Máster en Data Science

Curso académico 2017/2018

Trabajo de Fin de Máster:

Desambiguación de acrónimos clínicos

Autor: Javier Crespo Ortega

Tutor: Soto Montalvo Herranz

Índice general

1. Introducción **11**

2. Representación del texto **13**

2.1. TF-IDF 14

2.2. Embeddings 14

2.2.1. Obtención 15

2.2.2. t-SNE 16

3. Biomedical Abbreviation Recognition and Resolution **17**

3.1. Córpora 18

3.2. Enfoque adoptado 19

3.3. Creación de diccionarios 20

3.4. Localización de acrónimos 21

3.5. Desambiguación de acrónimos 22

3.5.1. Búsqueda de forma larga 22

3.5.2. No existe forma larga 23

3.6. Resultados 25

3.7. Fase I 25

3.8. Fase II 26

3.9. Fase III 26

3.9.1. Comentarios 27

4. Desambiguación mediante clasificación **29**

4.1. Dataset 29

4.1.1. Generación del Dataset 30

4.1.2. Web Scrapping 30

4.1.3. Composición del dataset 31

4.1.4. División del dataset 31

4.2. Validación y métricas de performance 32

4.2.1. Cross-Validation 32

4.2.2. Metricas de performance 33
4.3. Algoritmo Lesk 34
4.4. Coseno similarity 35
4.4.1. Coseno similarity TF-IDF 36
4.4.2. Coseno similarity Word2Vec 36
4.5. Aglomerative clustering 41
4.6. Naïve bayes 42
4.7. Support Vector Machines 43
4.8. Comentarios 45
4.8.1. Algoritmo Lesk vs Distancia Coseno 45
4.8.2. Codificación TF-IDF 46
4.8.3. Codificación Word2vec 46
4.8.4. TF-IDF vs Word2vec 47
4.8.5. Comparación acrónimo F1-Score 47

5. Sensegram **49**
5.1. Ego-graph 49
5.2. Word2vec y word Similarity Graph 50
5.3. Chinese Whispers y Community detection 51
5.3.1. Chinese Whispers 51
5.3.2. Community detection 51
5.4. Word Sense induction 52
5.5. Metodología 52
5.6. Resultados 53
5.6.1. PCR 53
5.6.2. Acrónimos L y T 56
5.6.3. Acrónimo AST 57

6. Conclusiones **59**

Índice de figuras

3.1. Ejemplo solución tarea BARR 18

4.1. Cross Validation 32
4.2. Confusion matrix 33
4.3. Representación Embedding Media PCR 38
4.4. SVM C TF-IDF 45
4.5. SVM C Word2Vec 45

5.1. Ego graph 50
5.2. PCR graph modelo I 53
5.3. PCR embeddings 55
5.4. PCR graph Modelo II 56
5.5. AST graph 57

Índice de cuadros

3.1. Resultados Fase I 26
3.2. Resultados Fase II 26
3.3. Resultados Fase III 27

4.1. Estadísticas dataset acrónimo T 31
4.2. Estadísticas dataset acrónimo PCR 31
4.3. Estadísticas dataset acrónimo L 31
4.4. Resultados del algoritmo lesk 35
4.5. Resultados del algoritmo lesk variando n 35
4.6. Resultados de la distancia coseno TF-IDF 36
4.7. Resultados de la distancia coseno Word2Vec 39
4.8. Resultados de la distancia coseno Word2Vec Decay 40
4.9. Resultados de la distancia coseno Word2vec Ventana 40
4.10. Resultados de la distancia coseno con reducción de cadena 41
4.11. Resultados de la Aglomerative clustering 42
4.12. Resultados del algoritmo Naive Bayes 43
4.13. Resultados del algoritmo SVM TF-IDF 44
4.14. Resultados del algoritmo SVM Word2Vec 44
4.15. Métrica F1 acrónimo PCR 47
4.16. Métrica F1 acrónimo L 48
4.17. Métrica F1 acrónimo T 48

5.1. Sensegram PCR Desambiguación Community detection 54
5.2. Sensegram PCR Desambiguación Chinese Whispers 54
5.3. Sensegram PCR f1-score 54

Resumen

El presente trabajo se centra en el estudio del problema de la desambiguación, un área en expansión dentro del procesamiento del lenguaje natural. Sus aplicaciones son muy variadas tanto en el terreno de la industria como en el de la investigación.

El estudio del proceso de desambiguación se centrará en la desambiguación de acrónimos en textos clínicos, para ello, se presentará una solución a la competición BARR2. Esta competición tiene como objetivo desarrollar un sistema automático de desambiguación de acrónimos en casos clínicos, debido al alto contenido de siglas que recogen este tipo de textos y a su alto grado de ambigüedad.

Además, sobre un grupo acotado de acrónimos se probarán y compararán distintas técnicas de clasificación, haciendo un repaso de técnicas tales como el Algoritmo Lesk hasta llegar al estado del arte de esta disciplina, la representación Word2Vec. También se estudiarán en este proceso algoritmos de machine learning tales como máquinas de vector soporte y Naïve-Bayes.

Aunque no obteniendo el mejor resultado, el ganador de este estudio como técnica de desambiguación es la distancia coseno, debido a su sencillez y rapidez. Bien es cierto que no es el proceso que mejor resultado obtiene, a la cabeza del proceso de desambiguación se encuentran las máquina de vector soporte con kernel lineal, pero debido a que han de ser entrenadas y parametrizadas para cada acrónimo hacen que sea un proceso imposible de llevar a cabo.

Por último, combinando la potencia de la representación Word2vec y la sencillez de la distancia coseno, se presenta un método no supervisado para la resolución de este proceso, Sensegram, que aun perdiendo precisión puede ser una técnica interesante.

Capítulo 1

Introducción

Diariamente se generan gran cantidad de documento clínicos, los cuales por motivos de ahorro de tiempo o con el fin de evitar repeticiones utilizan acrónimos o abreviaciones de términos clínicos. Estos acrónimos pueden llevar a confusiones, en mayor medida en el caso de ser ambiguos, como en el caso del acrónimo PCR cuyas desambiguaciones posibles son: Proteína C reactiva, Reacción en cadena de la Polimerasa y Parada Cardiorrespiratoria. Estas abreviaturas pueden llevar a confusiones incluso a profesionales del ámbito de la salud. Si pensamos en un sistema automático que clasifique los textos en función de su contenido o mantenga un registro de los mismos el problema se vuelve más interesante.

Partiendo de esta idea, se plantea la tarea BARR [1] [2] , cuyo objetivo es conseguir desarrollar un sistema automático de identificación y desambiguación de acrónimos clínicos. Esta competición cumple su segunda edición con lo que se apoya al crecimiento y desarrollo de este tipo de sistemas. En su primera edición el objetivo era detectar todos aquellos acrónimos cuya forma larga estuviese presente en el mismo [3], para la resolución de este año se aumenta la desambiguación de acrónimos a todos los que estén presentes en el texto, independeintemente de si aparece su forma larga en algún punto del documento o no. En el área de minería de texto, esta tarea se encuadraría dentro de las áreas de *Word Sense Disambiguation* (WSD) e *Retrival Information* (IR).

El presente trabajo pretende dar una solución a la tarea BARR2 [2], mostrando sus dificultades y avances intentando resolver el problema desde distintos puntos de vista. Además, pretende estudiar la solución a este tipo de problemas planteándolo como un sistema de clasificación utilizando herramientas de las áreas de *text classification*, utilizando para ello herramientas tales como la matriz TF-IDF y la representación Word2Vec.

Capítulo 2

Representación del texto

Para generar una representación de los textos se suelen aplicar diferentes técnicas lingüísticas, las más comunes son la tokenización y también lematización y stemming para reducir el tamaño del vocabulario y, por tanto, de la representación final.

Se conoce por tokenizacion la técnica donde el texto se divide en tokens, que pueden ser palabras o conjuntos de palabras o conjuntos de caracteres, etc. En ocasiones se eliminan lo signos de puntuación y las palabras vacías o stopwords. En este trabajo se obtendrán los tokens utilizando la librería NLTK de Python [4] .

El proceso de reducción del vocabulario mediante lematización consiste dada una forma flexionada obtener el lema correspondiente que representa por convenio a todas esas formas flexionadas. Un ejemplo de este método es el siguiente:

dije - decir
dire - decir

El proceso de reducción del vocabulario mediante steamming consiste en dada una palabra obtener su raíz en el caso del español o su stem el caso de la lengua inglesa, de ahí el nombre de esta técnica. Un ejemplo en español de esta técnica es el siguiente:

Bibliotecas - Bibliotec
Bibliotecario - Bibliotec

En este trabajo se trabajará con un corpus del dominio médico. Se denomina corpus a la colección de textos con la que se trabaja.

Debido al orgien de los textos y que son textos generados y anotados por distintos profesionales, nos podemos encontrar en situaciones en las que palabras o sentencias que no son exactamente iguales se refieren al mismo término.

Para controlar esta situación se ha optado por la utilización en Python del paquete difflib, su función SequenceMatcher. Para poder llevar esta comprobación a cabo, esta función nos

muestra matemáticamente como de iguales o de diferentes son dos cadenas de caracteres siendo el valor 1 el máximo, el cual identifica a dos cadenas exactamente iguales.

En muchas ocasiones para poder trabajar con los textos es necesario convertirlos en un ente matemático con el cual podamos trabajar. Existen distintas técnicas para llevar este proceso a cabo tales como: representación TF-IDF y representación mediante embeddings Word2vec.

2.1. TF-IDF

Es muy habitual representar los textos mediante vectores, de tal forma que primero se genera el vocabulario para todos los documentos del corpus. El vocabulario serían todos los tokens diferentes presentes en el corpus. Una vez se tiene el vocabulario, se genera un vector para cada documento, donde cada posición del vector representa uno de los tokens del vocabulario, por tanto, el valor de cada posición contiene el peso de ese token dentro del documento.

La matriz TF-IDF, *Term frequency– Inverse document frequency*, es una matriz documentos-términos, donde se unen todos los vectores que representan los documentos del corpus y donde el peso de cada componente viene dado por la medida TF-IDF. Esta medida tiene en cuenta la aparición de los términos en cada documento respecto al global de vocabulario en el corpus, pudiendo representar cómo de importante es una palabra para un documento en una colección. Un factor importante sobre esta matriz es que sus valores aumentan de manera proporcional al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia de la palabra en la colección de documentos.

2.2. Embeddings

En los últimos años ha habido una explosión en el área de *Natural Langage Processing* (NLP) en gran medida debida al avance en las técnicas de aprendizaje profundo, ya sea utilizadas para la obtener la representación del texto mediante técnicas como Word2vec y Glove o mediante la resolución de problemas de clasificación de textos con el uso de Convolutional neural networks (CNN) y Recurrent Neural Networks (RNN). En este problema nos centraremos en la representación Word2vec [6] dejando a un lado las arquitecturas neuronales como algoritmos de clasificación debido a que estas no se ajustan bien a nuestro problema, sin embargo, han demostrado una gran capacidad y un salto cualitativo a la hora de trabajar en el terreno del lenguaje natural.

Una de las técnicas actualmene más utilizada es la representación de las palabras mediante embeddings, Word2Vec, debido a los buenos resultados que ofrece. El output de esta

técnica es un vector de n dimensiones siendo n un número parametrizable. Este vector guarda información sobre las palabras que suelen aparecer cercanas a esta, es decir, su contexto. El número de palabras a tener en cuenta para la generación de los embeddings y el número de veces que una palabra debe aparecer en el texto para ser tomada en cuenta también son factores parametrizables, como veremos más adelante es necesario un ajuste en la validación de estos parámetros.

En los últimos años esta técnica ha tenido un gran impacto en NLP debido a las ventajas que presenta respecto de las representaciones *one-hot-encoding* [1]:

- Dense vector, no tenemos vectores de tipo sparse, con muchos ceros, lo cual desde el punto de vista computacional es una ventaja.
- Según aumenta nuestro corpus aumentan su eficacia.
- Nos dan información sobre el contexto.

2.2.1. Obtención

La potencia de los embeddings se basa en la asunción de que el significado de una palabra puede ser inferido por las palabras que la acompañan, para obtener esta representación debemos seguir los siguientes pasos:

- Entrenaremos una red neuronal de una sola capa oculta, con el objetivo de obtener dada una palabra cualquiera la probabilidad de que esta se encuentre dentro de una ventana espacial fijada.
- El proceso de "aprendijaze" consiste en conocer el número de veces que dentro de esta ventana temporal están apareciendo el par de palabras deseado. Para realizar esta tarea lo que desarrollamos es una red neuronal cuyo objetivo sea dada una entrada repetir lo mismo a la salida.
- ¿ Cómo introducimos la información en la red neuronal?
    - Fijamos el número de palabras de nuestro diccionario y representamos mediante un vector *one-hot vector* que sea 1 en la posición que representa a nuestra palabra en cuestión y cero el resto del vector.
    - No utilizamos función de activación en el hidden layer pero si usamos softmax para la neurona de salida.

