud coseno Glove (mapeado a FT)-FT	10 palabras más cercanas Glove->FT	10 palabras más cercanas Glove(mapeado a FT) ->FT
Dog	0.028	0.785	ABREU, self-trained, blewits, bioplastics, professional, Thomaz, biorefineries, bio-plastics, yStats.com, PEFC-certified	dog, dogs, puppy, Dog, canine, pet, pup, doggie, kennel, doggy
Car	0.022	0.794	SBSTA, Mungatana, Waiganjo, FSDC, Lamfalussy, Blackden, Koech, SG17, Batie, Uplyme	car, Car, cars, automobile, vehicle, car., Vehicle, Cars, vehicles, truck
Love	0.004	0.690	Clappison, 26yo, Lazarowicz, Koosh, Westmacott, Lightbown, Stipetic, Noff, encouaged, Saska	love, loving, LOVE, adore, Love, loved, loves, hate, lovers, lover

Tabla 7.1: Comparativa entre las palabras más cercanas de Dog, Car y Love extraídas de Glove en FastText y de Glove mapeado a FastText

Como podemos observar en la Tabla 7.1 cuando calculamos las diez palabras más cercanas en FastText de las representaciones de palabras extraídas de Glove, puesto que ambos word embeddings se encuentran en espacios vectoriales diferentes las 10 palabras más cercanas no guardan ninguna relación con las palabras extraídas. Además la similitud coseno entre las representaciones de “dog”, “car” y “love” extraídas de Glove y FastText es muy baja. En cambio, si mapeamos Glove a FastText conseguimos que ambos word embeddings se encuentren en un mismo espacio vectorial, y este caso, las palabras más cercanas en FastText guardan una gran relación con las extraídas de Glove, siendo la más cercana la propia palabra extraída de Glove. Además, la similitud coseno entre “dog”, “car” y “love” de las representaciones de Glove mapeado a FastText y FastText es ahora alta. Por lo tanto, hemos comprobado que las rotaciones actúan de la forma que esperamos.

7.3.1. Comparativa de los métodos de normalización:

Como hemos mencionado anteriormente, VecMap nos permite aplicar o no una serie de pasos opcionales. Estos pasos opcionales buscan que el mapeado de word embeddings sea lo más preciso posible. Sin embargo, producen cambios en los word embeddings alterando el conocimiento que contienen. Por lo tanto, nuestro primer paso va a ser determinar que tipo de rotación vamos a utilizar. El caso más básico es el de realizar una rotación ortogonal, en la que solo se aplique el paso de normalización de los word embeddings, y el mapeado de un word embedding a otro. Esta rotación tiene la ventaja de que no altera el conocimiento de los word embeddings. Llamaremos a este método ”Mapeado orotogonal´´. El segundo método se denomina ”OLS´´. Este método cosiste en aplicar el whitening (paso 2) a los dos word embeddings. Tras el mapeado se aplica el re-weight (paso 4) y de-whiten (paso 5) a ambos word embeddings. El objetivo de este método es que el mapeado sea la más preciso posible. Sin embargo puesto que modifica el conocimiento de los word embeddings, puede resultar complicado mapear más de dos word embeddings a un mismo espacio vectorial.

Hemos aplicado el método de generación de meta embeddings utilizando ambos mapeados y hemos evaluado los meta embeddings resultantes en los diferentes dataset. Todos los word embeddings han sido mapeados al espacio vectorial de FastText. Para la media de word embeddings se ha usado el algoritmo de aproximación de palabras y no se ha aplicado ninguna normalización a los word embeddings. Podemos ver los resultados en la Tabla 7.2

	OLS		Rotación ortogonal	
Embedding	Media	Media Ponderada	Media	Media Ponderada
FT + JointcHYB	0.662	0.599	**0.707**	**0.652**
FT + LEX-VEC	0.638	0.573	**0.666**	**0.607**
FT + UKB	0.698	0.653	**0.728**	**0.679**
FT + W2V	0.635	0.570	**0.661**	**0.601**
FT + GLO-VE	0.641	0.578	**0.668**	**0.605**

Tabla 7.2: Comparativa de los embeddings generados usando como mapeado OLS y una rotación ortogonal. Todos los word embeddings han sido mapeados al espacio de FastText. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

Como se puede observar en la Tabla 7.2, la rotación ortogonal ofrece un rendimiento significativamente superior a OLS en la tarea a las que estamos aplicando los word embeddings. El pre-procesamiento y post-procesamiento que aplica OLS provoca que se pierda parte del conocimiento de los word embeddings durante el mapeado. La rotación ortogonal preserva las distancias entre las palabras del word embeddings que se está mapeando, es decir, no altera su conocimiento. Puesto que se trata de una rotación de un word embedding al espacio vectorial de otro, es posible mapear varios word embeddings al espacio vectorial de un mismo word embeddings permitiendo aplicar nuestro método de generación de meta embeddings a el número de word embeddings que queramos. Por ejemplo, podemos rotar UKB, Glove y Word2Vec al espacio de FastText y realizar la media de los cuatro word embeddings.
Por lo tanto, a partir de este momento siempre realizaremos el mapeado de word embeddings usando el método de la rotación ortogonal.

7.3.2. Escogiendo los parámetros de la media de word embeddings

Una vez realizado el mapeado es necesario realizar la media de los word embeddings. En este punto debemos decidir si aplicar o no la normalización L_2 a los vectores para asegurar que todos los word embeddings tengan el mismo peso en la media. Y la segunda decisión que tenemos que tomar es respecto a las palabras fuera de las intersección del vocabulario de los word embeddings usados. En el caso de que un word embedding cuente con representación para una palabra pero uno o varios word embeddings no cuenten con representación para dicha palabra, debemos decir como realizar la media. Puestos que todos los word embeddings se encuentran en un mismo espacio vectorial podemos reali-

zar la media con las representaciones disponibles para cada palabra, es decir, si estamos realizando la media de cuatro word embeddings pero solo dos cuentan con representación para una palabra, realizaríamos la media de las dos representaciones disponibles. La segunda opción es aplicar el algoritmo de aproximación de palabras descrito en la Subsubsección 6.1.1, siguiendo el ejemplo anterior, este algoritmo nos permite generar una representación aproximada para los word embeddings que no cuentan con representación para la palabra para la que estamos calculando la media.

Normalización L_2

Lo primero que vamos a comprobar es como afecta la aplicación de la normalización L_2 a los word embeddings antes de realizar su media. Para ello hemos mapeado varios word embeddings al espacio vectorial de FastText y a continuación hemos realizado la media, tanto aplicando la normalización L_2 como no aplicándola. Para realizar la media hemos utilizado el algoritmo de aproximación de palabras. Una vez generados los meta embeddings hemos comprobado su rendimiento en los diferentes datasets. Los resultados se muestran en la Tabla 7.3

	Vectores no normalizados		Normalización L2 de los vectores	
Embedding	Media	Media Ponderada	Media	Media Ponderada
GLOVE + UKB	0.705	0.645	**0.707**	**0.650**
FT + jointcHYB	**0.707**	0.652	**0.707**	**0.654**
FT + UKB	**0.728**	**0.679**	0.726	0.678
UKB + LEXVEC	**0.708**	0.650	**0.708**	**0.652**
GLOVE + jointcHYB + UKB	0.707	0.649	**0.709**	**0.652**
FT + UKB + LEXVEC	0.716	0.657	**0.717**	**0.659**
GLOVE + FT + jointcHYB + UKB	0.718	0.659	**0.721**	**0.662**
FT + jointcHYB + UKB + LEXVEC	0.719	**0.662**	**0.721**	**0.662**
GLOVE + W2V + FT + jointcHYB + UKB + LEXVEC	0.704	0.640	**0.706**	**0.641**

Tabla 7.3: Comparativa de aplicar el método de generación de meta embeddings aplicando o no la normalización L_2 antes de realizar la media de los word embeddings.

Como se puede observar en la Tabla 7.3 aplicar o no la normalización L_2 supone una dife-

rencia de rendimiento despreciable. Tenemos que tener en cuenta, que antes del mapeado de los word embeddings ya aplicamos dos métodos de normalización, la normalización L_2 de los vectores y el centrado de variables. Esa normalización L_2 se preserva en las siguientes fases del algoritmo, por lo que no es necesario aplicarla de nuevo. Podemos omitir aplicar la normalización L_2 a los vectores antes de realizar la media.

Media de los vectores dispones vs algoritmo de aproximación de palabras

Como hemos explicado anteriormente, la rotación de word embeddings a un mismo espacio vectorial cuanta con una ventaja importante. Imaginemos que disponemos del word embedding A y el word embedding embedding B, y queremos calcular la media para los vectores que representan a la palabra w en A y B. En caso de que A cuente con representación para w, pero B no cuente con representación para w, podemos tomar directamente la representación de w en A. Esto es debido a que ambos word embeddings se encuentran en el mismo espacio vectorial, por lo que los que los vectores que representan a las mismas palabras en A y B cuentan con una similitud coseno muy alta entre ellas. Esto quiere decir que si B constase con representación para w, dicha representación debería ser similar a la de w en A, por lo tanto la media entre ambas representaciones debe ser al mismo tiempo similar a la representación de w en A y w en B. Por lo tanto, al representación de w en A es una buena aproximación de la media. Esto quiere decir, que cuando un embedding cuente con representación para una palabra y uno o varios no cuenten con representación para dicha palabra, podemos realizar la media entre las representaciones de los word embeddings que cuentan con representación para la palabra. Y dicha media será una buena aproximación del vector resultante en el hipotético caso en que todos los word embeddings contasen con representación para la palabra. Por otro lado, disponemos del algoritmo de aproximación de palabras, descrito en la Subsubsección 6.1.1. Este algoritmo nos permite aproximar la representación para cualquier palabra dentro de la unión de los vocabularios. Por lo tanto con este método podemos generar representaciones aproximadas para los word embeddings que no cuenten con representación para la palabra que estamos generando.

La comparativa entre estos dos métodos resulta interesante, ya que podremos analizar como afecta el mapeado de word embeddings a un mismo espacio vectorial a esta problemática. En este caso vamos a analizar lo que ocurre al realizar la media entre word embeddings con un vocabulario limitado, como UKB, en datasets donde al menos uno de los word embeddings no cuente con representación para un número significativo de

palabras. Buscamos que la diferencia entre la unión y la intersección de los vocabularios de los word embeddings usados sea lo mayor posible para que sea posible estudiar la diferencia entre ambos métodos. Hemos mapeado diferentes word embeddings al espacio de FastText y hemos realizado la media usando ambos métodos y sin normalizar los vectores mediante la norma L_2 antes de realizar la media. Los resultados se muestran en la Tabla 7.4.

