UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Uso de representaciones vectoriales de las palabras para la detección de dobles sentidos (Puns)

TRABAJO FIN DE MÁSTER

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Autor: Pascual Andrés Carrasco Gómez

Tutor: Dra. Encarnación Segarra Soriano
Dr. Ferran Pla Santamaría

Curso 2016-2017

Resumen

La desambiguación semántica o la comprensión del lenguaje natural son ámbitos dentro del procesamiento del lenguaje natural que aunque han sido ampliamente estudiados, siguen suponiendo un reto importante. Los enfoques tradicionales de la desambiguación semántica descansan en la suposición de que existe una única e inequívoca semántica subyacente a cada palabra en una oración. Sin embargo, existe una clase de construcciones en el lenguaje conocidas como juegos de palabras (*puns*), en los que la ambigüedad léxico-semántica es un efecto buscado en la oración. Es decir, el hablante o escritor pretende que una determinada palabra u otro elemento léxico sea interpretado simultáneamente con dos o más significados distintos.

En este proyecto proponemos abordar la localización y desambiguación de palabras con doble sentido (*puns*) en una serie de oraciones. Para ello usaremos diferentes representaciones vectoriales de las palabras obtenidas a partir de diferentes corpus, y se estudiarán diferentes métricas de similitud. Los conjuntos de datos pertenecen a la competición internacional Semeval 2017 y los resultados se podrán contrastar con los publicados por la competición.

Palabras clave: Desambiguación semántica, WSD, juego de palabras, puns, polisemia, embeddings, WordNet, lenguaje natural.

Abstract

Semantic disambiguation or the understanding of natural language are areas within the natural language processing that although they have been widely studied, continue to pose a major challenge. Traditional approaches to semantic disambiguation lie on the assumption that there is a unique and unequivocal semantic underlying each word in a sentence. However, there is a class of constructions in language known as puns, in which lexical-semantic ambiguity is a sought-after effect in the sentence. That is, the speaker or writer pretends that a particular word or other lexical element is interpreted simultaneously with two or more different meanings.

In this project we propose to approach the location and disambiguation of double meaning words (puns) in a set of sentences. To do this we will use different vector representations of words obtained from different corpus, and different metrics of similarity will be studied. The data sets belong to the Semeval 2017 international competition and the results can be compared with those published by the competition.

Key words: Semantic disambiguation, WSD, wordplay, puns, polysemy, embeddings, WordNet, natural language.

Resum

La desambiguació semàntica o la comprensió del llenguatge natural són àmbits dins del processament del llenguatge natural que encara que han estat àmpliament estudiats, segueixen suposant un repte important. Els enfocaments tradicionals de la desambiguació semàntica descansen en la suposició que hi ha una única e inequívoca semàntica subjacent a cada paraula en una oració. No obstant això, hi ha una classe de construccions en el llenguatge conegudes com jocs de paraules (*puns*), en els quals l'ambigüitat lexicosemàntica és un efecte buscat en l'oració. És a dir, el parlant o escriptor pretén que una determinada paraula o un altre element lèxic sigui interpretat simultàniament amb dos o més significats diferents.

En aquest projecte proposem abordar la localització i desambiguació de paraules amb doble sentit (*puns*) en una sèrie d'oracions. Per a això farem servir diferents representacions vectorials de les paraules obtingudes a partir de diferents corpus, i s'estudiaran diferents mètriques de similitud. Els conjunts de dades pertanyen a la competició internacional Semeval 2017 i els resultats es podran contrastar amb les publicades per la competició.

Paraules clau: Desambiguació semàntica, WSD, joc de paraules, puns, polisèmia, embeddings, WordNet, llenguatge natural.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

El Procesamiento del Lenguaje Natural (PLN) es la disciplina formada por la lingüística aplicada y la inteligencia artificial que tiene como objetivo intentar imitar la capacidad que tenemos los humanos de hablar y comprender el lenguaje natural.

Uno de los problemas con mayor complejidad en PLN es la comprensión del lenguaje natural. En la actualidad los sistemas informáticos no son capaces de hablar y entender a los humanos de igual forma como los humanos lo hacen entre sí. Cuando hablamos de lenguaje es importante conocer la diferencia entre lenguaje natural y lenguaje artificial, el lenguaje natural es el lenguaje verbal que utilizamos los seres humanos para comunicarnos entre si, por otra parte el lenguaje artificial es un lenguaje creado con una especificación detallada para ser utilizado en un entorno concreto. Algunos problemas simples pueden ser resueltos con un lenguaje artificial, como por ejemplo un sistema cuyo vocabulario es la afirmación *sí*, la negación *no* y el conjunto de dígitos [0-9] que se utiliza mediante un contestador automático para obtener el número de identidad de un cliente en una empresa.

El lenguaje natural esta formado por un vocabulario finito, donde las unidades básicas (palabras) se combinan formando un número infinito de oraciones (lenguaje). El procesamiento del lenguaje se ve influenciado por la dificultad lingüística del propio lenguaje y el *hardware* que se dispone actualmente para procesarlo.

Comprender el significado de una oración a nivel computacional es una tarea muy compleja, una buena práctica es analizar individualmente el sentido de cada palabra dentro de la oración utilizando la información que nos proporciona el contexto, posteriormente analizando el sentido de cada palabra podemos obtener una comprensión global de la oración. El contexto de la palabra es considerado como un conjunto de palabras que la acompañan y que le aportan información relevante acerca de su significado. Obtener el sentido de una palabra mediante el contexto es una tarea compleja y que no esta resuelta.

Uno de los problemas que hacen la tarea compleja es lo que se conoce en lingüística como polisemia. La polisemia se presenta cuando una misma palabra tiene más de un significado. Por ejemplo la palabra *cabo* en la lengua española puede significar «una punta de tierra que penetra en el mar», «un escalafón militar» o «una cuerda en jerga náutica». Los seres humanos, generalmente, somos capaces de distinguir el sentido de una palabra en el contexto de una oración debido a nuestra comprensión del habla desarrollada a lo largo de nuestra evolución como especie.

$\Rightarrow$ El cabo está bien atado.

$\Rightarrow$ El cabo no cumplió las ordenes y fue encarcelado.

La palabra *cabo* en la primera oración hace referencia a una cuerda en jerga náutica, observamos que el sentido de la palabra viene definido por el verbo *atar*, sin embargo la palabra *cabo* en la segunda oración hace referencia a un escalafón militar donde el sentido viene definido por el significado de *cumplir ordenes* y el verbo *encarcelar*. Este análisis los humanos lo hacemos inconscientemente pero un ordenador no dispone de nuestra ventaja.

Existen oraciones en las cuales los humanos tampoco somos capaces de conocer el significado de alguna de sus palabras.

⇒ Estuve esperándote en el banco.

En esta oración la palabra *banco* puede tener el significado de entidad bancaria y al mismo tiempo el significado de un asiento para varias personas. Esto sucede porque el contexto de la oración no nos aporta información suficiente para comprender el significado de la palabra.

⇒ Estuve esperándote en el banco Santander.

⇒ Estuve esperándote en el banco de madera del parque.

La asignación automática de los sentidos a las palabras de un texto se conoce en PLN como desambiguación semántica del significado de las palabras, en inglés *Word Sense disambiguation (WSD)*.

1.1 Motivación

La comprensión de texto por un ordenador tiene diversidad de aplicaciones como por ejemplo detectar y entender ironías, sarcasmo, retórica, humor, etc. En los últimos años se ha incrementado el interés acerca de un tema tradicional en el estudio de la lingüística y las ciencias cognitivas que se basa en la detección e interpretación de *puns*.

La detección e interpretación de *puns* se puede ver como un problema de *WSD*. Tradicionalmente cuando trabajamos con *WSD* se asume que todas las palabras del texto tienen únicamente un sentido que describe su significado, sin embargo existen una clase de construcciones en el lenguaje conocidas como paranomasia, o juego de palabras (*puns*), que hacen que una palabra pueda tener más de un sentido en el contexto en el que se encuentra.

Un *pun* se define como una forma de juego de palabras que tiene dos o más significados explotando la polisemia, homonimia o similitud fonética de otra palabra con la intención de generar un efecto humorístico o retórico.

Los escritores a lo largo de la historia han utilizado los *puns* como recurso para crear paralelismos entre dos conceptos, generar chistes, definir eslóganes e incluso utilizarlos como un recurso retórico y poético en la literatura. Aunque se pueden relacionar en diferentes aspectos de la literatura su aparición es más común en la comedia.

La detección e interpretación de *puns* permite poder resolver problemas relacionados con la generación y detección de humor por una máquina con la finalidad de mejorar la interacción persona-máquina, son de gran importancia para el análisis de sentimientos, para el reconocimiento de ambigüedad humorística en sistemas de traducción automática y para el análisis y crítica de autores de obras literarias.

SemEval es una competición internacional donde cada año propone un conjunto de tareas a resolver relacionadas con PLN que tienen gran interés por la comunidad científica. La competición SemEval-2017 ha planteado una tarea para el estudio de *puns* en el idioma inglés. Este constituye un problema de PLN en el que no existe mucha

experiencia anterior ni corpus en la comunidad científica, la novedad y dificultad lo convierten en un problema interesante y es el que se ha abordado en este trabajo final de máster.

1.2 Objetivos

En este apartado se presentan los objetivos que se pretenden alcanzar al finalizar este trabajo. Los objetivos propuestos se pueden agrupar en tres niveles: básico, intermedio y avanzado variando el nivel según la dificultad que presenta el objetivo para ser resuelto.

Los objetivos de nivel básico corresponden a la capacidad de asimilar conceptos para poder llevar a cabo las tres tareas que se resuelven de la competición SemEval-2017 siendo capaces de entender la necesidad de representar las palabras como vectores y cómo obtener las mejores representaciones vectoriales, la importancia de WordNet para los algoritmos de desambiguación semántica y adquirir conocimientos avanzados con las herramientas y lenguajes de programación que se utilizan hoy en día para resolver problemas de procesamiento del lenguaje natural.

El problema de *WSD* es uno de los problemas más complejos del procesamiento del lenguaje natural siendo un problema por resolver en la actualidad dentro de la comunidad científica. Los *puns* incrementan la complejidad al explotar la polisemia en el texto. Los objetivos de nivel intermedio se centran en la capacidad de resolver con resultados significativos las tareas relacionadas con *puns* que propone SemEval-2017 buscando alternativas para resolver cada tarea con la finalidad de intentar mejorar los resultados.

En el presente trabajo se presentan las soluciones que propusimos en la competición de SemEval-2017. Los objetivos de nivel avanzado se centran en intentar mejorar los resultados que obtuvimos en la competición manteniendo los criterios que realizamos utilizando la representación vectorial de palabras.

1.3 Estructura de la memoria

El presente trabajo se estructura en 11 capítulos que se describen brevemente en este apartado:

- **Capítulo 1, Introducción**: En este capítulo se introducen conceptos relacionados con el Procesado del Lenguaje Natural (PLN) que nos permiten explicar el problema de desambiguación semántica siendo un problema abierto y de gran interés por la comunidad científica. Los *puns* (juego de palabras) en el idioma inglés son un tipo de problema de desambiguación semántica de gran interés en la actualidad y son la motivación para la realización de este trabajo. También se presenta la estructura que tiene el trabajo y las asignaturas relacionadas del mismo que han sido impartidas en el máster.

- **Capítulo 2, Representación vectorial de palabras y frases**: En este capítulo se explican los métodos más utilizados para la representación de palabras en un ordenador. Se explican representaciones simples (*One-hot* y *BOW*) y representaciones más complejas basadas en redes neuronales (*CBOW y Skip-gram*).

- **Capítulo 3, Algoritmos de aprendizaje automático**: El capítulo explica la diferencia entre el tipo de aprendizaje deductivo y aprendizaje inductivo. En este

trabajo nos centramos en aprendizaje inductivo explicando en qué consiste el aprendizaje supervisado y el aprendizaje no supervisado. Por último se presentan los algoritmos de clasificación necesarios para abordar las tareas en este trabajo.

- **Capítulo 4, WordNet**: En este capítulo se presenta WordNet explicando como se estructura su conocimiento y en qué tipo de aplicaciones es de gran utilidad. WordNet es una de las herramientas más utilizadas para resolver problemas de desambiguación semántica.

- **Capítulo 5, Competición**: Este capítulo explica en qué consiste la competición SemEval-2017 haciendo hincapié en la tarea relacionada con *puns*. La tarea propone tres subtareas a resolver: detección, localización e interpretación de *puns*. También se explican las medidas de evaluación y los *baselines* que propone la organización para dicha tarea.

- **Capítulo 6, Corpus**: La organización de SemEval-2017 proporciona un corpus (*dataset*) para la tarea relacionada con *puns*. En este capítulo se explica como se han obtenidos los datos del corpus, se proporcionan características del corpus y se comenta el conjunto de estadísticas que proporciona SemEval-2017 acerca del corpus.

- **Capítulo 7, Aproximaciones al problema**: El capítulo presenta estudios previos basados en la detección y comprensión de juegos de palabras, centrándonos en técnicas basadas en análisis sintáctico, fonológico y semántico. Posteriormente, se explican las soluciones que han propuesto los participantes de la tarea relacionada con *puns* de SemEval-2017 que nos permite conocer un enfoque muy actual acerca de la detección y comprensión de *puns*.

- **Capítulo 8, Entorno de trabajo y herramientas**: En este capítulo se explica qué lenguaje de programación se ha utilizado para desarrollar el sistema y qué conjunto de herramientas se han utilizado para trabajar con PLN.

- **Capítulo 9, Experimentación**: Este capítulo se divide en tres partes que se corresponden con las tres subtareas de la tarea de *puns* de SemEval-2017. Para cada subtarea se aportan los estudios y soluciones realizados junto con los resultados obtenidos. Es de gran importancia comentar que en el presente trabajo no se realizan comparaciones con los participantes de la competición debido a que nuestros estudios utilizan parte del corpus etiquetado que ha sido liberado posteriormente a la competición.

- **Capítulo 10, Conclusiones**: En este capítulo se hace un resumen del trabajo realizado detallando los mejores resultados que se han obtenido para cada subtarea relacionada con *puns* de la competición SemEval-2017.

- **Capitulo 11, Trabajo futuro**: En este capítulo se exponen un conjunto de problemas similares que pueden ser abordados en un futuro utilizando como base los estudios que se realizan en el presente trabajo.

1.4 Asignaturas relacionadas

Los conocimientos proporcionados en el máster han sido de gran ayuda para el sistema desarrollado y la elaboración de la memoria del presente trabajo. Las asignaturas más relacionadas con este trabajo son las que pertenecen a la rama de tecnologías del lenguaje (TL) y a la rama de reconocimiento de formas (RF).

Todas estas asignaturas han proporcionado una base sólida sobre la cual se ha abordado los problemas que presenta este trabajo ayudando a comprender conceptos nuevos de forma más rápida. A continuación se describe la aportación de conocimientos previos de cada asignatura relacionados con este trabajo:

Asignaturas de la rama TL:

- Lingüística computacional: Esta asignatura define el significado y la necesidad de la lingüística computacional a lo largo de la historia y en la actualidad. Se estudian conceptos de análisis sintácticos, análisis semánticos y desambiguación semántica. También se estudia diferentes modelos de lenguaje.
- Aplicaciones de lingüística computacional: Esta asignatura presenta herramientas para resolver problemas en lingüística computacional como WordNet, DBpedia, Wikipedia, etc. Se estudian diferentes representaciones de palabras y diferentes métricas de distancia y similitud. Como proyecto de la asignatura se ha de participar en una competición como SensEval, SemEval o IberEval aportando conocimientos de cómo funcionan las competiciones y qué problemas existen en la actualidad en lingüística computacional.
- Traducción automática: Se estudia la representación de palabras mediante *embeddings* proporcionando conocimientos sobre las topologías *CBOW* y *Skip-grams* mediante redes neuronales.

Asignaturas de la rama RF:

- Reconocimiento de formas y aprendizaje computacional: En esta asignatura se proporcionan conceptos acerca de aprendizaje no supervisado y aprendizaje supervisado.
- Redes neuronales artificiales: Se estudian las representaciones de palabras mediante redes neuronales a lo largo de la historia, empezando con las primeras aproximaciones mediante redes *feedforward* hasta las topologías *CBOW* y *Skip-grams* actuales.

CAPÍTULO 2
Representación vectorial de palabras y frases

Muchos sistemas y técnicas actuales de PLN utilizan las palabras como unidad básica. Un ordenador trabaja con números, por lo tanto, cuando queremos trabajar con palabras es necesario obtener una representación numérica para cada palabra o conjunto de palabras. Esta representación consiste habitualmente en un vector de números enteros o en coma flotante. La representación discreta (números enteros) se obtiene mediante técnicas más simples sin embargo la representación continua (números en coma flotante) requiere de técnicas más complejas.

2.1 Representaciones simples

Las representaciones simples de palabras utilizan el conjunto de palabras del vocabulario (V) para obtener una representación. Estas representaciones se obtienen directamente de la información con la que se trabaja sin necesidad de obtener un modelo con una fase de entrenamiento previa.

2.1.1. One-hot

Un vector *one-hot* es un vector de tamaño |V| donde cada componente del vector identifica una palabra del vocabulario, es decir, el vector *one-hot* de una palabra tiene un 1 en la posición que identifica dicha palabra y los |V|-1 elementos restantes a 0. Un ejemplo ilustrativo para un vocabulario formado por tres palabras V = {Casa, Coche, Mesa} se muestra en la Tabla 2.1:

Palabra	**Vector**		
Casa	1	0	0
Coche	0	1	0
Mesa	0	0	1

Tabla 2.1: Ejemplo de representación *one-hot*.

La representación *one-hot* no aporta información acerca de relaciones entre palabras y presenta problemas de dimensionalidad cuando trabajamos con un tamaño de vocabulario grande ya que la dimensión de los vectores es |V|. Esta representación contiene un gran número de ceros que obliga a trabajar con matrices dispersas que pueden generar un gran coste computacional.

2.1.2. Bag Of Words (BOW)

La representación basada en bolsa de palabras (*BOW*) codifica un conjunto de palabras de un texto. La representación consiste en vectores de tamaño |V| que tienen 1 en aquellas posiciones correspondientes a las palabras que aparecen en el texto. Un ejemplo ilustrativo para un vocabulario formado por trece palabras V = {El, coche, de, carreras, es, Valencia, María, tiene, la, camisa, más, bonita, tienda} y dos frases se muestra en la Tabla 2.2:

Frase	**Vector**												
El coche de carreras es de Valencia.	1	1	1	1	1	1	0	0	0	0	0	0	0
María tiene la camisa más bonita de la tienda.	0	0	1	0	0	0	1	1	1	1	1	1	1

Tabla 2.2: Ejemplo de representación *BOW*.

