Universidad de Zaragoza

Trabajo de Fin de Máster

Identificación unívoca de filiaciones en datos de I+D+i utilizando herramientas de Inteligencia Artificial

Autor:

Muñoz Jordán, David[†]

Directores:

Iñiguez Dieste, David[‡] Durán Batalla, Juan Luis[§]

Ponente:

Alcalá Nalvaiz, Jose Tomás[*]

Financiado por la Unión Europea-NextGenerationEU

Curso 2023/2024

[†]759379@unizar.es
[‡]david.iniguez@bifi.es
[§]jduran@kampal.com
[*]jtalcala@unizar.es

Índice

Resumen

Los investigadores firman de muchas formas diferentes y distintas revistas han establecido también diferentes estándares en ese sentido. En este proyecto nos centraremos en la correcta identificación de las filiaciones, es decir, del centro al que pertenecen los investigadores y la unificación de investigadores. Un mismo investigador puede escribir el nombre del centro al que pertenece de forma distinta en unas u otras publicaciones, y distintos investigadores de un mismo centro van a escribirlo de forma distinta con total seguridad, más aún si queremos trabajar a un nivel de detalle como facultad, departamento o instituto de investigación. Por otro lado, un investigador puede estar adscrito a varios centros de forma simultánea, o en diferentes etapas de su vida profesional, lo que introduce el factor tiempo como una variable más a tener en cuenta. En este proyecto analizaremos distintas herramientas y algoritmos de Inteligencia Artificial para identificar de forma unívoca los distintos centros de investigación, y sus estructuras internas en algunos casos, así como su asociación a los investigadores en las distintas etapas de su vida profesional. Para lograr este objetivo, se construirá una cadena completa donde descargaremos artículos científicos de Web of Science (WoS), se procesará y normalizará la información, se aplicará un proceso de regularización, y posteriormente se aplicarán los diferentes algoritmos desarrollados con el fin de conseguir la unificación de autores e identificación de filiaciones.

Researchers sign in many different ways, and different journals also have varying standards in this regard. In this project we will focus on the identification of affiliations, i.e., the center which the researchers belong to and the researcher unification. The same researcher may write the name of the center differently in various publications, and different researchers from the same center will certainly write it differently as well, especially if we aim to work at a more granular level such as faculty, department, or research institute. On the other hand, a researcher may be affiliated with several centers simultaneously or at different stages of their professional life, which introduces the factor of time as an additional variable to consider. In this project, we will analyze several tools and algorithms in Artificial Intelligence to uniquely identify different research centers, and their internal structures in some cases, as well as their association with researchers at different stages of their professional career. To achieve this objective, a complete pipeline will be developed where we will download scientific papers from Web of Science (WoS), process and normalize the information, apply a regularization process, and subsequently apply the different developed algorithms with the objective of achieving author unification and affiliation identification.

Agradecimientos

Quiero agradecer al apoyo de los directores del trabajo, así como al equipo de Kampal que me ha apoyado con su experiencia en estos problemas y facilitado con recursos de hardware para poder realizar las pruebas de los diferentes modelos.

Siglas

ARI Adjusted Rand Index. 20, 37

ASCII American Standard Code for Information Interchange. 7, 9, 11

AUC Area Under the Curve. 19, 24, 27, 28, 31, 33–36, 45, 47

BERT Bidirectional Encoder Representations from Transformers. 15–17

CPU Central Processing Unit. 49

FMI Fowlkes-Mallows Index. 20, 37

FN False Negative. 19, 44, 46

FP False Positive. 19, 44, 46

GPT Generative Pre-trained Transformers. 15, 49

GPU Graphics Processing Unit. 37, 49

INE Instituto Nacional de Estadística. 5, 6, 11

NLP Natural Language Processing. 14

NMI Normalized Mutual Information. 20, 37

NoSQL Not Structured Query Language. 8, 48

ORCID Open Researcher and Contributor ID. 36, 37

ROC Receiver Operating Characteristic. 19, 24, 26, 28, 30, 31, 33–35

SBERT Sentence BERT. 16

SQL Structured Query Language. 8, 48

TN True Negative. 19, 44, 46

TP True Positive. 19, 44, 46

WoS Web of Science. 1, 3, 4, 8, 12, 17, 41, 46

1. Introducción

El interés por los datos ha crecido durante los últimos años por diferentes factores, como pueden ser el aumento de la capacidad de computación, su mayor asequibilidad, así como las soluciones efectivas implementadas y contrastadas basadas en modelos estadísticos. A raíz de esto, hoy en día está en boca de todos el mundo de la ciencia de datos, llenándose periódicos con titulares que incluyen "*Machine Learning*", "*Inteligencia Artificial*", "*Big Data*", "*Internet of Things*", "*Deep Learning*", "*Computer Vision*" y muchos más ejemplos.

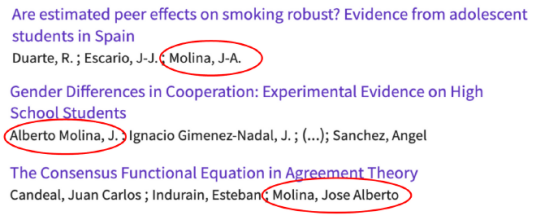
Como consecuencia, en muchos ámbitos se ha visto la importancia que tiene el almacenamiento masivo de datos de forma que se puedan analizar con diferentes propósitos como puede ser inferir y estudiar relaciones entre variables o construir modelos predictivos de forma que permita anticipar el valor esperado de una variable o la probabilidad de un suceso de que ocurra dado un contexto que lo rodea.

Los modelos estadísticos cubren un gran abanico de posibilidades donde en función de lo que se quiera modelar, unos se ajustan mejor a las necesidades del problema que otros. En el caso de este trabajo tenemos que la base de partida es texto, por lo que muchos modelos que se nos puedan ocurrir no son directamente aplicables debido a que las variables que manejan son numéricas, bien sean variables continuas o discretas. Por tanto, requeriremos de modelos más específicos para nuestro problema.

Cuando un investigador publica un artículo científico, éste lo hace bajo el nombre de una filiación a la que pertenece o tiene algún tipo de vínculo en el momento de la publicación. Cada investigador puede tener una o más filiaciones en una publicación, y no tienen por qué coincidir entre publicaciones, pues puede haberse movido de filiación, haber realizado una estancia en otra o una colaboración puntual, por ejemplo.

Existen bases de datos de publicaciones científicas como Web of Science (WoS) o Scopus de las cuales podemos obtener información sobre la actividad desarrollada por los investigadores de miles de centros a nivel mundial, el problema de esta información es que su formato no está estandarizado ni para las personas ni paras las filiaciones.

(a) Ejemplo con tres extractos de tres publicaciones diferentes donde se ha publicado bajo el nombre del BIFI.



(b) Ejemplo con tres extractos de tres publicaciones diferentes donde ha participado la misma persona.

Figura 1: Ejemplos extraidos de WoS.

En la Figura 1 se presenta un caso para personas y otro para filiaciones. En primer lugar, en la Figura 1b podemos ver un ejemplo de una misma persona que ha trabajado en tres artículos científicos diferentes. Se observa cómo la forma de referenciar el nombre para la misma persona es distinta en cada uno de los casos.

Por otra parte, en la Figura 1a se muestran tres extractos de artículos científicos donde una de las filiaciones participantes es el Instituto de Biocomputación y Física de Sistemas Complejos

(BIFI). Vemos que, aunque para un humano es casi inmediato identificar el BIFI en los tres casos, la cadena de texto completa en la que aparece dicha filiación es totalmente distinta, lo que dificulta hacer esa identificación de manera automatizada.

El objetivo de este trabajo es que, dado un conjunto de artículos científicos, seamos capaces de identificar de manera unívoca las filiaciones que han participado en cada uno de dichos artículos, independientemente de cómo los autores o revistas hayan reflejado las mismas.

Para conseguir este objetivo, descargaremos un conjunto de artículos científicos desde la base de datos de Web of Science (WoS), y estandarizaremos la información de cada artículo obteniendo para cada autor su nombre y la filiación con la que ha firmado. Se dispondrá también de un *maestro* de filiaciones, el cual no es más que un repositorio o base de datos con información estandarizada paras las filiaciones nacionales, el cual nos servirá como referencia a la hora de identificar las diferentes filiaciones.