	Media de los vectores disponibles			Algoritmo de aproximación de palabras		
Embedding	MTurk-287	VERB-143	RW	MTurk-287	VERB-143	RW
W2V + UKB	0.722	**0.467**	0.518	**0.736**	0.443	**0.539**
FT + UKB	**0.751**	**0.468**	0.549	**0.751**	**0.468**	**0.584**
GLOVE + UKB	0.750	0.408	0.504	**0.754**	**0.416**	**0.536**
UKB + LEX-VEC	0.756	0.424	0.516	**0.768**	**0.435**	**0.544**
jointcHYB + UKB	0.652	**0.283**	0.377	**0.651**	0.264	**0.384**
FT + UKB + LEXVEC	0.769	0.470	0.568	**0.776**	**0.479**	**0.581**
W2V + FT + UKB + LEX-VEC	0.752	0.467	0.569	**0.763**	**0.468**	**0.572**
GLOVE + W2V + FT + jointcHYB + UKB + LEX-VEC	0.748	0.454	0.545	**0.752**	**0.465**	**0.553**

Tabla 7.4: Comparativa entre realizar la media de las representaciones disponibles y el método de aproximación de palabras. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

Como podemos observar en la Tabla 7.3, el algoritmo de aproximación de palabras produce mejores resultados que realizar la media de las representaciones disponibles. Sin embargo realizar la media de las representaciones disponibles muestra un buen rendimiento, en muchos casos similar al del algoritmo de generación de palabras. Esto muestra que la alineación de espacios vectoriales permite tratar de forma efectiva las palabras fuera de la intersección del vocabulario de los word embeddings usados. Sin embargo, gracias al método de aproximación de palabras descrito en la Subsubsección 6.1.1 podemos generar mejores aproximaciones, los que nos permite mejorar el rendimiento cuando combinamos word embeddings con un vocabulario reducido.

7.3.3. Mapeado a diferentes espacios vectoriales

En las pruebas realizadas hasta ahora hemos mapeado todos los word embeddings al espacio vectorial de FastTex, sin embargo, es posible mapear los word embeddings al espacio

vectorial de cualquier word embeddings. En esta sección queremos comprobar si mapear word embeddings a diferentes espacios vectoriales puede afectar al rendimiento. Concretamente vamos a comprobar el rendimiento de mapear los word embeddings al espacio vectorial de FastText y al espacio vectorial de jointcHYB. Para ello hemos mapeado varios word embeddings primero a un espacio y luego al otro y hemos probado a realizar la media entre ellos. En ambos casos no se va a aplicar la normalización L_2 de los vectores antes de realizar la media, y se va a usar el algoritmo de aproximación de palabras. Los resultados se muestran en la Tabla 7.5.

	Mapeado al e.v de FastText		Mapeado al e.v de jointcHYB	
Embeddings	Media	Media Ponderada	Media	Media Ponderada
FT + jointcHYB	**0.707**	**0.652**	**0.707**	**0.652**
FT + UKB	0.728	**0.679**	**0.731**	0.675
UKB + LEXVEC	**0.708**	**0.650**	0.704	0.649
GLOVE + jointcHYB	0.687	**0.623**	**0.688**	**0.623**
GLOVE + FT + UKB	**0.716**	**0.654**	**0.715**	0.651
GLOVE + jointcHYB + UKB	0.707	0.649	**0.710**	**0.650**
GLOVE + W2V + jointcHYB + UKB	0.705	0.643	**0.709**	**0.647**
FT + jointcHYB + UKB + LEXVEC	0.719	0.662	**0.720**	**0.663**
GLOVE + W2V + FT + jointcHYB + UKB + LEXVEC	0.704	0.640	**0.706**	**0.642**

Tabla 7.5: Comparativa de aplicar el método mapeando los word embeddings al espacio vectorial de FastText vs mapear los word embeddings al espacio vectorial de JointcHYB. Se ha usado el algoritmo de aproximación de palabras, y no se ha aplicado la normalización L_2 a los vectores antes de realizar la media. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

Como se puede observar en la Tabla 7.5, mapear los word embeddings al espacio vectorial de FastText o al espacio vectorial de jointcHYB produce resultados muy similares. Hemos analizado también el rendimiento dataset por dataset buscando si existe alguna diferencia en el rendimiento en función del tipo de dataset, pero tampoco existe ninguna diferencia significativa en este aspecto, por lo que no se muestra el rendimiento en cada dataset por brevedad. Por lo tanto, la elección del word embedding al que se va a realizar el mapeo no tiene una gran importancia, al menos usando buenos word embeddings como FastText y JointcHYB, que además cuentan con un vocabulario muy extenso.

7.3.4. Conclusiones

Durante esta sección hemos estudiado todas las variantes del método de la media de word embeddings en un mismo espacio vectorial. En esta sección resumiremos los aspecto más importantes a tener en cuenta para aplicar este método.

En cuanto al mapeado de los word embeddings hemos estudiado dos formar de realizarlo. La primera la rotación ortogonal, donde tan solo realizamos un mapeado mediante una rotación ortogonal de un word embedding a otro. Y la segunda denomina OLS que aplicaba diferentes métodos de preprocesado y postprocesado para tratar de conseguir que la rotación sea lo más precisa posible. Hemos podido comprobar que la rotación ortogonal ofrece resultados significativamente mejores que OLS, además nos permite mapear varios word embeddings al mismo espacio vectorial de forma sencilla. Por lo tanto recomendamos aplicar la rotación ortogonal como método de mapeado.

En cuanto a la media de word embeddings, hemos podido comprobar que no es necesario aplicar una normalización L_2 a los vectores antes de realizar la media, ya que dicha normalización ya se realiza antes del mapeado y sus efectos se mantienen tras el mapeado. También hemos podido comprobar que dado que todos los word embeddings se encuentran en un mismo espacio vectorial, durante la media si existe uno o varios word embeddings que cuentan con representación para una palabra y uno o varios word embeddings que no cuentan con representación para dicha palabra, podemos realizar la media con solo las representaciones de los embeddings que cuentan con representación. A diferencia de cuando realizábamos la media sin mapear los word embeddings al mismo espacio, la palabra no terminará en un espacio vectorial diferente al resto al hacer esto, si no que será una buena representación de la palabra en el meta embeddings generado. Sin embargo, podemos hacer aún más precisa la media en esta situación usando el algoritmo de generación de palabras. Sin embargo dicho algoritmo requiere de una gran capacidad de cómputo y memoria. Recomendamos el uso del algoritmo de generación de palabras siempre que sea posible.

Por último hemos comprobado que el espacio vectorial al que mapeemos los word embeddings no afecta de forma significativa al rendimiento de los meta embeddings generados. Sin embargo recomendamos que siempre se realice el mapeado a word embeddings que cuenten con un buen rendimiento, y que cuenten con un vocabulario extenso.

7.4. Resultados obtenidos

Una vez estudiado en profundidad el método de generación de meta embeddings y escogida la forma en la que vamos a aplicarlo, podemos pasar a comprobar su rendimiento aplicado a diferentes word embeddings. En esta sección buscamos determinar si el método es capaz de producir buenos meta embeddings en comparación a los meta embeddings generados en el capítulo anterior.

En la primera comparativa que vamos a realizar vamos a comparar el rendimiento el método con el rendimiento del mejor dataset usado en la media, para comprobar si los meta embeddings generados mejoran el rendimiento de los word embeddings originales. Y además, vamos a comparar el rendimiento con la media sin aplicar la alineación de espacios vectoriales. Tomar como comparación los resultados del capítulo anterior para la media de word embeddings en diferentes espacios vectoriales no sería una comparativa justa. El primer paso del método descrito en este capítulo es aplicar los métodos de normalización de centrado de variables, y normalización L_2 de los vectores. Esto puede provocar una mejora en el rendimiento de los word embeddings. Por lo tanto, vamos a comparar el método de la media de word embeddings tras aplicar la normalización, con el método de la media de word embeddings tras aplicar el mapeado al mismo espacio vectorial. De esta forma estaremos aplicando la media a exactamente los mismos embeddings, con la diferencia de que en un caso cada uno estará en un espacio vectorial diferente y en el segundo caso ambos estarán en el mismo espacio vectorial. Hemos realizado la media sin aplicar la normalización L_2 a los vectores y usando el algoritmo de generación de palabras. Todos los embedding han sido mapeados al espacio vectorial de FastText. La metodología de las pruebas ha sido la descrita en la Sección 3.1. En la Tabla 7.6 se muestra el rendimiento, media y medio ponderada, obtenidos por los diferentes meta embeddings. Además se muestran cuatro columnas más. En la primera se muestra la diferencia del rendimiento obtenido por la media de word embeddings alineados a un mismo espacio vectorial y la media de los mismos word embeddings sin aplicar el mapeado al mismo espacio vectorial. En las segundas columnas, se muestra la diferencia de el rendimiento obtenido por la media de word embeddings alineados a un mismo espacio vectorial respecto al rendimiento del mejor word embedding usado en la media.

En la tabla Tabla 7.6 podemos observar el rendimiento obtenido por el método de la media de word embeddings alineados a un mismo espacio vectorial. Lo primero que llama la atención es en todos los casos, realizar la media de word embeddings alineados resulta significativamente mejor que realizar la media de word embeddings no alineados. La com-

	Media sin mapeado a un mismo espacio vectorial		Rendimiento del mejor word embedding usado			
Embeddings	Media (Diferencia)	Media Ponderada (Diferencia)	Media (Diferencia)	Media Ponderada (Diferencia)	Media	Media Ponderada
GLOVE + W2V	0.041	0.048	0.016	0.006	0.645	0.577
GLOVE + FT	0.041	0.044	-0.007	-0.015	0.668	0.605
GLOVE + jointcHYB	0.046	0.050	0.033	0.017	0.687	0.623
GLOVE + UKB	0.054	0.064	0.075	0.035	0.705	0.645
GLOVE + LEXVEC	0.046	0.054	0.012	0.005	0.642	0.578
W2V + FT	0.016	0.010	-0.014	-0.019	0.661	0.601
W2V + jointcHYB	0.021	0.020	0.027	0.014	0.681	0.620
W2V + UKB	0.035	0.025	0.067	0.033	0.697	0.643
W2V + LEX-VEC	0.017	0.019	0.007	0.005	0.637	0.578
FT + jointcHYB	0.030	0.028	0.032	0.032	0.707	0.652
FT + UKB	0.021	0.031	0.053	0.059	0.728	0.679
FT + LEXVEC	0.006	0.003	-0.009	-0.013	0.666	0.607
jointcHYB + LEXVEC	0.033	0.023	0.040	0.025	0.694	0.631
UKB + LEX-VEC	0.031	0.023	0.078	0.040	0.708	0.650
GLOVE + FT + UKB	0.048	0.049	0.041	0.034	0.716	0.654
GLOVE + jointcHYB + UKB	0.050	0.056	0.053	0.039	0.707	0.649
FT + jointcHYB + UKB	0.039	0.044	0.042	0.047	0.717	0.667
FT + UKB + LEXVEC	0.027	0.021	0.041	0.037	0.716	0.657
GLOVE + FT + jointcHYB + UKB	0.041	0.047	0.043	0.039	0.718	0.659
W2V + FT + jointcHYB + UKB	0.042	0.036	0.059	0.048	0.713	0.658
FT + jointcHYB + UKB + LEXVEC	0.036	0.034	0.065	0.052	0.719	0.662
W2V + FT + jointcHYB + UKB + LEXVEC	0.040	0.030	0.033	0.029	0.708	0.649
GLOVE + W2V + FT + jointcHYB + UKB + LEXVEC	0.044	0.038	0.029	0.020	0.704	0.640

Tabla 7.6: Comparativa del rendimiento obtenido por la media de word embeddings alineados a un mismo espacio vectorial en comparación a la media de word embeddings sin alinear sus espacios vectoriles y el mejor word embedding usado en la media. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

parativa no deja lugar a dudas, alinear los espacios vectoriales de los word embeddings antes de realizar la media resulta mucho que mejor que no hacerlo.