Se puede observar que la representación basada en bolsa de palabras se obtiene aplicando la unión de la representación *one-hot* de cada palabra de la frase. Es muy habitual trabajar con frecuencias de palabras en *BOW* utilizando la suma de la representación *one-hot* de las palabras de la frase.

Frase	**Vector**												
El coche de carreras es de Valencia.	1	1	2	1	1	1	0	0	0	0	0	0	0
María tiene la camisa más bonita de la tienda.	0	0	1	0	0	0	1	1	2	1	1	1	1

Tabla 2.3: Ejemplo de representación *BOW* basada en frecuencias de la Tabla 2.2

Esta representación también presenta problemas de dimensionalidad cuando |V| es grande y además no mantiene la información de orden en la que aparecen las palabras en la oración. Los vectores de representación siguen conteniendo un gran número de ceros que afectan a nivel de computo al trabajar con matrices dispersas.

2.2 Representaciones basadas en redes neuronales

Los modelos de representación simple, aparte de los problemas de dimensionalidad y de operaciones con matrices dispersas, no tienen en cuenta información acerca del contexto de la palabra que representan, este factor es crucial a la hora de representar una palabra ya que nos aportará información sintáctica y semántica. Los seres humanos cuando utilizamos el lenguaje natural en nuestra vida cotidiana inconscientemente analizamos palabras a través de sus palabras más próximas (contexto).

Cuando hablamos del contexto de una palabra es necesario mencionar los modelos de N-gramas. Un modelo N-grama nos aporta información de una palabra utilizando las N-1 palabras que la preceden, estos modelos aportan información sintáctica y aunque son sencillos consiguen resultados muy buenos siempre y cuando se les apliquen técnicas de suavizado. Los N-gramas nos permiten inferir conocimiento sobre una palabra mediante probabilidades y tienen gran importancia en la definición de modelos de lenguaje.

Los modelos que se van a presentar en este apartado se centran en la obtención de modelos de lenguaje. Un modelo de lenguaje permite predecir una palabra a partir de un conjunto de palabras que forman su contexto. Cuando trabajamos con modelos de lenguaje utilizando redes neuronales la capa de entrada corresponde al conjunto de palabras que forman el contexto de la palabra a predecir y la capa de salida esta formada por |V| neuronas donde cada neurona corresponde a una palabra del vocabulario.

En una red neuronal cambiar la función de activación *softmax* en la capa de salida por funciones de activación lineales (regresión) permite obtener una representación de la palabra en vez de predecir la misma. Otra forma de obtener la representación de la palabra con una red neuronal es descartar la capa de salida de la red y quedarnos con una de las capas ocultas, esta topología recibe el nombre de *encoder*.

La representación de una palabra obtenida a partir de la red neuronal recibe el nombre de *word embedding*. Un *word embedding* también conocidos como *embedding* es un vector de números en coma flotante con una dimensión muy inferior a las dimensiones de las representaciones simples.

Los primeros modelos propuestos de *embeddings* [10] se basaban en topologías *feedforward*. Una red *feedforward* cumple la restricción que las neuronas de una capa solo pueden conectarse con las neuronas de capas posteriores (Fig. 2.1).

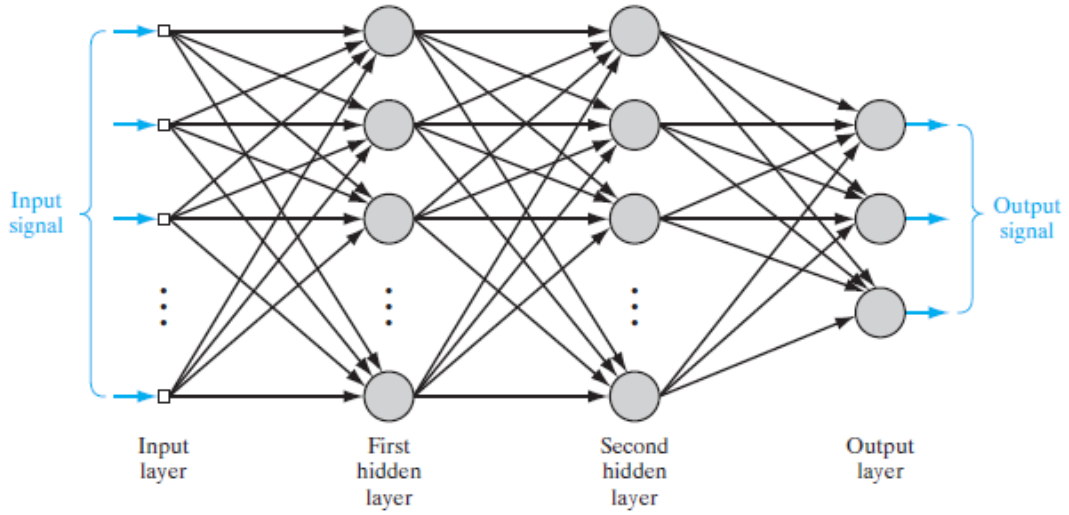

Figura 2.1: Topologia de una red neuronal *feedforward*.

Uno de los modelos más utilizados empleando la topología *feedforward* es el modelo *Feedforward Neural Net Language Model (NNLM)* que se compone de cuatro capas: una capa de entrada, una capa de proyección, una capa oculta y una capa de salida. La capa de entrada esta constituida por las N palabras previas representadas mediante la codificación *one-hot* las cuales son proyectadas sobre la capa de proyección mediante una matriz de proyección compartida cuyo coste computacional es pequeño debido a que en cada momento solo se consideran N entradas activas. El problema computacional de este modelo se encuentra en el gran número de conexiones (parámetros a aprender) entre la capa de proyección y la capa oculta y también entre la capa oculta y la capa de salida.

Las redes neuronales recurrentes (RNN) intentan cubrir las limitaciones de la redes *feedforward*. Permiten trabajar con una entrada a la red que no sea de tamaño fijo (N) y son capaces de representar de forma eficiente patrones más complejos (Fig. 2.2).

El modelo *Recurrent Neural Net Language Model (RNNLM)* está compuesto por tres capas: la capa de entrada, la capa oculta y la capa de salida eliminado la capa de proyección respecto a las redes *feedforward*. Las RNN tienen la peculiaridad de que las neuronas de la capa oculta se conectan con otras neuronas de la misma capa con conexiones con retardo. En la Fig. 2.2 se muestra un ejemplo simple de la topología de una red neuronal recurrente formada por una capa de entrada y una capa oculta siendo la salida de la capa oculta la salida de la red. En el instante n, la capa oculta esta conectada con las neuronas de la capa de entrada ($\boldsymbol{x}_n$) y con las neuronas que corresponden a la salida de la red en el instante anterior ($\boldsymbol{y}_{n-1}$). Este tipo de conexiones otorga a la red una

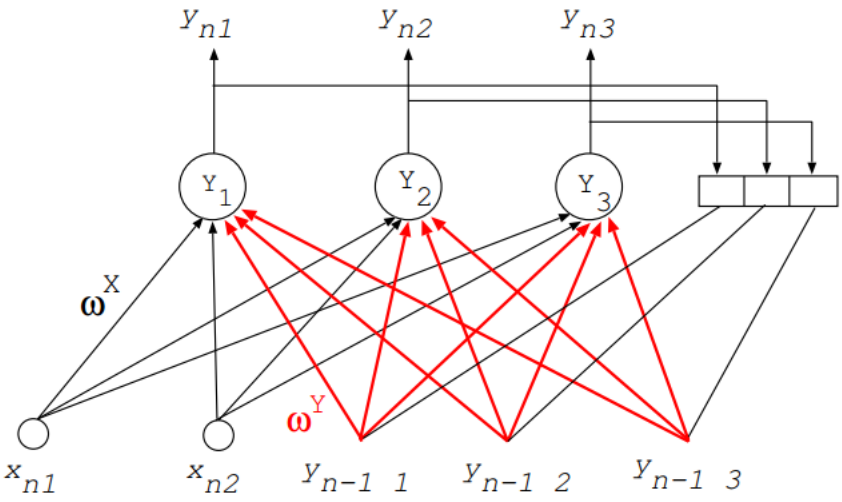

Figura 2.2: Topologia de una red neuronal recurrente.

memoria a corto plazo que le permite obtener el nuevo estado utilizando el estado actual y el estado anterior.

Los modelos anteriores requieren aprender un número elevado de parámetros. Para tratar de minimizar esta complejidad computacional [10, 11] propone dos modelos: el modelo continuo de bolsa de palabras (*CBOW*) y el modelo *Skip-gram*.

Dejando de lado los primeros enfoques de redes neuronales en PLN, nos centramos en el uso de redes neuronales en la actualidad donde se ha conseguido un gran avance con la representación de palabras basada en *embeddings*. El entrenamiento de *embeddings* con redes neuronales se centra en dos modelos: el modelo continuo de bolsa de palabras (*CBOW*) y el modelo *Skip-gram*.

2.2.1. CBOW

El modelo continuo de bolsa de palabras (*CBOW*) es el primer modelo propuesto por [10, 11]. Es similar a la topología *feedforward (NNLM)*. En este modelo se elimina la capa oculta de las redes neuronales *NNLM* y la capa de proyección es compartida entre todas las palabras ocasionando que las palabras sean proyectadas en la misma posición.

El modelo *CBOW* busca obtener la representación (o predicción) de una palabra central a partir de las palabras que se sitúan alrededor de ella (contexto), es decir, dada la frase «El perro juega con la pelota» y dada la palabra «juega» lo que se busca con el modelo *CBOW* es que el contexto pueda obtener una representación (o predicción) de la palabra «juega» a partir de las palabras {El, perro, con, la, pelota}.

En este modelo como entrada se define una ventana de análisis que define el número de palabras que se sitúan alrededor de la palabra central las cuales se proyectan y suman sobre el vector de proyección. El vector de proyección se conecta al vector de salida que nos proporciona la representación (o predicción) de la palabra central.

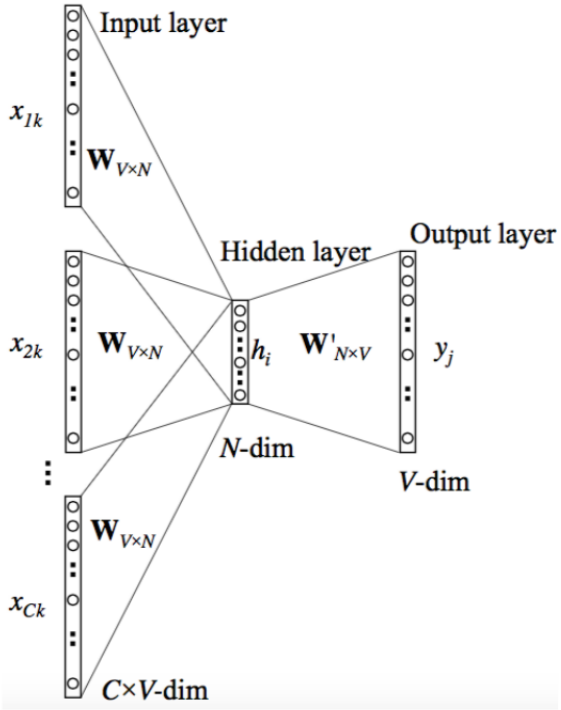

Figura 2.3: Topología de bolsa continua de palabras (*CBOW*).

Dado un vocabulario inicial $V = w^{(1)}, w^{(2)}, ..., w^{(m)}$ con $|V| = m$. Cada palabra del vocabulario $w^{(i)}$ se representa como un vector *one-hot* $x^{(i)}$. Los parámetros que necesita aprender el modelo corresponden a dos matrices, $W^{(1)} \in \mathbb{R}^{mxn}$ y $W^{(2)} \in \mathbb{R}^{nxm}$ donde n es la dimensión de los vectores de representación de las palabras.

Obtenemos los vectores codificados de entrada por cada vector *one-hot* utilizando la matriz $W^{(1)}$ siendo $2C$ el tamaño de la ventana de análisis.

$$(u^{(i-C)} = x^{(i-C)T}W^{(1)}, ..., u^{(i-1)} = x^{(i-1)T}W^{(1)}, u^{(i+1)} = x^{(i+1)T}W^{(1)}, ..., u^{(i+C)} = x^{(i+C)T}W^{(1)})$$

Se calcula la media de los vectores obteniendo un vector h:

$$h = \frac{u^{(i-C)} + u^{(i-C+1)} + ... + u^{(i+C)}}{2C}$$

Obtenemos un vector de *scores* z:

$$z = h^T W^{(2)}$$

Transformamos el vector z en un vector de probabilidades:

$$y' = softmax(z)$$

Buscamos que las probabilidades $y'^{(i)}$ sean iguales a las probabilidades correctas $y^{(i)}$.

Para aprender las matrices $W^{(1)}$ y $W^{(2)}$ se define una función a minimizar [5] basada en la entropía cruzada y se utiliza descenso por gradiente como técnica de minimización.

2.2.2. Skip-gram

El modelo *Skip-gram* corresponde al segundo modelo presentado por [10, 11]. Se basa en el concepto inverso propuesto por el modelo *CBOW*. Este modelo consiste en obtener el contexto a partir de una palabra, es decir, dada la oración «El perro juega con la pelota» y dada la palabra «juega» lo que se busca con el modelo *Skip-gram* es obtener la representación de las palabras (o predicción) {El, perro, con, la, pelota} a partir de la palabra «juega».

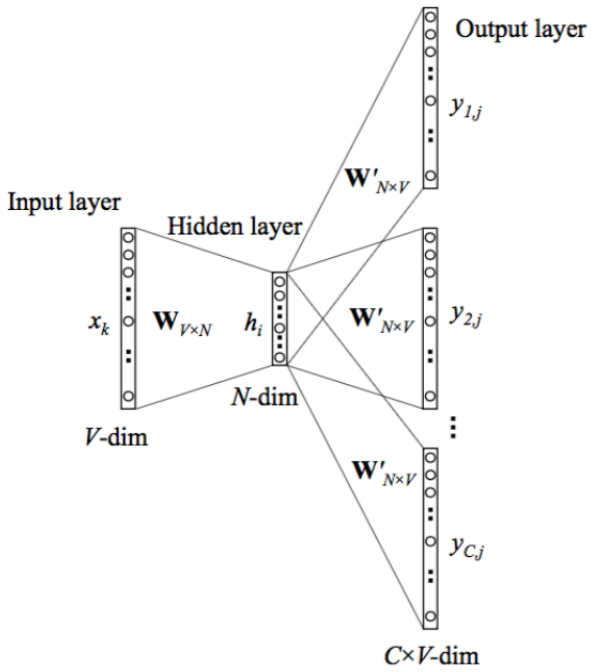

Figura 2.4: Topología (*Skip-gram*).

En este modelo como entrada se utiliza la palabra central que se conecta a la capa de proyección la cual proyecta las palabras de la capa de salida (contexto) cuyo número de palabras en la capa de salida viene definido por una ventana de análisis.

En este modelo la entrada es la palabra central del conjunto de palabras de la ventana de análisis. La codificación en la entrada se realiza únicamente sobre el vector *one-hot* de entrada:

$$u^{(i)} = x^{(i)T} W^{(1)}$$

Al ser únicamente un vector de entrada no se obtiene la media:

$$h = u^{(i)}$$

Se generan los 2C vectores de *scores*:

$$(z^{(i-C)}, ..., z^{(i-1)}, z^{(i+1)}, ..., z^{(i+C)})$$

Siendo $z^{(j)} = h^T W^{(2)}$
Se transforman cada vector z en un vector de probabilidad:

$$y'^{(j)} = softmax(z^{(j)})$$

Buscamos que nuestros vectores de probabilidad generados $y'^{(j)}$ sean iguales que los vectores de probabilidad real $y^{(j)}$.

Al igual que en el modelo *CBOW*, lo que necesitamos es aprender las matrices $W^{(1)}$ y $W^{(2)}$. Para ello se define una función [5] de minimización que corresponde a la negativa de una función de verosimilitud logarítmica que se minimiza aplicando descenso por gradiente.

El tamaño del corpus de entrenamiento es importante para comparar los modelos *CBOW* y *Skip-gram*. Con un corpus de entrenamiento con grandes cantidades de muestras se obtienen representaciones de mayor calidad utilizando el modelo *Skip-gram* debido a que el modelo *CBOW* suaviza mucho las distribuciones de probabilidad al promediar todos los contextos de las palabras. [10, 11] recomiendan el uso del modelo *Skip-gram* ya que capta mejor las propiedades y relaciones semánticas entre términos obteniendo como resultado que el significado de palabras semejantes aparezcan próximas en el espacio vectorial.

Calcular el gradiente de la función *softmax* para entrenar los modelos no es viable en la práctica debido a su coste computacional. Por este motivo [11] propone dos aproximaciones que sean más eficientes, que se conocen como *Softmax Jerárquico* y *Negative Sampling*. La aproximación *Softmax Jerárquico* representa la capa de salida del modelo como un árbol binario aprovechando esta estructura de datos para reducir el coste computacional. La aproximación *Negative Sampling* evalúa únicamente en la capa de salida la neurona que representa la clase correcta más algunas neuronas muestreadas al azar. Las neuronas de salida funcionan como clasificadores independientes de regresión logística. Esto hace que el coste computacional sea independiente del tamaño del vocabulario.

CAPÍTULO 3
Algoritmos de aprendizaje automático

Los métodos de aprendizaje automático se pueden clasificar en dos tipos: deductivo e inductivo. Un sistema de aprendizaje automático deductivo tiene a priori conocimiento proporcionado por una persona experta y a través de dicho conocimiento es capaz de inferir nuevo conocimiento. Por otra parte un sistema de aprendizaje inductivo no tiene conocimiento a priori y mediante un conjunto de datos de entrada genera un modelo para resolver una tarea determinada.

En el presente trabajo nos centramos en algoritmos de aprendizaje inductivo. Los algoritmos de aprendizaje inductivo requieren de un conjunto de datos de entrada para aprender un modelo. Dependiendo del tipo de datos de entrada podemos dividir el aprendizaje automático inductivo en dos tipos: aprendizaje supervisado y aprendizaje no supervisado.