El resultado de este trabajo es la base para poder hacer estudios más profundos sobre la producción científica de las diferentes filiaciones y personas, así como de las relaciones entre ellas.

En la segunda sección se explicará toda la cadena operativa para poder conseguir este objetivo, con sus diferentes partes. Se explicarán los distintos algoritmos desarrollados y las diferentes etapas de la cadena. En la tercera sección nos centraremos en los modelos matemáticos utilizados para resolver el problema y las pruebas realizadas. Por último, en la cuarta sección tendremos un apartado de conclusiones donde se hará un resumen de los resultados obtenidos, aspectos a mejorar, y posibles vías de continuación en el futuro.

2. Formulación

El problema a resolver consiste en conseguir, dado un artículo científico, identificar de manera unívoca tanto los autores que han participado en él, como las filiaciones de los mismos.

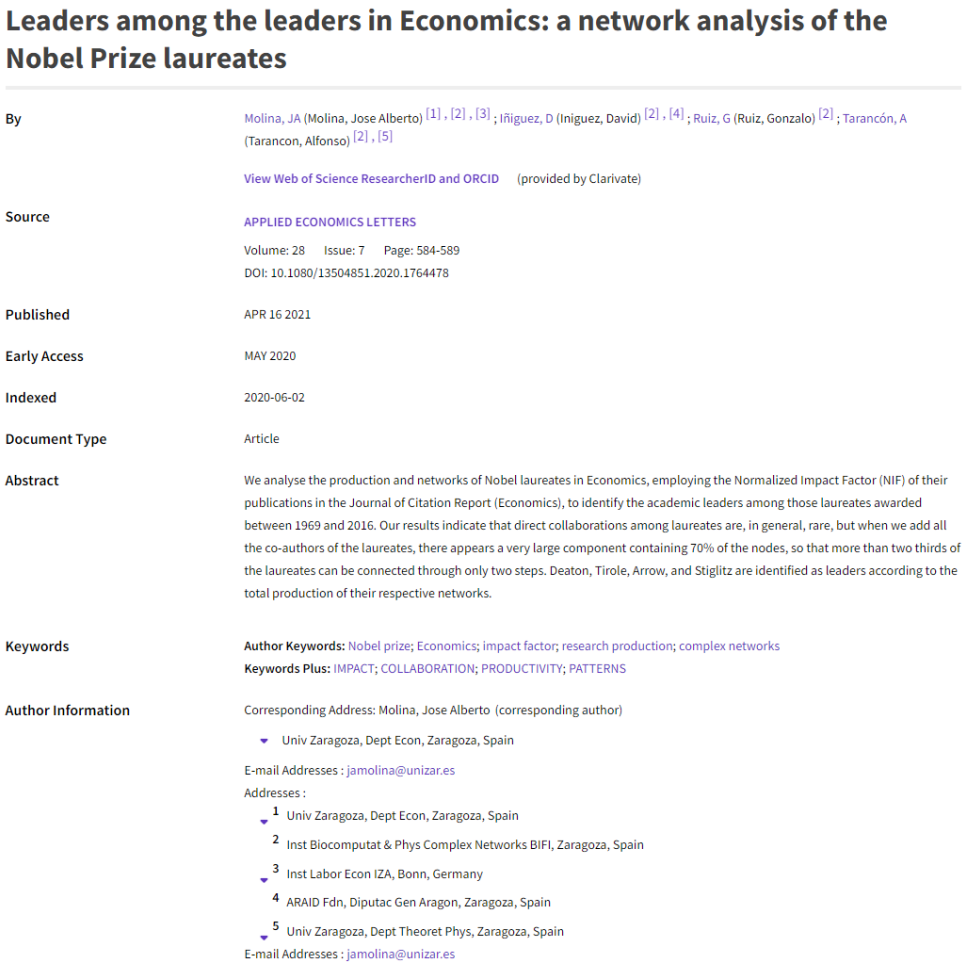
En la Figura 2 se muestra un ejemplo típico extraído de WoS de la información que se dispone para un artículo científico. En él tenemos información como el título, resumen o palabras clave, además de lo que concierne al problema que queremos resolver, que son sus autores y filiaciones correspondientes.

2.1. Cadena del proceso

Como se ha comentado, la finalidad de este trabajo es la correcta identificación de personas y filiaciones. Para resolver este problema, se divide en tres grandes partes como se muestra en la Figura 3. Lo que se hace en cada parte de esta cadena es:

1. Descarga. El repositorio bibliográfico WoS tiene un buscador avanzado[1] donde se pueden hacer búsquedas con diferentes criterios. Descargaremos artículos entre 2019 y 2023 donde haya participado algún autor español (es decir, con al menos una filiación con valor "*Spain*" el campo dirección) y con temáticas en "*Physics, Mathematical*", "*Engineering, Mechanical*", "*Computer Science, Artificial Intelligence*", "*Radiology, Nuclear Medicine & Medical Imaging*", "*Psychology, Psychoanalysis*", "*Nutrition & Dietetics*", "*Language & Linguistics*", "*Clinical Neurology*", "*Materials Science, Paper & Wood*" y "*Energy &*

[1]Enlace al buscador: https://www.webofscience.com/wos/woscc/advanced-search



Leaders among the leaders in Economics: a network analysis of the Nobel Prize laureates

By Molina, JA (Molina, Jose Alberto) [1] , [2] , [3] ; Iñiguez, D (Iniguez, David) [2] , [4] ; Ruiz, G (Ruiz, Gonzalo) [2] ; Tarancón, A (Tarancon, Alfonso) [2] , [5]

View Web of Science ResearcherID and ORCID (provided by Clarivate)

Source APPLIED ECONOMICS LETTERS

Volume: 28 Issue: 7 Page: 584-589

DOI: 10.1080/13504851.2020.1764478

Published APR 16 2021

Early Access MAY 2020

Indexed 2020-06-02

Document Type Article

Abstract We analyse the production and networks of Nobel laureates in Economics, employing the Normalized Impact Factor (NIF) of their publications in the Journal of Citation Report (Economics), to identify the academic leaders among those laureates awarded between 1969 and 2016. Our results indicate that direct collaborations among laureates are, in general, rare, but when we add all the co-authors of the laureates, there appears a very large component containing 70% of the nodes, so that more than two thirds of the laureates can be connected through only two steps. Deaton, Tirole, Arrow, and Stiglitz are identified as leaders according to the total production of their respective networks.

Keywords **Author Keywords:** Nobel prize; Economics; impact factor; research production; complex networks

Keywords Plus: IMPACT; COLLABORATION; PRODUCTIVITY; PATTERNS

Author Information Corresponding Address: Molina, Jose Alberto (corresponding author)

Univ Zaragoza, Dept Econ, Zaragoza, Spain

E-mail Addresses : jamolina@unizar.es

Addresses :

1 Univ Zaragoza, Dept Econ, Zaragoza, Spain

2 Inst Biocomputat & Phys Complex Networks BIFI, Zaragoza, Spain

3 Inst Labor Econ IZA, Bonn, Germany

4 ARAID Fdn, Diputac Gen Aragon, Zaragoza, Spain

5 Univ Zaragoza, Dept Theoret Phys, Zaragoza, Spain

E-mail Addresses : jamolina@unizar.es

Figura 2: Ejemplo de artículo ciéntifíco. [1]

Fuels". Posteriormente, se organizará la información en 4 tablas, una de personas, una de filiaciones, una de artículos y otra que contenga la relación entre estas tres anteriores.

2. Regularización. Como en muchos procesos se va a utilizar la distancia de Levenshtein como métrica para comparar dos textos, es necesario hacer un proceso de depurado y normalización sobre dichos textos a comparar que llamaremos "regularización". Este proceso se aplicará a los nombres y apellidos de personas, nombres de las filiaciones, y lugares. Además, en el caso de los nombres y apellidos de personas, se dispondrá de un corpus de nombres y personas obtenidos del INE[2] el cual nos ayudará a corregir posibles erratas en los nombres y apellidos. Para las filiaciones, se intentará extraer el tipo al que corresponde a partir del propio nombre, en concreto las clasificaremos como Universidad, Hospital, Facultad, Escuela, Instituto, Centro, Departamento. Se utilizará un tipo especial llamado Otros para aquellas filiaciones que no correspondan a ninguna de las clasificaciones anteriores.