[1] 'Se denomina representación *One-hot encoding* a aquella representación que representa cada palabra por un vector de longitud igual al vocabualrio, siendo 0 en todas las posiciones del mismo salvo la que se refiere a la palabra en cuestión que tiene un valor 1."

- Un parámetro importante en el modelo es la dimensión del vector que representará a cada palabra, se pueden fijar en distintos valores y para cada aplicación este valor debería ser contrastado. Como punto de partida elegiremos el valor de 100.

Se ha presentado una breve introducción a Word2vec, un análisis más intensivo de esta técnica se puede encontrar en [6].

Para el desarrollo de esta técnica existen librerías que implementan ya el proceso en forma de una función, en nuestro caso se utilizará una implementación sobre la librería Tensorflow [7] y la implementación de la función en la librería Gesim, una librería en Python desarrollada para el tratamiento de textos.

Es importante destacar que puede ser interesante para tareas concretas realizar un implementación de los embeddings con texto del mismo contexto u origen sobre los que se van a trabajar, en este caso con los textos clínicos. Existe también la opción de empezar utilizando unos embeddings ya entrenados y luego reentrenarlos con los propios textos. En ingles existen bastante recursos, Google tiene disponibles diferentes versiones de ellos.

2.2.2. t-SNE

T-distributed Stochastic Neighbor Embedding (t-SNE) es un algoritmo de machine learning desarrollado por Laurens van der Maaten y Geoffrey Hinton enfocado al terreno de la visualización [8]. La principal característica de este algoritmo es conseguir realizar una reducción de dimensionalidad no lineal, esta propiedad lo hace verdaderamente interesante frente a otros algoritmos en este campo, como puede ser PCA.

Debido a que nuestros embeddings tienen una alta dimensionalidad, mediante este algoritmo podemos realizar una visualización de los mismos. Además, visualmente podemos ver una distribución tipo cluster en la cual se puede encontrar el sentido a la técnica de los embeddings, palabras que aparezcan en contextos parecidos tenderán a aparecer juntas en esta visualización.

Es una técnica empleada a menudo y la visualización generada nos puede llegar a ofrecer mucha información.

Capítulo 3

Biomedical Abbreviation Recognition and Resolution

La resolucción de la tarea BARR2 [2], *Biomedical Abbreviation Recognition and Resolution 2nd Edition*, se centra en la desambiguacion automática de acrónimos en textos del ámbito clínico. El objetivo es conseguir un sistema automático que detecte y desambigue acrónimos. Esta competición cumple su segunda edición siendo en 2017 su primera disputa [1].

En la primera edición se buscaba encontrar y desambiguar acrónimos cuya forma larga se encontrara en el texto, para esta segunda edición se ha ampliado el rango, no solo se deben desambiguar aquellos cuya forma larga se encuentre en el texto sino todos los que estén presentes en el mismo. A continuación se presentan unos ejemplos de este problema:

"Se describe la relación entre diferentes factores de riesgo cardiovasculares (FRVC) y la obesidad a partir de una muestra representativa de la población adulta de Madrid."

En el ejemplo anterior el acrónimo "FRVC" aparece con su forma larga en la misma frase "factores de riesgo cardiovasculares", este tipo de situaciones son las más fáciles de detectar.

"Varón de 82 años que ingresa para realización de CPRE por un cuadro de colangitis aguda."

En este otro ejemplo no encontramos la forma larga correspondiente al acrónimo "CPRE" en el texto, por tanto, en función del contexto debemos apuntar su desambguación correcta, en este caso "colangio-pancreatografía retrógrada endoscópica".

El formato de la salida que deben generar los sitemas que se presenten a la tarea BARR2 se presenta en la Figura 3.1. A continuación, se describe brevemente el contenido de cada parte de la salida.

1. Document id: Identifica el número del documento, cada caso clínico tiene uno asignado.

2. StartOffset: Indica la posición del comienzo del acrónimo en el texto.

# Document_ID	StartOffset	EndOffset	Abbrev.	Definition	Definition_lemmatized
04-061420050010000	1034	1036	Kg	kilogramo	kilogramo
04-061420050010000	1031	1033	mg	miligramo	miligramo
04-061420050010000	196	199	IgA	inmunoglobulina a	inmunoglobulina a
04-061420050010000	2057	2060	LCR	líquido cefalorraquídeo	líquido cefalorraquídeo
04-061420050010000	1594	1598	EEII	extremidades inferiores	extremidad inferior
04-061420050010000	1009	1011	gr	gramo	gramo
04-061420050010000	982	984	Kg	kilogramo	kilogramo
04-061420050010000	979	981	mg	miligramo	miligramo
04-061420050010000	1963	1968	ELISA	enzyme-linked immunosorbent assay	enzyme-linked immunosorbent assay

FIGURA 3.1: *Ejemplo solución tarea BARR*

3. EndOffset: Indica la posición del finalización del acrónimo en el texto.

4. Abbreviation: Indica forma corta o abreviaturan identificada en el texto.

5. Definition: Identifica la definición porpuesta para el acrónimo.

6. Definition_lemmatized: Identifica la definición lematizada propuesta para el acrónimo.

3.1. Córpora

La organización es la encargada de proporcionar diferentes textos clínicos, que son en realidad casos clínicos reales además de sus metadatos, tanto para desarrollar el sistema como para la evaluación final del mismo. Un ejemplo de un caso clínico de la tarea es el siguiente:

"Se presenta un caso con patología litiásica compleja en el que se utilizó el 'stone sweeper'como derivación urinaria interna de forma bilateral. Se trata de un paciente varón de 54 años con 3 litiasis renales derechas, la mayor de ellas de 1.2 cm alojada en el cáliz inferior, que presenta además moderada dilatación pielocalicial y ureteral hasta el cruce con los vasos ilíacos, donde se localiza litiasis ureteral de 1.1 cm.; en esta unidad renal se colocó un catéter 'stone sweeper'previo al tratamiento con ESWL. En lado izquierdo se objetiva litiasis de 1 cm en polo inferior, al que se realizó tratamiento con ESWL con anterioridad, consiguiéndose fragmentación parcial de la misma; en esta unida renal también se colocó el catéter 'stone sweeper'previa a una segunda sesión de ESWL. El calibre de este catéter es de 6,5 Fr y se mantuvo durante el periodo de un mes. Durante la permanencia de los dos catéteres 'stone sweeperél paciente no refirió síntomas relacionados con el tracto urinario inferior ni superior indicando una buena tolerancia. La extracción de los mismos fue fácil, sin precisar de anestesia y observando pequeños fragmentos litiásicos dentro de las cestillas."

Para la creación de un sistema automático que lleve acabo este proceso se nos ofrecen dos conjuntos de textos y sus respectivas soluciones, el primer conjunto de textos denominado *train* compuesto de 318 casos clínicos y el conjunto *development* formado por 146 casos clínicos.

El conjunto de datos de *test* sobre el que se va a evaluar nuestro proceso será un subconjunto que esta formado por 220 textos, que a su vez están contenidos en otro conjunto mucho más grande, el conjunto *background* que está formado por 2879 casos clínicos.

Basándonos en estos textos se busca desarrollar un sistema automático de detección y desambiguación de acrónimos, a continuación se explicarán los detalles del mismo.

3.2. Enfoque adoptado

Debido a la casuística del problema en el enfoque adoptado se desarrollará un algoritmo rule-based que detecte acrónimos, es decir, que dado un texto pueda identificar si existe algún acrónimo y devolver el mismo con su posición en el texto, seguido de un proceso no supervisado de desambiguación. El proceso completo está dividido en cinco fases:

1. Localización de acrónimos en el texto.

2. Busqueda de la existencia de su forma larga o desambiguación en el texto.

3. División de los acrónimos cuyas formas largas no se hayan encontrado en el texto en dos grupos: únicos (aquellos acrónimos que solo tiene una desambiguación en el diccionario) y múltiples (aquellos acrónimos que tienen más de una desambiguación en el diccionario).

4. Al grupo de acrónimos únicos se les asignará la definición registrada en el diccionario.

5. Al grupo de acrónimos múltiples se les someterá a un proceso de desambiguación mediante las técnicas distancia coseno y Algoritmo Lesk.

Un punto importante en la tarea es la gran variedad de acrónimos que se encuentran en los textos, los cuales son muy complicados de cubrir en su totalidad. Otro punto importante a tener en cuenta para la resolución es que no existe una lista cerrada de acrónimos, es decir, en el conjunto de test pueden aparecer acrónimos que nunca se dieron en los conjuntos de *train* o *development* y, por tanto, no nos podemos ceñir a solo centrarnos en los acrónimos que ya han sido vistos con anterioridad.

3.3. Creación de diccionarios

Tal y como se ha comentado en el apartado anterior el sistema desarrollado para la detección y desambiguación de acrónimos tiene una gran dependencia de poder trabajar con un buen diccionario que guarde información de acrónimos y sus definiciones, la ventaja de este enfoque del problema es que suelen funcionar razonablemente bien si existe un diccionario diseñado con conocimiento del área.

En este trabajo no se ha buscado realizar un diccionario que sea completo o cubriese todos los casos posibles, se ha buscado integrar diferentes fuentes de información para comenzar a trabajar, debido a que no existe conocimiento previo alguno de los textos. Las fuentes elegidas han sido:

- Diccionario de siglas médicas en español creado por el doctor Yetano [5].
- Textos de la tarea BARR 2017.

Para cada acrónimo el diccionario tendrá guardados tres tipos de información o registros:

- Siglas.
- Definición o forma larga.
- Si existe un texto en el cual aparezca, este texto es necesario, como más adelante se verá, para la desambigucación de dicho acrónimo.

Se define la forma larga de un acrónimo como la desambiguación del mismo. A la hora de construir el diccionario, en el caso de que exista el mismo registro en ambas fuentes se dará mayor importancia al diccionario de siglas médicas. En la decisión de la toma de un texto como referencia para cada significado del acrónimo se utilizarán los de la tarea BARR 2017.

Para controlar que las definiciones en ambas fuentes no se dupliquen, se utilizará la función SequenceMatcher, presentada en el capítulo 2 de esta memoria, para comparar cadenas de caracteres de tal forma que si dos definiciones están por debajo de un límite establecido de forma automática se tomarán como diferentes, en caso contrario se establece que esa definición ya esta registrada, y por tanto, no se introduce.

En esta primera fase con el grupo de textos de *train*, vamos a actualizar nuestro diccionario con el fin de realizar mejores predicciones en un futuro. Si se ha encontrado un nuevo acrónimo que no existía en nuestra base de datos se incluirá, si se ha encontrado una definición nueva para un acrónimo se incluirá también. Para lidiar con el problema de la duplicidad de acrónimos se utilizará de nuevo la estrategia de comparar la cadena de caracteres. En cuanto a los textos

ejemplo, aquellos que se guardan en el diccionario para desambguar los acrónimos, se irán introduciendo de forma similar en nuestro diccionario.

Una vez que tengamos el diccionario actualizado, pasaremos a trabajar con los textos de *development*, realizando la misma actualización al final del proceso. Esta será la forma de mejorar el diccionario antes de afrontar la etapa de test.

Será de gran ayuda realizar un proceso de eliminación de *stop-words* sobre las siglas en el diccionario. Es posible que existan acrónimos que sean iguales a palabras comunes en Castellano, como el caso de *Se* que identifica a *Selenio*. Si no eliminamos estas siglas nos encontraremos con un sistema que tiene una gran tasa de falsos positivos.

3.4. Localización de acrónimos

Para la detección de acrónimos en el texto, se ha utilizado un algoritmo rule-based. Para que un token sea detectado como acrónimo no se tendrá en cuenta las palabras de alrededor y se limpiará el texto de los siguientes caracteres: (,),], [, % , ; , : , ® . Partiendo de esta situación inicial se han definido las siguientes categorías de tokens que serían clasificados como acrónimos:

- No tiene carazteres numéricos y cuenta con más de un (.) entre los caracteres que lo componen: Por ejemplo, T.A.C
- No tiene carazteres numéricos y está formado por más de un caracter en mayúsculas. Por ejemplo, EMG.
- No tiene carazteres numéricos, tiene longitud dos y sólo una mayúscula, eliminando de las posibilidades las stop-words. Por ejemplo, Na.
- Cuente con el carazter '/ ', puede ser que se detecte como acrónimio solo una parte de él o las dos. Por ejemplo, mmHg/mL o 2/L.
- Cuente con el carazter '- ', puede ser que se detecte como acrónimio solo una parte de el o las dos. Por ejemplo, B-CLL.
- Esté dentro del sistema internacional de unidades ["m","mm", "l", "L", "cm", "Kg", "g", "C", "ml", "mg", "dl", "gr", "U", "^{o}C", "h", "min", "seg", "ug", "T^{a}", "T", "dL", "u'l', "Km", "ppm", "mm3", "mmol", "m2", "mmHg", "mseg", "mEq", "pg", "ng"]

Siguiendo esta serie de reglas un acrónimo puede ser detectado como tal en varias de estas categorías, por tanto, es importante realizar una fase de filtrado para no tener desde el primer momento duplicados en nuestro sistema. Para dar la solución a la tarea no solo hay que

dar la definición del acrónimo y el acrónimo en cuestión, también el texto en el que aparece y la posición de inicio y fin. Con esta última información podemos controlar la duplicidad de los acrónimos encontrados.