Aprendizaje supervisado:

Dado un conjunto de entrada S de tamaño N, para cada entrada $x \in X$ se conoce la salida $y \in Y$. Siendo X los datos de entrada al sistema e Y los datos de salida del sistema.

$$S = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$$

Aprendizaje no supervisado:

Dado un conjunto de entrada S de tamaño N, únicamente se conocen los datos de entrada al sistema $x \in X$.

$$S = \{(x_1), (x_2), ..., (x_N)\}$$

Lo que necesitamos en este trabajo son algoritmos de aprendizaje inductivo que nos permitan clasificar un dato de entrada $x \in X$ en una clase determinada c dentro de un conjunto de C clases que ha sido previamente predefinido $c \in \{1, 2, ..., C\}$. Los algoritmos de clasificación que se han utilizado en el presente trabajo son *K-neighbours* y *SVM* para aprendizaje supervisado y *K-means* para aprendizaje no supervisado.

3.1 K-Means

K-Means [1] es un algoritmo no supervisado de agrupamiento (*clustering*) cuya finalidad es asignar una etiqueta (clase) a cada dato de entrada $x \in X$ dependiendo del grupo (*cluster*) donde se posicione. Para un conjunto de *clusters* C, el agrupamiento de cada dato x en cada *cluster* $c \in C$ se obtiene a partir del centroide más cercano. El centroide m_c de un *cluster* c se define como el punto en el espacio de representación que corresponde a la media del conjunto de datos agrupados en c.

Se asume una función criterio J para evaluar la calidad de cualquier partición de N datos en C *clústers*:

$$J(\Pi) : \Pi = \{X_1, ..., X_C\}$$

El problema de *clustering* se aproxima como:

$$\Pi^* = \underset{\Pi=\{X_1,...,X_C\}}{\arg\min} \; J(\Pi)$$

La SEC de una particion $\Pi = \{X_1, ..., X_C\}$ se define como:

$$J(\Pi) = \sum_{c=1}^{C} J_c$$

Donde J_c es la distorsión del *cluster* c:

$$J_c = \sum_{x \in X_c} \|x - m_c\|^2$$

Siendo m_c la media o centroide del *cluster* c y n_c su talla:

$$m_c = \frac{1}{n_c} \sum_{x \in X_c} x$$

$\Rightarrow$ Algoritmo *K-means* (Duda y Hart):

Dada una particion $\Pi = \{X_1, ..., X_C\}$, el incremento de la SEC debido a la transferencia de un dato x del *cluster* i al j es:

$$\Delta J = \frac{n_j}{n_j + 1} \|x - m_j\|^2 - \frac{n_i}{n_i - 1} \|x - m_i\|^2$$

La transferencia es provechosa si $\Delta J < 0$, que se cumple cuando:

$$\frac{n_j}{n_j + 1} \|x - m_j\|^2 < \frac{n_i}{n_i - 1} \|x - m_i\|^2$$

El algoritmo se muestra en la Fig. 3.1.

Algorithm 1 Algoritmo *K-means* (Duda y Hart)

Require: $\Pi = \{X_1, ..., X_C\}$
Ensure: $\Pi^* = \{X_1, ..., X_C\}$
1: **while** $\Delta J < 0$ **do**
2: **for** $x \in X$ **do**
3: Sea i el *cluster* en el que se encuentra x
4: Hallar un $j^* \neq i$ que minimice ΔJ al transferir x de i a j^*
5: **if** $\Delta J < 0$ **then**
6: transferir x de i a j^* y actualizar medias y J
7: **end if**
8: **end for**
9: **end while**

Figura 3.1: Algoritmo K-Means (Duda y Hart).

3.2 K-Nearest-Neighbours

K-Nearest-Neighbours [1] es un algoritmo de clasificación basado en la proximidad de los datos. Los datos de entrada son supervisados donde para clasificar un dato de entrada $x \in X$ se tiene en cuenta las etiquetas (clases) de sus k datos más cercanos, siendo k un parámetro a especificar. La etiqueta con la que se clasifica a x es la etiqueta más frecuente de sus k datos más cercanos, en el caso de existir empates un buen criterio es escoger del conjunto de etiquetas más frecuentes la etiqueta del dato más cercano. El algoritmo se muestra en la Fig. 3.2

Algorithm 2 Algoritmo *K-Nearest-Neighbours*

Require: Datos de entrada $S = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ y un nuevo dato x' a clasificar
Ensure: Clase a la que pertenece x'
1: **for** Para todo dato de entrada $(x_i, y_i) \in S$ ya clasificado **do**
2: Calcular $d_i = distancia(x_i, x')$
3: **end for**
4: Ordenar de menor a mayor d_i para $i = 1, ..., N$.
5: Utilizar las K etiquetas de los datos ms cercanos a x'.
6: Asignar la etiqueta (clase) más frecuente, del conjunto de etiquetas K, al dato x'.

Figura 3.2: Algoritmo K-Nearest-Neighbours.

Otro parámetro de gran importancia en este algoritmo es la distancia que se utiliza para calcular la proximidad entre los datos. Los datos se representan como puntos en un espacio de representación, por lo tanto se pueden definir diferentes distancias para calcular la proximidad. Ejemplo de distancias que se utilizan en este algoritmo son la distancia Euclídea, Manhattan, Chebyshev, Mahalanobis, etc.

Cuando $k = 1$ cada dato contribuye directamente sobre la frontera de clasificación creando fronteras de clasificación lineales definidas a trozos como se muestra en la Fig. 3.3.

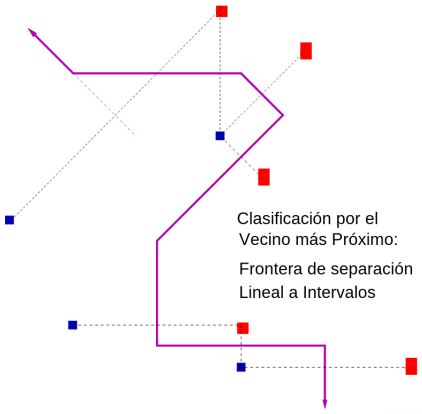

Figura 3.3: Frontera de clasificación lineal definida a trozos por el algoritmo K-Nearest-Neighbours con k = 1.

3.3 SVM

Las máquinas vector soporte o *support vector machines (SVM)* [6] son un tipo de clasificador basado en funciones discriminantes lineales (ϕ) que buscan obtener un máximo margen de separación entre clases.

$$\phi : \mathbb{R}^d \rightarrow \mathbb{R} : \phi(\boldsymbol{x}; \boldsymbol{\Theta}) = \boldsymbol{\theta}^t \boldsymbol{x} + \theta_0 = \sum_{i=1}^{d} \theta_i x_i + \theta_0$$

Donde $\boldsymbol{\Theta} = (\boldsymbol{\theta}, \theta_0)$ siendo $\boldsymbol{\theta} \in \mathbb{R}^d$ un vector de pesos y $\theta_0 \in \mathbb{R}$ un umbral.
Se define un clasificador (f) en dos clases: +1 y -1, y un vector de características $\boldsymbol{x} \in \mathbb{R}^d$ que representa un objeto a clasificar.

$$f(x) = \begin{cases} +1 & \text{si } \phi(\boldsymbol{x}, \boldsymbol{\Theta}) \geq 0 \\ -1 & \text{si } \phi(\boldsymbol{x}, \boldsymbol{\Theta}) < 0 \end{cases}$$

Una función discriminante lineal define el hiperplano de decisión H.

$$H = \{\boldsymbol{x} | \phi(\boldsymbol{x}, \boldsymbol{\Theta}) = 0\}$$

Para un hiperplano H dado, hay multiples posibilidades de definirlo mediante diferentes funciones discriminantes lineales (ϕ), lo que buscan las *SVM* es obtener una función discriminante lineal que maximice la separación entre ambas clases. Si definimos el margen del hiperplano separador H respecto al conjunto de muestras de aprendizaje S como $2r^* = \frac{2}{\|\theta\|}$ siendo r^* la distancia del vector $x^* \in S$ más próximo al hiperplano separador el objetivo de aprendizaje se basa en resolver un problema de maximización del margen $\frac{2}{\|\theta\|}$.

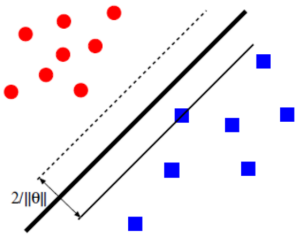

Figura 3.4: Hiperplano que máximiza la distancia entre ambas clases.

El margen optimo que buscamos esta sujeto a ciertas restricciones de canocidad lo que genera la utilización de multiplicadores de *Lagrange* que nos permiten obtener una optimización con restricciones. Los multiplicadores de *Lagrange* nos proporcionan como solución al problema de máximo margen una función discriminante lineal $\phi_{SVM}(\boldsymbol{x};\boldsymbol{\Theta})$ donde $\boldsymbol{\Theta} = (\boldsymbol{\theta}^*, \theta_0^*)$ siendo $\boldsymbol{\theta}^*$ el vector de pesos óptimo y θ_0^* el umbral óptimo obtenidos.

$$\phi_{SVM}(\boldsymbol{x},\boldsymbol{\Theta}) = \sum_{n\in\nu} \alpha_n^* c_n \boldsymbol{x_n^t x} + \theta_0^*$$

Donde ν es el conjunto de multiplicadores de *Lagrange* y α_n^* es el vector soporte n-ésimo.

Los *SVM* introducen dos alternativas para solucionar los problemas generados cuando las muestras no son linealmente separables. La primera solución es la definición de un parámetro C que permite deteminar el nivel de tolerancia que tiene el sistema ante las muestras mal clasificadas. La segunda opción es utilizar un *kernel* que transforme las muestras a un espacio alternativo con la finalidad de que sean linealmente separables.

$$\phi_k(\boldsymbol{x},\boldsymbol{\Theta}) = \sum_{n\in\nu} \alpha_n^* c_n k(\boldsymbol{x_n},\boldsymbol{x}) + \theta_0^*$$

Siendo la función $k(\boldsymbol{x_n},\boldsymbol{x})$ la función que define el *kernel*.

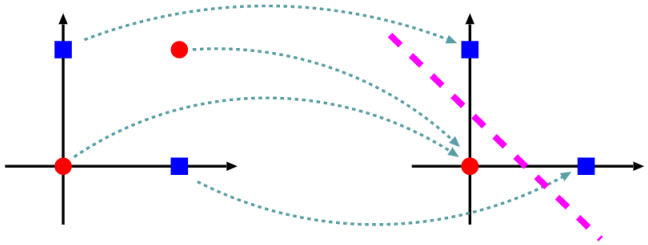

Figura 3.5: Ejemplo visual de la transformación aplicada a las muestras para que sean linealmente separables en el nuevo espacio de representación.

CAPÍTULO 4
WordNet

4.1 Definición

WordNet [7] es una base de datos léxica (creada y mantenida por el *Cognitive Science Laboratory* de la Universidad de *Princeton*) cuya finalidad es proporcionar información acerca de los sentidos (*synsets*) que tienen las palabras. La base de datos de WordNet almacena información unicamente para el idioma Inglés pero existen variantes para otros idiomas, se nombran algunas de estas variantes a continuación:

- WOLF: Versión francesa.
- JAWS: Versión francesa basada en Wiktionary y espacios semánticos.
- BalkaNet: WordNet para seis lenguas europeas (búlgaro, checo, griego, rumano, turco y serbio).
- FinnWordNet: Versión finlandesa.
- GermaNet: Versión alemana.
- EuroWordNet: Proyecto de WordNet para lenguas europeas conectadas entre sí.

La base de datos de WordNet es el léxicon computacional más usado en tareas de Procesamiento de Lenguaje Natural (PLN) principalmente cuando se trabaja con problemas de desambiguación del significado de palabras (*Word Sense Desambiguation (WSD)*).

La licencia con la que se ha liberado tanto la base de datos de WordNet como sus correspondientes herramientas permite que sean descargadas y usadas libremente (Licencia BSD).

4.2 Estructura

WordNet tiene organizadas las palabras de acuerdo a sus categorías gramaticales. Las categorías son: nombres, verbos, adjetivos y adverbios. Para cada palabra de la base de datos de Wordnet se recupera información sobre todos los sentidos que puede tener dicha palabra. Para cada sentido se puede consultar una definición del mismo y un conjunto de ejemplos.

La mayoría de sentidos están conectados a otros sentidos mediante relaciones semánticas:

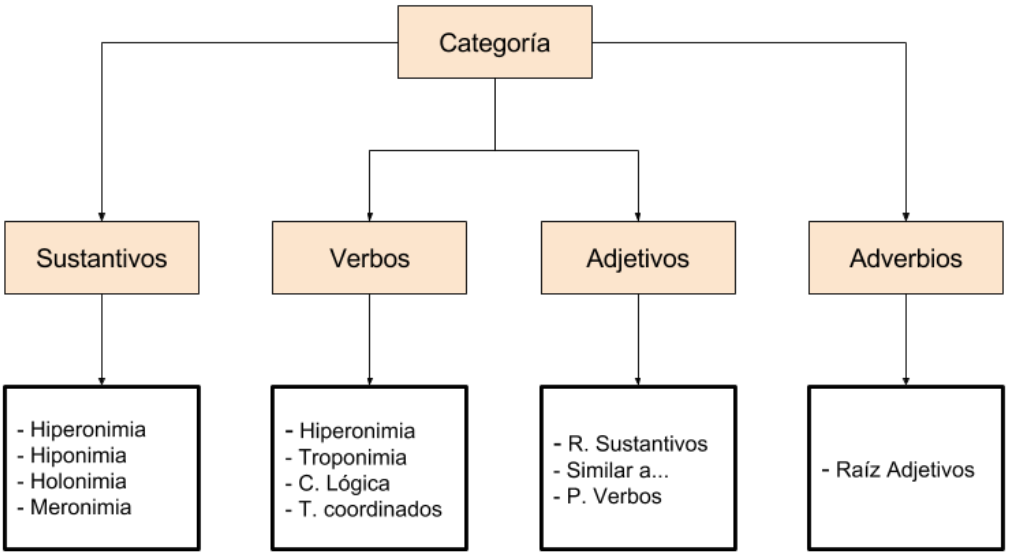

Figura 4.1: Conexión de sentidos en WordNet por categoría gramatical.

- Hiperonimia: X es un hiperónimo de Y si X engloba a Y.
  Ejemplo: Vehículo es un hiperónimo de coche.
- Hiponimia: X es un hipónimo de Y si Y engloba a X.
  Ejemplo: Coche es un hipónimo de vehículo.
- Holonimia: X es un holónimo de Y si Y es una parte de X.
  Ejemplo: Coche es un holónimo de rueda.
- Meronimia: X es un merónimo de Y si X es una parte de Y.
  Ejemplo: Rueda es un merónimo de coche
- Troponimia: El verbo X es un tropónimo del verbo Y si la acción de Y la realiza X de alguna forma.
  Ejemplo: Balbucear es un tropónimo de hablar.
- Consecuencia Lógica (C. Lógica): El verbo X es una consecuencia lógica del verbo Y si haciendo X se debe estar haciendo Y.
  Ejemplo: Pasear es una consecuencia lógica de caminar.
- Términos Coordinados (T. coordinados): Dos verbos son términos coordinados si ambos comparten un hiperónimo común.
  Ejemplo: Susurrar y gritar son verbos coordinados.
- Sustantivos relacionados (R. Sustantivos): Sustantivos que guardan relación con dicho adjetivo.
- Adjetivos similares (Similar a...).
- Participios de verbos (P. Verbos): El participio es la forma no conjugable de un verbo que desempeña generalmente la función de adjetivo.

La relación léxica principal entre palabras en WordNet es la sinonimia aunque también se pueden encontrar palabras relacionadas por antonimia. Dos palabras son sinónimas si tienen el mismo significado (bonito, hermoso), por otra parte, dos palabras son antónimas si tienen significados opuestos (luz, oscuridad).

WordNet trabaja a nivel de sentidos (*synsets*) de palabras. Como entrada a WordNet se le proporciona una palabra y devuelve como salida el conjunto de *synsets* que puede tener

dicha palabra. La Fig. 4.2 muestra el conjunto de *synsets* que proporciona el buscador de WordNet 3.1 para la palabra *dog*.