Otro detalle muy importante es que la media de word embeddings alineados a un mismo espacio vectorial, no pierde rendimiento a medida que añadimos más word embeddings a la concatenación, como observamos en la Sección 6.3. De hecho obtenemos muy buenos resultados al combinar cuatro e incluso cinco word embeddings.

También podemos observar un comportamiento similar al que hemos observado en el resto de combinaciones de word embeddings hasta el momento. Cuando combinamos word embeddings generados a partir de corpus de texto con word embeddings generados a partir de corpus de texto, o word embeddings generados a partir de grafos, con word embedings generados a partir de grafos, en general no obtenemos una mejora de rendimiento significativa respecto al mejor word embedding de la combinación. De hecho hay casos como la media de FastText y Glove, o FastText y LexVec donde podemos llegar a perder rendimiento. Las mejoras más significas las obtenemos al combinar word embedings calculados a partir de grafos con word embedding calculados a partir de corpus de texto. También observamos que añadir más word embeddings a la media no tiene por que suponer que el rendimiento aumente. Lo más importante es combinar word embeddings lo más diferentes posibles, no tratar de combinar la mayor cantidad de word embeddings similares.

En cuanto al rendimiento obtenido, las puntaciones son muy altas, más que las optenidas por cualquier otro método estudiado hasta el momento. Las puntuaciones obtenidas por los meta embeddings generados se encuentran muy cerca del rendimiento que obtiene el meta embedding Numberbatch actualmente consigerado el estado del arte. Esto quiere decir que nuestro método es capaz de generar meta embedings con un rendimiento similar al estado del arte actual, incluso sin requerir del conocimiento de bases de datos como ConceptNet para mejorar el rendimiento de los word embeddings antes de la generación del meta embedding.

El vocabulario del meta embedding generado por nuestro método es igual a la unión de los vocabularios de los diferentes word embeddings empleados, es decir, el meta embedding generado cuenta con un vocabulario más extenso que los word embeddings empleados para su generación de forma individual. Por lo tanto, si realizamos la misma comparativa, usando la segunda métrica detallada en la Sección 3.1, la cual se basa en multiplicar el rendimiento obtenido en cada dataset por el porcentaje de palabras al que se ha podido dar respuesta en el dataset, la diferencia de rendimiento aumenta ligeramente respecto al

mejor word embedding usado en la media como se puede observar en la Tabla 7.7

Embeddings	Media (Diferencia)	Media Ponderada (Diferencia)	Media	Media Ponderada
GLOVE + UKB	0.076	0.076	0.705	0.645
W2V + UKB	0.090	0.072	0.694	0.640
FT + jointcHYB	0.033	0.034	0.707	0.652
FT + UKB	0.054	0.060	0.728	0.678
UKB + LEXVEC	0.079	0.080	0.708	0.650
GLOVE + FT + UKB	0.042	0.036	0.716	0.654
FT + jointcHYB + UKB	0.043	0.049	0.717	0.667

Tabla 7.7: Rendimiento de el método de la media de word embeddings alineados a un mismo espacio vectorial respecto al rendimiento del mejor word embedding usado en la media. Metrica usada: Resultado de cada dataset multiplicado por el porcentaje de pares de palabras respondidas

Por último vamos a realizar una última comparativa. Al comienzo de este capítulo hemos asegurado que alinear los word embedding a un mismo espacio vectorial, provoca que la media deje de depender del ángulo que forman los espacios vectoriales en los que se sitúan los diferentes word embeddings. Y esto provoca que la media se convierta en un equivalente a la concatenación. Para comprobarlo, hemos mapeado los word embeddings al espacio vectorial de FastText, y tras esto, hemos realizado tanto la media de los word embeddings como la concatenación. En ambos casos no hemos aplicado la normalización L_2 de los vectores antes de realizar la media o la concatenación y hemos utilizado el algoritmo de aproximación de palabras. En la Tabla 7.8 podemos ver los resultados obtenidos por ambos métodos.

Como se puede observar en la Tabla 7.8, la media de word embeddings cuyos espacios vectoriales han sido alineados no solo ofrece un rendimiento similar a la concatenación, si no que en ocasiones incluso ofrece un rendimiento ligeramente superior. Esto es muy importante, ya que hemos conseguido combinar la principal ventaja de la concatenación, que es que el conocimiento de todos los word embeddings tiene el mismo peso en la concatenación y no se pierde el conocimiento de ninguno de ellos. Con la principal ventaja de la media de word embeddings, que es poder combinar una gran cantidad de word embeddings sin que las dimensiones del meta embedding generado aumenten con cada word embedding añadido.

Para terminar esta sección creemos que resulta interesante mostrar el rendimiento de la media, la concatenación y el mapeado y posterior media de la combinación de word embeddings que mejor resultado ha obtenido en los diferentes datasets. Como en las tablas anteriores mostraremos el rendimiento de la media y la concatenación usando los word

	Media de word embeddings alineados		Cocatenación	
Embeddings	Media	Media Ponderada	Media	Media Ponderada
GLOVE + W2V	**0.645**	0.577	0.644	**0.579**
GLOVE + FT	**0.668**	**0.605**	**0.668**	**0.605**
GLOVE + jointcHYB	0.687	**0.623**	**0.688**	0.622
GLOVE + UKB	**0.705**	**0.645**	0.704	0.638
GLOVE + LEXVEC	**0.642**	**0.578**	0.641	0.577
W2V + FT	0.661	0.601	**0.667**	**0.606**
W2V + jointcHYB	0.681	0.620	**0.682**	**0.623**
W2V + UKB	0.697	**0.643**	**0.698**	0.639
W2V + LEXVEC	0.637	0.578	**0.640**	**0.582**
FT + jointcHYB	**0.707**	**0.652**	0.702	0.639
FT + UKB	**0.728**	**0.679**	0.727	0.673
FT + LEXVEC	0.666	0.607	**0.672**	**0.611**
jointcHYB + LEXVEC	**0.694**	**0.631**	0.688	0.630
UKB + LEXVEC	**0.708**	**0.650**	0.703	0.647
GLOVE + FT + UKB	**0.716**	**0.654**	0.714	0.650
GLOVE + jointcHYB + UKB	**0.707**	**0.649**	0.705	0.643
FT + jointcHYB + UKB	**0.717**	**0.667**	0.712	0.660
FT + UKB + LEXVEC	**0.716**	**0.657**	0.715	0.656
GLOVE + FT + jointcHYB + UKB	**0.718**	**0.659**	**0.718**	0.656
W2V + FT + jointcHYB + UKB	**0.713**	**0.658**	**0.713**	0.657
FT + jointcHYB + UKB + LEXVEC	**0.719**	**0.662**	0.718	0.661
W2V + FT + jointcHYB + UKB + LEXVEC	0.708	0.649	**0.714**	**0.653**
GLOVE + W2V + FT + jointcHYB + UKB + LEXVEC	0.704	0.640	**0.708**	**0.644**

Tabla 7.8: Comparativa del rendimiento de la media y la concatenación de word embedding alineados al mismo espacio vectorial. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

embeddings normalizados que se han usado para realizar el mapeado, de esta forma todos los métodos se encuentran en igualdad de condiciones al usar los mismos word embeddings. También mostraremos el rendimiento de los word embeddings normalizados individualmente. Como todos los meta embeddings responden a casi la totalidad de los pares de palabras mostrar las dos métricas de las que disponemos es redundante. Mostraremos tan solo la primera métrica, la cual no penaliza que un word embedding no se capaz de dar respuesta para un determinado número de pares de palabras. Podemos ver los resultados en la Tabla 7.9.

Embedding	Media	Media Ponderada	Sim Lex 999	MTurk 287	MTurk 771	MEN ALL	WS 353 all	WS 353 relatedness	WS 353 similarity	RG65	VERB 143	MC 30	YP 130	Sim Verb 3500	RW
FT (normalizado)	0.683	0.626	0.507	0.737	0.775	0.846	0.781	0.724	0.840	0.887	0.462	0.863	0.620	0.424	**0.606**
UKB (normalizado)	0.630	0.619	0.537	0.663	0.682	0.768	0.721	0.640	0.786	0.851	0.042	0.887	0.784	0.556	0.442
FT+UKB (Media)	0.707	0.648	0.546	0.703	0.766	0.830	0.781	0.724	0.835	0.895	**0.518**	**0.903**	0.747	0.521	0.564
FT+UKB (Concatenación)	0.727	0.673	0.570	**0.752**	0.791	**0.860**	0.816	**0.775**	**0.855**	**0.905**	0.501	0.900	0.782	0.548	0.576
FT+UKB (Map + Media)	**0.728**	**0.679**	**0.592**	0.751	**0.796**	**0.860**	**0.817**	**0.775**	**0.865**	**0.905**	0.468	0.889	**0.792**	**0.556**	0.584

Tabla 7.9: Comparativa entre diferentes métodos de generación de meta embeddings y los word embddings originales en los diferentes datasets. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

7.5. Conclusiones

En esta sección hemos presentado un numero método de generación de meta embeddings, que se basa en realizar la media de word embeddings que previamente han sido mapeados a un mismo espacio vectorial. Este método ha resultado ser superior a los métodos mostrados en el Capítulo 6, y nos aporta varias ventajas muy importantes.

La primera es que la media de word embeddings deja de ser dependiente del ángulo que forman los word embeddings entre ellos. De esta forma la media pasa a ser un método equivalente a la concatenación, por los que los meta embeddings generados serán muy similares a los generados por la concatenación de word embeddings. Este método por lo tanto, no provocará la pérdida de información que observábamos la sección Sección 6.3, al aplicar el método de la media de word embeddings. Y respecto a la concatenación este método cuenta con una ventaja muy importante, no aumenta la dimensionalidad del meta embedding generado a media que añadimos más word embeddings a la combinación. Por lo que los meta embeddings generados por este método cuentan con un número de dimensiones que facilita su aplicación en todo tipo de tareas.

La segunda ventaja, derribada de la primera es que elimina la posibilidad de que diferentes word embeddings se "contraresten" entre ellos. En la Sección 6.3 pudimos observar que la medida perdía rendimiento a media que añadíamos word embeddings. Si realizásemos la media de infinitos word embeddings en diferentes espacios vectoriales la longitud de todos los vectores generados tendería a cero. Dicho comportamiento hemos comprobado que no ocurre si todos los word embeddings se encuentran en un mismo espacio vectorial, por lo que este método nos permite realizar la media de tantos word embeddings queramos, siempre y cuando hayan sido mapeados al mismo espacio vectorial previamente.