3.5. Desambiguación de acrónimos

Una vez tenemos los acrónimos detectados debemos identificar a cada acrónimo con su desambiguación correcta, este será el segundo bloque del proceso global. Existen dos grupos de acrónimos a desambiguar: aquellos cuya forma larga esta presente en el texto y aquellos que no.

La primera línea de actuación será comprobar si esta forma larga existe, en ese caso se tomará esta como definición del acrónimo. En caso contrario, nos encontramos de nuevo ante dos situaciones posibles: el acrónimo solo tiene una definición posible o el acrónimo tiene varias definiciones. Si el acrónimo sólo tiene una posible definición se tomará esta como desambiguación del mismo, en caso contrario, se deberá recurrir a los textos de referencia para decidir qué desambiguación es la elegida.

3.5.1. Búsqueda de forma larga

Cuando el acrónimo es detectado se guarda una cadena de palabras o tokens alrededor de él, 10 en ambos sentidos, con el fin de poder desambiguarse. Debemos buscar en esta secuencia su forma expandida previamente habiendo eliminado las *stop-words* de dicha secuencia. Para llevar a cabo este proceso se han planteado las siguientes estrategias:

1. Realizamos grupos de n palabras, el valor de n coincidirá con la longitud del acrónimo, buscando si el primer carácter de cada una de las palabras coincide con el acrónimo, independientemente de su orden.

    - Un ejemplo de esta casuística sería *Reacción cadena polimerasa*, cuyas siglas se corresponden con *PCR* y no guarda el orden de las palabras. Sin embargo, para el mismo acrónimo tenemos la forma expandida, *proteina C reactiva* en el cual el orden sí es mantenido.

    Si nuestro acrónimo en cuestión pasa la condición anterior, estamos ante dos situaciones posibles: este acrónimo está registrado en nuestra base de datos o no:

    a) Si no esta registrado se aceptará esta definición como desambiguación del mismo, en caso contrario, se chequeará en la base de datos y si esta definición esta catalogada, se utilizará para dicho chequeo el método de comparación de cadena de carazteres.

b) Si la definición encontrada no coincide con ninguna de las registradas esta no se considerará como válida.

2. Existe también la situación de que un acrónimo formado por n caracteres esté desambiguado por una cadena de carazteres que sea menor que n.

   - Un ejemplo de esta casuística sería *EMG* que teniendo una longitud de tres caracteres solo se desambigua con una sola palabra *electromiograma*.

   Debemos generalizar este ejemplo haciendo posible que con que una de las n palabras cumpla la condición de coincidir con alguna del acrónimo se debe chequear esa posibilidad. En ese caso, se buscará su definición en el diccionario y de nuevo si esta existe o la comparativa sobre la cadena de caracteres la cataloga a la misma, se aceptará como desambiguación del mismo.

Terminado este proceso, todos los acrónimos que no hayan sido identificados, pasarán a la siguiente fase.

3.5.2. No existe forma larga

En el caso de que la forma larga no exista en el texto o no haya sido encontrada, debemos realizar un proceso de desambiguación basado en la secuencia extraída del texto original. Estamos en este momento ante tres situaciones posibles:

1. Si el acrónimo no se encuentra en la base de datos, no se puede dar significado y por tanto, se dará por no válido. Esta es una situación por la cual tener un diccionario bien confeccionado cobra gran importancia.

2. En el caso de que si tenga registro en nuestra base de datos y solo existe una posible desambiguación, se le asignará esta.

3. En el caso de que exista más de una definición posible se tomarán en cuenta dos procesos para la desambiguación, algoritmo Lesk y Distancia coseno.

Algoritmo Lesk

El algoritmo Lesk es un método Word Sense Disambiguation [9], este algoritmo esta basado en comparar el texto en el cual aparece la palabra que se busca desambiguar respecto de la definición de dicha palabra.

La aproximación que se ha decidido realizar para esta tarea es comparar el texto completo guardado en nuestro diccionario respecto de la cadena extraída del texto para la desambiguación del acrónimo. Sobre ambos textos se realiza una limpieza y un proceso de stemming

para reducir el vocabulario. Cada coincidencia aumenta $+1$ al valor de ese significado, al final del proceso se compararán las puntuaciones de cada categoría y aquella que obtenga el valor más alto será la definición elegida.

Unos de los problemas que nos enfrentamos en este método es la elección del texto de comparación, debido a que no tenemos una definición oficial para cada acrónimo. Aquí es donde vuelve a aparecer la importancia del conocimiento de la rama que se va a estudiar, ya que podemos elegir un texto que sea una excepción o no sea representativo de dicho acrónimo.

Coseno similarity

El segundo método propuesto de desambiguación consiste en el uso de la distancia coseno [10]. El primer paso es convertir los texos en vectores mediante la utilización de la matriz TF-IDF, una vez nos encontramos en esa situación podemos definir y aplicar la métrica disancia coseno.

Matemáticamente la distancia coseno se define como la medida de la similitud existente entre dos vectores en un espacio que posee un producto interior con el que se evalúa el valor del coseno del ángulo comprendido entre ellos. El valor de la similitud coseno esta definido entre [-1,1].

Si los vectores son ortogonales, forman 90^o entonces el valor de esta cantidad es cero, si el ángulo comprendido entre ellos es 0^o entonces el valor de esta métrica sería 1, caso de estar formando un ángulo de 180^o el valor de la métrica sería -1.

Sean *u* y *v* dos vectores definidos en un espacio que posee producto interno, se define la distancia entre ambos mediante la siguiente ecuación:

$$D_c(u, v) = 1 - \frac{u \cdot v}{||u||_2||v||_2}$$

El interés en este estudio por esta métrica viene generado debido a que una vez convertido el texto en un vector, esta medida nos compara de forma efectiva cómo de similares son dos textos. Cuanto menor sea esta métrica, mayor será el valor de la similitud coseno y por tanto, mayor debería ser el parecido entre los textos.

Una vez que el texto ha sido representado en forma de vector se calcula la distancia coseno del texto a las diferentes definiciones, siendo la que menor distancia coseno tenga la que se tomará como definición. Surge de nuevo en la utilización de esta técnica el problema de la elección del texto.

3.6. Resultados

Se presentan diferentes aproximaciones a la solución con el fin de ver cómo va mejorando nuestro sistema. El total de los acrónimos mostrados se dividen en tres grupos diferentes, aquellos para los cuales se ha encontrado la posición correcta en el textos, aquellos acrónimos cuya posición es incorrecta y los acrónimos que no están reguistrados en el diccionario.

Si un acrónimo está localizado correctamente en el texto, nos referiremos a este grupo de soluciones como soluciones tipo 1, puede estar en tres categorías:

- Todos los campos son correctos, grupo A.
- La desambiguación no es exactamente la misma, pero se refiere al mismo término, Grupo B.
- La desambiguación es incorrecta, grupo C.

Si un acrónimo no está localizado correctamente en el texto, nos referiremos a este grupo de soluciones como soluciones tipo 2, puede estar en tres categorías:

- El resto de campos de la solución son correctos, Grupo A.
- La desambiguación no es exactamente la misma, pero se refiere al mismo término, Grupo B.
- La desambiguación es incorrecta Grupo C.

Para que un acrónimo sea del grupo B, su comparación mediante la función SequenceMacher debe obtener un resultado igual o mayor a 0,7. Antes de pasar a la evaluación es conveniente saber que el dataset contiene 3414 acrónimos a desambiguar.

3.7. Fase I

No utiliza el diccionario generado para la localización de acrónimo, solo los pasos presentados en la sección 3.4 de la memoria y se estudian los tres procesos: formas largas, únicos y múltiples. Utilizamos para el proceso de desambiguación el diccionario de partida. Se presentan los resultado en la tabla 3.1.

Aproximación	Grupo A	Grupo B	Grupo C
Formas largas 1	62	7	17
Únicos 1	111	130	204
Únicos 2	0	21	66
Lesk 1	27	300	433
Lesk 2	0	6	80
Distancia coseno 1	26	355	196
Distancia coseno 2	0	10	29

TABLA 3.1: *Resultados Fase I*

3.8. Fase II

No utiliza el diccionario generado para la localización de acrónimo, solo los pasos presentados en la sección 3.4 de la memoria y se estudian los tres procesos:formas largas, únicos y múltiples. Utilizamos para el proceso de desambiguación el diccionario mejorado como se ha especificado anteriormente. Se presentan los resultado en la tabla 3.2.

Aproximación	Grupo A	Grupo B	Grupo C
Formas largas 1	63	22	8
Únicos 1	66	12	21
Únicos 2	0	1	6
Lesk 1	959	346	219
Lesk 2	0	61	115
Distancia coseno 1	812	309	197
Distancia coseno 2	0	57	121

TABLA 3.2: *Resultados Fase II*

3.9. Fase III

Utilizamos el diccionario para detectar los acrónimos. Además de las reglas introducidas en la sección 3.4, si un token esta reguistrado en nuestro diccionario automáticamente se catalogará como acrónimo, y no evaluamos buscamos formas largas, solo dividimos en únicos y múltiples. Utilizamos para el proceso de desambiguación el diccionario mejorado como se ha especificado anteriormente. Se presentan los resultado en la tabla 3.3.

Aproximación	Grupo A	Grupo B	Grupo C
Únicos 1	4	27	34
Únicos 2	0	0	11
Lesk 1	1119	430	305
Lesk 2	0	92	423
Distancia coseno 1	1077	410	301
Distancia coseno 2	0	90	359

TABLA 3.3: *Resultados Fase III*

3.9.1. Comentarios

En la resolución de la tarea existen dos grandes dificultades a superar para obtener un resultado notable en la resolución de la misma:

1. Por una parte, detectar el mayor número posible de acrónimos sin caer en muchas situaciones de falsos positivos y calcular correctamente su posición en el texto.

2. La creación de un diccionario que abarque el mayor número de casos evitando en todo momento las duplicaciones de desambiguaciones. Además, el diccionario debe tener unos textos de referencia sobre los que se pueda realizar una serie de buenas comparaciones en la tarea de desambiguación.

El mayor problema en esta etapa viene en la comparación de las anotaciones médicas con la desambiguación predichas. Este problema viene motivado debido a que en el diccionario las formas largas de los acrónimos pueden que no se ajusten al cien por cien con la desambiguación que ha sido anotada en la tarea, incluso en algunos casos dos acrónimo iguales, con la misma forma larga, han sido anotados de forma distinta e incluso en idiomas distintos. El caso de *Reacción en cadena de la polimerasa* y *Polimerase chain reaction* es un claro ejemplo de esta casuística.

Por otra parte, esta solución es una solución estática, una vez este bien calibrada puede llegar a funcionar muy bien para el conjunto cerrado de acrónimos. Si se quiere ajustar para abarcar más acrónimos, el diccionario debería ser actualizado.

Los resultados mejoran notablemente cuando mejoramos el diccionario con los textos que nos van llegando, sin embargo, el resultado obtenido respecto del total de la tarea no es bueno. Uno de los problemas de este enfoque es que el acrónimo debe estar registrado en el diccionario o ser detectado en la fase de formas largas, ya que en caso contrario aunque sea detectado no se puede desambiguar.

En la comparación entre la fase II y fase III, tabla 3.2. y tabla 3.3. respectivamente, la fase III es capaz de encontrar más acrónimos, sin embargo, también tiene tasas de errores muy

superiores, por tanto, la fase elegida como mejor resultado será la fase II.

Respecto a la elección de un método de desambiguación , este estudio no arroja un resultado definitivo, en el siguiente capítulo se tratará este tema en detalle.

Capítulo 4

Desambiguación mediante clasificación

Debido a la naturaleza de la tarea a resolver, se hace complicado obtener un resultado óptimo con un enfoque de machine learning supervisado por el número escaso de textos para entrenar cada uno de los acrónimos y a tener una solución abierta (en el conjunto de datos de test encontramos acrónimos que no han aparecido antes). Por tanto, para una resolución eficaz de la tarea se ha decidido enfocarla como un proceso *rule-based* y *unsupervised learning*.

En este capítulo se busca realizar una comparación sobre un conjunto acotado de acrónimos de los métodos utilizados en el capítulo anterior con algunos algoritmos de clasificación. Para ello, se cotejarán los sistemas utilizados en la tarea, Algoritmo Lesk y distancia coseno, con los algoritmos supervisados SVM y Naive Bayes y no supervisados como el clustering aglomerativo.

Para realizar la representación del texto se estudiarán los dos métodos presentados al comienzo de este documento TF-IDF y Word2Vec, además de explorar la utilización de la representación embedding en el área de Word Sense Disambiguation.