3. Unificación. En este punto se aplicarán diferentes algoritmos con el fin de ir unificando tanto las personas como identificando las filiaciones contra el maestro mencionado anteriormente. Para las personas es muy interesante tener identificadas las filiaciones, pues a

[2]Disponible en https://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736177009&menu=resultados&idp=1254734710990

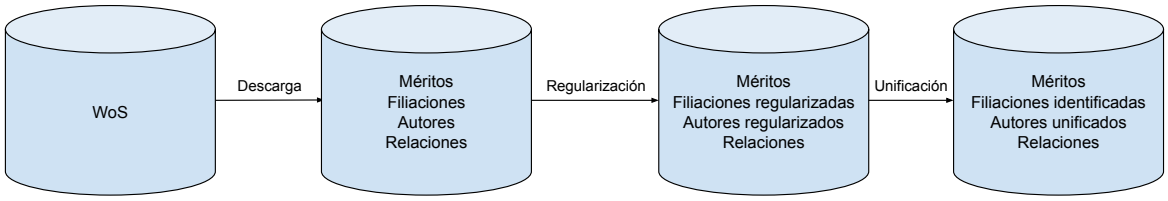


Figura 3: Visión a alto nivel de la cadena completa para resolver el problema.

la hora de compararlas es una información muy útil saber si dos personas han trabajado en la misma filiación o no. Para filiaciones también es útil tener las personas unificadas, pues una misma persona, al firmar diferentes artículos, puede escribir la misma filiación de diferentes formas.

2.2. Maestro de filiaciones

Para tener un marco de referencia de filiaciones, dispondremos de lo que llamaremos un *maestro* de filiaciones. En él se dispondrá de diferente información para cada filiación como son diferentes formas de escritura del nombre, su ubicación o ubicaciones geográficas y también relaciones de dependencia de padres e hijos, esto es, dada una filiación, tendremos las relaciones con las filiaciones de las que depende, o las filiaciones que dependen de ella. Por ejemplo, dada la Universidad de Zaragoza, tendremos como hijos todas sus facultades, departamentos y centros que dependan de ella. De la misma forma, para un departamento de la Universidad de Zaragoza, tendremos que es de la Universidad de Zaragoza. Nótese que el nombre de una universidad es suficiente para identificarla, pero el nombre de un departamento o facultad no, ni siquiera conociendo la ciudad a la que pertenecen, ya que puede haber dos filiaciones de este tipo con el mismo nombre en el mismo lugar que no sean la misma por pertenecer a dos universidades diferentes. Por tanto, en los casos de facultades y departamentos es necesario dar también la universidad a la que pertenecen, pues "Facultad de Ciencias" es un nombre que se repetirá entre las facultades de diferentes universidades.

2.2.1. Fuente geográfica

Una información útil para complementar la información de las filiaciones, es su ubicación geográfica. En el caso de las filiaciones nacionales, cada una estará en uno o más municipios, provincias y comunidad autónoma, como la UNED. Para ello se dispone de un catálogo de 253 países a nivel mundial. En el caso particular de España se dispone de municipios junto con su provincia y comunidad autónoma proveniente del Instituto Nacional de Estadística (INE)[3]. Con esto se construirá un corpus de lugares de forma que para cada lugar tendremos:

- id_lugar: Identificador del lugar.
- tipo: Tipo de lugar, 0 país, 1 comunidad autónoma, 2 provincia y 3 ciudad.
- Nombres: Lista de nombres del lugar.

[3]Datos disponibles en https://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736177031&menu=ultiDatos&idp=1254734710990

- Nombres regularizados. Lista anterior, pero con el texto convertido a ASCII y en minúsculas.
- Padres: Lista de identificadores de lugares con un nivel superior (siendo el 0 el superior de todos)
- Hijos: Lista de identificadores de lugares con un nivel inferior (siendo el 3 el inferior de todos)

En total el corpus cuenta con 8455 registros, donde 253 corresponden a países, 19 a comunidades autónomas, 52 a provincias y 8131 a municipios. Todos los municipios del corpus son españoles. En la Figura 4 se muestra cómo se distribuyen estos municipios españoles entre las provincias.

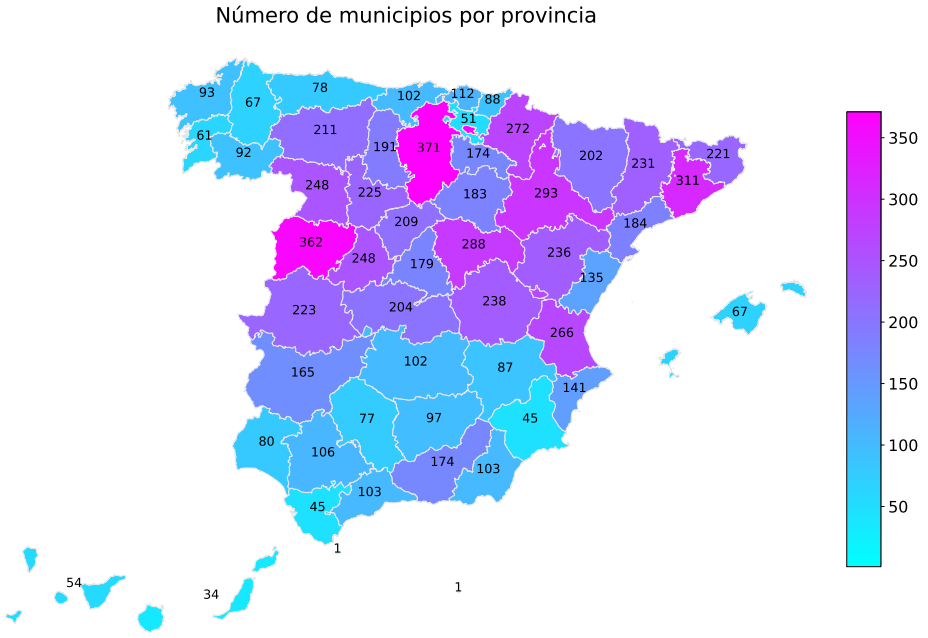


Figura 4: Distribución del número de municipios por provincia en el corpus de lugares.

2.2.2. Maestro de filiaciones

El maestro de filiaciones contiene 17016 registros en total y todos son de ámbito nacional. En la Tabla 1 se muestra cómo se distribuyen en las diferentes clasificaciones disponibles.

Este maestro de filiaciones es fruto del trabajo de Kampal a lo largo de los años. En él hay una clasificación por tipo de filiación, se dispone de algunos acrónimos, CIFs e información varia relacionada con cada filiación. Nótese que por filiación se entiende hasta un departamento por sí solo como puede ser "Departamento de Métodos Estadísticos". Junto a cada tipo hay asignado un nivel, el cual hace referencia a una jerarquización de las filiaciones. Como nivel 0 tenemos los tipos universidad, hospital y otros. Como nivel 1 están las facultades, institutos, centros y escuelas. Finalmente, como nivel 2 tenemos los departamentos. Esto tiene relación con otra información que disponemos en este maestro, que son las relaciones entre filiaciones. Para cada filiación tenemos cuáles son sus "hijos" y "padres". Por ejemplo, para la Universidad de Zaragoza, sus hijos serán todas las facultades, departamentos, centros, institutos y escuelas que

Tipo	Número de filiaciones	Nivel
Universidad	96	0
Hospital	81	0
Facultad	695	1
Instituto	618	1
Centro	1417	1
Escuela	467	1
Departamento	3319	2
Otros	11018	0
Total	17016	

Tabla 1: Número de filiaciones por tipo y en total en el maestro de filiaciones. También se muestra el nivel de la filiación, el cual hace referencia a la jerarquización de los diferentes tipos.

dependan de ella, mientras que para el Departamento de Métodos Estadísticos sus padres serán la Facultad de Ciencias y la Universidad de Zaragoza.