Por último, nuestro método trata de forma adecuada la situación en la que un word embedding cuente con representación para una palabra para la que otro word embedding no cuenta. Imaginemos que disponemos del embedding A y el embedding B, y queremos calcular la media para los vectores que representan a w en A y B. En caso de que A cuente con representación para w, pero B no cuente con representación para w, podemos tomar directamente la representación de w en A. Esto es debido a que ambos word embeddings se encuentran en el mismo espacio vectorial, por lo que los que los vectores que representan a las mismas palabras en A y B cuentan con una similitud coseno muy alta entre ellas. Esto quiere decir que si B constase con representación para w, dicha representación debería ser similar a la de w en A, por lo tanto la media entre ambas representaciones

debe ser al mismo tiempo similar a la representación de w en A y w en B. Esto hace que la representación de w en A sea una buena aproximación de la media de los vectores para w en A y B. Aunque hemos comprobado que, el algoritmo de aproximación de palabras descrito en la Subsubsección 6.1.1, puede generar aproximaciones todavía más precisas, aumentando así el rendimiento.

Nuestro método además ha demostrado ser capaz de generar meta embeddings con un rendimiento al nivel del estado del arte actual.

8. CAPÍTULO

Aplicando lo aprendido para generar el mejor meta embedding posible

Durante los capítulos anteriores hemos realizado una evaluación del rendimiento de diferentes word embeddings, diferentes métodos de normalización y diferentes métodos de generación de word embeddings. Gracias a ello hemos llegado a una serie de conclusiones que se han expuesto en cada capítulo y se resumirán en el Capítulo 9. Teniendo toda esta información como conclusión del proyecto, vamos a aplicar lo aprendido para intentar generar los mejores word embeddings posibles. Hemos realizado lo siguiente:

1. Hemos escogido los word embeddings FastText y UKB, ya que en los capítulos anteriores han sido la combinación que mejor rendimiento alcanzaba.

2. Hemos usado retrofitting para adaptarlos a la base de conocimiento PPDB Sección 6.4

3. Hemos aplicado el método de generación de meta embeddings descrito en el Capítulo 7. Hemos aplicado a ambos una normalización L_2 de sus vectores y un centrado de variables. Los hemos mapeado al espacio de jointcHYB ENES, y hemos realizado la media de ambos word embeddings sin aplicar la normalización L_2 y aplicando el algoritmo de aproximación de palabras (Subsubsección 6.1.1).

4. Tras esto hemos aplicado la normalización L_1 a las columnas del word embedding resultante, llamaremos a este embedding "'FTUKB". En una segunda versión

hemos aplicado también el algoritmo de normalización PPA [51], ya que hemos observado que en algunos dataset resulta beneficioso, llamaremos a este embedding "FTUKB + PPA".

A continuación mostraremos el rendimiento alcanzado por el embedding y una comparativa con Numberbatch [69], considerado el estado del arte actual.

Como podemos observar en la Tabla 8.1, nuestro word embedding es capaz de superar a Numberbatch en diferentes datasets. Si hacemos la media de todos los dataset obtenemos un resultado mayor que el que obtiene Numberbatch.

En caso de que penalicemos que un word embedding no sea paza de dar respuesta para un determinado número de pares de palabras, nuestro word embedding gracias a contar con un vocabulario mayor amplia la diferencia con Numberbatch en el dataset RW.

Esto muestra que nuestro método es capaz de generar word embedding al mismo o nivel o ligeramente superior al estado del arte actual. Y prueba la utilidad de los diferentes estudios realizados en este proyecto. Creemos que si en Numberbatch se sustituyese la concatenación y posterior reducción de dimensionalidad por nuestro método, los resultados de Numberbatch mejorarían, sin embargo no disponemos de todos los recursos necesarios para comprobar esto. Creemos esto ya que nuestros embeddings generados, pese a incluir menos conocimiento que numberbatch obtienen un mejor resultado rendimiento en varios dataset.

Ambos word embeddings generados están disponibles para su descarga en GitHub bajo la licencia MIT [1].

[1] https://github.com/ikergarcia1996/RotEmbeddings

Embedding	Media	Media Pon-dera-da	Sim Lex 999	MTurk 287	MTurk 771	MEN ALL	WS 353 all	WS 353 rela-ted-ness	WS 353 simi-larity	RG65	VERB 143	MC 30	YP 130	Sim Verb 3500	RW
FastText CC	0.675	0.620	0.503	0.726	0.762	0.837	0.791	0.737	0.840	0.863	0.447	0.852	0.625	0.426	0.595
UKB	0.631	0.610	0.525	0.644	0.679	0.756	0.720	0.627	0.785	0.823	0.153	0.873	0.768	0.545	0.443
Numberbatch	0.743	**0.708**	**0.651**	0.720	**0.817**	0.860	0.813	0.758	0.838	**0.910**	0.521	0.887	0.763	**0.602**	0.626
FTUKB	**0.749**	0.700	0.643	**0.773**	0.810	**0.862**	0.818	0.769	**0.875**	0.885	0.568	0.874	**0.809**	0.576	0.625
FTUKB + PPA	0.752	0.704	0.644	0.728	0.795	0.856	**0.828**	**0.776**	0.857	0.898	**0.617**	**0.89**	0.790	0.593	**0.635**

Tabla 8.1: Comparativa entre el meta embedding generado, los word embeddings usados para generarlo y Numberbatch. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

Embedding	Media	Media Pon-dera-da	Sim Lex 999	MTurk 287	MTurk 771	MEN ALL	WS 353 all	WS 353 rela-ted-ness	WS 353 simi-larity	RG65	VERB 143	MC 30	YP 130	Sim Verb 3500	RW
FastText CC	0.674	0.618	0.503	0.726	0.762	0.837	0.791	0.737	0.840	0.863	0.447	0.852	0.625	0.426	0.584
UKB	0.596	0.568	0.525	0.516	0.678	0.756	0.705	0.612	0.769	0.823	0.064	0.873	0.768	0.545	0.305
Numberbatch	0.737	0.695	**0.651**	0.720	**0.817**	0.860	0.813	0.758	0.838	**0.910**	0.521	0.887	0.763	**0.602**	0.562
FTUKB	0.749	0.700	0.643	**0.773**	0.810	**0.862**	0.818	0.769	**0.875**	0.885	0.568	0.874	**0.809**	0.576	0.624
FTUKB + PPA	**0.752**	**0.704**	0.644	0.728	0.795	0.856	**0.828**	**0.776**	0.857	0.898	**0.617**	**0.890**	0.790	0.593	**0.634**

Tabla 8.2: Comparativa entre el meta embedding generado, los word embeddings usados para generarlo y Numberbatch. Métrica utilizada: Resultado obtenido en cada datataset multiplicado por la cobertura.

9. CAPÍTULO

Conclusiones

Como capítulo final presentaremos las conclusiones generales del proyecto y las líneas de investigación futuras.

9.1. Conclusiones y objetivos alcanzados

El objetivo general de este proyecto era el de realizar una evaluación exhaustiva de diferentes word embeddings y métodos de generación de meta-embeddings. Más concretamente buscábamos responder a las preguntas ¿Cuales son los mejores word embedding que puedo utilizar? y Tengo varios word embeddings, ¿cuál es la mejor forma de combinarlos? Gracias al estudio exhaustivo realizado durante este trabajo de diferentes word embeddings y métodos de combinación de word embeddings hemos sido capaces de dar respuesta a ambas preguntas. En concreto:

- Hemos realizado un estudio del estado del arte actual donde hemos recopilado las publicaciones más relevantes en el ámbito de la generación de word embeddings y meta-embeddings.
- Hemos realizado una evaluación empírica del rendimiento de un gran número de word embeddings. Esta evaluación nos ha permitido determinar qué word embeddings y métodos de generación de word embeddings obtienen un mejor rendimiento en la tarea de similitud entre palabras. Una de las conclusiones más interesantes de

este esta evaluación es que diferentes métodos de generación de word embeddings obtienen diferente rendimiento en tareas específicas. Más concretamente, los word embeddings entrenados en corpus de texto muestran un mejor rendimiento en tareas relacionadas con la relación entre palabras, mientras que word embeddings entrenados a partir de la información obtenida mediante Random Walks sobre WortNet obtienen un mejor rendimiento en el calculo de la similitud.

- Hemos realizado un estudio de diferentes métodos de normalización de word embeddings. En este estudio hemos mostrado que para usar embeddings generados mediante GloVe, es necesario aplicar una normalización L_2 a sus columnas para obtener el mejor rendimiento posible en la tarea de la similitud entre palabras. Sin embargo aplicar métodos de normalización a word embeddings generados mediante otros métodos, no aporta una mejora de rendimiento significativa. También hemos comprobado que la información más relevante para el cálculo de la similitud entre palabras es el ángulo que forman, siendo de esta forma la similitud coseno la mejor métrica para medirla. Métricas como el producto escalar que además del ángulo que forman los vectores tienen en cuenta sus magnitudes resultan significativamente peores. Por lo tanto, para obtener los mejores resultados posibles en la tarea de la similitud entre palabras debemos usar la similitud coseno, y en caso de usar otras métricas como el producto escalar, es necesario normalizar los vectores usando la norma L_2.

- Hemos propuestos dos métodos para combinar el conocimiento de diferentes word embeddings sin generar un nuevo word embeddings. La aplicación en cascada y la media de las similitudes. La media de las similitudes se ha mostrado como un método atractivo si nuestra tarea a la que vamos a aplicar los word embeddings no requiere de la generación de un meta-embedding. Nos permite mejorar el rendimiento de los word embedding combinados y ampliar el vocabulario disponible.

- Hemos realizado un estudio de los métodos de combinación de word embeddings más populares. Hemos encontrado que la concatenación resulta ser el mejor método para generar un word embedding, ya que es capaz de combinar diferentes word embeddings sin que se pierda o degrade el conocimiento que contienen. Sin embargo, produce meta-embeddings con un número de dimensiones igual a la suma de las dimensiones de los word embeddings usados. Esto provoca que el uso de los meta-embeddings generados en diferentes tareas pueda ser complicado o imposible. Por ello hemos estudiado dos métodos que buscan resolver esto. El primero

es la reducción de dimensionalidad. El mejor método para reducir la dimensionalidad ha resultado ser DRA. Sin embargo todos los métodos probados han provocado una pérdida significativa en el rendimiento en los datasets que miden la similaridad entre palabras respecto a los meta-embeddings sin aplicar la reducción de dimensionalidad. El segundo ha sido la media de word embeddings. Este método al igual que la reducción de dimensionalidad obtiene un rendimiento inferior a la concatenación de word embeddings, aunque con la ventaja de que las dimensiones del word embedding generado no aumentan a medida que combinamos un número mayor de word embeddings. La pérdida de rendimiento respecto a la concatenación es más homogénea que al aplicar la reducción de dimensionalidad. Sin embargo a medida que realizamos la media de un número mayor de word embeddings el rendimiento obtenido por los meta-embeddings generados se reduce.