En los últimos años las redes neuronales tales como CNN y RNN están siendo el estado del arte de esta disciplina [11], pero debido a la cantidad limitada de datos con la que trabajamos no se ajustarán bien a la resolución del problema.

4.1. Dataset

La limitación en el número de ejemplos anotados para entrenar y validar un clasificador nos lleva a elegir un pequeño conjunto de acrónimos que cumplan un mínimo de apariciones y tengan más de una posible desambiguación. Después de realizar un estudio se eligieron los siguientes acrónimos: L, PCR y T. Para la generación de este dataset se deben tener en cuenta las siguientes consideraciones:

- Las distintas posibles definiciones que pueden tener nuestro acrónimo, L y PCR tienen

3 posibles definiciones y en el caso de T llegamos a tener 5 posibles definiciones.

- La descompensación en las posibles desambiguaciones de cada acrónimo, algunas son mucho más comunes que otras. Es un problema desbalanceado.
- Son lo suficientemente comunes como para tener un conjunto amplio de trabajo con ellos.

4.1.1. Generación del Dataset

Como se comentó en los textos de la tarea no tenemos un conjunto suficientemente amplio para entrenar, por esta razón se decide tomar en cuenta las siguientes consideraciones:

- Cambiar las formas largas de los acrónimos por sus formas cortas (el propio acrónimo). Partimos tomando cierta la hipótesis de que las palabras próximas al acrónimo o a su forma larga no varían en función de la utilización de una u otra. Esta estratégia es utilizada en estudios del mismo ámbito.
- Utilizamos los textos de la tarea BARR 2017 para aumentar nuestro conjunto de datos.
- Utilizamos los textos completos que se han obtenido mediante web scraping de la tarea 2018, más adelante se detalla cómo se realiza.
- Se tomarán distintas cadenas de caracteres alrededor del acrónimo para poder ser desambiguado. Tomaremos cadenas de 10, 20, y 30 tokens o palabras alrededor del acrónimo en cuestión si es posible, por lo tanto, trabajaremos con textos de extensión 61 , 41 y 21 tokens.

El punto fundamental para que este sistema funcione es que la primera condición se cumpla, en caso contrario nuestros algoritmos de clasificación no funcionarían correctamente. Sólo tenemos acrónimos anotados de los textos correspondientes a la tarea BARR 2018, por tanto, sobre el resto de textos obtenemos más ejemplos mediante la sustitución de las formas largas.

4.1.2. Web Scrapping

En la resolución de la tarea BARR2 se nos brinda un conjunto de soluciones donde viene registrada la dirección de la que se pueden encontrar los textos completos de los cuales han sido extraídos nuestros textos de trabajo.

Mediante un proceso de Web Scrapping se ha realizado la descarga de los documentos al completo. Los documentos de la tarea se corresponden a la introducción del caso clínico, por tanto, puede ser de utilidad el resto del texto para la generación de un dataset para la clasificación.

4.1.3. Composición del dataset

Finalmente, la composición de los datasets para los acrónimos T, PCR y L se presentan, respectivamente, en las Tablas 4.1, 4.2 y 4.3.

Acrónimo	Desambiguación	Número
T	Onda t	11
T	Tubular	994
T	Thyminar	1
T	Temperatura	2858
T	Tesla	19

TABLA 4.1: *Estadísticas dataset acrónimo T*

Acrónimo	Desambiguación	Número
PCR	Parada cardiorrespiratoria	669
PCR	Proteina C reactiva	113
PCR	Reacción en cadena de la polimerasa	1673

TABLA 4.2: *Estadísticas dataset acrónimo PCR*

Acrónimo	Desambiguación	Número
L	Litro	115
L	Leucocito	59
L	Linfocito	114

TABLA 4.3: *Estadísticas dataset acrónimo L*

Como se puede apreciar en las Tablas 4.1, 4.2 y 4.3 tenemos un importante desbalanceo entre las distintas desambiguaciones de los acrónimos, este será un dato a tener en cuenta en el proceso de desambiguación.

4.1.4. División del dataset

Antes de comenzar a trabajar debemos dividir el conjunto en train y test con el fin de poder asegurar la calidad y veracidad del modelo y evitar situaciones de overfitting. La partición de los datos tomará los porcentajes de 80 % en train y el 20 % restante en test, debido a los pocos datos de los que disponemos no se pueden reservar más datos para la fase de test.

4.2. Validación y métricas de performance

En esta sección antes de comenzar con el estudio de los modelos y la soluciones aportadas, evaluaremos distintas posibilidades de medir la performance de nuestro algoritmo (*F1-Score* , *Accuracy* y *Adjusted Rand Score*) y una técnica de validación y ajuste (*Cross-Validation*)

4.2.1. Cross-Validation

Se denomina Cross-Validation a la técnica utilizada para evaluar el resultado y realizar una hiperparametrización de un modelo de machine learning, con el fin de evaluar que su resultado o su eficacia no este sesgada por los datos con los cuales el modelo haya sido entrenado.

Para llevar a cabo esta técnica debemos realizar los siguientes pasos (ilustrados de alguna forma en la Figura 4.1):

- El total de casos disponibles se dividen en k grupos diferentes.
- Realizamos k iteraciones de ajuste y validación del modelo, en cada iteración se usa uno de los grupos de observaciones para validación utilizando el resto para ajustar el modelo.
- La estimación del error ”realista” cometido por el modelo será el promedio de los errores calculados en cada iteración.

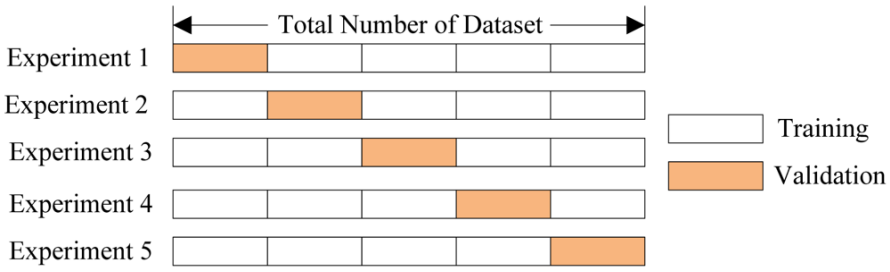

FIGURA 4.1: *Cross Validation*

Si el valor de k es igual a n, siendo n el número total de datos estaremos utilizando la versión *leave-one-out cross-validation (LOOCV)*, en caso de que k sea menor que n estaremos realizando *k-folds Cross-Validation*.

Normalmente se utiliza *k-folds Cross-Validation* más a menudo que *LOOVC* debido a que el proceso de ajuste y de entrenamiento de un modelo puede llegar a ser muy costoso.

Una vez realizado el modelo, el error promedio realista se calculará basándose en la siguiente ecuación:

$$CV(k) = \frac{1}{k}\sum_{i=1}^{k} MSE(i)$$

4.2.2. Metricas de performance

En esté apartado estudiaremos las dos métricas que se van a utilizar para cuantificar el funcionamiento del los modelos predictivos supervisados, estas son *F1-Score* y *Accuracy*. Para explicar estas cantidades nos valemos de la matriz de confusión, representada en la Figura 4.2, por simplicidad se presenta en su versión binaria $0, 1$:

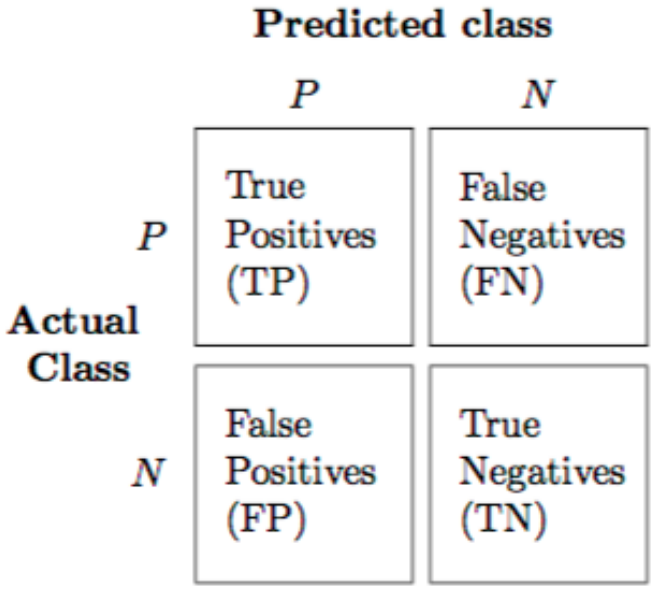

FIGURA 4.2: *Confusion matrix*

Los cuatro conceptos que se manejan en esta matriz son los siguientes:

- **TN**: Aquellos valores predichos como 1 siendo su verdadera clase 1.
- **FN**: Aquellos valores predichos como 1 siendo su verdadera clase 0.
- **TP**: Aquellos valores predichos como 0 siendo su verdadera clase 0.
- **FP**: Aquellos valores predichos como 0 siendo su verdadera clase 1.

Accuracy

Partiendo de la matriz de confusion, se define la métrica accuracy:

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

F1-Score

La medida F1-Score se puede interpretar con una media entre la precisión y recall, el rango de valores de esta métrica esta entre $[0, 1]$, siendo 1 su mejor valor.

$$F1 - Score = 2\frac{precision * recall}{precision + recall} = \frac{2TP}{2TP + FP + FN}$$

Adjusted rand score

Está métrica calcula la similaridad entre dos clusters mediante la ecuación:

$$ARI = \frac{RI - Expected(RI)}{max(RI) - Expected(RI)}$$

Por tanto, para evaluar esta métrica debemos introducir la formación real del cluster para poder contrastar los resultados. Los posible valores se encuentran cercanos a 0 cuando la clasificación es aleatoria y es exactamente 1 cuando los clusterings son identicos.

4.3. Algoritmo Lesk

Como primera técnica de desambiguación realizaremos una implementación del Algoritmo Lesk, como paso previo se utilizará la técnica de Stemming para realizar una reducción de vocabulario.

A la hora de comparar esta desambiguación no se realizará sobre un solo texto de referencia para evitar información sesgada, elegir un texto que no represente correctamente al acrónimo. Con todos nuestros textos de train vamos a calcular para cada clase un *word-count* y quedarnos solo el *top n* de palabras por clase, las cuales se utilizaran para comparar el solapamiento. En caso de que en algún texto no se encuentre ninguna palabra de solapamiento sobre ninguna categoría, se clasificara este texto como correspondiente a la clase mayoritaria.

Se ha decidido utilizar diferentes valores de n con el fin de ver como varía el resultado, escogiendo los valores del top 30 palabras provenientes del proceso *word-count*. Los resultados obtenidos con este algoritmo se presentan en la Tabla 4.4.

Acrónimo	Longitud de cadena	Accuracy
PCR	30	0.801
PCR	20	0.761
PCR	10	0.584
L	30	0.448
L	20	0.482
L	10	0.435
T	30	0.502
T	20	0.506
T	10	0.632

TABLA 4.4: *Resultados del algoritmo lesk*

Basándonos en esta tabla vamos a elegir la longitud de cadena qué mejor resultado ha obtenido, comparando que resultado obtiene en el caso de variar n sobre los siguientes valores: 50, 60 y 80. Los resultados obtenidos con este algoritmo se presentan en la Tabla 4.5.

Acrónimo	Longitud de cadena	Top n	Accuracy
PCR	30	30	0.801
PCR	30	50	0.871
PCR	30	60	0.882
PCR	30	80	0.808
L	30	30	0.448
L	30	50	0.534
L	30	60	0.603
L	30	80	0.655
T	10	30	0.632
T	10	50	0.751
T	10	60	0.778
T	10	80	0.807

TABLA 4.5: *Resultados del algoritmo lesk variando n*

4.4. Coseno similarity

En este apartado la técnica de clasificación estará basada en el calculo de la distancia coseno al vector de referencia de cada uno de los posibles significados.

Para crear dicho vector de referencia se realizará la media de todos los vectores correspondientes a esa desambiguación en el grupo de train. Aunque es una forma sencilla en esencia encaja con el problema que intentamos resolver, encontrar el significado más cercano a nuestro dato a desambiguar.

Por tanto, dado un nuevo acrónimo a desambiguar se deberá calcular la distancia coseno respecto de todos los vectores de referencia y se escogerá aquella definición que obtenga menor distancia coseno.

4.4.1. Coseno similarity TF-IDF

Una vez tenemos el conjunto de datos dividido en train y test, tomamos el conjunto de train y calculamos los vectores correspondientes a cada texto creando asi la transformación para obtener la matriz TF-IDF. Con todos los vectores calculados realizamos la media para cada desambiguación posible, es decir, generamos el vector referencia de cada desambiguación.

Aplicamos la transformación fijada en la fase de train a nuestros textos de test y calculamos para cada uno de ellos la distancia coseno a los vectores referencia, eligiendo como desambiguación aquella que menor valor obtenga en esta métrica. Los resultados obtenidos en este proceso se presentan en la Tabla 4.6.