Al ser éste un maestro completo con toda esta información, el objetivo del trabajo será comparar las filiaciones que descarguemos de WoS contra éste y asignarlas contra una filiación del mismo. Si no se encuentra ninguna filiación del maestro con suficiente evidencia de que sean la misma, significará que la filiación que se quiere identificar no existe a nivel nacional o no es ninguna de las que están el maestro.

2.3. Regularización de la información de los artículos

La información que viene en cada artículo tiene un formato que puede tener pequeñas variaciones entre artículos y de acceso no inmediato para nuestro problema. Es por ello que lo primero será hacer un proceso que llamaremos *regularización* el cual consiste en estandarizar la información en cada artículo de forma que toda ella esté estructurada de igual forma.

En este trabajo emplearemos "*PySpark*"[4], que es una herramienta extensamente utilizada en el ámbito del "*Big Data*". Esta herramienta incorpora lenguaje SQL y NoSQL, permitiendo organizar la información en "*dataframes*", los cuales se pueden ver como tablas en un esquema de bases de datos relacional con la ventaja de que en sus atributos o campos se pueden incluir estructuras de datos más complejas que en los lenguajes SQL incorporando funcionalidades propias de lenguajes NoSQL.

Tras la descarga de los diferentes artículos, se organizará la información en tres tablas con la información de personas, artículos y filiaciones respectivamente y una cuarta tabla adicional que contenga la relación entre ellas. Para cada artículo, se extraerán los autores que participan en él, sus filiaciones y sus relaciones. Por ejemplo, en el artículo mostrado en la Figura 2 tenemos una primera persona con tres referencias a filiaciones, la segunda con dos, la tercera con una y la cuarta con otras dos. De esta forma, este artículo producirá un registro en la tabla de artículos y cuatro registros en la tabla de personas. En cuanto a las filiaciones, se introduce el concepto de "participación".

Una participación es una cadena completa que puede incluir una o varias filiaciones en ella y que pueden tener relación entre ellas. Por ejemplo, la primera participación de la Figura 2 se

[4]Documentación disponible en https://spark.apache.org/docs/latest/api/python/index.html

correspondería con la cadena de texto "Univ Zaragoza, Dept Econ, Zaragoza, Spain". De esta participación se generan dos registros en la tabla de filiaciones, uno para "Univ Zaragoza" y otro para "Dept Econ". Estos dos registros tendrán un identificador de la filiación único "f_id" y un identificador único de la participación "id_part". De esta forma, en el artículo de ejemplo hay cinco participaciones y cada una de ellas tiene dos filiaciones. La penúltima y última parte de la participación son la ciudad y el país, por lo que no se incluyen como filiaciones, sino que se añaden como el lugar de esa participación.

A modo esquemático, se presenta en la Figura 5 el esquema relacional de la información para los diferentes artículos.

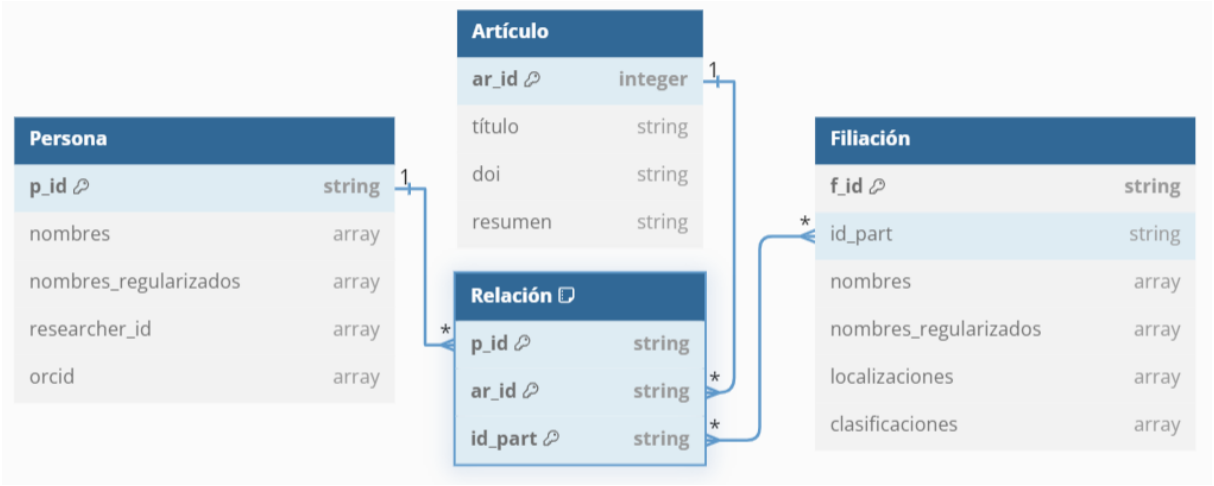


Figura 5: Esquema de las tablas que dispondremos con sus relaciones.

En este proceso de regularización también se procesa y enriquece la información para las personas y filiaciones. En todo el proceso de regularización se convierten las cadenas de texto a ASCII y minúsculas para una mejor comparación entre ellas. De forma particular para personas y filiaciones se realiza lo siguiente:

- Regularización de personas. Para las personas se realiza un proceso de regularización del nombre. Este proceso consiste en corregir posibles erratas en los nombres y apellidos con ayuda de un corpus de nombres y apellidos externo. Cada palabra en el nombre de la persona se compara con este corpus para identificar si se trata de un nombre o un apellido y corregir posibles erratas en el nombre. Para esto, se utiliza la distancia de Levenshtein[5], la cual es una distancia de edición. Esto permite corregir pequeños errores del tipo "Davod" que debería ser "David" por ejemplo. Además, tener identificado el nombre en el corpus nos ayudará en la unificación de personas, pues, por ejemplo, aunque "Antonio" y "Antonia" tengan distancia de Levenshtein 1 por diferenciarse en una letra, sabemos que son nombres completamente distintos.

  También en este punto se extraerán las palabras que se corresponden al nombre y las que se corresponden al apellido, además de las iniciales, pues puede haber casos como "Muñoz, D." donde no sabemos su nombre pero sí su inicial que es "D".

- Regularización de filiaciones. En este proceso, además de convertir el texto a ASCII y minúsculas, se eliminan "*stop words*" y se intenta extraer una clasificación a partir de las

[5] https://en.wikipedia.org/wiki/Levenshtein_distance

palabras que contiene la filiación. Si contiene palabras como “universidad”, “universitat” o “university”, se clasifica como Universidad, y además, la palabra se reemplaza por “univ” para poder hacer una mejor comparación en los siguientes procesos entre diferentes formas de escritura. Lo mismo se hace para extraer la clasificación de Departamento, Facultad, Instituto, Escuela o Centro, las cuales se abrevian como “dept”, “fac”, “inst”, “sch” y “ctr” respectivamente. En la Tabla 2 se muestra la conversión de las palabras que se abrevian. En el caso de los artículos y algunas preposiciones, se eliminan de la cadena de texto.

Abreviatura	**Palabra maestra**	**Equivalentes**
univ	Universidad	panepistimio, univerzite, egyetemi, universiteit, haskola, univerzitet, universitario, universitaria, universite, universidade, universitat, universidad, university, sveuciliste, universitate, univerzita, universitet, uniwersytet, ollscoile, universita, universiteti, unibertsitatea, agr-universitat
		la, el, les, du, del, the, las, di, of, los, da, de, a, y, en
ctr	Centro	center, centro, cntr, centre
politecn	Politécnica	politecnica, politecnico, polytecn
dept	Departamento	departamento, dep, depto, dpto, department, departament
sch	Escuela	escuela, school, escola, colegio, xescuela
inst	Instituto	instituto, institute, institut, inst, insituto
fac	Facultad	facultad, faculty, facultat, xfacultad
hosp	Hospital	hospital, agr-hospital, hospitalaria, hospitalario
adm	Administración	administration, administracion, administracio
inf	Infraestructura	infraestructure, infraestructura
fund	Fundación	fundación, fundation, fundacio, fundacion, agr-fundacio, fundaciones
asoc	Asociación	asociación, asociacion, aso, asoc
bib	Biblioteca	biblioteca, biblio, bilioteca
ud	Unidad	unidad, ud, udad

Tabla 2: Abreviaturas utilizadas en la regularización de las filiaciones. Las palabras que se encuentren en la columna “Equivalentes” se sustituyen la palabra en la columna “Abreviatura”.