Tanto al aplicar la concatenación de word embeddings como al realizar la media, hemos comprobado que es necesario normalizar los vectores mediante la norma L_2 antes de aplicar cualquiera de los dos métodos. Esto evita que los word embeddings que se encuentran en una escala superior tiendan a dominar al resto y provoquen que se pierda el conocimiento del resto de word embeddings. También hemos propuesto un algoritmo de aproximación de palabras que nos permite conseguir que el vocabulario del meta-embedding generado sea igual a la unión de los vocabularios usados en la generación del meta-embeddings. Este método se aplica cuando un word embedding cuenta con representación para una palabra pero uno o varios word embeddings usados en la combinación no cuentan con representación para dicha palabra. Nuestro algoritmo ha mostrado un rendimiento significativamente superior a la asignación de un vector predefinido.

Dentro de los métodos de generación de meta embeddings también hemos estudiado el retrofitting, método que permite ajustar el conocimiento de un word embedding a la información contenida en una base de conocimiento. Aplicar este método usando la base de datos PPDB produce muy buenos resultados, y nos permite mejorar el rendimiento de un word embeddings. En cambio, si usamos WordNet, aunque mejoramos el rendimiento en datasets que se centran en la similaridad, perdemos rendimiento en datasets que estudian la relación entre palabras.

- Hemos propuesto un nuevo método para la generación de meta-embeddings que utiliza proyecciones en espacios vectoriales. Dicho método realiza una rotación ortogonal de los word embeddings a un mismo espacio vectorial. Una vez todos los word embeddings se encuentran en un mismo espacio vectorial se realiza la media.

La alineación de espacios vectoriales provoca que la media de word embeddings se convierta en un equivalente a la concatenación, pero con la importante ventaja de que los meta-embeddings generados no aumentan su número de dimensiones a medida que añadimos word embeddings. Además el propio método soluciona la situación en la que un word embedding cuente con representación para una palabra para la que otro word embedding no cuenta (en inglés OOV de Out of Vocabulary). Ya que todos los word embeddings se encuentran en un mismo espacio vectorial, podemos realizar la media de las representaciones disponibles sin necesidad de asignar vectores predefinidos o generar aproximaciones para los word embeddings que no cuentan con representación para dicha palabra (aunque podemos usar el algoritmo de aproximación de palabras para obtener un mejor rendimiento).

- Una de las conclusiones más importantes a la que hemos llegado en esta investigación ha sido que para generar los mejores word embeddings, lo mejor es combinar word embeddings lo más diferentes posibles. Si dos word embeddings han sido entrenados usando el mismo corpus o corpus muy similares, el conocimiento que codificarán será muy similar. Por lo tanto, combinarlos no generará seguramente un meta-embedding con un mejor rendimiento que los word embedding originales. Sin embargo, si combinamos word embeddings entrenados usando corpus muy diferentes, como word embeddings entrenados usando el corpus Common crawl y word embeddings entrenados usando la información obtenida de Ramdom Walks sobre WordNet, conseguiremos un meta-embedding que codifica el conocimiento de las dos fuentes de conocimiento diferentes. Al contener más conocimiento conseguiremos que el meta-embedding resultante obtenga un mejor rendimiento que los word embedding originales. Los mejores resultados no se obtienen tratando de combinar el mayor número de word embeddings posible o tratando de usar los mejores word embeddings, si no combinando la mayor cantidad de conocimiento posible, usando para ello word embeddings de naturaleza distinta.

- Aplicando lo aprendido durante el proyecto y el nuevo método de generación de meta-embedding propuesto, se ha generado un meta-embedding que se encuentra a la altura del estado del arte actual y es capaz de superar el rendimiento de Numberbatch en una gran cantidad de datasets. Dicho meta-embedding se encuentra disponible para que cualquiera pueda descargarlo y utilizando bajo la licencia MIT [1].

[1] https://github.com/ikergarcia1996/RotEmbeddings

- Se ha desarrollado un software que permite la evaluación de word embeddings en la tara de la similaridad entre palabras, y la generación de meta-embeddings de forma sencilla. Dicho software se ha usado para la realización de este trabajo. Ha sido escrito en el lenguaje python usando el entorno de desarrollo Jupyter Notebook. El código se encuentra en la plataforma github [2], y cualquiera puede utilizarlo bajo la licencia MIT.

Gracias a estos resultados, creemos que hemos clarificado un poco la metodología a seguir para generar mejores meta-embeddings, y de esta forma contribuir a mejorar el rendimiento de todo tipo de tareas dentro del ámbito del procesamiento del lenguaje natural.

9.2. Trabajo futuro

El estudio realizado abre nuevas vías de investigación que podrían resultar interesantes:

- La primera es el estudio del rendimiento que obtienen los distintos word embeddings y meta-embeddings propuestos en otros tipos de tareas. En este trabajo, solo hemos analizado su rendimiento en profundidad en la tarea del cálculo de la similitud entre palabras en inglés.

- Hemos comprobado que los mejores resultados se obtienen al combinar el conocimiento de word embeddings entrenados usando corpus de textos, y word embeddings calculados usando la información obtenida mediante ramdom walks sobre WordNet. Es decir, los word embedding que codifican un conocimiento más diferente entre ellos. De esta forma conseguimos que el meta embedding generado codifique la mayor cantidad de conocimiento posible. Existe un tipo de word embedding que codifica un tipo de conocimiento completamente diferente a los utilizados durante este trabajo: los word embeddings generados a partir de imágenes. Estos Word Embeddings normalmente son entrenados usando usa red neuronal convolucional entrenada con la información contenida en ImageNet [32]. Este tipo de word embeddings pueden añadir todavía más conocimiento a los meta embeddings generados y creemos que esto puede mejorar el rendimiento de los mismos. Sin embargo, los word embeddings calculados a partir de imágenes disponibles actualmente cuentan con un rendimiento muy pobre, por lo que no resultan aptos para

[2] https://github.com/ikergarcia1996/RotEmbeddings

combinarlos con word embeddings como FastText, ya que estamos añadiendo información de muy baja calidad en comparación al conocimiento que codifica FastText. Hemos realizado algunas pruebas que no hemos documentado en esta memoria, sin conseguir hasta el momento buenos resultados. Sin embargo, creemos que si es posible generar buenos word embeddings a partir de imágenes, pueden resultar muy beneficiosos en la generación de meta-embeddings.

- Otra forma de conseguir combinar más conocimiento para obtener los mejores meta-embeddings puede ser usando word embeddings en diferentes idiomas. Aunque hayan sido entrenados en un corpus similar y usando el mismo método, cada idioma expresa las ideas de forma diferente, por lo que el conocimiento contenido en word embeddings en diferentes idiomas será diferente. Combinarlo puede ayudarnos a generar mejores meta-embeddings. Un ejemplo de como podríamos hacer esto es tomar un embedding entrenado en corpus en inglés, otro en corpus en castellano y otro en corpus en euskera. Usando VecMap podemos alinearlos a un mismo espacio vectorial. Una vez hecho esto, para cada palabra del embedding en inglés realizamos la media con la palabra más cercana en el embedding en castellano y el embedding en euskera. Como todos los word embedding están alineados en un mismo espacio vectorial esperamos que la palabra más cercana en castellano y euskera sea la que expresa el mismo concepto en esos idiomas. De esta forma, con una pequeña modificación de nuestro algoritmo de generación de meta-embeddings podríamos combinar el conocimiento de diferentes idiomas. Esto es algo que tenemos la intención de estudiar en el futuro.