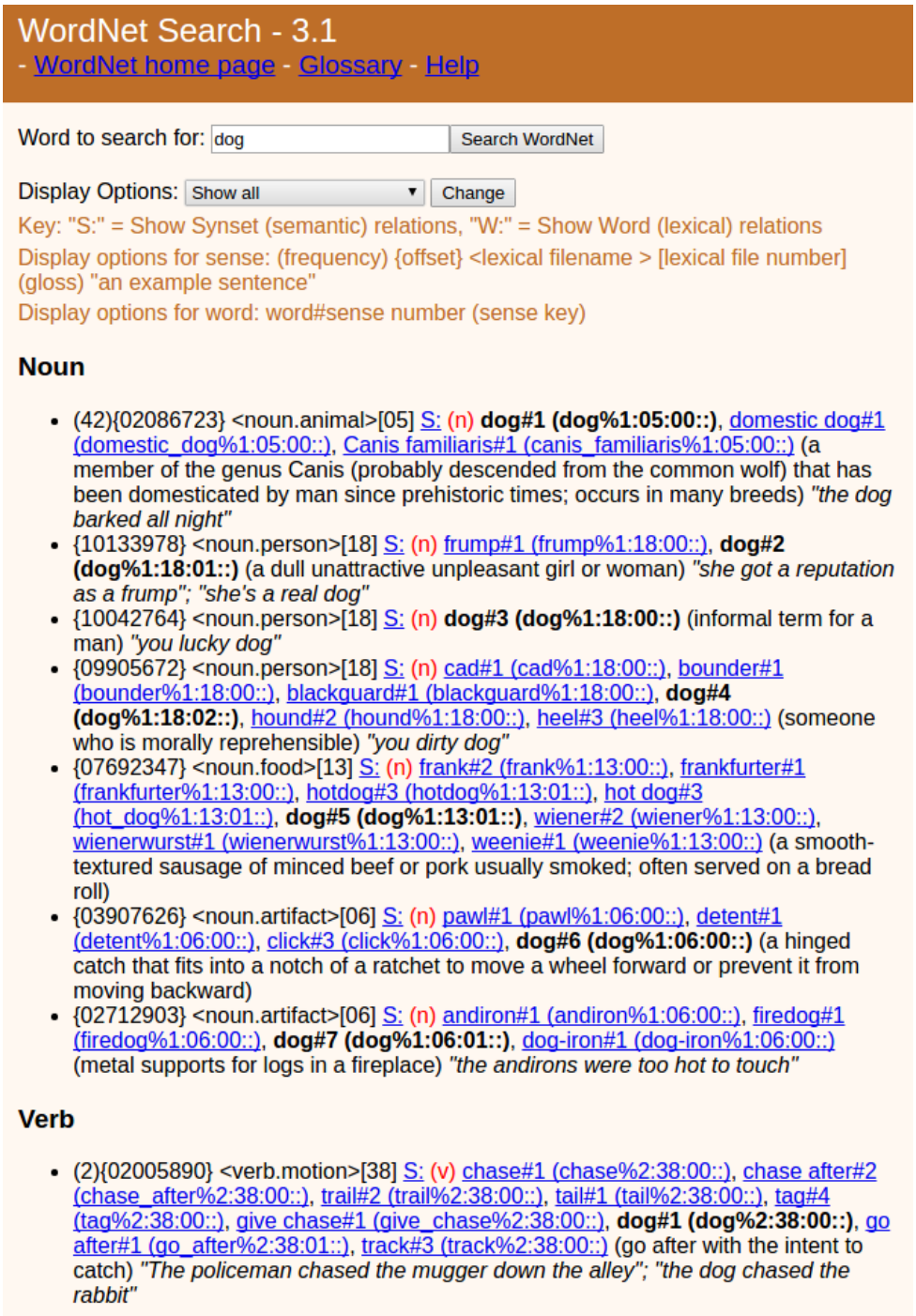

Figura 4.2: *Synsets* que proporciona WordNet para la palabra *dog*

Para la palabra *dog* WordNet devuelve siete sentidos que corresponden a la categoría gramatical «nombre» y un sentido que corresponde a la categoría gramatical «verbo». Basándonos en el primer *synset* obtenido para la palabra *dog* explicamos a continuación la información que proporciona WordNet:

- (42): Es el número de veces (frecuencia) que la palabra *dog* tiene dicho sentido. Es un valor aproximado porque WordNet calcula la frecuencia a partir de un corpus de textos propio.

- {02086723}: Es una codificación única que tiene WordNet para cada *synset*.
- <noun.animal>[05]: Nos aporta información léxica del sentido. Indica que la palabra *dog* es un nombre y que es un hipónimo de *animal*. [05] es el código interno que utiliza WordNet para referencia a «noun.animal».
- S: Es un enlace que nos permite preguntar a WordNet sobre hiperónimos, hipónimos, holónimos y merónimos para la palabra *dog*.
- (n): Nos indica que la categoría gramatical del sentido es un nombre.
- dog#1: Es una enumeración interna de WordNet para enumerar los sentidos que correponden a la categoría gramatical nombre para la palabra *dog*.
- (dog %1:05:00::): Es una clave (*key*) única que tiene WordNet para cada *synset*.
- domestic dog#1 (domestic_dog %1:05:00::): Es el primero de los dos sinónimos que contiene WordNet para el *synset*.
- Canis familiaris#1 (canis_familiaris %1:05:00::): Es el segundo sinónimo que contiene WordNet para el *synset*.
- Definición del *synset*: *(a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds)*
- Ejemplos del *synset*: "the dog barked all night".

4.3 Aplicaciones y restricciones

WordNet es una herramienta de gran utilidad en sistemas de PLN. A continuación presentamos algunos ejemplos de aplicaciones que requieren el uso de análisis semántico utilizando herramientas como WordNet:

- **Traducción automática**: Las traducciones entre dos idiomas no siempre se pueden resolver con análisis sintáctico ya que existen traducciones que no son literales.
- **Recuperación de información**: Los sistemas de recuperación de información necesitan una entrada con la consulta a realizar, que el sistema sea capaz de comprender la consulta mejora la calidad respecto a la interacción máquina-persona.
- **Detección de plagio**: Cuando un texto dice lo mismo que otro pero solo puede ser detectado al comprender el significado de los mismos.
- **Sistemas de reconocimiento**: Los sistemas de reconocimiento del habla (o escritura) utilizan un modelo de lenguaje para que la transcripción resultante cumpla las restricciones sintácticas del lenguaje. Si somos capaces de obtener información semántica a la hora de obtener las transcripciones los sistemas de reconocimiento mejoran considerablemente.
- **Clasificación automática de texto**: Si el sistema es capaz de comprender la temática del texto la clasificación realizada obtiene mejores resultados.

WordNet no contiene todas las palabras del idioma Inglés en su base de datos, aún así la base de datos se actualiza constantemente y el número de palabras almacenadas es muy grande siendo una herramienta de gran relevancia en PLN.

CAPÍTULO 5
Competición

En el presente capítulo se presenta la competición de SemEval con la finalidad de dar a conocer la utilidad que aporta este tipo de competiciones en PLN. SemEval-2017 esta constituida por un conjunto de tareas que tienen gran interés actualmente en la comunidad científica. De forma más detallada se presenta la tarea 7: Detección e interpretación de *puns* en el idioma inglés que corresponde al conjunto de problemas que se van a intentar solucionar en el presente trabajo. Cuando se trabaja con *WSD* la evaluación de los sistemas no es trivial, por este motivo en este apartado se describen las métricas que ha propuesto la competición para poder establecer una comparación de nuestros resultados respecto al resto de participantes.

5.1 SemEval-2017

El nombre de SemEval es una combinación entre los terminos en inglés *Semantic* y *Evaluation* y consiste en una serie de evaluaciones de sistemas computacionales de análisis semántico. Las evaluaciones corresponden a la resolución de tareas planteadas por la organización *Special Interest Group on the Lexicon of the Association for Computational Linguistics (SIGLEX)*.

El objetivo principal de SemEval es explorar la naturaleza del sentido en el lenguaje a causa de la dificultad que tienen las máquinas en realizar un análisis semántico sobre un texto. Las tareas que se han propuesto en SemEval-2017 se dividen en tres grupos:

Semantic comparison for words and texts

- Task 1: Semantic Textual Similarity
- Task 2: Multilingual and Cross-lingual Semantic Word Similarity
- Task 3: Community Question Answering

Detecting sentiment, humor and truth

- Task 4: Sentiment Analysis in Twitter
- Task 5: Fine-Grained Sentiment Analysis on Financial Microblogs and News
- Task 6: HashtagWars: Learning a Sense of Humor
- Task 7: Detection and Interpretation of English Puns
- Task 8: RumourEval: Determining rumour veracity and support for rumours

Parsing semantic structures

- Task 9: Abstract Meaning Representation Parsing and Generation
- Task 10: Extracting Keyphrases and Relations from Scientific Publications
- Task 11: End-User Development using Natural Language
- Task 12: Clinical TempEval

5.2 Detección e interpretación de *puns* en el idioma inglés

Un *pun* se define como una forma de juego de palabras que tiene dos o más significados explotando la polisemia, homonimia o similitud fonética de otra palabra con la intención de generar un efecto humorístico o retórico.

Los *puns* se dividen en dos categorías:

- Homofónicos (perfectos): Los dos significados que tiene el *pun* comparten la misma pronunciación. Ejemplo de *pun* perfecto:

  I used to be a banker but I lost the ***interest***

  Se observa que el *pun* en esta oración es la palabra *interest* que tiene el sentido de «valor o utilidad que en sí tiene una persona» y al mismo tiempo también tiene el sentido de «cantidad que se paga por un préstamo». En esta oración la palabra *banker* es la palabra del contexto que más información nos aporta acerca de la desambiguación del *pun*.
- Heterofónicos (imperfectos): El *pun* corresponde a una entidad que representa a otra entidad fonológicamente similar que recibe el nombre de objetivo, en inglés *target*. Ejemplo de *pun* imperfecto :

  The lobe was the original ***earring*** *aid.*

  En este caso la palabra que corresponde al *pun* es *earring* que sustituye a la palabra objetivo (*target*) *hearing*. La palabra más importante del contexto para detectar el *pun* en la oración es *lobe*. El doble sentido en esta oración corresponde al significado que tiene la palabra lóbulo en la oración que hace referencia a la «utilización del lóbulo como audífono» o como «soporte para un pendiente en la anatomía de los seres humanos».

La detección e interpretación de las dos categorías es una tarea compleja existiendo un mayor grado de complejidad en los *puns* imperfectos. En los *puns* perfectos sabemos que la palabra que corresponde al *pun* se encuentra en la oración convirtiendose en un problema de *WSD* tradicional con el detalle de tener que obtener de forma automática dos sentidos para la palabra en vez de uno, esto implica mayor complejidad respecto al problema de *WSD* tradicional. Para los *puns* imperfectos aparte de las dificultades que encontramos en el análisis de *puns* perfectos se añade la dificultad de encontrar una palabra que tenga una similitud fonológica similar al *pun* cuyo significado guarde también una relación directa con el contexto del *pun*.

En el caso de trabajar con texto en vez de con pronunciaciones de las palabras las dos categorías reciben el nombre de *puns* homográficos y *puns* heterográficos que son los términos adecuados para referirnos a las categorías de *puns* en esta tarea.

La organización de la tarea proporciona un conjunto de frases que contienen *puns* homográficos y otro conjunto de frases que contienen *puns* heterográficos. Esta separación convierte el problema en dos problemas independientes facilitando un poco las cosas debido a que si tuviéramos los tipos de *puns* mezclados necesitaríamos saber diferenciarlos.
La tarea a su vez se divide en tres subtareas que se describen a continuación:

Detección: Dada una frase que puede contener *pun* o no, la tarea propone detectar si la frase contiene *pun* o no. Se puede abordar como un problema de clasificación en dos clases, tiene *pun* (1), no tiene *pun* (0).

Localización: Dada una frase que contiene *pun*, la tarea propone localizar la palabra en la frase que corresponde al *pun*.

Interpretación: Dada una frase que contiene *pun* y conociendo a priori qué palabra es el *pun* en la frase, la tarea propone obtener los dos sentidos que tiene el *pun* en la frase.

5.3 Medidas de evaluación

La evaluación de un sistema nos permite obtener información acerca de cómo de buena es la solución propuesta para resolver el problema, también permite comparar diferentes soluciones al mismo problema para conocer qué solución es mejor. SemEval-2017 propone métricas de evaluación para cada subtarea de la tarea 7: Detección e interpretación de *puns* en el idioma inglés que explicamos en este apartado.

Detección:

Como se ha comentado anteriormente la tarea de detección se puede estudiar como un problema de clasificación en dos clases donde una clase indica que la frase contiene *pun* (1) y la otra clase que no contiene *pun* (0).

Es necesario conocer una serie de términos que nos permiten entender las métricas para esta tarea. Estos términos hacen referencia a lo que se conoce como verdaderos positivos (TP), falsos positivos (FP), verdaderos negativos (TN) y falsos negativos (FN).

- TP: Si la frase contiene *pun* (1) y la respuesta de nuestro sistema es que contiene *pun* (1).
- FP: Si la frase no contiene *pun* (0) y la respuesta de nuestro sistema indica que contiene *pun* (1).
- TN: Si la frase no contiene *pun* (0) y la respuesta de nuestro sistema indica que no contiene *pun* (0).
- FN: Si la frase contiene *pun* (1) y la respuesta de nuestro sistema indica que no contiene *pun* (0).

Se define *Precision* (P) como el cociente entre el número de aciertos que tiene nuestro sistema respecto a las frases que tienen *pun* (TP) y el número total de frases que nuestro sistema clasifica como *puns* (TP + FP).

$$P = \frac{TP}{TP + FP}$$

Se define *Recall* (R) como el cociente entre el número de aciertos que tiene nuestro sistema respecto a las frases que tienen *pun* (TP) y el número de frases que tienen *pun* en

la subtarea (TP + FN).

$$R = \frac{TP}{TP + FN}$$

Se define *Accuracy* (A) como el cociente entre el número de frases que nuestro sistema clasifica correctamente (TP + TN) y el número total de frases que resuelve nuestro sistema (TP + TN + FP + FN).

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

Se define *F-score* (F_β) como una media armónica que combina los valores de *precision* y *recall*. Si $\beta = 1$ se le da la misma importancia a la *precision* que al *recall*. F_1 es la medida más utilizada.

$$F_1 = \frac{2PR}{P + R}$$

Localización:

En la tarea de localización trabajamos únicamente con frases que contienen *puns*, nuestro sistema devuelve un resultado correcto cuando acierta la palabra que es *pun* en la frase y retorna un resultado erróneo en caso contrario.

Se define *Coverage* (C) como el cociente entre el número total de frases que resuelve nuestro sistema (# *of guesses*) y el número total de frases que tiene la tarea (# *of contexts*).

$$C = \frac{\text{\# of guesses}}{\text{\# of contexts}}$$

Se define *Precision* (P) como el cociente entre el número de aciertos que devuelve nuestro sistema (# *of correct guesses*) y el número total de frases que resuelve nuestro sistema (# *of guesses*).

$$P = \frac{\text{\# of correct guesses}}{\text{\# of guesses}}$$

Se define *Recall* (R) como el cociente entre el número de aciertos que devuelve nuestro sistema (# *of correct guesses*) y el número total de frases que tiene la tarea (# *of contexts*).

$$R = \frac{\text{\# of correct guesses}}{\text{\# of contexts}}$$

Se define F_1-*score* de la misma forma que en la subtarea de detección.

$$F_1 = \frac{2PR}{P + R}$$

Interpretación:

En la tarea de interpretación todas las frases tienen *pun* y conocemos qué palabra de la frase corresponde al *pun*. Cuando nuestro sistema devuelve dos sentidos correspondientes al *pun* la evaluación considera una solución correcta si los dos sentidos son correctos y errónea en caso contrario.

Para que ambos sentidos sean correctos se tiene que cumplir lo siguiente:

$$(\{sentido1\} \cap \{sentidos1\_gold\} \neq \emptyset \wedge \{sentido2\} \cap \{sentidos2\_gold\} \neq \emptyset)$$

$$\vee$$

$$(\{sentido1\} \cap \{sentidos2\_gold\} \neq \emptyset \wedge \{sentido2\} \cap \{sentidos1\_gold\} \neq \emptyset)$$

Se tiene que cumplir que el sentido1 aparezca en el conjunto de sentidos válidos para el primer sentido del *pun* y el sentido2 aparezca en el conjunto de sentidos válidos para el segundo sentido del *pun* o también es una solución válida si el sentido1 aparece en el conjunto de sentidos válidos para el segundo sentido del *pun* y el sentido2 aparezca en el conjunto de sentidos válidos para el primer sentido del *pun*. Donde sentido1 y sentido2 hacen referencia al primer y segundo sentido que devuelve nuestro sistema, sentidos1_gold hace referencia al conjunto de sentidos validos que proporciona la organización para el primer sentido del *pun* y sentidos2_gold corresponde al conjunto de sentidos validos que proporciona la organización para el segundo sentido del *pun*. Observamos que no existe importancia acerca del orden de los sentidos que se obtienen como solución.

Las métricas de evaluación en esta subtarea son *Coverage, Precision, Recall* y F_1-*score* y se calculan de la misma forma que en la tarea de localización.

5.4 Baseline

Los soluciones propuestas como *baselines* se proporcionan en una competición con la finalidad de otorgar a los participantes una referencia acerca de los resultados mínimos que deben obtener para resolver la tarea propuesta. En este apartado se van a explicar los *baselines* que ha propuesto SemEval-2017 sobre cada una de las subtareas de la *tarea 7: Detección e interpretación de puns en el idioma ingles*.

Detección:

Para la tarea de detección de *puns* solo se presenta un *baseline* que se basa en un clasificador aleatorio (*random*) que tiene la misma probabilidad para clasificar una frase que contiene *pun* (50 %) como una frase que no contiene *pun* (50 %).

Localización:

En la tarea de localización se presentan tres *baselines*. El primer *baseline* simplemente selecciona de forma aleatoria (*random*) una palabra de la frase y es la que devuelve como *pun*. El segundo *baseline* siempre devuelve como *pun* la ultima palabra de la frase (*last word*). El tercer *baseline* es un poco más sofisticado, se selecciona como *pun* la palabra que tenga la polisemia más alta (*max. polysemy*). Para determinar la polisemia de una palabra se cuentan todos los sentidos que tenga, en el caso de que haya empates en la frase se elige la palabra más polisémica que se encuentre más cercana al final de la frase.

Interpretación:

El primer *baseline* consiste en escoger dos sentidos del *pun* de forma aleatoria (*random*), la obtención de la *precision* y *recall* se muestra a continuación:

$$P = R = \frac{1}{n} \sum_{i=1}^{n} \frac{G_1^i \cdot G_2^i}{S^i}$$

Donde n es el número de frases, G_1^i es el número de *keys* que tiene el primer sentido del *pun* que buscamos (*gold*) en la frase i, G_2^i es el número de *keys* que tiene el segundo sentido del *pun* que buscamos (*gold*) en la frase i y S^i es el número de keys que tienen todos los sentidos del *pun* en la frase i. El número de *keys* corresponde al número de sinónimos que tiene el sentido. Este *baseline* (random) únicamente se aplica al corpus que contiene *puns* homográficos debido que el gran número de palabra objetivo (*target*) en el corpus con *puns* heterográficos hacen que los resultados sean insignificantes.

El segundo *baseline* se basa en utilizar el sentido más frecuente (MFS) a partir de un corpus sobre el cual se consultan la frecuencia de los sentidos. Para el caso de *puns* homográficos se lematiza el *pun* y posteriormente se selecciona los dos sentidos más frecuentes que proporciona WordNet sobre dichos lemas. Para los *puns* heterográficos únicamente el primer sentido se selecciona usando el criterio para los *puns* homográficos. Una segunda lista de lemas es construida para encontrar aquellos lemas que tengan menor distancia respecto a los lemas de la primera lista. El sentido más frecuente de los lemas en la segunda lista se selecciona como el segundo significado del *pun*.

El tercer *baseline* es un sistema que puntúa los sentidos del *pun* dependiendo de la intersección léxica que se obtiene mediante la información del contexto en la frase. Los dos sentidos con mayor puntuación son los seleccionados, en el caso de que los dos sentidos sean iguales se busca un sentido con puntuación alta que guarde una distancia semántica respecto al primer sentido.