Además, para las filiaciones contamos muchas veces con su ciudad y país, por lo que se buscan estas ciudades y países en el corpus de lugares que tenemos usando también la distancia de Levenshtein para identificarlas en cada filiación.

2.4. Unificación de autores

Un autor publica artículos en repetidas ocasiones bajo las mismas filiaciones, pero las escribe de manera distinta, por tanto, unificar autores nos permitirá tener diferentes ejemplos de escritura de una misma filiación.

Como se puede ver en la Figura 1b una misma persona ha escrito su nombre como "Molina, J-A", "Alberto Molina, J." ó "Molina, Jose Alberto". Mediante los algoritmos de regularización implementados, somos capaces de extraer y estandarizar la información de estos nombres, dividiéndolo en iniciales, nombres y apellidos.

Nombre de entrada	Iniciales	Nombres regularizados	Apellidos regularizados
Karakas, S. Pinar	sp	[(0,pinar)]	[(1,karakas)]
Aguado-Linares, P	p	[]	[(0,aguado),(0,linares)]
Cuadrado, Jrge Pérez	j	[(0,jorge)]	[(0,perez),(0,cuadrado)]

Tabla 3: Ejemplo del funcionamiento de la regularización de nombres de personas. Se convierte el texto a ASCII en minúsculas, se extraen las iniciales del nombre, se corrigen posibles erratas y se divide en nombres y apellidos. Los nombres y apellidos es una lista de duplas, donde el primer elemento de la misma es un flag que significa que dicho nombre o apellido está fuera del corpus de nombres y apellidos. 1 indica que el nombre está fuera del corpus y 0 que está contenido en el mismo.

En la Tabla 3 se muestra un ejemplo de cómo funciona este algoritmo de regularización para tener una idea más clara. El objetivo del algoritmo de regularización de nombres es estandarizar la información para que toda tenga la misma estructura y sea más sencilla la comparación entre personas. La información del nombre de una persona se divide en 3 grupos, las iniciales, los nombres y los apellidos. Añadir las iniciales es necesario debido a que muchas veces los autores escriben sus nombres acortados por la inicial, por lo que aunque no sepamos su nombre, sí sabemos su inicial, lo cual es de ayuda. Los nombres y apellidos pueden ser uno o varios por persona, por lo que se almacenan listas de nombres y apellidos. Además, al disponer de un corpus de nombres y apellidos a nivel nacional extraído del INE, podemos corregir erratas y además determinar si una cadena de texto se trata o no de un nombre o un apellido. Esto se puede ver en la Tabla 3 en el último ejemplo. El autor ha escrito su nombre con una errata, poniendo "Jrge" en lugar de "Jorge", que se puede corregir gracias al corpus de nombres. En segundo lugar, tenemos la cadena de texto "Pérez", que por el lugar que ocupa, se podría confundir con un segundo nombre, sin embargo, de nuevo con la ayuda del corpus de nombres y apellidos, determinamos que se trata de un apellido muy probablemente. Esto último en ocasiones no es posible, pues hay nombres como "Martín" que pueden ser un apellido perfectamente.

Por otra parte, gracias a la regularización de filiaciones y sus respectivos lugares, somos capaces de recuperar con qué filiaciones ha firmado un autor y dónde, de esta forma, se añaden como información a la persona una lista de filiaciones y otra de lugares. Los lugares obtenidos tras el proceso de regularización están bien identificados contra el corpus de lugares disponible y además a cuatro niveles, ciudad, provincia, comunidad autónoma y país. Para poder usar la información de las filiaciones, primero se aplica el algoritmo presentado en 2.5.1 Identificación de filiaciones con similaridad por distancia de edición que sirve para hacer una primera identificación de las mismas.

Además, existe la posibilidad de usar palabras clave las cuales vienen de la propia fuente.

Con todo esto se construyó un modelo que tiene estas 4 variables en un inicio: nombres regularizados, lugares, filiaciones y palabras clave. Sin embargo, tras realizar diferentes pruebas de distintos modelos, se decidió dejar de usar las palabras clave. El motivo es que no eran muy relevantes en estos modelos, principalmente debido a que estas palabras clave provenientes de WoS son muy específicas y no aportan mucha información a la hora de comparar dos personas con dos artículos distintos. Además, ralentizaban el proceso.

De esta forma, se decidió usar únicamente estas 3 variables, nombres regularizados, lugares y filiaciones. Cada una de estas variables se subdivide en más, quedando la siguiente información para cada persona:

- Nombres regularizados (Lista).
  - Iniciales (texto).
  - Nombres (Lista de duplas, el primer elemento es un flag de si el nombre está en el corpus o no y el segundo el nombre)
  - Apellidos (Lista de duplas, el primer elemento es un flag de si el apellido está en el corpus o no y el segundo el apellido)
- Lugares (Lista).
  - id_ciudad (entero).
  - ciudad (texto).
  - flag_ciudad: 1 si está desinformado, 0 si está informado (entero).
  - id_provincia (entero).
  - provincia (texto).
  - flag_provincia: 1 si está desinformado, 0 si está informado (entero).
  - id_ccaa (entero).
  - ccaa (texto).
  - flag_ccaa: 1 si está desinformado, 0 si está informado (entero).
  - id_pais (entero).
  - pais (texto).
  - flag_pais: 1 si está desinformado, 0 si está informado (entero).
- Filiaciones (Lista)
  - id_nivel_0 (entero).
  - nivel_0 (texto).
  - flag_nivel_0: 1 si está desinformado, 0 si está informado (entero).
  - id_nivel_1 (entero).
  - nivel_1 (texto).
  - flag_nivel_1: 1 si está desinformado, 0 si está informado (entero).
  - id_nivel_2 (entero).
  - nivel_2 (texto).
  - flag_nivel_2: 1 si está desinformado, 0 si está informado (entero).

2.4.1. Algoritmo de unificación

Para unificar personas se implementó un algoritmo que es común a todos los modelos que presentaremos, la única diferencia es el modelo que calcula la distancia entre dos personas. Sea un conjunto de N personas que deseamos unificar, no es viable calcular la matriz de todas las distancias, ya que supondría un cálculo de orden N^2, y N va a ser un número alto ya que se quieren analizar grandes cantidades de información. El alcance de este algoritmo es el de unificar millones o decenas de millones de personas, que son las que participan en la producción científica a nivel nacional e internacional. Esto supone hacer una enorme cantidad de operaciones y además un almacenamiento masivo de datos.
Dado un conjunto de N personas se sigue como procede.

1. Se inicializa la lista de salida de personas unificadas con la primera persona del conjunto.
2. Se lee la siguiente persona del conjunto de personas de entrada.
3. Se recorre la lista de personas unificadas de salida y se calcula la distancia entre la persona de entrada y las personas de la lista de salida.
4. Se toma la menor distancia de todas junto con el identificador de la lista de salida que se corresponda.
   a) Si la distancia es menor que un umbral, entonces se consideran que son la misma persona y se actualiza dicho registro de la lista de salida añadiendo el identificador de la persona de entrada y combinando la información de ambas personas para posteriores comparaciones.
   b) Si la distancia es mayor o igual al umbral, entonces no se ha encontrado una persona igual y, por tanto, se añade a la lista final de personas unificadas.
5. Se vuelve al segundo punto hasta recorrer todo el conjunto de personas de entrada.
6. Se devuelven los clústeres formados, esto es, las personas que se deben unificar entre ellas.