10. CAPÍTULO

Anexo

Embedding	Media	Media ponderada	SimLex 999	MTurk-287	MTurk-771	MEN DEV	MEN TEST	MEN ALL	WS353 all	WS353 relatedness	WS353 similarity	WS353 set1	WS353 set2	RG65	VERB 143	MC-30	YP-130	SimVerb 3500	RW
Numberbatch	**0.743**	**0.708**	**0.651**	0.720	**0.817**	**0.857**	**0.865**	**0.860**	**0.813**	**0.758**	0.838	**0.802**	0.761	**0.910**	**0.521**	**0.887**	0.763	**0.602**	**0.626**
FastText CC	0.675	0.620	0.503	0.726	0.762	0.835	0.839	0.837	0.791	0.737	**0.840**	0.743	0.767	0.863	0.447	0.852	0.625	0.426	0.595
jointcHYB EN ES	0.654	0.606	0.541	0.625	0.692	0.777	0.783	0.780	0.756	0.668	0.825	0.773	**0.668**	0.863	0.304	**0.877**	**0.816**	0.558	0.378
jointcHYB EN EU	0.644	0.607	0.541	0.639	0.687	0.763	0.779	0.769	0.729	0.648	0.787	0.754	0.641	0.836	0.288	0.839	0.802	0.559	0.394
FastText Wiki	0.630	0.562	0.450	0.705	0.710	0.791	0.789	0.791	0.733	0.682	0.812	0.729	0.663	0.846	0.463	0.836	0.519	0.357	0.523
UKB	0.630	0.610	0.525	0.644	0.679	0.755	0.756	0.756	0.720	0.627	0.785	0.773	0.601	0.823	0.153	0.873	0.768	0.545	0.443
Subword LexVec CC	0.630	0.573	0.477	0.717	0.738	0.807	0.805	0.807	0.752	0.696	0.809	0.723	0.719	0.744	0.441	0.784	0.578	0.352	0.539
GLOVE CC 840B	0.629	0.571	0.453	0.721	0.736	0.829	0.829	0.829	0.784	0.738	0.819	0.773	0.727	0.757	0.392	0.788	0.597	0.338	0.530
GLOVE CC 42B	0.628	0.553	0.453	0.700	0.740	0.813	0.816	0.814	0.759	0.733	0.794	0.773	0.689	0.829	0.374	0.836	0.596	0.319	0.487
jointcHYB EN IT	0.617	0.565	0.467	0.631	0.671	0.761	0.766	0.763	0.758	0.695	0.806	0.766	0.673	0.821	0.263	0.842	0.774	0.515	0.288
Word2Vec	0.616	0.555	0.442	0.684	0.671	0.771	0.770	0.771	0.700	0.635	0.772	0.665	0.658	0.761	0.497	0.788	0.559	0.364	0.534
SketchEngine web	0.610	0.545	0.412	**0.743**	0.714	0.789	0.790	0.789	0.743	0.686	0.770	0.729	0.709	0.714	0.470	0.712	0.582	0.315	0.515
LexVec CC WordsVectors	0.606	0.543	0.444	0.716	0.720	0.793	0.799	0.795	0.719	0.668	0.762	0.750	0.646	0.752	0.376	0.785	0.550	0.304	0.508
LexVec CC WordsVectors + ContexVectors	0.601	0.534	0.419	0.712	0.724	0.807	0.813	0.809	0.741	0.702	0.775	0.764	0.663	0.765	0.315	0.800	0.553	0.277	0.496
PDC. Dims: 300	0.599	0.529	0.427	0.670	0.688	0.774	0.768	0.773	0.765	0.700	0.796	0.766	0.671	0.790	0.343	0.828	0.511	0.296	0.500
HDC. Dims: 300	0.593	0.515	0.407	0.656	0.670	0.763	0.754	0.760	0.745	0.674	0.801	0.760	0.651	0.806	0.416	0.808	0.491	0.278	0.489
Contex to vec ukwac	0.588	0.531	0.408	0.678	0.678	0.753	0.761	0.756	0.702	0.645	0.724	0.667	0.661	0.737	0.475	0.689	0.524	0.320	0.500
GLOVE WIKI. Dims 300	0.577	0.511	0.408	0.651	0.681	0.771	0.774	0.772	0.658	0.621	0.711	0.698	0.585	0.779	0.361	0.727	0.580	0.277	0.459
PDC. Dims 100	0.573	0.494	0.361	0.709	0.663	0.752	0.760	0.755	0.746	0.684	0.785	0.762	0.658	0.774	0.332	0.809	0.434	0.240	0.475
SketchEngine British National	0.563	0.492	0.329	0.688	0.661	0.747	0.739	0.744	0.680	0.649	0.735	0.648	0.644	0.739	0.412	0.724	0.481	0.255	0.475
Multilingual512	0.562	0.503	0.405	0.537	0.647	0.758	0.760	0.759	0.683	0.646	0.743	0.730	0.609	0.756	0.261	0.817	0.598	0.292	0.423
LexVec Wiki+NewsCrawls WordsVectors + ContexVectors	0.559	0.499	0.362	0.649	0.666	0.765	0.766	0.765	0.683	0.663	0.734	0.706	0.608	0.793	0.235	0.742	0.520	0.255	0.476
HDC. Dims 100	0.559	0.465	0.324	0.651	0.633	0.737	0.740	0.738	0.708	0.647	0.780	0.755	0.609	0.804	0.369	0.822	0.452	0.199	0.443
LexVec Wiki+NewsCrawls WordsVectors	0.554	0.506	0.384	0.655	0.663	0.753	0.749	0.751	0.661	0.619	0.727	0.680	0.606	0.747	0.279	0.699	0.491	0.279	0.489
GLOVE Wiki. Dims 200	0.551	0.483	0.376	0.658	0.656	0.748	0.755	0.750	0.637	0.597	0.688	0.687	0.560	0.744	0.335	0.692	0.545	0.239	0.432
Contex to vec mscc	0.540	0.462	0.409	0.552	0.589	0.663	0.667	0.664	0.585	0.435	0.732	0.643	0.450	0.840	0.331	0.879	0.452	0.294	0.345
PDC. Dims 50	0.538	0.456	0.309	0.701	0.614	0.720	0.720	0.720	0.703	0.641	0.750	0.745	0.595	0.763	0.315	0.815	0.338	0.196	0.447
HDC. Dims 50	0.518	0.433	0.281	0.654	0.601	0.709	0.708	0.708	0.661	0.588	0.736	0.720	0.558	0.723	0.346	0.768	0.372	0.173	0.410
GLOVE Wiki. Dims 100	0.517	0.452	0.329	0.649	0.613	0.716	0.721	0.718	0.594	0.573	0.660	0.661	0.521	0.694	0.331	0.653	0.485	0.212	0.407
GLOVE twitter. Dims 200	0.511	0.449	0.275	0.653	0.635	0.738	0.753	0.743	0.645	0.592	0.730	0.661	0.600	0.714	0.283	0.758	0.323	0.188	0.404
GLOVE Wiki. Dims 50	0.473	0.422	0.292	0.657	0.581	0.678	0.696	0.684	0.567	0.530	0.639	0.636	0.489	0.603	0.250	0.583	0.403	0.183	0.395
GLOVE twitter. Dims 100	0.458	0.388	0.199	0.638	0.603	0.661	0.680	0.667	0.600	0.543	0.696	0.607	0.559	0.687	0.233	0.651	0.254	0.120	0.381
SW2V words and sense wikipedia	0.444	0.370	0.246	0.488	0.502	0.637	0.656	0.643	0.528	0.386	0.679	0.560	0.448	0.777	0.261	0.775	0.199	0.145	0.320
Senna	0.414	0.370	0.270	0.587	0.496	0.572	0.564	0.569	0.498	0.401	0.607	0.475	0.466	0.493	0.356	0.574	0.160	0.166	0.385
GLOVE twitter. Dims 50	0.389	0.332	0.156	0.576	0.523	0.587	0.607	0.594	0.519	0.481	0.607	0.543	0.495	0.608	0.199	0.496	0.197	0.079	0.338
GLOVE twitter. Dims 25	0.326	0.267	0.122	0.542	0.425	0.484	0.493	0.487	0.404	0.344	0.522	0.430	0.377	0.538	0.168	0.458	0.087	0.040	0.311
Turian. Dims: 100	0.307	0.248	0.221	0.421	0.317	0.272	0.382	0.309	0.361	0.317	0.425	0.355	0.370	0.352	0.305	0.414	0.302	0.178	0.191

Tabla 10.1: Rendimiento de todos los word embeddings evaluados en cada dataset. Métrica utilizada: Descartar los pares de palabras para los que no se puede dar respuesta.

Embedding	Media	Media ponde-rada	SimLex 999	MTurk-287	MTurk-771	MEN DEV	MEN TEST	MEN ALL	WS353 all	WS353 rela-tedness	WS353 simila-rity	WS353 set1	WS353 set2	RG65	VERB 143	MC-30	YP-130	SimVerb 3500	RW
Numberbatch	0.737	0.695	0.651	0.720	0.817	0.857	0.865	0.860	0.813	0.758	0.838	0.802	0.761	0.910	0.521	0.887	0.763	0.602	0.562
FastText CC	0.674	0.618	0.503	0.726	0.762	0.835	0.839	0.837	0.791	0.737	0.840	0.743	0.767	0.863	0.447	0.852	0.625	0.426	0.584
jointcHYB EN ES	0.651	0.597	0.541	0.625	0.692	0.777	0.783	0.780	0.756	0.668	0.825	0.773	0.668	0.863	0.304	0.877	0.816	0.558	0.347
jointcHYB EN EU	0.638	0.590	0.541	0.639	0.687	0.763	0.779	0.769	0.729	0.648	0.787	0.754	0.641	0.836	0.288	0.839	0.802	0.559	0.333
GLOVE CC 840B	0.629	0.569	0.453	0.721	0.736	0.829	0.829	0.829	0.784	0.738	0.819	0.773	0.727	0.757	0.392	0.788	0.597	0.338	0.521
Subword LexVec CC	0.629	0.570	0.477	0.717	0.738	0.807	0.805	0.807	0.752	0.696	0.809	0.723	0.719	0.744	0.441	0.784	0.578	0.352	0.528
FastText Wiki	0.628	0.559	0.450	0.700	0.710	0.791	0.789	0.791	0.733	0.682	0.812	0.729	0.663	0.846	0.463	0.836	0.519	0.357	0.506
GLOVE CC 42B	0.627	0.552	0.453	0.700	0.740	0.813	0.816	0.814	0.759	0.733	0.794	0.773	0.689	0.829	0.374	0.836	0.596	0.319	0.482
jointcHYB EN IT	0.614	0.553	0.467	0.629	0.671	0.761	0.766	0.763	0.758	0.695	0.806	0.766	0.673	0.821	0.263	0.842	0.774	0.515	0.254
SketchEngine web	0.610	0.545	0.412	0.743	0.714	0.789	0.790	0.789	0.743	0.686	0.770	0.729	0.709	0.714	0.470	0.712	0.582	0.315	0.515
LexVec CC WordsVectors	0.606	0.541	0.444	0.716	0.720	0.793	0.799	0.795	0.719	0.668	0.762	0.750	0.646	0.752	0.376	0.785	0.550	0.304	0.501
Word2Vec	0.604	0.541	0.442	0.655	0.671	0.757	0.758	0.757	0.700	0.635	0.772	0.665	0.658	0.761	0.466	0.788	0.559	0.364	0.479
LexVec CC WordsVectors + ContexVectors	0.601	0.533	0.419	0.712	0.724	0.807	0.813	0.809	0.741	0.702	0.775	0.764	0.663	0.765	0.315	0.800	0.553	0.277	0.489
UKB	0.596	0.568	0.525	0.516	0.678	0.755	0.756	0.756	0.705	0.612	0.769	0.763	0.586	0.823	0.064	0.873	0.768	0.545	0.305
PDC. Dims 300	0.595	0.521	0.427	0.668	0.688	0.774	0.768	0.773	0.765	0.700	0.796	0.766	0.671	0.790	0.343	0.828	0.511	0.295	0.457
HDC. Dims 300	0.589	0.507	0.407	0.653	0.670	0.763	0.754	0.760	0.745	0.674	0.801	0.760	0.651	0.806	0.416	0.808	0.491	0.278	0.446
Contex to vec ukwac	0.575	0.504	0.407	0.676	0.678	0.753	0.761	0.756	0.702	0.645	0.724	0.667	0.661	0.737	0.475	0.689	0.524	0.320	0.363
GLOVE WIKI	0.572	0.500	0.408	0.651	0.681	0.771	0.774	0.772	0.658	0.621	0.711	0.698	0.585	0.779	0.361	0.727	0.580	0.277	0.402
PDC. Dims 100	0.569	0.487	0.361	0.706	0.663	0.752	0.760	0.755	0.746	0.684	0.785	0.762	0.658	0.774	0.332	0.809	0.434	0.240	0.434
HDC. Dims 100	0.555	0.457	0.324	0.649	0.633	0.737	0.740	0.738	0.708	0.647	0.780	0.755	0.609	0.804	0.369	0.822	0.452	0.199	0.404
LexVec Wiki+NewsCrawls WordsVectors + ContexVectors	0.553	0.486	0.361	0.647	0.666	0.765	0.766	0.765	0.683	0.663	0.734	0.706	0.608	0.793	0.235	0.742	0.520	0.255	0.411
SketchEngine British National	0.551	0.469	0.328	0.685	0.659	0.747	0.736	0.743	0.680	0.649	0.735	0.648	0.644	0.739	0.412	0.724	0.477	0.255	0.357
Multilingual512	0.550	0.476	0.404	0.536	0.647	0.758	0.760	0.758	0.683	0.646	0.743	0.730	0.609	0.756	0.261	0.817	0.598	0.292	0.298
LexVec Wiki+NewsCrawls WordsVectors	0.548	0.493	0.384	0.653	0.663	0.753	0.749	0.751	0.661	0.619	0.727	0.680	0.606	0.747	0.279	0.699	0.491	0.279	0.422
GLOVE Wiki. Dims 200	0.546	0.472	0.376	0.658	0.656	0.748	0.755	0.750	0.637	0.597	0.688	0.687	0.560	0.744	0.335	0.692	0.545	0.238	0.379
PDC. Dims 50	0.535	0.448	0.309	0.699	0.614	0.720	0.720	0.720	0.703	0.641	0.750	0.745	0.595	0.763	0.315	0.815	0.338	0.196	0.408
HDC. Dims 50	0.515	0.426	0.281	0.651	0.601	0.709	0.708	0.708	0.661	0.588	0.736	0.720	0.558	0.723	0.346	0.768	0.372	0.173	0.374
GLOVE Wiki. Dims 100	0.512	0.441	0.329	0.649	0.613	0.716	0.721	0.718	0.594	0.573	0.660	0.661	0.521	0.694	0.331	0.653	0.485	0.212	0.357
Contex to vec mscc	0.505	0.408	0.403	0.492	0.563	0.633	0.629	0.631	0.522	0.390	0.663	0.572	0.403	0.840	0.331	0.879	0.424	0.286	0.182
GLOVE twitter. Dims 200	0.493	0.411	0.275	0.651	0.635	0.738	0.752	0.743	0.643	0.589	0.730	0.661	0.597	0.714	0.283	0.758	0.323	0.188	0.211
GLOVE Wiki. Dims: 50	0.468	0.413	0.292	0.657	0.581	0.678	0.696	0.684	0.567	0.530	0.639	0.636	0.489	0.603	0.250	0.583	0.403	0.183	0.346
GLOVE twitter. Dims 100	0.441	0.355	0.199	0.636	0.603	0.661	0.680	0.667	0.598	0.541	0.696	0.607	0.556	0.687	0.233	0.651	0.254	0.119	0.199
SW2V words and sense wikipedia	0.436	0.366	0.246	0.432	0.502	0.637	0.656	0.643	0.501	0.364	0.656	0.531	0.426	0.777	0.261	0.775	0.199	0.145	0.311
Senna	0.398	0.339	0.270	0.583	0.496	0.572	0.563	0.569	0.498	0.401	0.607	0.475	0.466	0.493	0.356	0.574	0.159	0.162	0.216
GLOVE twitter. Dims 50	0.374	0.303	0.156	0.574	0.523	0.587	0.606	0.593	0.517	0.479	0.607	0.543	0.492	0.608	0.199	0.496	0.197	0.079	0.177
GLOVE twitter. Dims 25	0.312	0.244	0.121	0.540	0.425	0.484	0.493	0.487	0.403	0.343	0.522	0.430	0.375	0.538	0.168	0.458	0.087	0.040	0.163
Turian. Dims 100	0.287	0.223	0.221	0.329	0.314	0.260	0.369	0.296	0.358	0.316	0.421	0.350	0.368	0.347	0.305	0.414	0.288	0.175	0.107