Acrónimo	Longitud de cadena	Accuracy
PCR	30	0.963
PCR	20	0.964
PCR	10	0.813
L	30	0.931
L	20	0.946
L	10	0.826
T	30	0. 950
T	20	0.942
T	10	0.849

TABLA 4.6: *Resultados de la distancia coseno TF-IDF*

El mejor resultado a nivel general se obtiene sobre las cadenas de 41 tokens sobre las cuales la matriz TF-IDF es capaz de identificar de una forma bastante exacta la desambiguación correcta del acrónimo.

4.4.2. Coseno similarity Word2Vec

Debido a la potencia que tiene la representación embeding y su capacidad de expresar matemáticamente el contexto, mediante la técnica Coseno Similarity se pueden llevar a cabo diferentes soluciones al problema. Antes de comenzar con las soluciones presentamos cómo se han generado los dos modelos de embeddigns a utilizar en este apartado.

Generación

Obteniendo diferentes fuentes de textos del ámbito médico, textos de la propia tarea y fuentes derivadas de Web Scraping de artículos de la Wikipedia, se han generado dos modelos de embeddigns con diferentes parámetros con el fin de ver qué impacto tienen estos parámetros en el funcionamiento y el número de información sobre nuestro modelo [12].

1. **Modelo I**: Se han generado embeddings respecto de un corpus con una longitud cercana a los 34 millones de palabras. La dimensionalidad de los vectores toma el valor de 100, la ventana se ha fijado en 5 palabras en cada sentido y el mínimo número de apariciones para tener en cuenta a una palabra se ha fijado en 5. Con estos parámetros, formamos un vocabulario con 88766 términos.

2. **Modelo II**: Se han generado embeddings respecto de un corpus con una longitud cercana a los 34 millones de palabras, coincidiendo con el copus utilizado en el modelo I. La dimensionalidad de los vectores toma el valor de 100, la ventana se ha fijado en 10 palabras en cada sentido y el mínimo número de apariciones para tener en cuenta a una palabra se ha fijado en 2. Con estos parámetros, formamos un vocabulario con 129163 términos.

Media embeddings

Realizamos el mismo proceso de clasificación mediante la distancia coseno aunque ahora mediante la codificación basada en Word2Vec. En este caso para construir un vector que represente a un texto, o en nuestro caso a una frase. Se va a realizar mediante el cálculo de la media, ya que cada palabra está representada por un vector.

$$SBE(w) = \frac{1}{n} \sum_{i=0}^{n} Emb(i)$$

A modo de ilustración ilustración (ver Figura 4.3) vamos a comprobar cómo han quedado nuestros textos en el espacio generado por el algoritmo t-SNE tomando como punto de partida la media de los embeddings.

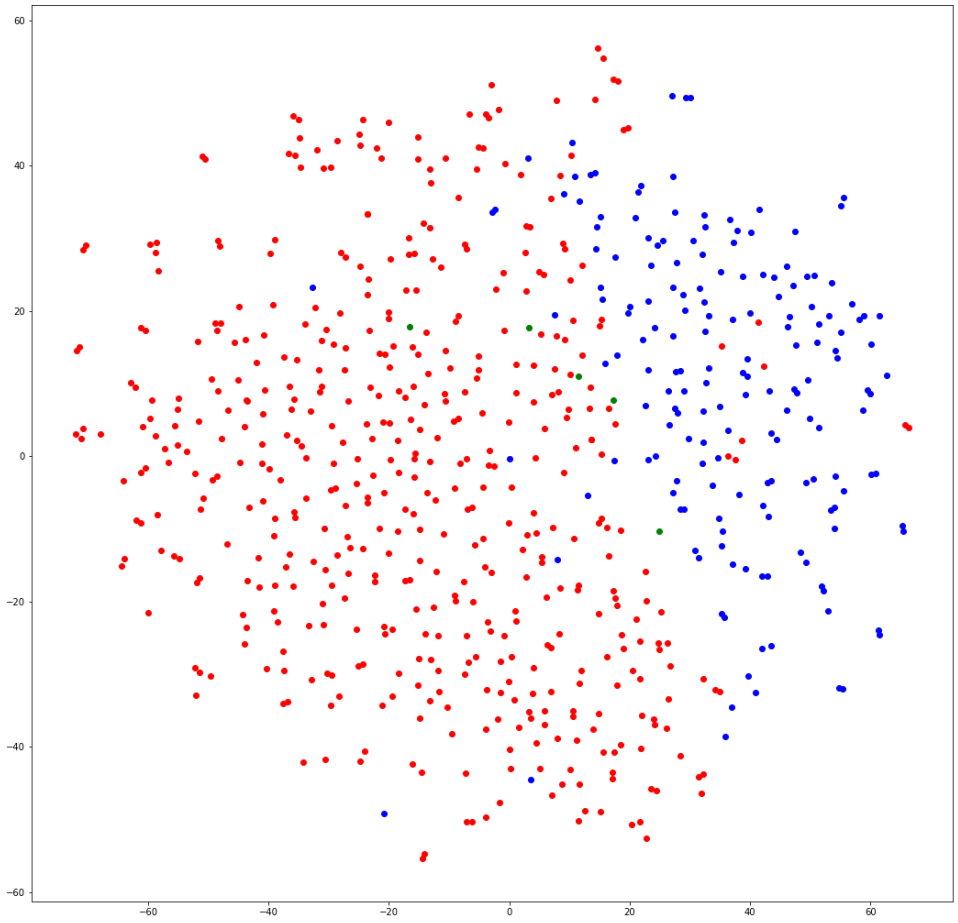

FIGURA 4.3: *Representación Embedding Media PCR*

Como vemos en la figura 4.3, el grupo ”rojo” es ampliamente mayoritario, la distribución es a modo de clusters entre las clases, salvo en alguna excepciones. Sin embargo, estos clusters sí tienen puntos de contactos, fronteras difíciles de resolver mediante un algoritmo del tipo clustering. En este caso, las cadena cortas de 21 tokens son las que mejor resultado obtienen, sin embargo, el resultado obtenido es inferior a los obtenidos mediante la matriz TF-IDF, más adelante explicaremos en detalle este fenómeno. Los resultados se muestran en la tabla 4.7.

Acrónimo	Longitud de cadena	Accuracy modelo 1	Accuracy modelo 2
PCR	30	0.781	0.797
PCR	20	0.809	0.806
PCR	10	0.909	0.903
L	30	0.775	0.775
L	20	0.660	0.625
L	10	0.708	0.708
T	30	0.702	0.810
T	20	0.646	0.665
T	10	0.780	0.810

TABLA 4.7: *Resultados de la distancia coseno Word2Vec*

Weighted media embeddings

Existe también otras formas de generar el vector correspondiente a la frase completa, en este caso exploramos introducir un factor de decaimiento, de tal manera que cuanto más alejado esté la palabra de la posición del acrónimo, menor peso tendrá. Este efecto local se va a introducir ponderando la media en función de la distancia al acrónimo usando para este proceso la siguiente ecuación:

$$SBE(w) = \frac{1}{n}\sum_{i=0}^{n}\frac{x - d(i)}{x}Emb(i)$$

Siendo n la longitud de la frase, x la posición del acrónimo a desambiguar en la frase y $d(i)$ la distancia de la palabra i al acrónimo a desambiguar.

Como la representación Word2Vec genera la codificación basándose en una ventana de palabras alrededor de la palabra a codificar, tiene sentido que cuanto más cercana esté una palabra al acrónimo a desambiguar mayor sera la importancia de esa palabra debido al efecto local generado por esta representación.

Los resultado obtenidos por este nuevo proceso se presentan en la Tabla 4.8.

Acrónimo	Longitud de cadena	Accuracy Modelo 1
PCR	30	0.783
PCR	20	0.806
PCR	10	0.909
L	30	0.775
L	20	0.642
L	10	0.666
T	30	0.687
T	20	0.648
T	10	0.798

TABLA 4.8: *Resultados de la distancia coseno Word2Vec Decay*

Ventana de contexto embeddings

Otra opción que surge de la posibilidad de tener un vector asignado a cada palabra es realizar la desambiguación del texto basándonos solo en las palabras más extremas del mismo. Denominamos palabras extremas a aquellas cuya distancia coseno a los acrónimos de referencia se maximiza entre las diferentes clases.

Para llevar a cabo este proceso calcularemos nuestros vectores de referencia y a la hora de obtener la desambiguación, calcularemos la distancia coseno a cada una de las palabras de la frase, obteniendo para cada frase una distancia máxima y una distancia mínima. Aquella categoría que maximize la resta de ambas cantidades será la elegida como desambiguación.

Realizaremos este proceso sobre las ventanas de menor longitud, ya que han sido las que mejores resultados obtuvieron en las fases anteriores.

Acrónimo	Longitud de cadena	Accuracy modelo 1	Accuracy modelo 2
PCR	10	0.301	0.240
L	10	0.130	0.130
T	10	0.002	0.022

TABLA 4.9: *Resultados de la distancia coseno Word2vec Ventana*

Comparación longitud de ventana

En todo este proceso de desambiguación mediante la distancia coseno, se ha visto como variaba el accuracy de nuestro algoritmo en función de la longitud de la cadena de tokens. El máximo valor se obtenía con 10 tokens en ambos sentidos, es decir, una cadena de 21 tokens. Esta cadena puede qué sea incluso demasiado larga, vamos a ver si vamos reduciendo el tamaño de esta ventana que efecto tiene sobre nuestro poder predictivo.

Acrónimo	Longitud de cadena	Accuracy TF-IDF	Accuracy modelo 1	Accuracy modelo 2
PCR	10	0.813	0.909	0.903
PCR	8	0.765	0.909	0.903
PCR	6	0.759	0.909	0.921
PCR	4	0.759	0.897	0.897
T	10	0.849	0.780	0.810
T	8	0.832	0.773	0.807
T	6	0.832	0.795	0.815
T	4	0.827	0.788	0.795
L	10	0.826	0.708	0.708
L	8	0.739	0.75	0.708
L	6	0.739	0.666	0.666
L	4	0.782	0.541	0.625

TABLA 4.10: *Resultados de la distancia coseno con reducción de cadena*

4.5. Aglomerative clustering

Continuamos con el estudio entrando ya en el territorio de machine learning, en este caso con la utilización de esta herramienta en su forma no supervisada mediante un algoritmo de clustering aglomerativo. Este tipo de clustering es jerárquico, por tanto, debemos fijar el número de clusters que se quieren generar. Este valor debe coincidir con el número de desambiguaciones posibles.

El funcionamiento de este tipo de clustering se basa en tratar cada documento como un cluster en si mismo. A medida que va avanzando va mezclando los distintos clusters en grupos más amplios hasta tener todos los documentos en un solo cluster, construyendo así un dendograma. Para ir construyendo los sucesivos clusters dada una medida de similaridad se construye la matriz de similaridad NxN, siendo N el número de documentos y en cada iteración se unen los dos clusters cuyo coeficiente de similitud sea mayor.

Para la valoración del algoritmo utilizaremos la métrica adjusted rand score, presentada en la sección 4.2. Los resultados obtenidos mediante esta técnica se presentan en la tabla 4.11.

Acrónimo	Longitud de cadena	Adjusted Rand Score
PCR	30	0.008
PCR	20	0.002
PCR	10	0.001
L	30	-0.001
L	20	-0.002
L	10	-0.011
T	30	0.003
T	20	0.02
T	10	0.068

TABLA 4.11: *Resultados de la Aglomerative clustering*

Los resultados mostrados por este algoritmo y reflejados en la Tabla 4.11 son muy pobres, esto es debido a que clasifica mayoritariamente todos los textos como pertenecientes a la clase mayoritaria.Lo que convierte a este clasificador en un clasificador sin valor alguno.

4.6. Naïve bayes

Una vez estudiado el funcionamiento del clustering sobre este problema nos centramos en el área de los modelos supervisados. A la hora de trabajar con textos siempre es buena idea revisar el funcionamiento de este algoritmo, aunque no tan potente como otros puede obtener buenos resultados. Se utilizará el algoritmo de Naïve-Bayes en su versión multinomial, ya que estamos ante un problema de clasificación que tiene más de dos clases.

Para obtener unos resultados correctos y no caer en una zona de overfitting, debemos parametrizar el parámetro α del clasificador. Pero antes de realizar la prueba empírica de obtener su mejor valor, vamos a explicar qué es este parámetro y cómo afecta al funcionamiento del modelo.

Dada una sentencia a evaluar en las clases a y b, si algunas de de las palabras que forman la secuencia no han aparecido en la fase de train la probabilidad condicionada toma el valor de cero.

Una solución para este problema es aumentar la probabilidad de todos los bi-gramas en un término α, asegurando así que todos los valores serán distintos de cero. El valor de este parámetro es el que debemos validar para un correcto funcionamiento. De nuevo la codificación del texto a utilizar será la matriz TF-IDF. Los resultados obtenidos se presentan en la Tabla 4.12.