2.4.2. Unificación de autores con C

Los primeros modelos que se han probado están basados en un modelo elaborado en lenguaje de programación C. El motivo de usar este lenguaje de programación es la eficiencia del mismo, pues en lenguajes interpretados como Python la velocidad del algoritmo desciende considerablemente. Este modelo se basa en una media ponderada donde se calcula una distancia entre nombres, otra entre lugares y otra entre filiaciones y se calcula una distancia final mediante la fórmula

$$d = \frac{\sum_{i \in V} \beta_i f_i d_i}{\sum_{i \in V} \beta_i f_i} \tag{1}$$

donde $V = \{\text{nombre}, \text{lugar}, \text{filiación}\}$ es el conjunto de variables, β_i son los coeficientes asociados a cada variable i, d_i son las distancias calculadas para cada variable i y f_i es un flag que toma valor 1 cuando esa variable i está informada y 0 cuando lo está, todo con $i \in V$. La introducción de la variable f_i es porque puede haber casos donde, por ejemplo, no tengamos la filiación de

esa persona porque los algoritmos para detectar las filiaciones todavía no han sido capaces de detectar ninguna filiación.

No se entrará en detalles de cómo se calculan estas distancias, pues hay muchos detalles y se podría extender mucho la explicación. La base de la distancia en todos los casos es la distancia de Damerau-Levenshtein, la cual se diferencia en que la transposición de dos letras tiene distancia 1 mientras que con la distancia de Levenshtein tendría distancia 2. En orden de optimizar el tiempo del cálculo de la distancia, para los lugares y las filiaciones, no se tiene en cuenta el texto de las mismas, sino los identificadores correspondientes. Cuando se comparan dos ciudades, provincias, comunidades autónomas, países o filiaciones entre niveles iguales, se toma distancia 0 si ambos identificadores son el mismo, y distancia 1 si no. De esta forma, la distancia entre, por ejemplo, "palencia" y "valencia" es la misma que entre "palencia" y "zaragoza", siendo 1 en ambos casos.

En el caso de la distancia entre los nombres sí se emplea la distancia de Damerau-Levenshtein y ésta se normaliza por la longitud de la mayor cadena que se compara, así, la distancia de Damerau-Levenshtein entre "ejemplo" y "examples" es de $4/8 = 0.5$, pues la distancia de Damerau-Levenshtein es 4 y la longitud máxima entre las dos cadenas es 8. Por otra parte, se tienen en cuenta los flags asociados a los nombres, pues si dos nombres que se comparan tienen el flag que indica que son nombres del corpus, hay una penalización en la distancia. La distancia entre "mario" y "maria" no puede ser la misma que entre "mari" y "mario". En ambos casos la distancia es de $1/5 = 0.2$, pero en el primer caso los dos son nombres del corpus y por tanto sabemos que son diferentes, y en el segundo caso el primer nombre no está en el corpus y seguramente se trate de una errata en el mismo. También se tienen en cuenta otros aspectos y soluciones específicas a patrones encontrados en los datos donde la distancia no se corresponde con lo esperado en un principio.

La distancia entre palabras clave, aunque finalmente se han desechado, se basaba en la distancia de Levenshtein normalizada también.

2.4.3. Unificación de autores con redes neuronales

La distancia de Levenshtein tiene sus limitaciones para nuestro problema, pues es una distancia de edición. En la actualidad, cuando se trata de analizar texto, se recurre a redes neuronales capaces de extraer información ya no solo de una palabra, sino del contexto o resto de palabras que la rodea. Los modelos de redes neuronales que tratan sobre texto son conocidos como modelos de procesamiento de lenguaje natural, en inglés *Natural Language Processing* (NLP). Estos modelos de procesamiento de lenguaje natural se basan en la *tokenización*, la cual consiste en dividir un texto en *tokens*. Existen diversas formas de tokenizar un texto, la primera forma que se nos puede ocurrir es dividirlo por palabras, lo cual es una buena primera aproximación. Otra opción podría ser dividirlo por letras, pero esta tokenización generaría muchísimos tokens por cada texto, y además los tokens que se generarían no tendrían información semántica, por lo que no es aconsejable. Los tokenizadores también se pueden ajustar para que capturen cuál es la mejor manera de tokenizar un texto. Un ejemplo de tokenización es

"This is how tokenization works": ["this", "is", "how", "token", "##ization", "works"].

En este ejemplo, además de dividir el texto en palabras, también se parte la palabra "tokenization" en "token" y "ization", las dos # sirven para indicar que dicho token se ha partido del

texto anterior. Esto tiene sus ventajas, pues, por ejemplo, en otra palabra como "polarization" ocurriría lo mismo dividéndolo en "polar" y "ization" y en la palabra "tokenize" se divide en "token" y "ize". Vemos así como este tokenizador es capaz, de forma aproximada, de dividir palabras en lexemas y morfemas, lo que le dota de un mayor entendimiento del lenguaje en sí.

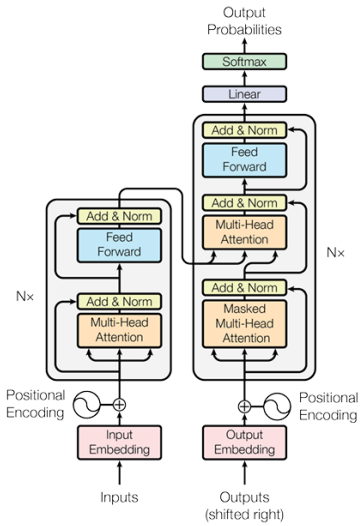


Figura 6: Esquema de la arquitectura de un Transformers. Figura extraída de [2].

Para poner en contexto la arquitectura de los modelos que hemos usado en este proyecto, vamos a hacer primero un pequeño repaso hablando de los *Transformers*. Los Transformers son una arquitectura de redes neuronales presentada en 2017 por Vaswani et al.[2]. Esta arquitectura se descompone principalmente en dos bloques, un codificador (*encoder*) y un decodificador (*decoder*). El codificador convierte la secuencia de entrada, texto, a una representación interna en forma de vector. El decodificador, toma esta representación interna y genera una secuencia de salida, la cual se convierte a texto de nuevo. La clave de esta arquitectura es su mecanismo de atención *Multi-Head Attention* que permite al modelo determinar qué partes de la secuencia de entrada son relevantes para cada token en la secuencia de salida. De manera visual, en la Figura 6 se muestran estos dos bloques con su mecanismo de atención.

A partir de la salida de los Transformers, Google desarrolló en 2018 un modelo de representación del lenguaje llamado BERT[3], siglas de *Bidirectional Encoder Representations from Transformers*, el cual proviene del codificador del Transformers. La característica de este modelo es su bidireccionalidad. Otros modelos, como GPT, siglas de *Generative Pre-trained Transformers*, el cual proviene del decodificador del Transformers y que hoy en día es ampliamente conocido desde la salida de chatGPT, son unidireccionales. Mientras que en modelos como GPT solo se tienen en cuenta los tokens a la izquierda de uno mismo, en BERT se tienen en cuenta tanto los tokens a la izquierda como los que tiene a la derecha. Por tanto BERT lee el texto en ambas direcciones, lo que permite capturar relaciones complejas y dependencias entre palabras distanciadas en el texto.

Al introducir un par de frases en BERT, en primer lugar, se tokeniza cada una de las frases y saca el *embedding* de cada token. El embedding es un vector que representa a dicho token, el cual es propio de cada modelo de BERT y depende de cómo se haya preentrenado. También añade un embedding para cada frase. Esto es una forma de que BERT sepa dónde empieza y termina cada una de las frases. Podría ser el embedding de la primera frase un vector de todo 0 y en la

segunda frase un vector de todo 1, por ejemplo. Como BERT por sí solo no tiene información de la posición de cada token en el texto, se añaden embeddings que dan información de la posición del token en el texto. De forma adicional, BERT utiliza tres tokens especiales para esta tarea. Para cada token de entrada, se suman el embedding del propio token, del segmento y de la posición para generar un embedding final. En la Figura 7 se muestra un ejemplo con las frases "my dog is cute" y "he likes playing".