CAPÍTULO 6
Corpus

Existen un conjunto de corpus para trabajar con *puns* en el idioma inglés nombrados en [13] que han sido creados por lingüistas y por personas que se dedican a ciencias sociales donde las tareas de interés se centran en la generación y detección de humor de forma automática. Los corpus mencionados no son completamente accesibles lo que genera que no se disponga de un corpus amplio para poder abordar la tarea. Además todos los corpus descritos no contienen ninguna anotación semántica (*synsets*).

La importancia de trabajar con *puns* en la comunidad científica es amplia, por este motivo, SemEval-2017 decide generar un corpus propio que sea de libre acceso, etiquetando los puns en las frases de forma manual, con la ayuda de un software propio denominado *Punnotator* [13], tanto para el *dataset* de *puns* homográficos como para el *dataset* de *puns* heterográficos.

El objetivo de SemEval-2017 era recolectar aproximadamente 2000 *puns* debido a que este número de casos es típico en competiciones de *WSD* como SensEval y SemEval en años anteriores. Es de gran importancia que un corpus cumpla con una serie de restricciones con la finalidad de ser robusto, consistente y con una especificación detallada. Por este motivo los *puns* del corpus generado por el SemEval-2017 cumplen los siguientes requisitos:

- El conjunto de oraciones que contienen *pun*, contienen como máximo un *pun*.
- Un *pun* formado por una única palabra contiene únicamente una categoría gramatical (nombre, verbo, adjetivo o adverbio). Para los *puns* formados por más de una palabra se mantienen aquellos que son verbos compuestos basándose en el criterio de que todas las palabras que componen el *pun* tengan la misma categoría gramatical.

  ⇒ *State of the art*: No es aceptado.
  ⇒ *Get off the ground*: No es aceptado.
  ⇒ *Take off*: Si es aceptado.
- Un *pun* debe tener exactamente dos significados.
- Todos los *puns* tienen una entrada en WordNet 3.1, pero no todos los *puns* tienen sentidos en WordNet 3.1.
- Los intereses de investigación en la tarea de SemEval-2017 se centran más en la semántica léxica que en la fonología para el estudio de *puns* heterográficos. Con la finalidad de facilitar la anotación manual y mantener un buen número de casos morfológicamente interesante se aplica la restricción basada en que las unidades léxicas correspondientes a los dos significados distintos deben deletrearse exactamente de la misma manera.

Es importante conocer la procedencia del conjunto de *puns* ya que nos aporta información acerca de la temática a la que pertenecen, si son similares o distintos entre sí y si son más complejos o más simples a nivel semántico. Las frases del corpus que contienen *pun* provienen de colecciones obtenidas a partir de humoristas profesionales (Stan Kegel, entre otros) y mediante páginas Web como *Pun of the Day*. Eliminados duplicados, el tamaño del corpus inicial, formado por frases que contienen *pun*, es de 7750 frases de las cuales se realiza un filtrado de forma manual exigiendo que todos los *puns* cumplan los requisitos anteriormente mencionados.

Respecto al conjunto de frases que no contienen *pun*, utilizadas en la tarea de detección, se obtienen a partir del trabajo de [9] reuniendo un total de 2972 proverbios y refranes famosos de varias páginas Web. Posteriormente se han filtrado aquellos que contengan juegos de palabras. La Tabla 6.1 muestra las características finales del corpus de la competición.

Subtarea	**Tipo (*pun*)**	**Nº Frases**
Detección	Homográfico	2250
	Heterográfico	1780
Localización	Homográfico	1607
	Heterográfico	1271
Interpretación	Homográfico	1298
	Heterográfico	1098

Tabla 6.1: Número de frases para cada subtarea de la tarea 7 de SemEval-2017

Además de conocer el tamaño que tiene el corpus en cada tarea es importante obtener información acerca de la longitud de las frases del corpus de SemEval-2017. Esta información se muestra en la Tabla 6.2.

			Nº Palabras / Frase		
Subtarea	**Tipo {*pun*}**	**Nº Frases**	**min**	**media**	**max**
Detección	Homográfico	2250	2	10.9	44
	Heterográfico	1780	2	10.9	69
Localización	Homográfico	1607	3	11.8	44
	Heterográfico	1271	3	11.9	69
Interpretación	Homográfico	1298	3	11.9	44
	Heterográfico	1098	3	12.1	69

Tabla 6.2: Número de palabras por frase para cada subtarea de la tarea 7 de SemEval-2017

La anotación manual del corpus ha sido realizada por dos jueces expertos mediante la aplicación *Punnotator* [13]. De un total de 1652 casos positivos de *pun* el porcentaje de acuerdo fue del 98.91 % (1634 casos) al seleccionar la palabra como *pun* en el texto. De estos 1634 casos se hizo un estudio acerca de la anotación de sus respectivos sentidos utilizando una métrica de corrección de probabilidad [13] obteniendo un resultado mucho más alto de lo que se ha informado en otros estudios de anotación de sentidos.

Para los casos en los que los sentidos anotados eran diferentes o contradictorios (también para la localización del *pun* en el texto) una tercera persona intentaba decidir entre la anotación de un juez o el otro, obteniendo en total 1607 casos positivos.

De los 1607 casos positivos los jueces fueron capaces de anotar ambos sentidos correctamente en 1298 casos (80.8 %) donde para los 303 restantes (18.9 %) WordNet no daba cobertura a uno de los dos sentidos, y para los últimos 6 casos restantes (0.4 %), WordNet no tenía cobertura para los dos sentidos.

SemEval-2017 proporciona un estudio acerca de cómo se distribuyen los sentidos de los *puns* a nivel de categoría gramatical. De los 2596 sentidos anotados, el 50.2 % corresponde a nombres, el 33.8 % corresponde a verbos, el 13.1 % corresponde a adjetivos y solo el 1.6 % corresponde a adverbios.

Un *pun* puede tener más de un sentido válido, todos los sentidos que se consideren válidos para el *pun* serán considerados correctos en la tarea de interpretación.

Por último, SemEval-2017 proporciona estadísticas sobre en qué posición de la frase se sitúa el *pun*. Todas las frases se han dividido en cuatro partes, siendo la primera parte el inicio de la frase y la cuarta parte el final de la frase. En la cuarta parte de la frase el *pun* aparece un 82.8 %, en la tercera parte aparece un 9.3 %, en la segunda parte aparece un 6.7 % y en la primera parte un 1.2 %.

CAPÍTULO 7
Aproximaciones al problema

Los únicos estudios que conocemos previos sobre la detección y comprensión computacional de juegos de palabras (*puns*) se centran en características fonológicas y sintácticas. [20] describe un sistema para detectar la presencia de juegos de palabras en el texto japonés, pero sólo funciona con juegos de palabras que son imperfectos y no gramaticales, basándose en anotaciones sintácticas y no en información léxico-semántica. [18], por un camino similar, describen un enfoque basado en N-gramas para reconocer cuándo se usan juegos de palabras imperfectos para el efecto humorístico en una clase muy pequeña de chistes ingleses. Su enfoque en juegos de palabras imperfectos y su uso de un contexto sintáctico fijo hace que su aproximación sea en gran medida inaplicable a los juegos de palabras arbitrarios dentro de un texto. Uno de los problemas que tienen estas aproximaciones es que son incapaces de asignar más de un significado al mismo objetivo. [9] defienden que el análisis semántico podría ayudar en la detección de la incongruencia humorística.

Actualmente los sistemas de detección de humor o ironía se centran en la realización de un análisis semántico que sea capaz de aportar información léxico-semántica al sistema para comprender el texto y de este modo interpretar el significado del mismo. Los algoritmos que realizan análisis semántico de un texto necesitan acceder a estructuras de datos como bases de datos, tesauros, diccionarios, etc. Cuando estas estructuras no pueden proporcionar el conocimiento que necesita el algoritmo no se debe considerar como una condición de fallo sino como una limitación en la tarea de desambiguación semántica.

Las aproximaciones al problema de *WSD* se centran en obtener el conjunto de sentidos que pueden tener todas las palabras de un texto y seleccionar para cada una el sentido que se considere más adecuado en ese contexto. Este problema es complejo y la dificultad aumenta cuando las palabras del texto pueden tener más de un sentido, como ocurre en los problemas abordados en este trabajo (*puns*). Una primera aproximación en la obtención de los dobles sentidos de un *pun* en un texto consiste en determinar a través del contexto los dos sentidos con mayor relación semántica, lo cual es erróneo debido a que guardan una relación muy directa y los dos sentidos que tiene un *pun* difieren proporcionando ese toque de humor que los caracteriza. Una mejor aproximación consiste en obtener una distancia semántica mínima entre los dos sentidos [4].

La competición de SemEval-2017 a través de la tarea 7: *Detección e interpretación de puns* nos proporciona una visión sobre las aproximaciones que existen actualmente para solucionar el problema de desambiguación de *puns* gracias a la participación y posterior publicación de los participantes de la misma. A continuación se resumen brevemente las soluciones propuestas por los participantes para las tres subtareas que propone la competición.

BuzzSaw (Oele and Evang, 2017)

BuzzSaw asume que los dos significados del *pun* tienen una alta similitud semántica con una única parte del contexto. Para ello propone dividir el texto en dos partes considerando todas las particiones posibles. A continuación, utiliza un algoritmo *WSD* que utiliza *embeddings* de palabras y sentidos para desambiguar el *pun* respecto a las palabras de la partición opuesta al *pun*.

Este enfoque también lo utiliza para localizar el *pun* en el texto. Aplica el sistema de interpretación para cada palabra polisémica en el contexto escogiendo como *pun* aquella palabra cuya representación en *embeddings* de sus dos sentidos tengan la máxima distancia coseno.

Duluth (Pedersen, 2017)

Para la tarea de detectar *pun*, Duluth asume que todos los sistemas *WSD* tendrán problemas para asignar más de un sentido al *pun*. Para ello utiliza el mismo algoritmo *WSD* pero con cuatro configuraciones diferentes, por lo tanto, si dos o más sentidos difieren entre las ejecuciones supone que el texto contiene *pun*.

Para localizar el *pun* el sistema selecciona la palabra cuyo sentido cambia en las ejecuciones. Si varias palabras cambian de sentido entonces el sistema escoge la palabra cuyo sentido sea más cercano al contexto del texto.

En la tarea de interpretación para *puns* homográficos se escogen aquellos dos sentidos más frecuentes al aplicar las diferentes configuraciones del algoritmo *WSD*. Para *puns* heterográficos el sistema intenta obtener la palabra objetivo (*target*) generando una lista de lemas de WordNet con una distancia de edición mínima al *pun* o bien utilizando Datamuse API para aquellas palabras con ortografía, pronunciación y significado similares.

ECNU (Xiu et al., 2017)

ECNU utiliza un enfoque supervisado para la detección de *puns*. Los autores recopilaron un conjunto de entrenamiento formado por 60 *puns* homográficos y 60 *puns* heterográficos además de 60 proverbios y refranes famosos de Internet. El corpus recopilado se utiliza para entrenar un clasificador utilizando características obtenidas mediante WordNet y *embeddings Word2Vec*.

En la tarea de localización ECNU utiliza un sistema basado en el conocimiento obteniendo qué probabilidad tiene cada palabra de ser *pun* basándose en la distancia entre los *embeddings* de los sentidos o entre los *embeddings* de los sentidos y los *embeddings* del contexto.

ELiRF-UPV (Hurtado et al., 2017)

Esta es la aportación de nuestro grupo (ELiRF-UPV) en la competición. Nos centramos en resolver los *puns* homográficos de las tareas de localización e interpretación. Para la tarea de localización proponemos dos hipótesis: 1) El *pun* nos lo proporciona el par de palabras no contiguas cuyos *embeddings* sean más similares. 2) Suponemos que el *pun* tiende a aparecer en las posiciones finales del texto, por lo tanto del par de palabras

con mayor similitud se escoge como *pun* la palabra situada en la posición más a la derecha de la frase.

En la tarea de interpretación obtenemos las dos palabras con mayor similitud respecto al *pun*. El primer sentido se obtiene de la palabra con mayor similitud respecto al *pun* y el segundo sentido con la segunda palabra con mayor similitud respecto al *pun*. Para determinar el mejor sentido de cada palabra respecto al *pun* se define una métrica de similitud basada en el contexto (definición y ejemplos) de los sentidos de WordNet. Nuestra aproximación se explica con más detalle en el Capítulo 9 del presente trabajo.

Fermi (Indurthi and Oota, 2017)

Fermi proporciona soluciones para las tareas de detección y localización de *puns* homográficos. En la tarea de detección Fermi propone una solución basada en aprendizaje supervisado. Del corpus (*dataset*) proporcionado por SemEval-2017 para esta tarea, realiza una partición cogiendo una parte para etiquetarla manualmente y entrenar una red neuronal bidireccional que funcione como un clasificador binario usando *embeddings* como características de entrada.

Para la tarea de localización propone una aproximación similar a la nuestra (ELiRF-UPV) utilizando la similitud entre pares de *embeddigns* de sentidos de palabras y basándose en que el *pun* suele encontrarse al final del texto.

Idiom Savant (Doogan et al., 2017)

Idiom Savant utiliza diferentes métodos dependiendo de la tarea a resolver y el tipo de *pun* de la misma. Generalmente todas ellas se basan en N-gramas de Google y *embeddings Word2Vec*. Para trabajar con *puns* heterográficos aplica una distancia fonética utilizando el diccionario de pronunciación de CMU. Su aproximación se centra en tratar especialmente los *puns* Tom Swifties, los cuales se conoce a priori que se encuentran en el corpus de la tarea [12].

JU_CSE_NLP (Pramanick and Das, 2017)

Propone soluciones para la tarea de detección y localización de *puns*. Utiliza aprendizaje supervisado con un *dataset* de 413 *puns* del proyecto Gutenberg etiquetado manualmente. Con el *dataset* entrena un modelo oculto de Markov (HMM) y una red de dependencia cíclica utilizando características extraídas a partir de un *part-of-speech tagger* (POS-tagger) y un *parser* sintáctico.

PunFields (Mikhalkova and Karyakin, 2017)

PunFields utiliza diferentes métodos para la detección, localización e interpretación de *puns* que comparten la propiedad de utilizar características a nivel semántico. El enfoque del sistema para la detección de *puns* se basa en aprendizaje supervisado utilizando características de los vectores que tabulan el número de palabras en el contexto que aparecen en cada una de las 34 secciones del Tesauro de Roget. PunFields utiliza un enfoque supervisado que clasifica a los candidatos en base a su presencia en las secciones de Roget, su posición dentro del contexto y su parte del discurso.

Para la tarea de interpretación del *pun*, el sistema particiona el contexto basándose en campos semánticos. Selecciona como el primer sentido del *pun* el sentido cuya definición en WordNet tiene el mayor número de palabras en común con la primera partición. En el

caso de *puns* homográficos, el segundo sentido seleccionado es el que tiene la frecuencias más alta en WordNet (o el siguiente con la frecuencias más alta, en caso de que el primer sentido seleccionado ya tenga la frecuencia más alta). Para los *puns* heterogéneos, utiliza una lista de palabras objetivo candidatas utilizando la distancia de Levenshtein. Entre sus correspondientes sentidos de WordNet, el sistema selecciona aquel cuya definición tenga la intersección léxica más alta con la segunda partición.

UWaterloo (Vechtomova, 2017)

Para localizar el *pun* UWaterloo propone un sistema basado en reglas que puntúa las palabras candidatas utilizando once heurísticas simples. Estas heurísticas obtienen la posición de la palabra a partir del contexto, la frecuencia de la palabra del documento en un gran corpus de referencia, la información mutua normalizada (PMI) con otras palabras del contexto y si la palabra existe en un conjunto de referencia de *puns* homófonos y palabras sonoras similares. Solo tiene en cuenta las palabras de la segunda mitad del contexto, en caso de empate el sistema escoge la palabra más cercana al final del texto.

UWAV (Vadehra, 2017)

Propone soluciones para las tareas de detección y localización. Para detectar el *pun* utiliza un sistema basado en aprendizaje supervisado utilizando los votos de tres clasificadores (máquina de vector soporte (SVM), Naive Bayes y regresión logística) entrenados utilizando características léxico-semanticas y *embeddings* de palabras. Para entrenar los modelos utiliza un *dataset* anotado manualmente.

Para la localización del *pun*, UWAV divide el contexto por la mitad y comprueba si cualquier palabra en la segunda mitad está en algunas listas predefinidas de homónimos, homófonos y antónimos. Si es así, una de esas palabras se selecciona como el *pun*. De lo contrario, la similitud se calcula entre cada par de *embeddings* de palabras en el contexto. En el par de palabras de mayor puntuación, se selecciona la palabra más cercana al final del texto.

N-Hance (Sevgili et al., 2017)

N-Hance presentó sus propuestas fuera de tiempo en la competición. N-Hance asume que el *pun* guarda una gran relación únicamente con una palabra del contexto. Para las tareas de detección y localización calcula el PMI para todo par de palabras de las frases. En la tarea de detección ordena los *scores* (PMI) de mayor a menor y obtiene el rango intercuartílico IQR. De todos los IQR obtiene la mediana y la utiliza como umbral para determinar si el texto contiene *pun* o no.

Para la tarea de interpretación, el primer sentido se selecciona encontrando la intersección máxima entre las definiciones de los sentidos del *pun* y los sentidos del contexto. N-Hance obtiene el segundo sentido a partir de la palabra que tiene la puntuación PMI más alta respecto el *pun*, seleccionando como segundo sentido aquel que tenga mayor similitud coseno entre los sentidos del *pun* y los sentidos de la palabra emparejada.

En la subtarea de detección de *puns* los mejores resultados los obtienen *N-Hance* para *puns* homográficos y *Idiom Savant* para *puns* heterográficos. En la subtarea de localización de *puns* los mejores resultados los obtiene *Idiom Savant* y *UWaterloo* para *puns* homográficos y *UWaterloo* para *puns* heterográficos. Por útlimo para la subtarea de interpretación de *puns* los mejores resultados los obtienen *BuzzSaw* y *Duluth* para *puns* homográficos y *Idiom Savant* para *puns* heterográficos.

CAPÍTULO 8
Entorno de trabajo y herramientas

Python es el lenguaje de programación utilizado en la experimentación del presente proyecto. Python es un lenguaje de programación interpretado, multiplataforma y con tipado dinámico. Es multiparadigma ya que soporta programación orientada a objetos, programación imperativa y en menor medida, programación funcional. Uno de sus puntos más fuertes es que se trata de un lenguaje de código abierto con una licencia (*Python Software Foundation License*) que es compatible con la licencia pública general de GNU.