Acrónimo	Longitud de cadena	Train Accuracy	Test Accuracy
PCR	30	0.938	0.988
PCR	20	0.931	0.975
PCR	10	1.0	0.740
L	30	0.8	0.913
L	20	0.8	1.0
L	10	1.0	0.782
T	30	0.965	0.967
T	20	0.963	0.958
T	10	0.995	0.874

TABLA 4.12: *Resultados del algoritmo Naive Bayes*

Si realizamos el mismo proceso mediante la técnica utilizando la técnica Word2Vec tendríamos un problema, dado que Naïve-Bayes no puede utilizarse con valores negativos y los embeddings incumplen esta condición.

4.7. Support Vector Machines

Utilizando la representación de los documentos mediante la matriz TF-IDF y Word2vec vamos a dar una solución al problema de clasificación mediante la técnica SVM *(support vector machines)* utilizando kernel lineal.

Tal y como se explica en el siguiente artículo de referencia [13] , las máquinas de vector soporte son un buen arma a utilizar en problemas de clasificación de textos, ya que no se ven tan afectadas por lidiar con dataset que tengan un gran número de *features*. Aguantan mejor estas situaciones en las que otras técnicas al tener más variables que ejemplos de estudio caerían irremediablemente en overfitting. Como modelo de codificación de los dos estudiados anteriormente utilizaremos el modelo II, debido a los resultados obtenidos.

Aun siendo resistente, si se entrenan con el valor por defecto del parámetro C la librería *sklearn* en *Python* que toma un valor 1, estamos en la región de overfitting, por tanto, es necesario realizar una búsqueda del correcto punto de operación. Este término se corresponde con *penalty term*. Para valores altos de C, elegirá el hiperplano que sea mínimo y para valores pequeños de C elegirá margenes de hiperplanos más amplios.

En cuanto al modo de entrenamiento de SVM, como tenemos más de dos clases podemos resolver el problema mediante dos enfoques distintos, *One vs One* o *One vs All*:

- En el caso de un entrenamiento *One vs All* el proceso que estamos realizando es un método de aprendizaje para poder conocer cuándo los textos en cuestión son de una clase o del resto. En el caso del uso de la técnica *One vs One* desarrollamos un método

de aprendizaje que entrena un SVM para distinguir entre cada clase. Siendo n el número de clases en *One vs All* se deben entrenar n modelos y en *One vs One* el número se eleva a $\frac{n(n-1)}{2}$.

- Por tanto, en el caso de *One vs All* tenemos que entrenar menos clasificadores, de hecho en este caso la diferencia de tiempo en resolver el problema mediante una estrategia u otra es bastante elevado.

- Tras realizar algunas pruebas y debido al tiempo de ejecución de la modalidad *One vs One*, se ha decidido utilizar la técnica *One vs All* .

Se han entrenado las SVM en dos situaciones, mediante una codificación matriz TF-IDF y mediante Word2Vec [14]. En el caso de Word2Vec el vector representativo de cada texto como en situaciones anteriores será la media de todas sus palabras componentes. Las Tablas 4.13 y 4.14 presentan los resultados obtenidos.

Acrónimo	Longitud de cadena	Train Accuracy	Test Accuracy
PCR	30	0.940	0.976
PCR	20	0.937	0.980
PCR	10	1.0	0.746
L	30	0.826	0.948
L	20	0.816	0.946
L	10	1.0	0.826
T	30	0.969	0.972
T	20	0.968	0.968
T	10	1.0	0.903

TABLA 4.13: *Resultados del algoritmo SVM TF-IDF*

Acrónimo	Longitud de cadena	Train Accuracy	Test Accuracy
PCR	30	0.927	0.917
PCR	20	0.928	0.881
PCR	10	0.995	0.933
L	30	0.869	0.517
L	20	0.883	0.535
L	10	1.0	0.791
T	30	0.913	0.903
T	20	0.912	0.904
T	10	0.953	0.918

TABLA 4.14: *Resultados del algoritmo SVM Word2Vec*

Los estudios de SVM muestran una clara diferencia en la realización del ajuste del parámetro C, los embeddings obtienen mejores resultado cuanto más cercanos al cero nos encontramos, si nos alejamos el resultado cae de manera pronunciada produciéndose subidas y bajadas. Sin embargo, para la codificación mediante la matriz TF-IDF nos encontramos en una solución totalmente diferente obteniendo sus mejores resultados en valores de C más alejados del cero. Los resultados se muestran en las figuras 5.4 y 5.5.

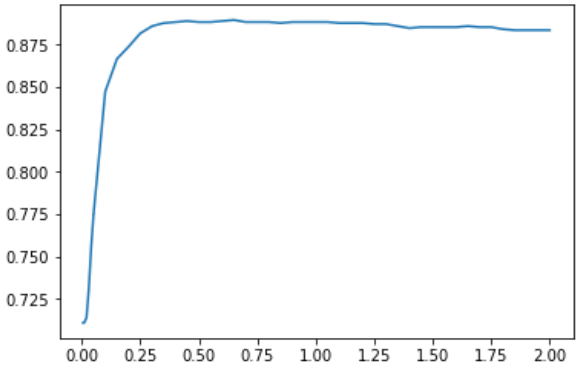

FIGURA 4.4: *SVM C TF-IDF*

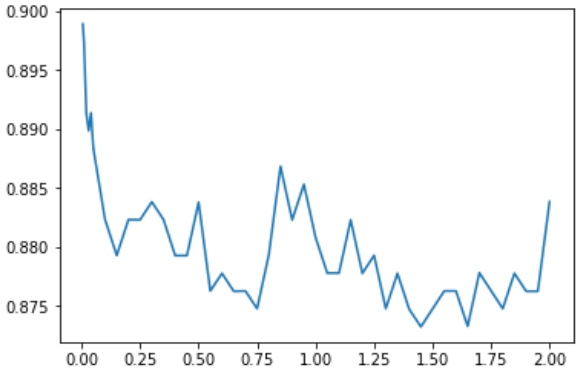

FIGURA 4.5: *SVM C Word2Vec*

4.8. Comentarios

Esta ultima sección del capítulo se centra en comparar los resultados obtenidos por los diversos clasificadores en los resuloción de este problema.

4.8.1. Algoritmo Lesk vs Distancia Coseno

Estos métodos son los utilizados en el proceso de desambiguación de la tarea BARR2, el estudio realizado en este capítulo postula como mejor sistema de desambiguación al cálculo

de la distancia coseno, mostrando los resultado obtenidos en las tablas 4.5. y 4.6.

El mejor resultado del Algoritmo Lesk alcanza un valor de accuracy de 0.882 en el acrónimo PCR, para ese mismo acrónimo el método distancia coseno supera en su mejor versión por un 8 % al algoritmo Lesk. Si nos movemos al estudio de otros acrónimos la diferencia se hace más evidente, alcanzando su máximo en el acrónimo L.

Con el apoyo de la matemática que subyace el principio de la distancia coseno, la forma óptima en este caso de realizar la codificación del texto es la matriz TF-IDF. Es una forma rápida y sencilla de llevar a cabo la desambiguación, en el caso de los acrónimo estudiados se encuentra en valores iguales o superiores al 90 % de acierto.

4.8.2. Codificación TF-IDF

Comparando los clasificadores utilizados el ganador de esta categoría sería la distancia coseno ya que obtiene una solución con un alto porcentaje de aciertos y no necesita todo el proceso de ajuste que deberíamos realizar en la SVM o Naïve-Bayes. Debemos remarcar que las máquinas de vector soporte con kernel lineal obtienen un mejor resultado, pero es necesario un proceso de ajuste, lo mismo ocurre en el caso del algoritmo Naïve-Bayes.

4.8.3. Codificación Word2vec

Basándonos en la idea de contexto brindada por la codificación embedding se han llevado a cabo distintos estudios: media embeddings, weighted media embeddings , ventana de contexto y SVM.

Los resultados obtenidos en estos procesos son dispares entre los dos primeros y el tercero. Media embeddings y weighted media embeddings obtienen resultados similares en este proceso de desambiguación, para obtener un resultado diferente la poderación según se alejase la palabra del acrónimo debería ser menor, la caida programada es demasiado suave como para afectar al resultado final. La técnica de ventana de contexto obtiene un resultado bastante pobre, uno de los problemas de esta técnica es que al depender solo de un palabra si los embeddings no están bien generados puede llevar a situaciones como esta, donde funcione muy mal.

En este apartado sí que se encuentra una gran diferencia entre la utilización de SVM respecto de la técnica distancia coseno, las máquinas de vector soporte obtienen resultados más precisos en la mayoría de sus comparaciones. Por tanto, en la utilización de Word2Vec se elegiria para trabajar las máquinas de vetor soporte.

Respecto a los dos modelos de embeddings generados para su comparación en esta tarea de clasificación no reviste de mayor importancia la utilización de uno u otro.

4.8.4. TF-IDF vs Word2vec

Si comparamos ambas codificaciones en la resolución de esta tarea de clasificación, la matriz TF-IDF sale ampliamente favorecida debido a que obtiene mejores resultados en todos los casos.

Respecto al estudido donde se muestran bajo qué condiciones muestran su verdadera potencia, la matriz TF-IDF tiende a obtener mejores resultados en una distancia de cadena entorno a las 20 palabras o tokens, es decir, para dar con el significado correcto del acrónimo necesita una extensión de 41 tokens, sin embargo, Word2Vec obtiene mejores resultados si se disminuye la longitud de la cadena. En la tabla 4.10. de comparación de ambas técnicas sobre la variación de distancia y la matriz TF-IDF se mantiene por encima salvo en alguna excepción.

La matriz TF-IDF muestra un resultado mejor que en el caso del uso de embeddings, la cantidad de textos que hemos utilizado para generar los embeddings no es la suficiente. Según aumenta el número de textos para la generación de los embeddigns mejor tienden a entender el contexto de las palabras, en este caso está claro que nuestro sistema peca de falta de textos. En el artículo [14] desarrollan los embeddings con unos textos de una longitud de 260 millones de palabras y aun con esa gran fuente de información en los textos detectan que su sistema puede mejorar con la inclusión de más información. En nuestro caso estamos con una generación cercana a los 35 millones de palabras, por tanto, tiene tendencia de que el aumento en el número de textos aumentaría significantemente los resultados de los sistemas que usan como fuente de codificación dichos vectores.

4.8.5. Comparación acrónimo F1-Score

La medida F1-score es una típica métrica para la medir el poder de desambiguación de este tipo de técnicas, por tanto, para la mejor parametrización de cada uno de los procesos de clasificación se muestra a continuación los resultados para cada acrónimo y su mejor clasificador en las tablas 4.15, 4.16 y 4.17.

Algoritmo de desambiguación	Clase 0	Clase 1	Clase 2	Accuracy
Algoritmo Lesk	0.867	0.137	0.924	0.882
Distancia coseno	0.935	0.5	0.976	0.964
Naive-Bayes	0.981	0.8	0.942	0.988
SVM	0.961	1	0.918	0.980
SVM Word2Vec	0.985	0.833	0.988	0.933
Distancia coseno Word2Vec	0.849	0.588	0.942	0.903

TABLA 4.15: *Métrica F1 acrónimo PCR*

Sobre el acrónimo PCR reflejado en la tabla 4.15 se observa que las clases 0 y 2 tienen valores parejos aunque siempre mejores para la clase 2 que es la mayoritaria. La clase 1 es la clase minoritaria, se muestra castigada por esta razón en los resultados de esta tabla 4.15, solo obteniendo un buen resultado mediante el uso de Naïve Bayes y en especial de máquinas de vector soporte.

Algoritmo de desambiguación	Clase 0	Clase 1	Clase 2	Accuracy
Algoritmo Lesk	0.72	0.14	0.73	0.655
Distancia coseno	0.936	1	0.94	0.946
Naive-Bayes	0.92	0.8	0.918	0.913
SVM	0.958	0.857	0.95	0.948
SVM Word2Vec	0.857	0.333	0.667	0.791
Distancia coseno Word2Ve	0.923	0	0.625	0.708

TABLA 4.16: *Métrica F1 acrónimo L*

El acrónimo L mostrado en la tabla 4.16 se puede considerar complicado en líneas generales debido a un menor dataset de entrenamiento y menor distancia de significado entre sus términos, la mayoría de las veces aparecen en forma de unidades. Aun con todo esto los resultados son bastante buenos en el caso de uso de SVM y distancia coseno ambas con codificación TF-IDF.

Algoritmo de desambiguación	Clase 0	Clase 1	Clase 2	Clase 3	Clase 4	Accuracy
Algoritmo Lesk	0.244	0.852	0	0.843	0	0.807
Distancia Coseno	0	0.904	0	0.964	0	0.950
Naive-Bayes	0	0.945	0	0.977	0	0.967
SVM	0	0.951	0	0.979	0	0.972
SVM Word2Vec	0	0.909	0	0.960	0	0.918
Distancia coseno Word2vec	0.166	0.792	0	0.980	0	0.810

TABLA 4.17: *Métrica F1 acrónimo T*

Por último, el acrónimo T mostrado en la tabla 4.17 llama la atención sus buenos valores de accuracy, sin embargo, las clases 0, 1 y 2 tienen valores nulos de medida F1. Esto es debido a que es un problema de 5 clases sin embargo, las clases 1 y sobre todo 3 son muy superiores en número al resto y por tanto, vuelven este problema mucho más difícil de resolver.