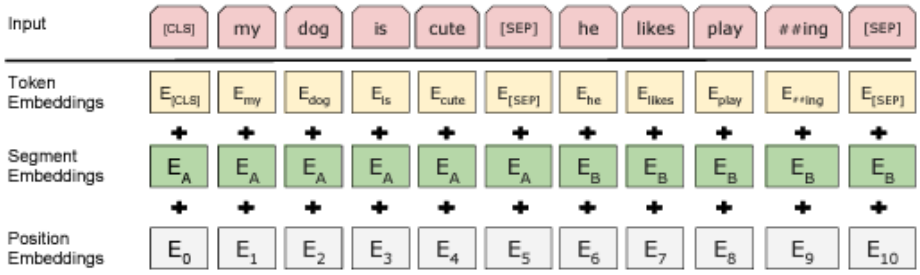


Figura 7: Representación de la entrada en BERT. Los embeddings de entrada son la suma de los embeddings de los tokens, los embedding de los segmentos y los embeddings de posición. Figura extraída de [3].

La arquitectura que usaremos en este trabajo se basa en BERT usando redes siamesas. En 2019 Nils Reimers e Iryna Gurevych sacaron una nueva arquitectura sobre la de BERT llamada Sentence Transformers o SBERT[4]. En los Sentence Transformers se añade una capa de *pooling* para generar un embedding de tamaño fijo de la frase, llamado vector semántico. En nuestro caso será un vector semántico de 512 componentes. El *pooling*, o agrupación, se puede hacer de diferentes formas, en nuestro caso será la media de todos los embeddings generados por el modelo BERT para una frase, aunque se podría tomar el embedding de salida del token CLS. Para hacer un ajuste fino y entrenar el BERT, se genera una red siamesa, es decir, una red compuesta de dos copias de la red las cuales comparten los coeficientes, para poder así generar vectores semánticos de forma que representen el texto de entrada y se puedan comparar textos mediante similaridad coseno, es decir, mediante el producto escalar normalizado de estos vectores.

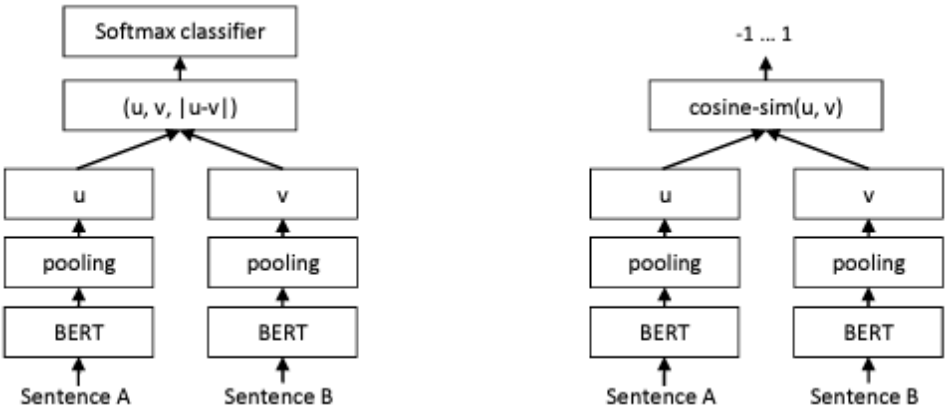


Figura 8: Arquitectura de los Sentence Tranformers en función del problema a resolver. En todas las arquitecturas se tiene una red siamesa de BERT, se aplica una agrupación o pooling para sacar un vector semántico y se comparan dichos vectores en función del objetivo del problema. Figura extraída de [4].

En la Figura 8 se muestra la arquitectura para un Sentence Transformers. En problemas en los que queramos comparar pares de frases, la arquitectura consta de dos redes siamesas de BERT, a las cuales se introduce una frase de la pareja a cada BERT por separado. Para cada frase se sacan los embeddings de cada token de entrada como en la Figura 7 y se le aplica el pooling para sacar un vector semántico para cada frase, $\vec{u}$ y $\vec{v}$ respectivamente. En función del problema, y la función de pérdida que se use en el entrenamiento, se opera con estos dos vectores para ver cuánto de parecido o diferentes son. En este trabajo también emplearemos funciones de pérdida que, en lugar de trabajar con parejas de frases, trabajan con ternas. Las ternas están constituidas por una frase ancla, una frase positiva y una frase negativa. En este caso, en lugar de usar redes siamesas, se usa una triple red, donde el esquema de la arquitectura es el mismo que en las redes siamesas pero añadiendo una red BERT más y teniendo al final tres vectores semánticos. En todos los casos, los pesos de las redes BERT son compartidos en toda la red completa.

2.5. Identificación de filiaciones

El objetivo de estos procesos es que para cada entrada en la tabla de filiaciones de la Figura 5, se añada una columna donde se recoja el identificador, o identificadores, de la filiación del maestro presentado en el apartado 2.2 Maestro de filiaciones. De esta forma las filiaciones extraídas del WoS se habrán identificado de forma unívoca. Para este objetivo se emplearán dos estrategias diferentes, la primera basada en una modificación de la distancia de Levenshtein para comparar conjuntos de palabras[6]. En la segunda estrategia se emplearán redes neuronales de procesamiento lenguaje natural con el objetivo de capturar de una mejor forma la información de dicha filiación.

2.5.1. Identificación de filiaciones con similaridad por distancia de edición

En el primer método para identificar filiaciones se usa como base la distancia de Levenshtein. La distancia de Levenshtein[7] es una distancia de edición que nos permite tener una métrica de cómo de similares son dos cadenas de texto. Mide el número mínimo de operaciones para pasar de una cadena de texto a otra siendo estas operaciones la inserción, eliminación o sustitución de un caracter. Como las filiaciones suelen ser una de cadena de texto de unas pocas palabras, en lugar de la distancia de Levenshtein como tal, se usará una distancia de Levenshtein "por palabras" de forma que dadas dos listas de palabras, por ejemplo ["dept","health","care"] y ["care", "hlth", "department"], encuentra la ordenación con menor distancia de Levenshtein sin penalizar por mover de posición las palabras de la lista. En este caso se emparejaría "dept" con "department", "care" con "care" y "health" con "hlth", calculando las distancias de Levenshtein de cada pareja para hacer un promedio con la longitud de dichas parejas.

En muchos casos, sobre todo a nivel de universidades, es suficiente con sacar la distancia anterior, pues el nombre de las universidades no suele variar mucho entre firmantes. Sin embargo, cuando tratamos de identificar facultades, centros, institutos, departamentos, hospitales u otras filiaciones, el problema es más complejo. Las formas diferentes de escritura de una misma filiación aumentan. Además, encontramos el problema del idioma, en ocasiones los investigadores, por

[6]Documentación de la librería de Python en https://github.com/rapidfuzz/Levenshtein

[7]Más información sobre la distancia de Levenshtein en https://en.wikipedia.org/wiki/Levenshtein_distance

ejemplo, escriben las filiaciones en español, otras veces en inglés, y minoritariamente en otros idiomas como catalán, euskera, etc.
El algoritmo planteado para identificar las filiaciones es el siguiente:

1. De manera paralela se analiza cada participación con todas sus filiaciones.
2. Se hace una primera pasada buscando acrónimos y filiaciones de nivel 0 (universidades, hospitales y otros).
    2.1 Búsqueda de acrónimos mediante expresiones regulares. Si lo encontramos, se elimina de la cadena. También se comprueba que el resto de la cadena se corresponda con la misma filiación del acrónimo.
    2.2 Búsqueda por distancia de Levenshtein filtrando por lugar y clasificación.
    2.3 Si no se encuentra a nadie, búsqueda filtrando por lugar solamente.
    2.4 Si no se encuentra a nadie, búsqueda filtrando por clasificación.
3. Se hace una segunda pasada buscando filiaciones de nivel 1 y 2 (facultades, institutos, centros, escuelas, y departamentos) apoyándonos en lo encontrado en el punto 2.
    3.1 Búsqueda por distancia de Levenshtein en filiaciones hijas de las filiaciones encontradas en el punto 2.
    3.2 Si no se encuentra ninguna, búsqueda filtrando por lugar y clasificación.
    3.3 Si no se encuentra ninguna, búsqueda filtrando por lugar solamente.
    3.4 Si no se encuentra ninguna, búsqueda filtrando por clasificación.

La búsqueda se basa en calcular la distancia de Levenshtein "por palabras" de la filiación con las del maestro que cumplan el filtro. Se toma la de menor distancia, y si está por debajo de un umbral, se considera que son la misma filiación.