Python tiene una gran variedad de herramientas para trabajar en *PLN*, las cuales se actualizan con frecuencia y son desarrolladas por grupos de investigación de gran relevancia en la materia. Las herramientas que se han utlizado para abordar la experimentación del presente trabajo se describen en este capítulo.

NLTK

NLTK [16] (*Natural Language Toolkit*) es un kit de herramientas para el procesamiento de lenguaje natural, formado por un conjunto de bibliotecas y programas que se pueden descargar y utilizar en Python. NLTK fué creado con la finalidad de apoyar la investigación y la docencia en el estudio de procesamiento de lenguaje natural.

NLTK nos permite acceder directamente a los recursos de WordNet a través de Python, pero además proporciona un conjunto de herramientas de gran interés en PLN como son *parsers*, *chunking*, *part-of-speech-tagging*, *tokenizers*, *classification algorithms*, *corpus*, etc.

Word2Vec

Word2Vec [15] es una herramienta integrada en la librería Gensim. Proporciona una implementación eficiente de los modelos *CBOW* y *Skip-gram* para la obtención de representaciones vectoriales de palabras. Esta compuesto por una *suite* de algoritmos que nos permiten generar modelos de forma simple y cargar modelos ya entrenados.

Dado como entrada un corpus textual la herramienta genera como salida un vector de representaciones vectoriales de palabras. El funcionamiento interno de la herramienta consiste en crear un vocabulario (V) a partir de los textos de entrenamiento y aprender los vectores de representación mediante los modelos *CBOW* y *Skip-gram*. Si se utiliza Python, Word2Vec proporciona como salida un diccionario de tamaño $|V|$ donde la clave es la palabra y el valor es el vector de representación. En el presente trabajo los vectores de representación proporcionados por Word2Vec (*embeddings*) se utilizan para obtener relaciones semánticas entre pares de palabras dentro de un texto.

Realizando operaciones con los *embeddings* obtenidos se obtiene una medida de calidad mediante la identificación de regularidades sintácticas y semánticas, es decir, si los *embeddings* están bien entrenados las palabras que guarden relaciones sintácticas y semánticas se posicionaran cerca en el espacio de representación, por ejemplo:

$\Rightarrow$ *embedding*(coche) - *embedding*(coches) $\approx$ *embedding*(casa) - *embedding*(casas)
$\Rightarrow$ *embedding*(rey) - *embedding*(hombre) $\approx$ *embedding*(reina) - *embedding*(mujer)

En el presente trabajo se han utilizado dos modelos Word2Vec ya entrenados mediante *Google News* y la *Wikipedia*. En la Tabla 8.1 se muestran las características de ambos modelos.

	Modelos Embeddings	
Características	**Google News**	**Wikipedia**
Autor	Google	[3]
Dimensiones	300	400
Corpus	Google News	Wikipedia
Vocabulario	3M (palabras)	2.2M (palabras)
Arquitectura	Word2Vec	Word2Vec
Algoritmo	Negative Sampling	Negative Sampling

Tabla 8.1: Características de los modelos *Wod2Vec* utilizados en este trabajo.

Scikit-learn

Scikit-learn [14] (*sklearn*) es un *toolkit* de aprendizaje automático para Python formado por un conjunto de herramientas sencillas y eficientes para resolver problemas de minería de datos y análisis de datos. Esta construido sobre la librería de *Numpy* de Python (librería para calculo vectorial y matricial) y tiene una licencia de código abierto y libre uso (Licencia BSD).

Scikit-learn proporciona un conjunto de algoritmos para resolver problemas de clasificación, regresión y *clustering*. Tiene un módulo dedicado a preprocesamiento de datos proporcionando técnicas de extracción de características y de normalización de datos. También tiene implementados algoritmos para la reducción de dimensionalidad de los datos.

Permite con facilidad evaluar los modelos que generamos y nos proporciona herramientas para poder realizar comparaciones entre modelos. Todos los algoritmos son considerablemente configurables y poseen una *API* amplia y de fácil comprensión.

En el presente trabajo se han utilizado los algoritmos de clasificación: *LinearSVC*, *KNeighborsClassifier* y *KMeans* proporcionados por la librería *Scikit-learn*.

CAPÍTULO 9
Experimentación

En este capítulo se explica la experimentación que se ha realizado en este proyecto para resolver los problemas propuestos en la tarea 7: Detección e interpretación de *puns* en el idioma inglés. En el trabajo nos centramos en resolver los *puns* de tipo homográfico donde la experimentación se estructura en tres partes, una para cada subtarea. Todas las herramientas y conceptos que se aplican en este capitulo se han explicado en capítulos anteriores de este proyecto.

En las subtareas de localización e interpretación de *puns* planteamos una primera aproximación basada en la participación de nuestro grupo de investigación ELiRF-UPV. El trabajo [8] que presentamos ha sido aceptado y publicado por SemEval-2017. Respecto a la tarea de interpretación aunque no se abordó en la participación de la competición hemos planteado nuevas soluciones en este proyecto.

En la experimentación se hace referencia a términos que son de gran importancia para obtener los resultados que mostramos en este capitulo, por este motivo se explican de forma detallada a continuación.

Tokenización: Cuando hacemos referencia a aplicar tokenización a las frases del corpus nos referimos al preproceso que se realiza sobre los datos antes de aplicar los métodos propuestos. La tokenización consiste en eliminar *stopwords*, eliminar símbolos de puntuación y convertir las palabras a minúsculas. El término de *stopwords* hace referencia al conjunto de palabras que normalmente no tienen semántica (artículos, pronombres, preposiciones, etc) y que no son relevantes para ciertos problemas de PLN.

Palabras no contiguas (no_contiguas): Dos palabras son contiguas si son adyacentes dentro del texto. Al estudiar algunas frases del corpus se observa que el *pun* y la palabra (o palabras) que aportan información sobre el contexto del *pun* no suelen ser adyacentes. Cuando mencionamos en la experimentación el termino *no contiguas* hacemos referencia a que en el momento de seleccionar los pares de palabras de la frase para obtener su distancia, no se tienen en cuenta aquellas palabras que son adyacentes.

Palabra1 es diferente de palabra2 ($p1 \neq p2$): Se ha observado que hay frases que tienen palabras repetidas. Al calcular la distancia de dos palabras que son iguales el resultado es cero. Que dos palabras sean iguales en esta tarea no nos aporta información relevante, en cambio nos genera problemas ya que el sistema las detecta como aquel par de palabras de la frase con mayor similitud semántica. Esta restricción obliga a que todo par de palabras analizado deba contener palabras distintas.

Métrica basada en lemas de los sentidos (métrica_sentidos): La métrica tiene como entrada dos palabras (palabra1,palabra2) y obtiene todos los lemas de todos los sentidos de la palabra1 y todos los lemas de todos los sentidos de la palabra2. Aplica la distancia coseno de los *embeddings* de todos los lemas de la palabra1 con los *embeddings* de todos los lemas de la palabra2 y devuelve la menor distancia coseno obtenida.

9.1 Localización de puns

Nuestra primera aproximación para solucionar esta tarea se presentó en la competición de SemEval-2017 [8] (Grupo de investigación ELiRF de la UPV). La hipótesis que defendimos consiste en que el *pun* en la frase se puede determinar mediante otra palabra de la frase, la cual denominamos disparador, que corresponde a aquella palabra que guarde mayor similitud semántica respecto al *pun*.

Una buena relación semántica nos la proporciona la distancia coseno, la cual se aplica entre todo par de *embeddings* de palabras no contiguas. Según nuestra hipótesis, aquel par de *embeddings* con menor distancia coseno contiene el *pun*. El algoritmo se ilustra en la Figura 9.1.

Algorithm 1: Selection of the pun of a sentence, task2

Input: s, the sentence that contains a pun
Result: w_k, the word in s that we guess is the pun

```
begin
    k, b ← -1, ∞
    t ← remove_stopwords(tokenize(s))
    foreach w_i ∈ t do
        e_i ← embedding(w_i)
        foreach w_j ∈ t: i < j − 1 do
            e_j ← embedding(w_j)
            d ← cosine_distance(e_i, e_j)
            if d < b then
                b, k ← d, j
    if k > −1 then
        return w_k
    else
        return w_{|t|−1}
```

Figura 9.1: Algoritmo para la subtarea de localización presentado en SemEval-2017.

Se tokeniza el texto de entrada y para cada par de *tokens* no contiguos se obtiene su representación de *embeddings*. Con el par de *embeddings* se calcula la distancia coseno. El algoritmo devuelve como *pun* aquella palabra situada en la posición derecha del par de palabras cuya distancia coseno sea menor. Esto se debe a que nos basamos en la hipótesis de que el *pun* aparece en las últimas posiciones de la frase. Se puede dar el caso que el algoritmo no obtenga distancias coseno de ningún par de palabras a causa de que la tokenización devuelva un número pequeño de *tokens* y ninguno se encuentren en el

diccionario *word2vec* de *embeddings*. En estos casos se decide devolver la última palabra de la frase basándonos en la hipótesis de que el *pun* frecuentemente se encuentra en las últimas posiciones.

	homographic			
system	**C**	**P**	**R**	**F_1**
BuzzSaw	**1.0000**	0.2775	0.2775	0.2775
Duluth	**1.0000**	0.4400	0.4400	0.4400
ECNU	**1.0000**	0.3373	0.3373	0.3373
ELiRF-UPV	**1.0000**	0.4462	0.4462	0.4462
Fermi	**1.0000**	0.5215	0.5215	0.5215
Idiom Savant	0.9988	**0.6636**	**0.6627**	**0.6631**
JU_CSE_NLP	**1.0000**	0.3348	0.3348	0.3348
PunFields‡	**1.0000**	0.3279	0.3279	0.3279
UWaterloo	0.9994	0.6526	0.6521	0.6523
UWAV	**1.0000**	0.3410	0.3410	0.3410
random	**1.0000**	0.0846	0.0846	0.0846
last word	**1.0000**	0.4704	0.4704	0.4704
max. polysemy	**1.0000**	0.1798	0.1798	0.1798
N-Hance	0.9956	0.4269	0.4250	0.4259

Figura 9.2: Resultados tarea de localización SemEval-2017.

Los resultados de la competición se muestran en la Figura 9.2. Los resultados se dividen en dos partes, la parte superior corresponde a los resultados de los participantes de la competición que entregaron sus propuestas dentro de la fecha que establece la organización, la parte inferior son los resultados de los *baselines* y los resultados de los participantes que entregaron sus resultados fuera de plazo. Nuestro equipo (ELiRF-UPV) obtuvo la 4 posición de un total de 10 participantes. Una vez finalizada la competición los corpus fueron liberados por eso es interesante contrastar las dos hipótesis que aplicamos en el algoritmo 9.1. El número de pares de *tokens* seleccionados para obtener el pun, es decir, aquel par de *tokens* con menor distancia coseno en la frase contiene el *pun* 767 veces donde 702 de esos pares (91.5 %) el *pun* corresponde a la palabra del par situada a la derecha.

Al disponer actualmente del corpus etiquetado (*gold*) proporcionado por la organización para la tarea de localización se ha decidido particionar el corpus. Utilizando el 30 % del corpus etiquetado (*gold*) como corpus de desarrollo (*dev*) y el 70 % restante para evaluación (*test*). Los datos se han barajado y se ha establecido una semilla para que la partición del corpus sea siempre la misma.

Nº Oraciones		
Total	***Dev***	***Test***
1607	482	1125

Tabla 9.1: Partición del corpus de la tarea localización en *dev* y *test*

Se han aplicado un conjunto de métodos que definen una distancia entre pares de palabras. Utilizando la partición *dev* se ha obtenido la frecuencia sobre el número de veces que el *pun* se encuentra en el par de palabras con menor distancia. La Tabla 9.2 muestra las frecuencias[1] descritas para cada modelo de *embeddings* (Google-News y Wikipedia).

[1] Los valores de las celdas tienen el formato X/Y donde X es el número de frases que cumplen la restricción e Y es el número de frases analizadas por el método.

Se observa que los dos modelos de *embeddings Word2Vec* están bien entrenados ya que obtienen resultados muy similares. Los mejores resultados se obtienen utilizando la métrica basada en la distancia coseno de lemas de sentidos (métrica_sentidos) aplicando las restricciones de palabras no contiguas (no_contiguas) y que la palabra p1 sea diferente a la palabra p2 del par de *tokens* de la frase ($p1 \neq p2$). A la distancia obtenida con este método la vamos a llamar en este apartado métrica_sentidos_v2.

Estadísticas		
Método	**Google News**	**Wikipedia**
d_coseno	233/481	231/481
d_coseno + no_contiguas	253/480	250/480
d_coseno + no_contiguas + $p1 \neq p2$	256/480	256/480
métrica_sentidos	279/481	289/478
métrica_sentidos + no_contiguas	291/480	294/475
métrica_sentidos + no_contiguas + $p1 \neq p2$	298/480	301/475

Tabla 9.2: Estadísticas de aparición del *pun* en el par de palabras con menor distancia.

Utilizar la información que nos aporta la posición de la palabra en el texto puede ser relevante basándonos en la hipótesis que el *pun* se encuentra en las últimas palabras del texto. Esto consiste en dar mayor importancia a las palabras situadas al final de la frase que a las palabras situadas al inicio. Manteniendo el criterio que el *pun* es la palabra situada a la derecha del par de palabras se ha realizado una ponderación sobre la distancia utilizando la posición de las palabras en la frase.

⇒ Ejemplo: Posición de las palabras para ponderar la distancia.
Frase: *I used to be a banker but I lost the interest*.

Analizando los pares de palabras de la frase se obtienen las siguientes distancias:
- d(*banker,interest*) = 0.689
- d(*banker,lost*) = 0.892

Manteniendo la hipótesis que el *pun* es la palabra de la derecha del par de palabras. Obtenemos las posiciones de los posibles *puns*:
- pos(*interest*) = 11
- pos(*lost*) = 9

La longitud de la frase es:
- len(frase) = 11

Obtenemos las ponderaciones sobre la posición de los posibles *puns*:
- ponderación(*interest*) = ((len(frase)-pos(*interest*))+1) / (len(frase) =
((11-11)+1) / 11 = 0.091
- ponderación(*lost*) = ((11-9)+1) / 11 = 0.273

La nueva métrica para obtener la distancia es la siguiente:
- d'(*banker,interest*) = 0.689 * 0.091 = 0.063
- d'(*banker,lost*) = 0.892 * 0.273 = 0.243

Esta ponderación basada en la posición da valores pequeños para palabras situadas al final de la frase y valores más grandes para palabras situadas en las primeras posiciones.

Este criterio es necesario para mantener la métrica de distancia en la nueva métrica obtenida.

Utilizando como distancia la métrica_sentidos_v2 multiplicada por la ponderación de la posición de la palabra situada a la derecha del par de palabras (ponderación_pos) se obtienen las estadísticas que mostramos en la Tabla 9.3.

Estadísticas		
Método	**Google News**	**Wikipedia**
métrica_sentidos_v2 * ponderación_pos	359/480	362/475

Tabla 9.3: Estadísticas de una primera aproximación de las posiciones de las palabras en el texto.

Observamos que al utilizar la posición para calcular la distancia se mejoran las estadísticas considerablemente. Esta mejora nos indica que utilizar la información de la posición de la palabra en la frase es importante. Se ha realizado un estudio para ver el peso que tiene la métrica_sentidos_v2 y la ponderación de la posición de las palabras en la localización del *pun*. Para ello se ha ponderado cada métrica de la siguiente forma:

La ponderación de cada métrica se obtiene con dos pesos wp y wd que son los parámetros que vamos a estudiar:

distancia = (wp*ponderacion_pos)+(wd*metrica_sentidos_v2)

Donde:
wp + wd = 1
ponderacion_pos(w) = ((len(sentencia)-pos(w))+1) / (len(sentencia)
w = palabra en la posición derecha del par de palabras.

Se ha realizado un barrido de los pesos wp y wd para estudiar la importancia de la métrica_sentidos_v2 y ponderación_pos. Los resultados se muestran en la Tabla 9.4.

Estadísticas		
Parámetros	**Google News**	**Wikipedia**
wd = 0.7; wp = 0.3	343/480	355/475
wd = 0.6; wp = 0.4	356/480	368/475
wd = 0.5; wp = 0.5	363/480	375/475
wd = 0.4; wp = 0.6	373/480	373/475
wd = 0.3; wp = 0.7	375/480	373/475
wd = 0.2; wp = 0.8	370/480	371/475

Tabla 9.4: Barrido de parámetros wd y wp para medir la importancia de las métricas métrica_sentidos_v2 y ponderación_pos en la subtarea de localización de *puns*

Los mejores parámetros corresponden a wd = 0.3, wp = 0.7 con *embeddings* de Google News y wd = 0.5, wp = 0.5 con *embeddigns* de la Wikipedia.

Las mejores estadísticas sobre el estudio realizado en la subtarea de localización se basan en utilizar la métrica:

$$distancia = (wp * ponderacion\_pos) + (wd * metrica\_sentidos\_v2)$$

con los mejores parámetros obtenidos para cada modelo de *embeddings*. Para evaluar el sistema se muestra en las tablas 9.5 y 9.6 los resultados obtenidos sobre la partición de

Resultados Google News			
Coverage	*Precision*	*Recall*	*F1-score*
0.9902	0.6032	0.5973	0.6002

Tabla 9.5: Resultados de la tarea de localización utilizando el modelo de Google News con wd = 0.3 y wp = 0.7 para la partición de *test*.

Resultados Wikipedia			
Coverage	**Precision**	**Recall**	**F1-score**
0.9822	0.6099	0.5991	0.6044

Tabla 9.6: Resultados de la tarea de localización utilizando el modelo de la Wikipedia con wd = 0.5 y wp = 0.5 para la partición de *test*.

test. Para escoger el *pun* del par de *embeddings* de palabras con menor distancia se ha mantenido la hipótesis de escoger la palabra de la posición derecha del par.

Observamos que con la partición de *test* se han obtenido mejores resultados utilizando el modelo de *embeddings* de Google News. Es interesante utilizar el corpus completo (1607 frases) para evaluar el sistema que mejor resultados nos ha proporcionado con la finalidad de comprobar la mejora real que hubiésemos obtenido en la competición. La Tabla 9.7 muestra los resultados.