Tabla 10.2: Rendimiento de todos los word embeddings evaluados en cada dataset. Métrica utilizada: Multiplicación del resultado obtenido por la cobertura

Embedding	SimLex 999	MTurk-287	MTurk-771	MEN DEV	MEN TEST	MEN ALL	WS353 all	WS353 related-ness	WS353 simila-rity	WS353 set1	WS353 set2	RG65	VERB 143	MC-30	YP-130	SimVerb 3500	RW
Word2Vec	1	0.95819	1	0.98100	0.98400	0.98200	1	1	1	1	1	1	0.93750	1	1	1	0.89725
GLOVE WIKI 6B	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.87611
GLOVE CC 42B	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.98968
GLOVE CC 840B	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.98230
FastText Wiki	1	0.99303	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.96657
FastText CC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.98083
UKB	1	0.80139	0.99870	1	1	1	0.98017	0.97619	0.98030	0.98693	0.97500	1	0.41667	1	1	1	0.68879
SketchEngine British National	0.99700	0.99652	0.99611	1	0.99600	0.99867	1	1	1	1	1	1	1	1	0.99231	0.99800	0.75025
SketchEngine web	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99951
Numberbatch	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.89823
Turiam. Dims	0.99900	0.78049	0.99222	0.95600	0.96600	0.95933	0.99150	0.99603	0.99015	0.98693	0.99500	0.98462	1	1	0.95385	0.98429	0.56146
Senna	0.99900	0.99303	1	1	0.99900	0.99967	1	1	1	1	1	1	1	1	0.99231	0.98029	0.55949
Multilingual512	0.99900	0.99652	1	1	0.99900	0.99967	1	1	1	1	1	1	1	1	1	0.99771	0.70600
SW2V words and sense wikipedia	1	0.88502	1	1	1	1	0.94901	0.94444	0.96552	0.94771	0.95000	1	1	1	1	0.99943	0.97247
Contex to vec ukwac	0.99900	0.99652	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99857	0.72517
Contex to vec mscc	0.98599	0.89199	0.95590	0.95500	0.94300	0.95100	0.89235	0.89683	0.90640	0.88889	0.89500	1	1	1	0.93846	0.97257	0.52901
jointcHYB EN ES	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.91888
jointcHYB EN EU	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.84464
jointcHYB EN IT	1	0.99652	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.88151
Subword LexVec CC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.97837
LexVec CC WordsVectors	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.98673
LexVec CC WordsVectors + ContexVectors	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.98673
LexVec Wiki+NewsCrawls WordsVectors	0.99900	0.99652	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.86382
LexVec Wiki+ NewsCrawls WordsVectors + ContexVectors	0.99900	0.99652	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.86382
PDC. Dims 50	1	0.99652	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.91298
PDC. Dims 100	1	0.99652	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.91298
PDC. Dims 300	1	0.99652	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.91298
HDC. Dims 50	1	0.99652	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.91298
HDC. Dims 100	1	0.99652	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.91298
HDC. Dims 300	1	0.99652	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.91298
GLOVE Wiki 50	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.87611
GLOVE Wiki. Dims 100	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.87611
GLOVE Wiki. Dims 200	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.99943	0.87611
GLOVE twitter. Dims 25	0.99900	0.99652	1	1	0.99900	0.99967	0.99717	0.99603	1	1	0.99500	1	1	1	1	0.99886	0.52262
GLOVE twitter. Dims 50	0.99900	0.99652	1	1	0.99900	0.99967	0.99717	0.99603	1	1	0.99500	1	1	1	1	0.99886	0.52262
GLOVE twitter. Dims 100	0.99900	0.99652	1	1	0.99900	0.99967	0.99717	0.99603	1	1	0.99500	1	1	1	1	0.99886	0.52262
GLOVE twitter. Dims 200	0.99900	0.99652	1	1	0.99900	0.99967	0.99717	0.99603	1	1	0.99500	1	1	1	1	0.99886	0.52262

Tabla 10.3: Porcentaje de pares de palabras de cada dataset para los que cada word emebdding ha sido capaz de dar respuesta

Normalización	Media	Media Ponderada	SimLex 999	MTurk 287	MTurk 771	MEN DEV	MEN TEST	MEN ALL	WS353 all	WS353 relatedness	WS353 similarity	WS353 set1	WS353 set2	RG65	VERB 143	MC 30	YP 130	SimVerb 3500	RW
Producto escalar	-0.028	-0.027	-0.025	-0.028	-0.047	-0.026	-0.026	-0.026	-0.022	-0.018	-0.029	-0.041	-0.010	-0.018	-0.023	-0.031	-0.033	-0.014	-0.046
L_1 por características	0.006	0.007	0.010	0.006	0.003	0.003	0.006	0.004	0.005	0.006	0.002	0.004	0.007	-0.001	0.008	0.003	0.005	0.009	0.011
L_2 por características	0.005	0.007	0.009	0.006	0.003	0.003	0.006	0.004	0.005	0.006	0.001	0.003	0.006	-0.002	0.009	0.001	0.005	0.009	0.011
Centrado de variables	-0.010	-0.012	-0.010	-0.014	-0.006	-0.007	-0.004	-0.006	-0.023	-0.022	-0.012	-0.003	-0.031	0.013	-0.024	0.007	-0.009	-0.015	-0.020
Centrado de vectores	-0.001	0.000	-0.001	0.000	-0.001	0.000	0.000	0.000	-0.001	-0.001	-0.001	0.000	-0.001	-0.002	-0.002	-0.002	0.000	0.000	0.000
PPA	0.005	0.007	0.011	-0.008	-0.002	0.001	0.002	0.001	0.009	0.011	-0.002	0.007	0.011	-0.002	0.010	0.005	0.001	0.011	0.013
L_2 por vectores + Centrado de variables	-0.003	-0.005	-0.001	-0.004	-0.001	-0.003	-0.002	-0.003	-0.008	-0.005	-0.005	0.000	-0.010	0.014	-0.019	0.005	-0.005	-0.004	-0.012
L_2 vectores + PPA	0.005	0.008	0.012	-0.005	-0.002	0.001	0.003	0.001	0.010	0.012	0.001	0.007	0.014	-0.004	0.006	0.006	0.002	0.013	0.015
L_2 por vectores + L_2 por características	0.006	0.007	0.010	0.005	0.003	0.003	0.006	0.004	0.006	0.006	0.002	0.004	0.007	-0.001	0.007	0.002	0.006	0.009	0.011
L_2 por columnas + Centrado de variables	0.002	0.001	0.001	0.001	0.003	0.003	0.007	0.004	-0.002	0.001	0.001	0.007	-0.007	0.016	-0.015	0.014	0.006	-0.001	0.001
L_2 por características + PPA	0.006	0.010	0.017	-0.008	-0.002	0.000	0.004	0.002	0.009	0.010	-0.003	0.008	0.014	-0.005	0.014	-0.002	0.009	0.017	0.017
Centrado de variables + PPA	0.002	0.002	0.009	-0.018	0.001	-0.002	0.004	0.000	-0.001	0.003	-0.005	0.006	0.000	0.013	-0.011	0.012	0.004	0.004	0.004
Centrado de variables + L_2 por características	-0.005	-0.008	-0.006	-0.009	-0.002	-0.003	0.000	-0.002	-0.015	-0.014	-0.006	0.001	-0.021	0.017	-0.018	0.008	-0.004	-0.011	-0.014
L_2 por vectores + L_2 por características + PPA	0.005	0.011	0.017	-0.008	-0.001	0.000	0.005	0.002	0.009	0.010	-0.001	0.009	0.014	-0.006	0.003	-0.001	0.010	0.018	0.017
PPA + Centrado de variables	0.002	0.002	0.009	-0.018	0.001	-0.002	0.004	0.000	-0.001	0.003	-0.005	0.006	0.000	0.013	-0.011	0.012	0.004	0.004	0.004
PPA + L_2 por caracteristicas	0.004	0.008	0.013	-0.008	-0.003	0.000	0.003	0.001	0.006	0.008	-0.003	0.005	0.008	-0.001	0.003	0.005	0.006	0.013	0.014
PPA + L_2 pro vectores + L_2 por caracteristicas	0.005	0.008	0.014	-0.008	-0.002	0.000	0.003	0.001	0.007	0.009	-0.003	0.006	0.009	0.000	0.004	0.007	0.008	0.013	0.014
Centrado de variables + L_2 por vectores + L_2 por caracteristicas	-0.004	-0.007	-0.005	-0.009	-0.001	-0.003	0.001	-0.002	-0.012	-0.011	-0.005	0.003	-0.019	0.018	-0.018	0.011	-0.004	-0.010	-0.012
L_2 por características +L_2 vectores + Centrado de variables	0.006	0.006	0.008	0.009	0.004	0.004	0.007	0.005	0.006	0.010	0.004	0.006	0.006	0.015	-0.010	0.014	0.007	0.008	0.004

Tabla 10.4: Media de la diferencia de rendimiento entre los word embeddings normalizados y los word embeddings originales en cada dataset. . Métrica utilizada: Ignorar pares para los que no se puede dar respuesta.

Bibliografía

[1] British National Corpus. http://www.natcorp.ox.ac.uk/corpus/.