Con todo esto, podemos afirmar que enfocar el problema como un problema de clasificación es una forma útil de trabajar siempre que no tengamos clases muy desbalanceadas, en ese caso, se podría estudiar resolver este sistema de clasificación de forma desbalanceada.

Capítulo 5

Sensegram

A lo largo de todo este estudio se reconoce el hecho de que no tenemos textos suficientes para entrenar o es complicado conseguir un conjunto de datos anotados. Además, una crítica a los embeddings en este proceso es que aglomeran todos los posibles significados del acrónimo en el mismo vector, es decir, el vector correspondiente al acrónimo L en nuestro caso tendrá contenida la información contextual referente a las palabras litro, linfocito y leucocito.

Teniendo en cuenta la potencia de la técnica coseno similarity, si se pudiese obtener para cada posible acrónimo ambiguo un embedding representativo de cada desambiguación posible de forma no supervisada, sería una buena aproximación para resolver este problema.

Debemos tener en cuenta que el dataset utilizado en el capítulo anterior ha sido desarrollado para trabajar sólo con tres acrónimos, si tuviesemos que resolver este problema con un número de desambiguaciones mayor el tiempo empleado en esta fase de creación del dataset podría llegar a ser insostenible.

La técnica a estudio en este capítulo se denomina Sensegram [15], la cual nos dará como resultado un vector embedding correspondiente a cada uno de los posibles significados del acrónimo contenidos en nuestro dataset. Una vez generados estos embeddigns, debemos comparar con cada una de las apariciones en el dataset mediante la métrica distancia coseno, eligiendo la definición cuya distancia sea menor al significado real del acrónimo.

Este proceso de desambiguación consta de cuatro pasos, los cuales se van a desarrollar en detalle a continuación.

5.1. Ego-graph

Un Ego-graph es aquel grafo que tiene como punto de partida un nodo del mismo, ego. Este grafo está formado el nodo ”ego” y sus primeros vecinos (todos conectados con él), es una técnica utilizada frecuentemente en los estudios de redes sociales. Se ilustra en la Figura 5.1. Además de con ”ego”, el resto de nodos del grafo pueden estar conectados entre

sí siempre que exista una relación entre ellos, lo cual convierte a este tipo de grafo en una buena opción para separar en grupos a distintos usuarios tomando a uno de estos como punto de referencia.

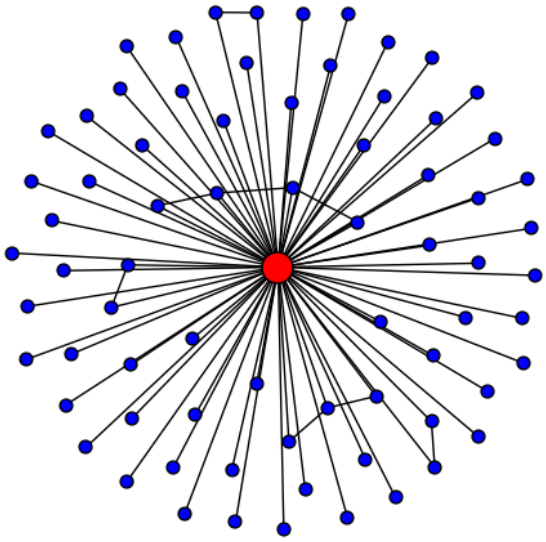

FIGURA 5.1: *Ego graph*

5.2. Word2vec y word Similarity Graph

El primer paso para llevar a cabo esta técnica es la generación de los embeddings, siguiendo el proceso estudiado en la sección 3.2.1.

A partir de estos vectores se debe construir la matriz $M[i, j]$ cuyo contenido será la similitud coseno entre los pares de palabras o tokens i, j. En ese proceso estamos construyendo una matriz que dadas dos palabras i, j la distancia coseno entre las mismas es la posición $[i, j]$ de la matriz M.

Una vez construida la matriz M, debemos fijar la palabra o acrónimo sobre la que se quiere realizar el proceso de desambiguación y construir entorno a ella un Ego-Graph. Esta palabra fijada pasará a ser ”ego”.

Fijado ”ego” obtenemos las 300 palabras más cercanas al mismo. Calculamos para cada par de nodos del grafo su similitud coseno, si esta está por encima de un determinado valor, ambos nodos se convertirán en vecinos en caso contrario no.

En el proceso de generación de este grafo debemos fijar dos valores además del número de palabras cercanas, estos valores son la distancia coseno a partir de la cual dos nodos que no sean ”ego” se convierten en primeros vecinos y el número de primeros vecinos que debe tener un nodo para mantenerlo en el grafo. Esta última condición se utiliza para evitar tener nodos que esten muy cercanos a ”ego” pero no tengan relación con el resto del grafo.

5.3. Chinese Whispers y Community detection

Una vez creado el grafo debemos obtener de manera automática los distintos significados encontrados en él. Estos significados no son más que ese conjunto de palabras que están relacionadas con ”ego” y además fuertemente relacionadas entre sí. Las cuales conforman un cluster o comunidad. Para llevar a cabo este proceso de forma automática existen dos posibilidades: utilizar el método de clustering Chinese Whispers o un algoritmo de detección de comunidades.

5.3.1. Chinese Whispers

Se propone para este proceso realizar clustering sobre el grafo mediante la técnica *Chinese Wisphered* [16], la cual nos devolverá las clases correspondientes a cada palabra respecto del grafo.

Esta técnica permite realizar clustering sobre un grafo, ya sea este ”weighted” o ”unweighted” y es non-hierarchical lo que significa que no tenemos que indicar el número de particiones o clusters que se deberían realizar sobre el grafo. Esta propiedad es muy importante ya que no estamos fijando el número de significados que el acrónimo puede tener en el corpus.

Otro factor importante es que su tiempo de ejecución es lineal respecto al número de nodos, lo cual puede llegar a ser un factor determinante a la hora de decantarnos por la utilización de uno u otro método.

5.3.2. Community detection

La tarea a resolver bien podría afrontarse como un problema de detección de comunidades, por tanto, también estudiaremos esta vía. Utilizaremos la implementación del algoritmo de detección de comunidades *Girvan Newman* [17] sobre nuestro grafo en su implementación en la librería Network X.

El algoritmo de detección de comunidades se construye sobre la idea de que la arista que une distintas comunidades, deberían tener un alto valor de edge betweenness, así realizamos recursivamente el siguiente paso:

1. Calcular la métrica *edge betweeness* para todas las artistas del grafo.

2. Eliminar del grafo aquella arista que tenga el valor máximo.

3. Repetir el proceso hasta eliminar todas las aristas.

Mediante este proceso estamos realizando la construcción de un dendograma como en el caso del clustering acumulativo del capítulo anterior.

5.4. Word Sense induction

Una vez hemos clusterizado a qué significado corresponde cada palabra, es hora de realizar para cada uno de estos significados su ”embeddings” característico. Se utilizará en este paso una versión ”unweighted”, que consiste en asignar como vector de significado la media de todos los nodos que conforman cada uno de los cluster detectados por las técnicas anteriores.

Con los embeddigns de referencia generados, dado un nuevo vector representación de cada nueva frase a clasificar, como en casos anteriores, podemos obtener su distancia coseno y de nuevo, clasificar hacia el significado que se encuentre más cercano, aquel que obtenga menor distancia coseno.

$$SBE(w) = \frac{1}{n} \sum_{i=0}^{n} Emb(i)$$

5.5. Metodología

Utilizando esta técnica existen dos opciones para llevar a cabo la desambiguación de acrónimos:

1. Si el corpus es lo suficientemente extenso, se puede realizar todo el proceso sobre él: calcular los embeddings y obtener mediante la clusterización del grafo los vectores referencia de cada una de las clases.

2. En caso de que el corpus no sea lo suficientemente extenso, pero tengamos textos de la misma materia, textos de background, generamos los vectores referencia utilizando los textos de background. Una vez que tengamos el proceso realizado y obtenidos los vectores referencia, aplicamos el modelo de embeddigns a nuestros textos y llevamos a cabo el proceso estándar de desambiguación.

En nuestro caso, como los textos no son suficientes se tomará optará por la resolución mediante la metodología número 2.

5.6. Resultados

Sobre un conjunto de acrónimos vamos a ver qué resultado nos ofrece esta aproximación:

5.6.1. PCR

Este acrónimo tiene 3 posibles desambiguaciones en nuestro dataset, por tanto, deberíamos encontrar 3 clusters diferentes. El modelo de embeddings generados no ha conseguido obtener desarrollar todo el potencial que brinda esta técnica. Debido a este suceso encontramos nodos del grafo que solo están conectados con ”ego” y no con el resto del grafo.

Se ha realizado un proceso de limpieza partiendo de la condición de que solo nos interesan aquellos nodos que tienen una importante conexión en el grafo y decidimos quedarnos solo con aquellos nodos que tienen más de 5 vecinos. Para que dos nodos sean vecinos en este caso su similitud debe de ser superior a 0,7.

La técnica utilizada para separar las distintas comunidades ha sido el algoritmo *Girvan Newman* de la librería Network X.

Modelo I

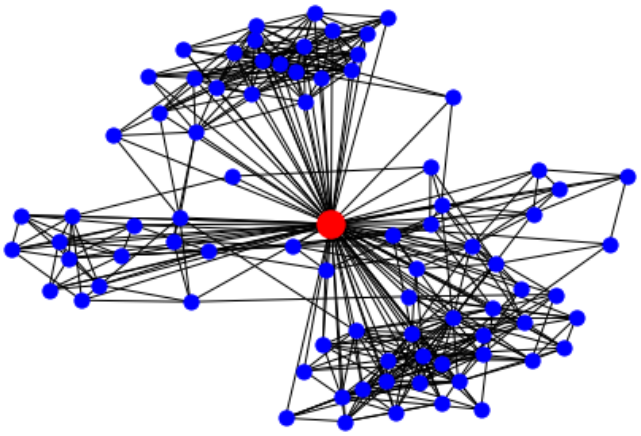

FIGURA 5.2: *PCR graph modelo I*

En la gráfica de la Figura 5.2 se puede intuir que el sistema ha funcionado correctamente, ya que a simple vista tiene sentido diferenciar entre 3 nubes de nodos. El algoritmo de detección de comunidades refrenda esta suposición y encuentra tres comunidades distintas.

Si probamos a desambiguar con los vectores de referencia todos los ejemplos del acrónimo PCR en el capítulo anterior podemos ver cómo funciona esta técnica, los resultados se muestran en la Tabla 5.1.

Acrónimo	Longitud de cadena	Accuracy
PCR	30	0.716
PCR	20	0.722
PCR	10	0.774

TABLA 5.1: *Sensegram PCR Desambiguación Community detection*

Se ha lllevado a cabo el mismo proceso utilizando como técnica de clusterización el algoritmo *Chinese Wisphered*, esta técnica encuentra también 3 comuniades pero su valor predictivo es inferior. Los resultados se muestran en la Tabla 5.2.

Acrónimo	Longitud de cadena	Accuracy
PCR	30	0.664
PCR	20	0.655
PCR	10	0.668

TABLA 5.2: *Sensegram PCR Desambiguación Chinese Whispers*

Tal y como se muestra en las tablas 5.1 y 5.2, la comparación para esta situación entre el algoritmo *Chinese Wisphered* y commuty detection decanta la balanza en favor del algoritmo de detección de comunidades *Girvan Newman*. Mostramos a continuación en la Tabla 5.3 los valores de la métrica f1-Score:

Clase 0	Clase 1	Clase 2
0.501	0.421	0.885

TABLA 5.3: *Sensegram PCR f1-score*

Si realizamos una representación de embeddigns mediante el agoritmo t-SNE veremos que situación hemos alcanzado, Figura 5.3.