Resultados			
Coverage	**Precision**	**Recall**	**F1-score**
0.9919	0.6324	0.6273	0.6298

Tabla 9.7: Resultados de la tarea de localización utilizando el modelo de Google News con wd = 0.3 y wp = 0.7 sobre el corpus completo de la tarea.

Observamos que si hubiésemos utilizado la mejor aproximación que se ha obtenido en el estudio del presente trabajo los resultados que hubiésemos obtenido en la competición mejorarían considerablemente.

Los resultados que hemos obtenido, tanto en la partición de *test* cómo utilizando el corpus completo para *test*, no se pueden comparar con los resultados que obtuvieron los participantes en la competición debido a que no se disponía del corpus etiquetado para poder realizar un estudio con una partición del mismo.

9.2 Detección de puns

La tarea de detección consiste en determinar de un conjunto de frases cuales contienen *pun* y cuales no. Este problema se puede interpretar como un problema de clasificación en dos clases: (1) la frase contiene *pun* y (0) la frase no contiene *pun*.

En la competición SemEval-2017 se obtuvieron los resultados que se muestran en la Fig. 9.3 para esta tarea, en la cual no participamos.

	homographic			
system	**P**	**R**	**A**	**F_1**
Duluth	0.7832	0.8724	**0.7364**	**0.8254**
Idiom Savant	—	—	—	—
JU_CSE_NLP	0.7251	**0.9079**	0.6884	0.8063
PunFields	**0.7993**	0.7337	0.6782	0.7651
UWAV	0.6838	0.4723	0.4671	0.5587
random	0.7142	0.5000	0.5000	0.5882
ECNU*	0.7127	0.6474	0.5628	0.6785
Fermi†	**0.9024**	0.8970	**0.8533**	**0.8997**
N-Hance	0.7553	**0.9334**	**0.7364**	**0.8350**

Figura 9.3: Resultados tarea de detección SemEval-2017.

El corpus de esta tarea esta formado por frases que contienen *pun* y frases que no contienen *pun*. La información acerca del corpus se muestra en la Tabla 9.8.

Corpus Detección		
Nº. Total Textos	**Nº. Textos con Pun**	**Nº. Textos sin Pun**
2250	1607	643

Tabla 9.8: Información del corpus de la tarea de detección.

Se observa que el corpus esta desbalanceado donde hay aproximadamente un 40 % más de frases que contienen *pun* respecto a las frases que no contiene *pun*. Al trabajar con un corpus que esta desbalanceado los resultados se ven influenciados por la clase que predomina en el conjunto de datos. Para esta tarea en concreto imaginemos que nuestra solución es clasificar a todas las frases del corpus como frases que contienen *pun* donde los resultados que obtendríamos serían TP = 1607, FP = 643, TN = 0, FN = 0 obteniendo una P = 0.7142, R = 1.0, A = 0.7142, F_1 = 0.8333 y C = 1.0. Esto demuestra que, en esta tarea con este corpus, el sistema que obtenga mejores resultados será aquel que sea más propenso a clasificar las frases como frases que contienen *pun*.

Partiendo de que el corpus esta desbalanceado los resultados que se han obtenido en la competición no nos aportan una información real acerca de como discriminan los sistemas propuestos por los participantes. Por este motivo en esta experimentación se ha balanceado el corpus y se ha realizado una evaluación mediante validación cruzada. Para que el corpus esté balanceado ha de tener el mismo número de frases que contienen *pun* como frases que no contienen *pun*. El criterio que se ha escogido para balancear el corpus ha sido particionar el conjunto de frases que contienen *pun* en tres partes, dos de las partes contienen 643 frases cada una y una tercera parte que contiene las 321 frases restantes. Se han juntado las 643 frases que contienen *pun* con las 643 frases que no contienen *pun* obteniendo dos particiones de *test* formadas por un total de 1286 frases cada una.

Para realizar la experimentación se ha generado un corpus propio de desarrollo (*dev*) que nos permite analizar y tunear parámetros antes de realizar la evaluación (*test*) del sistema. Para este corpus se han obtenido de la Web[2] frases que no contiene *pun* que corresponden a refranes y proverbios. Se han obtenido un total de 422 frases que no contienen (*pun*) las cuales se han comparado entre si y se han eliminado repetidas quedando un total de 318 frases. De las 318 frases se han eliminado aquellas frases que

[2]https://bristolenos.com/2015/06/07/refranes-y-proverbios-en-ingles/
http://cogweb.ucla.edu/Discourse/Proverbs/Spanish-English.html

aparecen en el corpus de *test* quedando finalmente un total de 277 frases. El conjunto de frases que contienen *pun* en el corpus de desarrollo se ha obtenido escogiendo 277 frases de las 321 frases de la tercera partición obtenida al balancear el corpus de *test*.

Para realizar las particiones de *test* y generar el corpus *dev* se han barajado las frases y se ha establecido una semilla. Barajar los datos antes de realizar las particiones nos permite evitar correlaciones entre los datos y establecer una semilla nos permite volver a realizar las particiones y/o la generación del corpus de desarrollo asegurando que los resultados obtenidos por los diferentes sistemas propuestos sean siempre los mismos.

Al utilizar un corpus balanceado los resultados que hemos obtenido en esta tarea no se pueden comparar con los resultados que han obtenido los participantes de la competición.

Las soluciones que se proponen en el presente trabajo para resolver la tarea de detección se basan en tres aproximaciones que se detallan a continuación:

- Umbral (*threshold*): La clasificación de las dos clases se realiza aplicando un umbral. Un umbral es un valor que se obtiene a partir de los datos y define un límite entre las dos clases. Es muy utilizado en técnicas de binarización de imágenes.

Siendo c_x una característica obtenida a partir de un objeto $x \in X$
if $c_x \geq umbral$ **then**
$clase(x) == 1$
else
$clase(x) == 0$
end if

- Aprendizaje no supervisado: Utilizando un conjunto de datos sin etiquetar se realizan técnicas de *clustering* que nos permiten obtener dos centroides. Es necesario definir un criterio que defina la etiqueta de cada *cluster*.
- Aprendizaje supervisado: A partir de un conjunto de datos de entrenamiento (etiquetados) se obtiene un modelo. El modelo entrenado se utiliza para clasificar los datos de evaluación (*test*).

Todas las experimentaciones para el estudio y *tuning* de parámetros se han realizado sobre el corpus de desarrollo (*dev*) obteniendo un conjunto de resultados. El método que mejor resultado nos ha proporcionado es el que hemos utilizado para evaluar el sistema con el corpus de *test*. Como se ha explicado en este punto el corpus de *test* se divide en dos particiones con la finalidad de evaluar el sistema con corpus balanceados, por este motivo la evaluación resultante es una evaluación cruzada que consiste en evaluar las dos particiones de forma independiente y obtener la media como resultado.

En esta tarea cuando hablamos de la distancia aplicada a los *embeddings* del par de palabras hacemos referencia a la distancia y parámetros que mejor resultado nos han proporcionado en la tarea de localización del *pun*.

$$distancia = (wp * ponderacion\_pos) + (wd * metrica\_sentidos\_v2)$$

Clasificación a partir de un umbral:

N-Hance [12] propone un criterio interesante para clasificar la clases utilizando un umbral. El criterio consiste en obtener todos los pares de palabras posibles de una frase y utilizar como factor de discriminación la diferencia entre el primer par de palabras y el segundo par de palabras con mayor similitud. La hipótesis que propone se basa en

que el *pun* se encuentra en el par de palabras con mayor similitud y la diferencia entre la similitud de dicho par con el resto ha de ser significativa si la frase contiene *pun*.

Nuestra primera aproximación consiste en utilizar el método propuesto por N-Hance utilizando como métrica de similitud semántica nuestra distancia y obteniendo los resultados a partir de un corpus balanceado (*dev*). El método consiste en obtener de cada frase del corpus el rango intercuartílico (IQR). Para una frase de entrada se calcula la distancia entre todo par de palabras representadas como *embeddings*, las distancias se ordenan de menor a mayor y el IQR corresponde a la diferencia entre la distancia que pertenezca al tercer cuartil y la distancia que pertenezca al primer cuartil. El umbral corresponde a la mediana del conjunto de rangos intercuartílicos obtenidos a partir de las frases del corpus.

Para clasificar una frase se obtiene la distancia entre todo par de palabras representadas como *embeddings*. Las distancias se ordenan de menor a mayor y se obtiene la diferencia entre el segundo par de palabras y el primer par de palabras con menor distancia. Si la diferencia es mayor o igual que el umbral la frase se clasifica como frase que contiene *pun*.

Se ha obtenido un umbral = 0.2283 utilizando el modelo de *embeddings* de Google News y un umbral = 0.1838 utilizando el modelo de *embeddings* de la Wikipedia. Los resultados obtenidos se muestran en la Tabla 9.9.

Resultados					
Embeddings	**Coverage**	**Precision**	**Recall**	**Accuracy**	**F1-score**
Google News	0.7635	1.0	0.0192	0.3948	0.0376
Wikipedia	0.7599	0.8636	0.0730	0.4204	0.1347

Tabla 9.9: Experimentación utilizando el método IQR propuesto por *N-Hance*

.

Observamos que el criterio que propone N-Hance utilizando nuestra distancia y un corpus balanceado (*dev*) no nos proporciona buenos resultados. Antes de descartar este criterio se ha realizado una experimentación utilizando *clustering*.

Aprendizaje no supervisado:

Nuestra segunda aproximación consiste en clasificar las frases utilizando *clustering*. Se ha utilizado el algoritmo *KMeans* de la librería *sklearn* para realizar el *clustering*. Como parámetros se ha especificado obtener 2 *clusters*, un máximo de 300 iteraciones y una semilla en el parámetro *random_state* para que la inicialización del algoritmo sea siempre la misma. El resto de parámetros son los que el algoritmo trae por defecto.

Manteniendo el criterio que propone N-Hance, una frase se representa con un número que corresponde a la diferencia entre el segundo par de palabras y el primer par de palabras (usando representación de *embeddings*) con menor distancia. Esta característica es la que se le proporciona al algoritmo para que agrupe las frases en dos clases.

El algoritmo *KMeans* nos devuelve dos centroides que determinan cada *cluster*. Al tratarse de un aprendizaje no supervisado hemos de buscar un criterio para determinar que centroide clasifica las frases que contienen *pun* y que centroide clasifica las frases que no contienen *pun*.

Basándonos en la hipótesis de N-Hance si la diferencia entre el segundo par de palabras y el primer par de palabras con menor distancia es grande la frase contiene *pun* por lo tanto el centroide que clasifica a la clase de frases que contienen *pun* es aquel centroide con mayor valor. Los resultados obtenidos se muestran en la Tabla 9.10.

Resultados					
Embeddings	**Coverage**	**Precision**	**Recall**	**Accuracy**	**F1-score**
Google News	0.7635	0.7125	0.2183	0.4634	0.3343
Wikipedia	0.7599	0.6250	0.2308	0.4394	0.3371

Tabla 9.10: Experimentación utilizando clustering con *KMeans*.

Observamos que aunque los resultados mejoren respecto a la primera aproximación siguen siendo poco prometedores. Con estos dos estudios podemos verificar que el criterio que propone N-Hance para clasificar las frases no es bueno utilizando nuestra distancia y un corpus balanceado.

Aprendizaje supervisado

Para entrenar y evaluar los modelos obtenidos con aprendizaje supervisado se han representado las frases como vectores de tres componentes:

- 1º Componente: Es la menor distancia de todo par de palabras (representadas como *embeddings*) de la frase. La hipótesis de esta componente consiste en que el par de palabras con menor distancia contiene el *pun*, y si la frase contiene *pun* la distancia entre el *pun* y la palabra disparadora debe ser pequeña, en cambio si dicha distancia es grande es un primer indicio de que la frase no contiene *pun*.
- 2º Componente: Del par de palabras con menor distancia nos quedamos con la palabra de la derecha (según nuestro criterio es el *pun*). La 2º componente del vector es la posición de dicha palabra en la frase. La hipótesis de esta componente consiste en que si dicha palabra es el *pun* su posición corresponderá a una posición cercana al final de la frase.
- 3º Componente: Es la diferencia de posiciones en la frase entre la palabra de la izquierda y la palabra de la derecha del par de palabras de menor distancia. La hipótesis se basa en que el *pun* y la palabra disparador se suelen encontrar separadas en la frase.

Los algoritmos de aprendizaje supervisado necesitan un conjunto de datos etiquetados para entrenar el modelo y un conjunto de datos etiquetados para evaluar dicho modelo. Para realizar la experimentación utilizando aprendizaje supervisado se ha particionado el corpus de desarrollo (*dev*) en dos partes, una parte para entrenamiento formada por 394 frases donde 197 frases contienen *pun* y 197 frases no contienen *pun* y una parte para evaluación del modelo formada por 160 frases de las cuales 80 contienen *pun* y las 80 restantes no contienen *pun*.

El primer estudio se ha realizado utilizando como clasificador el algoritmo de *k*-vecinos más cercanos. Para su implementación se ha utilizado el módulo *KNeighborsClassifier* que nos proporciona la librería *sklearn* donde se ha utilizado como distancia la distancia euclidea y se ha realizado un estudio variando el parámetro *k*. Los resultados se muestran en la Tabla 9.11.

Resultados Google News					
k	**Coverage**	**Precision**	**Recall**	**Accuracy**	**F1-score**
1	0.95	0.6666	0.725	0.6645	0.6946
2	0.95	0.7302	0.575	0.6645	0.6434
3	0.95	0.7159	0.7875	0.7237	0.75
4	0.95	0.7722	0.7625	0.7566	0.7673
5	0.95	0.6989	0.8125	0.7171	0.7514
10	0.95	0.6947	0.825	0.7171	0.7543
20	0.95	0.7415	0.825	0.7566	0.7811
30	0.95	0.7234	0.8134	0.735	0.7658
Resultados Wikipedia					
k	**Coverage**	**Precision**	**Recall**	**Accuracy**	**F1-score**
1	0.95	0.6354	0.7625	0.6447	0.6932
2	0.95	0.7077	0.575	0.6513	0.6345
3	0.95	0.6966	0.775	0.7039	0.7337
4	0.95	0.725	0.725	0.7105	0.725
5	0.95	0.6966	0.775	0.7039	0.7337
10	0.95	0.6818	0.75	0.6842	0.7142
20	0.95	0.7204	0.8375	0.7434	0.7746
30	0.95	0.7186	0.8125	0.7238	0.7627

Tabla 9.11: Barrido del parámetro *k* en el clasificador *KNeighborsClassifier* de la tarea de detección utilizando la partición de *test* del corpus *dev*.

Observamos que basándonos en este estudio, los mejores resultados los obtenemos con el modelo de *embeddings* entrenado con Google News y utilizando los 20 vecinos más cercanos (k = 20).

El segundo estudio que hemos realizado consiste en entrenar un clasificador *SVM*. Se ha utilizado el algoritmo que tiene implementado el módulo *svm.SVC* de la librería *sklearn*. Se ha realizado *tuning* de los parámetros que hacen referencia al *kernel* y al factor *C*. Los demás parámetros que tiene el algoritmo se han dejado por defecto.

El primer parámetro de estudio corresponde al tipo de *kernel* que utilizamos en el modelo *SVM*, para ello se ha realizado un barrido de los *kernels*: *rbf, linear, poly y sigmoid* que corresponden a los *kernels* que nos permite utilizar el módulo *sklearn.svm.SVC*. Para estudiar cual es el mejor *kernel* se ha establecido el parámetro *C* con el valor que trae por defecto el algoritmo ($C = 1$). Los resultados obtenidos se muestran en la Tabla 9.12.

Resultados Google News					
Kernel	**Coverage**	**Precision**	**Recall**	**Accuracy**	**F1-score**
rbf	0.95	0.7143	0.8125	0.7303	0.7602
linear	0.95	0.7701	0.8375	0.7829	0.8024
poly	0.95	0.7866	0.7375	0.7566	0.7613
sigmoid	0.95	0.5263	1.0	0.5263	0.6896
Resultados Wikipedia					
Kernel	**Coverage**	**Precision**	**Recall**	**Accuracy**	**F1-score**
rbf	0.95	0.7097	0.825	0.7303	0.7630
linear	0.95	0.7816	0.85	0.7960	0.8144
poly	0.95	0.7647	0.65	0.7105	0.7027
sigmoid	0.95	0.5263	1.0	0.5263	0.6896

Tabla 9.12: Estudio del parámetro *kernel* en el clasificador *SVM* para la tarea de detección utilizando la partición de *test* del corpus *dev*.

Observamos que el mejor *kernel* obtenido es de tipo lineal tanto para *embeddings* de Google News como para *embeddings* de la Wikipedia. Es interesante estudiar como afecta

el parámetro *C* en el clasificador *SVM* por este motivo se ha realizado un barrido de dicho parámetro utilizando como *kernel* el tipo lineal. Se muestran los resultados obtenidos en la Tabla 9.13.

Resultados Google News					
C	**Coverage**	**Precision**	**Recall**	**Accuracy**	**F1-score**
0.1	0.95	0.76471	0.8125	0.7697	0.7879
0.2	0.95	0.7674	0.825	0.7763	0.7952
0.3	0.95	0.7674	0.825	0.7763	0.7952
0.4	0.95	0.7674	0.825	0.7763	0.7952
0.5	0.95	0.7674	0.825	0.7763	0.7952
0.6	0.95	0.7674	0.825	0.7763	0.7952
0.7	0.95	0.7674	0.825	0.7763	0.7952
0.8	0.95	0.7701	0.8375	0.7829	0.8024
0.9	0.95	0.7701	0.8375	0.7829	0.8024
1.0	0.95	0.7701	0.8375	0.7829	0.8024
Resultados Wikipedia					
C	**Coverage**	**Precision**	**Recall**	**Accuracy**	**F1-score**
0.1	0.95	0.7816	0.85	0.7960	0.8144
0.2	0.95	0.7816	0.85	0.7960	0.8144
0.3	0.95	0.7816	0.85	0.7960	0.8144
0.4	0.95	0.7816	0.85	0.7960	0.8144
0.5	0.95	0.7816	0.85	0.7960	0.8144
0.6	0.95	0.7816	0.85	0.7960	0.8144
0.7	0.95	0.7816	0.85	0.7960	0.8144
0.8	0.95	0.7816	0.85	0.7960	0.8144
0.9	0.95	0.7816	0.85	0.7960	0.8144
1.0	0.95	0.7816	0.85	0.7960	0.8144

Tabla 9.13: Estudio del parámetro *C* en el clasificador *SVM* para la tarea de detección utilizando la partición de *test* del corpus *dev*.