[2] Commoncrawl. http://commoncrawl.org/.

[3] Conceptnet. http://www.conceptnet.io/.

[4] Google news dataset. https://cloud.google.com/bigquery/public-data/.

[5] Multilingual and cross-lingual semantic word similarity. http://alt.qcri.org/semeval2017/task2/.

[6] Sketch engine. https://www.sketchengine.eu/.

[7] Statmt.org. http://www.statmt.org/.

[8] Twitter. https://twitter.com/.

[9] ukWaC – British English corpus from the .uk domain. https://www.sketchengine.eu/ukwac-british-english-corpus/.

[10] Wikipedia. https://www.wikipedia.org/.

[11] MSR Sentence Completion Challenge. https://www.microsoft.com/en-us/research/project/msr-sentence-completion-challenge/, 2011.

[12] Robert Parker, David Graff, Junbo Kong, Ke Chen, Kazuaki Maeda. English Gigaword Fifth Edition. https://catalog.ldc.upenn.edu/LDC2011T07l, 2011.

[13] Marco Idiart Alexandre Salle and Aline Villavicencio. Enhancing the lexvec distributed word representation model using positional contexts and external memory. *arXiv:1606.01283v1*, 2016.

[14] Marco Idiart Alexandre Salle and Aline Villavicencio. Matrix factorization using window sampling and negative sampling for improved word representations. *ACL*, 2016.

[15] Marco Idiart Alexandre Salle and Aline Villavicencio. Incorporating subword information into matrix factorization word embeddings. *ACL*, 2018.

[16] Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. Unsupervised neural machine translation. In *Proceedings of the Sixth International Conference on Learning Representations*, April 2018.

[17] Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. Unsupervised neural machine translation. In *Proceedings of the Sixth International Conference on Learning Representations*, April 2018.

[18] Amir Bakarov. A survey of word embeddings evaluation methods. 01 2018.

[19] Amir Bakarov and Olga Gureenkova. Automated detection of non-relevant posts on the russian imageboard "2ch": Importance of the choice of word representations. *CoRR*, abs/1707.04860, 2017.

[20] Baker, Reichart and Korhonen. Verb Similarity Dataset . *EMNLP 2014*, 2014.

[21] Mohit Bansal, Kevin Gimpel, and Karen Livescu. Tailoring continuous word representations for dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 809–815. Association for Computational Linguistics, 2014.

[22] Tal Baumel, Raphael Cohen, and Michael Elhadad. Sentence embedding evaluation using pyramid annotation. pages 145–149, 01 2016.

[23] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[24] Danushka Bollegala, Kohei Hayashi, and Ken-ichi Kawarabayashi. Think globally, embed locally - locally linear meta-embedding of words. *CoRR*, abs/1709.06671, 2017.

[25] Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. Massive Exploration of Neural Machine Translation Architectures. *ArXiv e-prints*, March 2017.

[26] John A. Bullinaria and Joseph P. Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510–526, Aug 2007.

[27] R. Collobert. Deep learning for efficient discriminative parsing. In *AISTATS*, 2011.

[28] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. pages 160–167, 2008.

[29] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November 2011.

[30] Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. 2018.

[31] Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart and Anna Korhonen. A Large-Scale Evaluation Set of Verb Similarity . *EMNLP 2016*, 2016.

[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. 2009.

[33] Cícero Nogueira dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. pages 69–78, 2014.

[34] E. Bruni, N. K. Tran and M. Baroni. Multimodal Distributional Semantics. *Journal of Artificial Intelligence Research 49*, 2013.

[35] Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Pasca, Aitor Soroa. Relatedness Using Distributional and WordNet-based Approaches. *NAACL-HLT 2009*, 2009.

[36] Allyson Ettinger and Tal Linzen. Evaluating vector space models using human semantic priming results. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 72–77. Association for Computational Linguistics, 2016.

[37] Manaal Faruqui, Jesse Dodge, Sujay K. Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. Retrofitting word vectors to semantic lexicons. In *Proceedings of NAACL*, 2015.

[38] Felix Hill, Roi Reichart and Anna Korhonen. SimLex-999: Evaluating Semantic Models with (Genuine) Similarity Estimation. https://www.cl.cam.ac.uk/~fh295/simlex.html, 2014.

[39] Fellbaum, Christiane. WordNet and wordnets . *In: Brown, Keith et al. (eds.), Encyclopedia of Language and Linguistics, Second Edition, Oxford: Elsevier, 665-670*, 2005.

[40] J.R. Firth. A synopsis of linguistic theory 1930-1955. *Philological Society: 1–32*, 1957.

[41] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-burch. Ppdb: The paraphrase database. In *In HLT-NAACL 2013*, 2013.

[42] Josu Goikoetxea, Eneko Agirre, and Aitor Soroa. Single or multiple? combining word representations independently learned from text and wordnet. pages 2608–2614, 2016.

[43] Josu Goikoetxea, Aitor Soroa, and Eneko Agirre. Random walks and neural network language models on knowledge bases. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1434–1439. Association for Computational Linguistics, 2015.

[44] Josu Goikoetxea, Aitor Soroa, and Eneko Agirre. Bilingual embeddings with random walks over multilingual wordnets. *Know.-Based Syst.*, 150(C):218–230, June 2018.

[45] Josu Goikoetxea, Aitor Soroa, and Eneko Agirre. Bilingual embeddings with random walks over multilingual wordnets. *Knowledge-Based Systems*, 150:218 – 230, 2018.

[46] Guy Halawi, Gideon Dror, Evgeniy Gabrilovich and Yehuda Koren. Large-scale learning of word relatedness with constraints. 2012.

[47] Z. Harris. Distributional structure. *Word, 10(23): 146-162.*, 1954.

[48] Herbert Rubenstein and John B. Goodenough. Contextual correlates of synonymy. *Communications of the ACM, 8(10):627-633*, 1965.

[49] Stanislaw Jastrzebski, Damian Lesniak, and Wojciech Marian Czarnecki. How to evaluate word embeddings? on importance of data efficiency and simple supervised tasks. *CoRR*, abs/1702.02170, 2017.

[50] Jeffrey Pennington and Richard Socher and Christopher D. Manning. GloVe: Global Vectors for Word Representation. *Empirical Methods in Natural Language Processing (EMNLP). pages 1532–1543*, 2014.

[51] Pramod Viswanath Jiaqi Mu. All-but-the-top: Simple and effective postprocessing for word representations. 2017.

[52] Hwiyeol Jo and Stanley Jungkyu Choi. Extrofitting: Enriching word representation and its vector space with semantic lexicons. *arXiv preprint arXiv:1804.07946*, 2018.

[53] Danushka Bollegala Joshua N Coates. Frustratingly easy meta-embedding – computing meta-embeddings by averaging source word embeddings. 2018.

[54] Kira Radinsky, Eugene Agichteint, Evgeniy Gabrilovich, Shaul Markovitch. A Word at a Time: Computing Word Relatedness using Temporal Semantic Analysis. 2011.

[55] Arne Köhn. Evaluating embeddings using syntax-based classification tasks as a proxy for parser performance. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 67–71, Berlin, Germany, August 2016. Association for Computational Linguistics.

[56] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. 2018.

[57] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing Search in Context: The Concept Revisited. *ACM Transactions on Information Systems*, 2002.

[58] Omer Levy and Yoav Goldberg. Linguistic regularities in sparse and explicit word representations. *Eighteenth Conference on Computational Language Learning, pages 171–180, Baltimore, Maryland USA, June 26-27 2014.*, 2014.

[59] Luong, Minh-Thang and Socher, Richard and Manning, Christopher D. Better Word Representations with Recursive Neural Networks for Morphology . *CoNLL*, 2013.

[60] Tim Finin James Mayfield Lushan Han, Abhay L. Kashyap and Johnathan Weese. UMBC EBIQUITY-CORE: Semantic Textual Similarity Systems. In *Proceedings of*

the Second Joint Conference on Lexical and Computational Semantics. Association for Computational Linguistics, June 2013.

[61] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[62] Miller, G. A., & Charles, W. G. Contextual Correlates of Semantic Similarity. *Language and Cognitive Processes*, 1991.

[63] Ido Dagan Omer Levy, Yoav Goldberg. Improving distributional similarity with lessons learned from word embeddings. *ACL*, 2015-10-16, 2015.

[64] Ido Dagan Oren Melamud, Jacob Goldberger. context2vec: Learning generic context embedding with bidirectional lstm. 2016.

[65] Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Comput. Linguist.*, 31(1):71–106, March 2005.

[66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[67] Patrick Pantel Peter D. Turney. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research, (2010), 37, 141-188*, 2010.

[68] Catherine Havasi Robert Speer, Joshua Chin. Conceptnet 5.5: An open multilingual graph of general knowledge. *arXiv preprint arXiv:1612.03975*, 2016.

[69] Joanna Lowry-Duda Robert Speer. Conceptnet at semeval-2017 task 2: Extending word embeddings with multilingual relational knowledge. *arXiv:1704.03560*, 2017.

[70] Joshua Chin Robert Speer. An ensemble method to produce high-quality word embeddings. 2016.

[71] Herbert Rubenstein and John B. Goodenough. Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633, 1965.

[72] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. pages 696–699, 1988.

[73] Tobias Schnabel, Igor Labutov, David M. Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *EMNLP*, pages 298–307. The Association for Computational Linguistics, 2015.

[74] C. Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 15:88–103, 1904.

[75] Robert Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. pages 4444–4451, 2017.

[76] Robert Speer and Joanna Lowry-Duda. Conceptnet at semeval-2017 task 2: Extending word embeddings with multilingual relational knowledge. *CoRR*, abs/1704.03560, 2017.

[77] Fei Sun, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. Learning word representations by jointly modeling syntagmatic and paradigmatic relations. pages 136–145, 2015.

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

[79] Greg Corrado Tomas Mikolov, Kai Chen and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781v3*, 2013.

[80] Jianqing Fan Tony Cai and Tiefeng Jiang. Distributions of angles in random packing on spheres. *The Journal of Machine Learning Research 14(1):1837–1864*, 2013.

[81] Yulia Tsvetkov, Leonid Boytsov, Anatole Gershman, Eric Nyberg, and Chris Dyer. Metaphor detection with cross-lingual model transfer.

[82] Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Guillaume Lample, and Chris Dyer. Evaluation of word vector representations by subspace alignment. pages 2049–2054, 01 2015.

[83] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 384–394, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[84] Hinrich Schütze Wenpeng Yin. Learning meta-embeddings by using ensembles of embedding sets. *arXiv:1508.04257*, 2015.

[85] Benjamin J. Wilson and Adriaan M. J. Schakel. Controlled experiments for word embeddings. *CoRR*, abs/1510.02675, 2015.

[86] Yang, D., & Powers, D. M. W. Verb Similarity on the Taxonomy of WordNet. *Proceedings of the Third International WordNet Conference (GWC-06) (pp. 121-128)*, 2006.

[87] Wenpeng Yin and Hinrich Schütze. Learning word meta-embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1351–1360, Berlin, Germany, August 2016. Association for Computational Linguistics.

[88] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. pages 649–657, 2015.