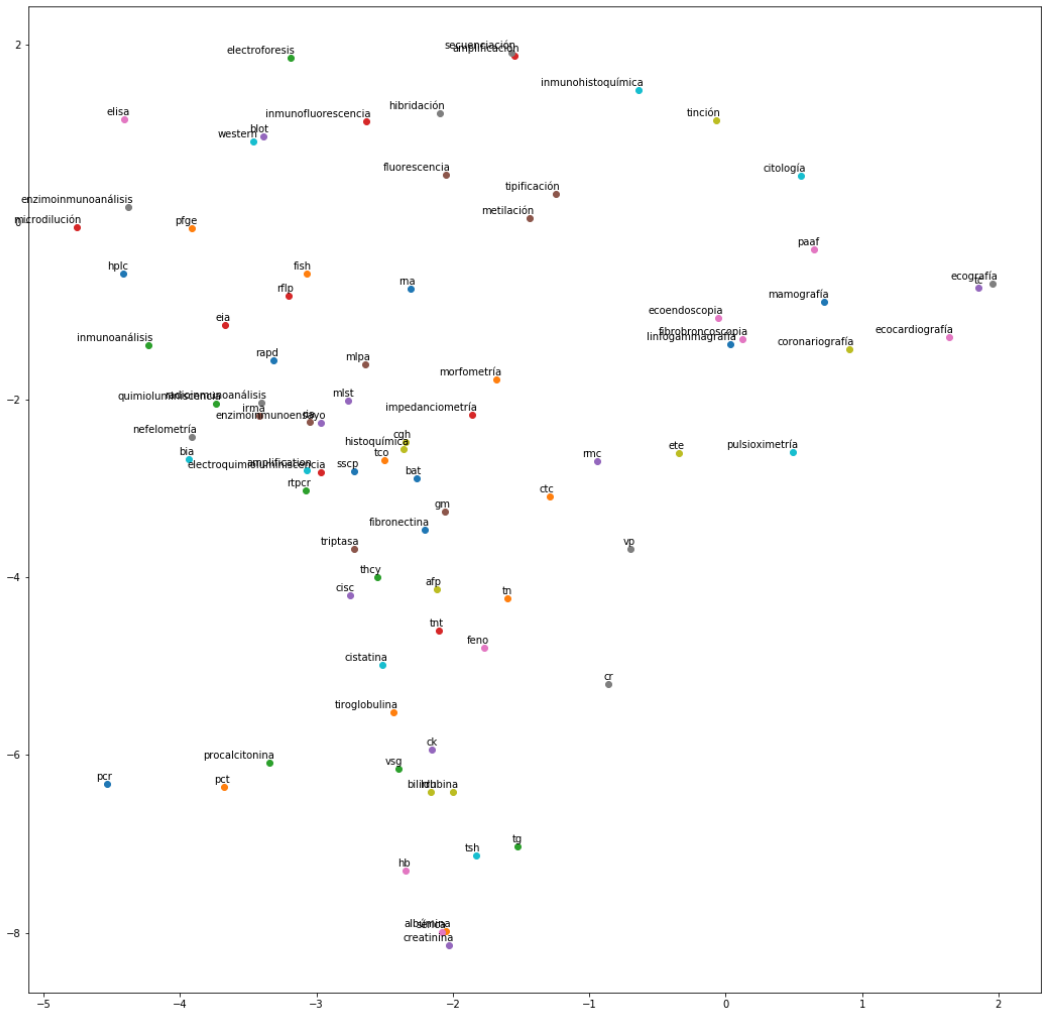

FIGURA 5.3: *PCR embeddings*

En la figura 6.3. podemos ver representados los tres contextos referentes a las desambiguaciones del acrónimo PCR. En la zona superior derecha encontramos el contexto referente a parada cardiorrespiratoria en palabras tales como ecocardiografía o coronariografía. En la zona inferior izquierda rodeando al acrónimo vsg encontramos el significado referente a Proteína C reactiva. Por último, el grupo más visible a la vez que mayoritario en el dataset, Reacción en cadena de la polimerasa en la zona superior izquierda, reflejado en palabras tales como elisa o enzimoinmunoanálisis.

Modelo II

Si repetimos el mismo proceso que en el apartado anterior pero utiliando el modelo II de Word2vec generando la figura 5.4.

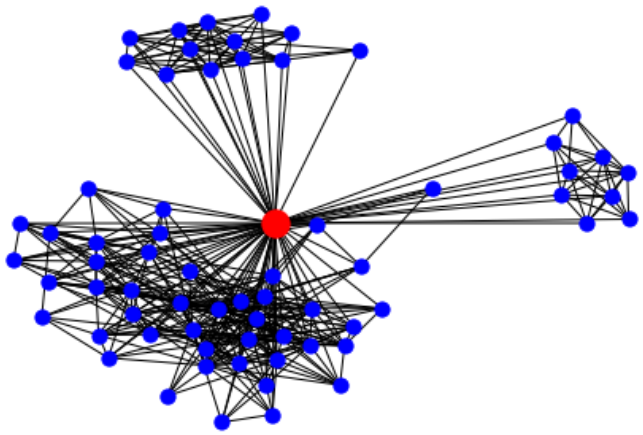

FIGURA 5.4: *PCR graph Modelo II*

En este caso los cluster son incluso más reconocibles, no tiene conexión entre ellos salvo por ”ego”. ¿Es este un mejor resultado? La verdad es que no, debido a que en este modelo el número de veces que una palabra para ser tomada en cuenta tiene que ser menor, la mayoria de lo nodos que aparecen no tienen relación alguna con el acrónimo PCR más que casual. Además, las comunidades ya no se refiren a las distintas desambiguaciones del acrónimo.

5.6.2. Acrónimos L y T

Repitiendo el proceso sobre los acrónimos L y T nos encontramos ante una situación bien distinta, en este caso nuestro sistema no es capaz de detectar más que una comunidad. No es capaz de distinguir los distintos significados de los acrónimos. Este problema puede ester motivado por la generación de los embeddings debido a que se ha realizado un proceso de transformación en minúsculas y tokenización, por esta razón podemos incurrir en errores, de hecho, las palabras más cercanas a estos acrónimos son letras sueltas.

Otro punto de vista es que nuestro sistema no llega a tener la capacidad de dividir en comunidades ya que al estar referidos a palabras muy comunes que no tienen ese valor de localidad que si puede tener el acrónimo PCR. Falla, por tanto, en el proceso de obtener una división correcta en comunidades.

5.6.3. Acrónimo AST

Debemos también contrastar que cuando utilizamos esta técnica sobre un acrónimo que no tiene más que una posible desambiguación, no es ambiguo, este sistema no detecta más de una comunidad o contexto. La elección de este acrónimo es debida a que cuenta con un número interesante de apariciones. Como se puede observar en la figura 6.5. encontramos un grafo mucho más conexo sobre el que no se detectan comunidades.

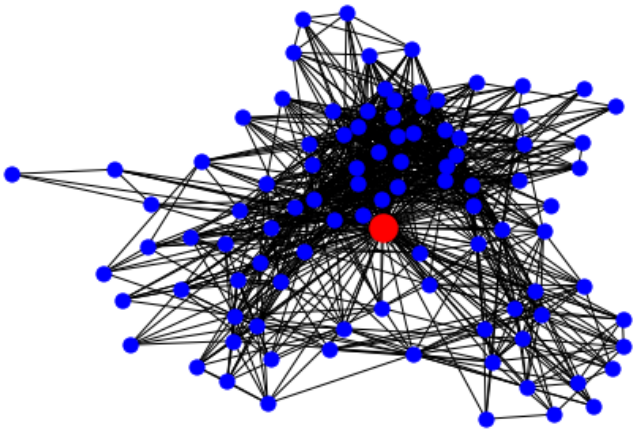

FIGURA 5.5: *AST graph*

Capítulo 6

Conclusiones

En el presente trabajo se han llevado a cabo diferentes estrategias para resolver el problema de la desambiguación de acrónimo clínicos. Para la resolución de la tarea BARR2 se ha optado por la utilización de una aproximación rule-based y un proceso de desambiguación utilizando el algoritmo Lesk y la distancia coseno. Partiendo de una primera situación en la cual con el diccionario era complicado obtener buenos resultados, se ha mejorado mediante técnicas de revisión del mismo hasta obtener un resultado muy superior al inicial.

A la hora de resolver el problema como un problema de clasificación se ha generado un dataset de trabajo mediante web scraping y los textos de la tarea, además de la generación de distintos modelos Word2Vec para probar su eficacia. Los resultados obtenidos en esta fase han sido favorables, sobre todo en la utilización de los algoritmos SVM, Naïve-Bayes y distancia coseno.

Por último, se ha utilizado la técnica de Sensegram sobre este conjunto de acrónimos, una técnica no supervisada compuesta de varias fases, aunque nuestros embeddings no han podido desarrollar todo su potencial, esta técnica no se ha vista tan penalizada y se ha podido aplicar de manera satisfactoria al acrónimo PCR.

Bibliografía

[1] ANDER INTXAURRONDO , MARTIN PEREZ-PÉREZ , GAEL PEREZ-RODRÍGUEZ, JOSE ANTONIO LOPEZ-MARTÍN, JESUS SANTAMARÍA, SANTIAGO DE LA PEÑA, MARTA VILLEGAS,AHMAD AKHONDI, ALFONSO VALENCIA, ANALIA LOURENC, MARTIN KRALLINGER, *The Biomedical Abbreviation Recognition and Resolution (BARR) track: benchmarking, evaluation and importance of abbreviation recognition systems applied to Spanish biomedical abstracts. SEPLN 2017, (2017)*

[2] ANDER INTXAURRONDO, MONTSERRAT MARIMON, AITOR GONZALEZ-AGIRRE,JOSE ANTONIO LOPEZ-MARTIN, HEIDY RODRIGUEZ, JESUS SANTAMARIA,MARTA VILLEGAS, MARTIN KRALLINGER, *Finding mentions of abbreviations and their definitions in Spanish Clinical Cases: the BARR2 shared task evaluation results. SEPLN 2018.*

[3] SOTO MONTALVO, MAITE ORONOZ , HORACIO RODRÍGUEZ , RAQUEL MARTÍNEZ, *Biomedical Abbreviation Recognition and Resolution by PROSA-MED. Proceedings of the Second Workshop on Evaluation of Human Language Technologies for Iberian Languages (IberEval 2017) co-located with 33th Conference of the Spanish Society for Natural Language Processing (SEPLN 2017), Murcia, Spain, September 19, 2017. CEUR Workshop Proceedings 1881, CEUR-WS.org 2017: 247-254.*

[4] STEVEN BIRD, EWAN KLEIN, AND EDWARD LOPER, *Natural Language Processing with Python. O'Reilly, June de 2009*

[5] JAVIER YETANO LAGUNA, VICENT ALBEROLA CUÑAT, *Diccionario de siglas médicas y otras abreviaturas, epónimos y términos médicos relacionados con la codificación de las altas hospitalarias. ISBN: 84-7670-667-7 (2012)*

[6] IGNACIO IACOBACCI , MOHAMMAD TAHER PILEHVAR, ROBERTO NAVIGLI, *Embeddings for Word Sense Disambiguation: An Evaluation Study. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 897–907, Berlin, Germany, August 7-12, 2016.*

[7] GOOGLE RESEARCH, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Preliminary White Paper, November 9, 2015. download.tensorflow.org/paper/whitepaper2015.pdf 07/09/2018*

[8] LAURENS VAN DER MAATEN, GEOFFREY HINTON, *Visualizing Data using t-SNE. Journal of Machine Learning Research 9 (2008) 2579-2605.*

[9] SATANJEEV BANERJEE , TED PEDERSEN, *An adapted Lesk ALgorithm for Word Sense Disambiguation Using WordNet. (2002), d.umn.edu/ tpederse/Pubs/cicling2002-b.pdf 07/09/2018*

[10] GRIGORI SIDOROV, ALEXANDER GELBUKH, HELENA GOMEZ-ADORNO, AND DAVID PINTO, *Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Mode. Computación y Sistemas Vol 18, No 3 (2014)*

[11] MARK HUGHES, IRENE LI, SPYROS KOTOULAS, TOYOTARO SUZUMURA, *Medical Text Classification using Convolutional Neural Networks. Apr 25, 2017. arxiv.org/ftp/arxiv/papers/1704/1704.06841.pdf 07/09/2018*

[12] YANSHAN WANG, SIJIA LIU, NAVEED AFZAL, MAJID RASTEGAR-MOJARAD, LIWEI WANG, FEICHEN SHEN, PAUL KINGSBURY, HONGFANG LIU, *A Comparison of Word Embeddings for the Biomedical Natural Language Processing. arXiv:1802.00400v3 [cs.IR] 18 Jul 2018*

[13] THORSTEN JOACHIMS, *Text Categorization with Support Vector Machines: Learning with Many Relevant Features. ECML 1998: Machine Learning: ECML-98 pp 137-142 (1998)*

[14] YONGHUI WU, JUN XU, YAOYUN ZHANG, HUA XU, *Clinical Abbreviation Disambiguation Using Neural Word Embeddings, Proceedings of the 2015 Workshop on Biomedical Natural Language Processing (BioNLP 2015), pages 171–176, Beijing, China, July 30, 2015.*

[15] MARIA PELEVINA, NIKOLAY AREFYEV, CHRIS BIEMANN, ALEXANDER PANCHENKO, *Making Sense of Word Embeddings. arXiv:1708.03390 [cs.CL] 10 Aug 2017*

[16] CHRISTIAN BIEMANN *Chinese Whispers - an Efficient Graph Clustering Algorithm and its Application to Natural Language Processing Problems. Workshop on TextGraphs, at HLT-NAACL 2006, pages 73–80, New York City, June 2006.*

[17] M. E. J. NEWMAN, *Detecting community structure in networks. M.E.J. Eur. Phys. J. B (2004) 38: 321. https://doi.org/10.1140/epjb/e2004-00124-y*