Se observa que para el estudio del parámetro *C* en el clasificador *SVM* los mejores resultados se obtienen con el modelo de *embeddings* de la Wikipedia donde el parámetro *C* no influye en los resultados. Que el parámetro *C* no influya en los resultados significa que no hay muestras mal clasificadas entre el hiperplano separador y los vectores soporte.

A partir del conjunto de estudios realizados se observa que los mejores parámetros obtenidos para abordar la tarea de detección es utilizar *embeddings* entrenados con la Wikipedia y un clasificador *SVM* con un *kernel* lineal cogiendo cualquier parámetro para *C*. Se ha decidido escoger $C = 0.8$ buscando que el parámetro mantenga el criterio de aceptar pocas muestras mal clasificadas pero no lo haga de forma estricta.

Utilizando el modelo entrenado con los mejores parámetros obtenidos en la experimentación se ha realizado la evaluación sobre el corpus balanceado de la tarea (test) mostrando los resultados en la Tabla 9.14.

Resultados					
Dataset	**Coverage**	**Precision**	**Recall**	**Accuracy**	**F1-score**
test_particion1	0.9378	0.6702	0.7050	0.6625	0.6872
test_particion2	0.9362	0.6631	0.6851	0.6520	0.6739
Media	0.9370	0.6667	0.6951	0.6573	0.6806

Tabla 9.14: Resultados obtenidos con el corpus de *test* balanceado en la tarea de detección.

9.3 Interpretación de puns

La primera aproximación para solucionar esta tarea la presentamos en la competición de SemEval-2017 [8] (Grupo de investigación ELiRF de la UPV). El algoritmo que propusimos en la competición se muestra en la Fig. 9.4.

begin
  w_i, w_j = get_closest_words(s, w_p)
  $sy_1, b \leftarrow null, -\infty$
  foreach $sy_p \in synsets(w_p)$ **do**
    foreach $sy_i \in synsets(w_i)$ **do**
      $s \leftarrow$synset_similarity(sy_p, sy_i)
      if $s > b$ **then**
        $sy_1, b \leftarrow sy_p, s$
  $sy_2, b \leftarrow null, -\infty$
  foreach $sy_p \in synsets(w_p)$ **do**
    foreach $sy_j \in synsets(w_j) \mid sy_j \neq sy_1$ **do**
      $s \leftarrow$synset_similarity(sy_p, sy_i)
      if $s > b$ **then**
        $sy_2, b \leftarrow sy_p, s$
  return (sy_1, sy_2)

Figura 9.4: Algoritmo ELiRF-UPV para tarea de interpretación en la competición.

En el algoritmo propuesto (Fig. 9.4) la función *get_closest_words* tokeniza la frase de entrada y devuelve los dos *tokens* (*token1*,*token2*) con menor distancia coseno respecto al *pun*, aplicando la restricción de que los *tokens* devueltos y el *pun* no sean contiguos en la frase. Para cada sentido (*synset*) WordNet del *pun*, *token1 y token2* se obtiene una bolsa de palabras (*BOW*) construida a partir de todos los lemas de la definición del *synset*, el propio nombre del *synset* y los lemas de todos los ejemplos del *synset*. La hipótesis que proponemos consiste en obtener el primer sentido a partir del *token1* y el segundo sentido a partir del *token2*. Para obtener el primer sentido se calcula la similitud entre todos los sentidos del *pun* y todos los sentidos del *token1* devolviendo el sentido con mayor similitud. El segundo sentido se obtiene siguiendo el mismo criterio pero utilizando el *token2* y devolviendo un sentido que sea diferente al sentido devuelto por el *token1*.

La métrica para medir la similitud entre dos sentidos (*synset_similarity*) se define de la siguiente forma:

$$similitud = \|BOW\_sentido_{pun} \cap BOW\_sentido_{token}\|$$

Los resultados que obtuvimos en la competición se muestran en la Fig. 9.5 donde quedamos 4 de un total de 6 participantes.

Se ha realizado un estudio variando las métricas que nos proporcionan los dos *tokens* con mayor similitud respecto al *pun*. Para realizar la experimentación se ha particionado el corpus en dos partes, una parte para desarrollo (30 %) y otra parte para evaluar el mejor sistema obtenido (70 %) donde la información de la partición se muestra en la Tabla. 9.15.

	homographic			
system	**C**	**P**	**R**	**F_1**
BuzzSaw	0.9761	0.1563	**0.1525**	0.1544
Duluth (DM)	0.8606	**0.1683**	0.1448	**0.1557**
Duluth (ED)	0.9992	0.1480	0.1479	0.1480
ELiRF-UPV	0.9646	0.1014	0.0978	0.0996
Idiom Savant	**0.9900**	0.0778	0.0770	0.0774
PunFields	0.8760	0.0484	0.0424	0.0452
random	**1.0000**	0.0931	0.0931	0.0931
MFS	**1.0000**	0.1348	0.1348	0.1348
Miller & Gurevych	0.6826	**0.1975**	0.1348	**0.1603**
N-Hance	0.9831	0.0204	0.0200	0.0202

Figura 9.5: Resultados de la tarea de interpretación de *puns* en la competición SemEval-2017.

Partición (Frases)		
Total	**Dev**	**Test**
1298	389	909

Tabla 9.15: Partición en *dev* y *test* del corpus de interpretación.

El primer estudio que hemos realizado consiste en cambiar la distancia coseno por la mejor distancia obtenida en la tarea de localización utilizando los mejores parámetros (wp,wd) obtenidos para cada modelo de *embeddings*.

$$distancia\_wp\_wd = (wp * ponderacion\_pos) + (wd * metrica\_sentidos\_v2)$$

Los resultados obtenidos en este estudio utilizando la partición *dev* se muestran en la Tabla 9.16.

Resultados				
Embeddings	**Coverage**	**Precision**	**Recall**	**F1-score**
Google News	0.928	0.108	0.1003	0.104
Wikipedia	0.9203	0.0978	0.09	0.0937

Tabla 9.16: Estudio utilizando la *distancia_wp_wd* para obtener los *tokens* en la tarea de interpretación.

El segundo estudio consiste en obtener el primer *token* respecto al *pun* utilizando como distancia la distancia coseno y el segundo *token* al *pun* utilizando como distancia la *distancia_wp_wd*. Si las dos distancias devuelven el mismo *token* el segundo *token* corresponde al siguiente *token* con menor *distancia_wp_wd* respecto al *pun*. Los resultados obtenidos utilizando la partición *dev* se muestran en la Tabla 9.17.

Resultados				
Embeddings	**Coverage**	**Precision**	**Recall**	**F1-score**
Google News	0.9229	0.1003	0.0925	0.0962
Wikipedia	0.9152	0.0927	0.0848	0.0886

Tabla 9.17: Estudio utilizando la distancia coseno para obtener el primer *token* y la distancia *distancia_wp_wd* para obtener el segundo *token* en la tarea de interpretación.

Es interesante ver si mejoran los resultados invirtiendo las distancias, es decir, obtener el primer *token* utilizando como distancia *distancia_wp_wd* y el segundo *token* utilizando como distancia la distancia coseno. Si las dos distancias devuelven el mismo *token* el segundo *token* corresponde al siguiente *token* con menor distancia coseno respecto al *pun*. Los resultados obtenidos utilizando la partición *dev* se muestran en la Tabla 9.18.

Resultados				
Embeddings	**Coverage**	**Precision**	**Recall**	**F1-score**
Google News	0.9177	0.1064	0.0977	0.1019
Wikipedia	0.9203	0.0978	0.09	0.0937

Tabla 9.18: Estudio utilizando la distancia *distancia_wp_wd* para obtener el primer *token* y la distancia coseno para obtener el segundo *token* en la tarea de interpretación.

Una alternativa interesante consiste en obtener el primer sentido a través de una herramienta de desambiguación semántica y el segundo sentido utilizando nuestro algoritmo (Fig. 9.4). Para obtener el primer sentido mediante desambiguación semántica se ha utilizado la herramienta *PYWSD* [17].

El tercer estudio que hemos realizado consiste en obtener el primer sentido del *pun* utilizando la herramienta *PYWSD*. El segundo sentido es el obtenido por nuestro algoritmo utilizando el *token* con menor *distancia_wp_wd* respecto al *pun*. Los resultados obtenidos utilizando la partición *dev* se muestran en la Tabla 9.19.

Resultados				
Embeddings	**Coverage**	**Precision**	**Recall**	**F1-score**
Google News	0.8046	0.0415	0.0334	0.037
Wikipedia	0.7969	0.0452	0.036	0.0401

Tabla 9.19: Estudio utilizando *PYWSD* y la distancia *distancia_wp_wd* en la tarea de interpretación

El último estudio realizado consiste en obtener el primer sentido del *pun* utilizando la herramienta *PYWSD*. El segundo sentido es el obtenido por nuestro algoritmo utilizando el *token* con menor distancia coseno respecto al *pun*. Los resultados obtenidos utilizando la partición *dev* se muestran en la Tabla 9.20.

Resultados				
Embeddings	**Coverage**	**Precision**	**Recall**	**F1-score**
Google News	0.8021	0.0481	0.0386	0.0428
Wikipedia	0.7943	0.0356	0.0283	0.0315

Tabla 9.20: Estudio utilizando *PYWSD* y la distancia coseno en la tarea de interpretación

Observamos a partir del estudio realizado que los mejores resultados obtenidos se basan en obtener los dos *tokens* mediante la distancia *distancia_wp_wd* y utilizando como *embeddings* el modelo entrenado con Google News. Se ha utilizado este criterio para evaluar el sistema con la partición de *test* donde los resultados obtenidos se muestran en la Tabla 9.21.

Resultados				
Dataset	**Coverage**	**Precision**	**Recall**	**F1-score**
Test	0.9618	0.1021	0.1008	0.1014

Tabla 9.21: Resultados tarea interpretación de *puns* sobre la partición de *test*.

Es interesante obtener los resultados que hubiésemos obtenido en la competición utilizando el estudio que hemos realizado. Para ello utilizamos el mejor sistema obtenido sobre el corpus completo (1298 frases) de la tarea. Los resultados se muestran en la Tabla 9.22.

Resultados			
Coverage	**Precision**	**Recall**	**F1-score**
0.9412	0.1024	0.1003	0.1013

Tabla 9.22: Resultados tarea interpretación de *puns* sobre el corpus completo de la tarea.

Se observa que aunque los resultados son mejores que los que obtuvimos en la competición la mejora no es significativa. Los bajos resultados de todos los participantes en la tarea de interpretación de *puns* nos aportan información acerca de su dificultad.

CAPÍTULO 10
Conclusiones

En el presente trabajo se han explicado los conceptos necesarios para poder abordar el conjunto de tareas relacionadas con *puns* que propone la competición SemEval-2017. Se ha explicado la importancia de resolver las tareas relacionadas con *puns* utilizando frases con sentido humorístico o retorico con la finalidad de realizar un estudio que mejore los sistemas capaces de detectar y/o generar humor de forma automática.

Los *embeddings* son la representación vectorial de palabras más utilizada actualmente para abordar problemas de procesamiento de lenguaje natural siendo uno de los pilares más importantes en los que se fundamenta el presente trabajo. Se han realizado estudios utilizando dos modelos de *embeddings* generados mediante la herramienta *Word2Vec* y entrenados a partir de la información de Google News y la Wikipedia. En todas las tareas se han realizado comparaciones entre los dos modelos observando que los resultados son muy similares debido a que los dos modelos están bien entrenados.

Para resolver la tarea de localización hemos mantenido la hipótesis que propusimos en la participación de la competición SemEval-2017. La hipótesis consiste en determinar el *pun* a partir de las dos palabras con mayor relación semántica dentro de la frase. Realizando una serie de estudios hemos sido capaces de mejorar aproximadamente 20 puntos los resultados que obtuvimos en la competición utilizando como métrica de similitud una combinación ponderada entre la distancia que hemos definido a partir de los lemas de los sentidos del par de palabras y la posición de la ultima palabra del par que es la que consideramos que es el *pun* en la frase. El mejor modelo de *embeddings* para resolver esta tarea ha sido el entrenado con Google News.

La tarea de detección la hemos abordado como un problema de clasificación de *Machine Learning* utilizando técnicas basadas en aprendizaje no supervisado y técnicas basadas en aprendizaje supervisado. Los mejores resultados que hemos obtenido han sido utilizando un clasificador *SVM* con *kernel* lineal con una $C = 0.8$ y el modelo de *embeddings* entrenado con la Wikipedia. Debido a que el corpus que proporciona SemEval-2017 para esta tarea no está balanceado no se pueden comparar los resultados obtenidos con los participantes de la competición.

La última tarea que hemos abordado en el presente trabajo corresponde a determinar los dos sentidos que tiene el *pun* en la frase. Esta tarea es la tarea más compleja de las tres viéndose reflejada su dificultad en los bajos resultados que obtuvimos todos los participantes en la competición. En esta tarea hemos realizado un estudio reutilizando el algoritmo que propusimos en la competición de SemEval-2017. La complejidad de la tarea se ve reflejada en la mejora que hemos obtenido respecto a los resultados que propusimos en la competición que aunque no sea una diferencia muy significativa ha sido todo un reto.

Los objetivos de nivel básico se han superado realizando la experimentación del presente trabajo donde los conceptos necesarios para llevarla a cabo se exponen en la memoria y la capacidad de utilizar un entorno de trabajo, utilizando las herramientas que se utilizan actualmente en PLN, queda reflejada en el código implementado que es accesible desde el siguiente enlace: https://github.com/PACarrascoGomez/TFM

Los objetivos de nivel intermedio y nivel avanzado relacionados con la capacidad de resolver las tareas propuestas en la competición de SemEval-2017 y ser capaces de mejorar los resultados que obtuvimos en la competición se han superado para la tarea de localización e interpretación.

La tarea de detección la hemos planteado balanceando el corpus y no podemos comparar los resultados con los resultados de la competición. En esta tarea los resultados que hemos obtenido consideramos que son buenos y pueden ser un *baseline* para trabajos futuros en los que se tenga en cuenta que el corpus este balanceado.

CAPÍTULO 11
Trabajo futuro

En el presente trabajo hemos decidido abordar únicamente los tipos de *pun* homográficos con la finalidad de centrarnos en un único tipo de problema e intentar obtener mejores resultados que los resultados que obtuvimos en la participación de la competición [8]. SemEval-2017 también ha proporcionado corpus con tipo de *puns* heterográficos para las subtareas propuestas en la *tarea 7: Detección e interpretación de puns en el idioma inglés*. En un futuro sería interesante utilizar las propuestas que se proporcionan en la experimentación del presente trabajo añadiendo la capacidad de consultar a bases de datos con información fonológica, para resolver los tipos de *puns* heterográficos.

El presente trabajo se centra en resolver *puns* que tienen un efecto humorístico o retorico en un texto. En un futuro se puede intentar reutilizar nuestro estudio con la finalidad de resolver problemas relacionados con otros recursos literarios cómo metáforas, sátiras, hipérboles, etc.

Nuestro grupo de investigación (ELiRF-UPV) es miembro del proyecto ASLP-MULAN [2]. Este proyecto propone una gran variedad de tareas de PLN a resolver. Muchas de estas tareas requieren de un análisis semántico que permita detectar palabras y expresiones que sean relevantes en la sentencia. Para las tareas que necesiten la detección de ironía y/o humor, la base de este trabajo puede ser relevante realizando las modificaciones necesarias a los métodos que hemos propuesto para resolver los nuevos problemas.

CAPÍTULO 12
Publicaciones

Lluís-F. Hurtado, Encarna Segarra, Ferran Pla, Pascual Carrasco, José A. González. ELiRF-UPV at SemEval-2017 Task 7: Pun Detection and Interpretation. 2017. [8]

Bibliografía

[1] Alfons Juan, Albert Sanchis, Jorge Civera. Reconocimiento de formas y aprendizaje computacional. MUIARFID, 2017.

[2] ASLP-MULAN. Audio, speech and language processing for multimedia analytics. http://www.aslp-mulan.com.

[3] Cedric De Boom, Steven Van Canneyt, Thomas Demeester, and Bart Dhoedt. Representation learning for very short texts using weighted word embedding aggregation. *CoRR*, abs/1607.00570, 2016.

[4] Alexander Budanitsky and Graeme Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Comput. Linguist.*, 32(1):13–47, March 2006.

[5] Cristian A. Cardellino. Representación de palabras mediante vectores. *Fundamentos de aprendizaje automático*, 2015.

[6] Enrique Vidal, Francisco Casacuberta. Aprendizaje automático. Grado de ingeniería informática, 2016.

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

[8] Lluís-F. Hurtado, Encarna Segarra, Ferran Pla, Pascual Carrasco, José A. González. Elirf-upv at semeval-2017 task 7: Pun detection and interpretation. In *Proceedings of the 11th International Workshop on Semantic Evaluation.*, SemEval '17, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[9] Rada Mihalcea and Carlo Strapparava. Learning to laugh (automatically): Computational models for humor recognition. *Computational Intelligence*, 22(2):126–142, 2006.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[12] Tristan Miller, Christian Hempelmann, and Iryna Gurevych. Semeval-2017 task 7: Detection and interpretation of english puns. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 59–69. Association for Computational Linguistics, August 2017.

[13] Tristan Miller and Mladen Turković. Towards the automatic detection and identification of english puns. *European Journal of Humour Research*, 4(1):59–75, January 2016.

[14] 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.

[15] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[16] Steven Bird, Ewan Klein, Edward Loper, Dan Garrette, Peter Ljunglöf, Joel Nothman, Mikhail Korobov, Steven Bird, Alexis Dimitriadis. The nltk project. http://www.nltk.org.

[17] Liling Tan. Pywsd: Python implementations of word sense disambiguation (wsd) technologies [software]. https://github.com/alvations/pywsd.

[18] Julia M. Taylor and Lawrence J. Mazlack. Computationally recognizing wordplay in jokes. In *In Proceedings of CogSci 2004*, pages 1315–1320, 2004.

[19] Princeton University. About wordnet., 2010. http://wordnet.princeton.edu.

[20] T. Yokogawa. Japanese pun analyzer using articulation similarities. *11th IEEE International Conference on Fuzzy Systems (FUZZ 2002)*, 2:1114–1119, 2002.

[21] Arnold M. Zwicky and Elizabeth D. Zwicky. Imperfect puns, markedness and phonological similarity. 1986.