Comentar que este algoritmo se ha ido mejorando. Como grandes cambios se mencionan tres, que posteriormente, en la sección 3.3.2 Identificación de filiaciones con similaridad por distancia de edición, se compararan. En la primera versión, solo existía una pasada y se buscaban todo tipo de filiaciones en la misma sin utilizar la información de las relaciones entre filiaciones del maestro. En la segunda versión se añadieron las dos pasadas buscando primero filiaciones de nivel 0, pues una vez identificadas estas filiaciones en la primera pasada, se usan las relaciones entre filiaciones del maestro para acotar más la búsqueda. Esto es muy útil para facultades, escuelas y departamentos, pues entre universidades se repiten los nombres y con la distancia de Levenshtein podemos tener múltiples coincidencias para un mismo nombre y lugar. La tercera mejora consistió en agregar como sinónimos al maestro de filiaciones los nombres traducidos al inglés. Esto también ayudo a nivel de facultades y departamentos, pues "health science dept" y "dept ciencias salud" tienen una distancia de Levenshtein grande entre sí, aunque realmente referencian la misma filiación.

2.5.2. Identificación de filiaciones con redes neuronales

De igual manera que con la unificación de personas, se ha probado un modelo de Sentence Transformers para calcular la similaridad entre dos filiaciones. El objetivo es desarrollar un algoritmo similar al del apartado anterior para la identificación de filiaciones por distancia de

edición, pero usando redes neuronales y, por tanto, adaptando el algoritmo a éstas. Los modelos de redes neuronales que se usan son los mismos que los presentados en 2.4.3 Unificación de autores con redes neuronales, donde están explicados.

En el apartado 3.3.3 Identificación de filiaciones con redes neuronales. Creación del algoritmo se exponen las diferentes versiones del algoritmo y diferentes pruebas realizadas con estos modelos.

2.6. Métricas utilizadas

A lo largo de la resolución se medirán las eficiencias de los modelos probados con diferentes métricas. En este apartado se explicarán dichas métricas.

El primer conjunto de métricas es propio de problemas de clasificación binaria. En estos modelos tenemos una variable objetivo y que toma valor 0 ó 1 y es conocido. Los modelos predicen el valor $\hat{y}$ de esta variable.

		$\hat{y}$ = 1	$\hat{y}$ = 0
y	1	TP	FN
	0	FP	TN

Tabla 4: Matriz de confusión. TP son los veraderos positivos, FN son los falsos negativos, FP son los falsos positivos y TN son los verdaderos negativos.

Dependiendo de las cuatro opciones que hay, salen cuatro posibles casos englobados en lo que se llama la matriz de confusión presentada en la Tabla 4. En nuestro contexto, el valor 1 representará cuando dos autores o dos filiaciones sean la misma y 0 cuando no. A partir de esta matriz de confusión, usaremos las siguientes tres métricas más para medir la bondad de nuestros modelos

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{F}_1 \text{ score} = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2}$$

La primera, precision, mide cuánto acierta el modelo cuando predice $\hat{y} = 1$, mientras que el recall o sensibilidad, mide cuántos casos con $y = 1$ acierta el modelo. A partir de estas dos, se define el F_1 score que no es más que la media armónica de precision y recall.

Los modelos que desarrollaremos, dados dos autores o dos filiaciones, devolverán un número que medirá la distancia o similaridad entre ambos. Por tanto, será necesario poner un corte sobre el cual por encima o debajo del mismo se considerará que ambas personas o filiaciones son la misma, $\hat{y} = 1$, o distintas $\hat{y} = 0$. De esta forma, para un mismo modelo, será necesario hacer un estudio para ver cuál es el mejor corte. De aquí sale el concepto de curva ROC, *Receiver Operating Characteristic*, la cual la construimos graficando el ratio de verdaderos positivos, TP/(TP+FN), frente al ratio de falsos positivos, FP/(FP+TN), para diferentes cortes. De esta curva podemos elegir el corte óptimo como el corte para el cual la diferencia entre el ratio de verdaderos positivos y el ratio de falsos negativos es máxima, por ejemplo. También, se define otra métrica a partir de esta curva ROC que es el área bajo la curva, AUC (*Area Under the Curve*). Un modelo perfecto tendría un AUC de 1, mientras que un modelo aleatorio donde predijese $\hat{y} = 1$ o $\hat{y} = 0$ con igual probabilidad independientemente de la entrada tendría un AUC de 0.5. De esta forma, modelos con un AUC más cercano a 1 son mejores.

El segundo conjunto de métricas está asociado a modelos de agrupación o *clustering*. En nuestro contexto, tendremos un conjunto de N datos que están agrupados en K clústeres. Por una parte, tendremos un vector $\vec{y}$ de N componentes donde cada componente tomará el valor del clúster al que pertenece $y_i = k$, $i = 1, ..., N$, $k \in \{1, ..., K\}$. Los algoritmos y modelos desarrollados harán su predicción de agrupación $\hat{\vec{y}}$, donde pueden predecir un número diferente de clústeres $\hat{K}$, y mediante la comparación de estos dos vectores sacaremos diferentes métricas.

La primera métrica es el ARI, *Adjusted Rand Index*, el cual mide la similaridad entre dos agrupaciones de clústeres considerando todas las parejas y contando parejas que están asignadas al mismo o diferente clúster entre el valor predicho y el real. Un valor de 1 significa una concordancia perfecta. La segunda métrica es el NMI, *Normalized Mutual Information*, la cual se basa en la información mutua cuantificando la información compartida entre un clúster predicho y otro real. De nuevo, el mejor valor es 1 al estar normalizada. La tercera métrica es la homogeneidad. Se dice que un clúster predicho es homogéneo si en él contiene solo miembros de un solo clúster de los reales. Para esta métrica es, por tanto, necesario conocer los clústeres reales. Una homogeneidad perfecta tiene valor 1. La cuarta y última métrica es el índice de Fowlkes-Mallows, FMI. Esta métrica se define como la media geométrica entre la precision y recall definidos anteriormente. Este índice va de 0 a 1, siendo 1 el mejor valor.

3. Resolución

3.1. Análisis de los artículos a analizar

Para poner a prueba los modelos desarrollados, se han descargado artículos entre 2019 y 2023 donde haya participado algún autor español (con filiación española) y con temáticas en "*Physics, Mathematical*", "*Engineering, Mechanical*", "*Computer Science, Artificial Intelligence*", "*Radiology, Nuclear Medicine & Medical Imaging*", "*Psychology, Psychoanalysis*", "*Nutrition & Dietetics*", "*Language & Linguistics*", "*Clinical Neurology*", "*Materials Science, Paper & Wood*" y "*Energy & Fuels*". La idea es tener un conjunto de datos de diferentes años para que se repitan las personas entre años, repartidos en diferentes temáticas y que no sea un volumen excesivo con el fin de acortar tiempos para hacer pruebas.

Tras el proceso de regularización, donde identificamos el país de las filiaciones, descartamos todas aquellas que no sean españolas para nuestros procesos. Tras este filtro, tenemos 53358 artículos, 183989 personas, 114042 participaciones y 238878 relaciones artículo-persona-participación. Recordamos que una participación es una cadena completa donde puede haber una o más filiaciones. De estas participaciones salen 292432 filiaciones.

En la Figura 9 se muestra un pequeño análisis del resultado de la regularización de personas. En él se muestra la distribución de las diferentes estructuras encontradas en los nombres regularizados, donde una estructura es la pareja (número de nombres, número de apellidos). Podemos ver cómo lo más común es un nombre y un apellido, seguido de un nombre y dos apellidos. Se observan también varios casos donde no tenemos el nombre pero sí uno o dos apellidos, aunque la inicial del nombre sí que la tenemos aunque no se muestre en la gráfica.

Respecto a la filiaciones, en la Tabla 5 se muestran las clasificaciones de las de entrada extraídas en el proceso de regularización. Se observa cómo hay muchas filiaciones que están categorizadas como "Otros". En esta categoría la casuística es muy grande, puede haber direcciones como "Av Complutense 40", unidades como "Alzheimers Dis & Other Cognit Disorders

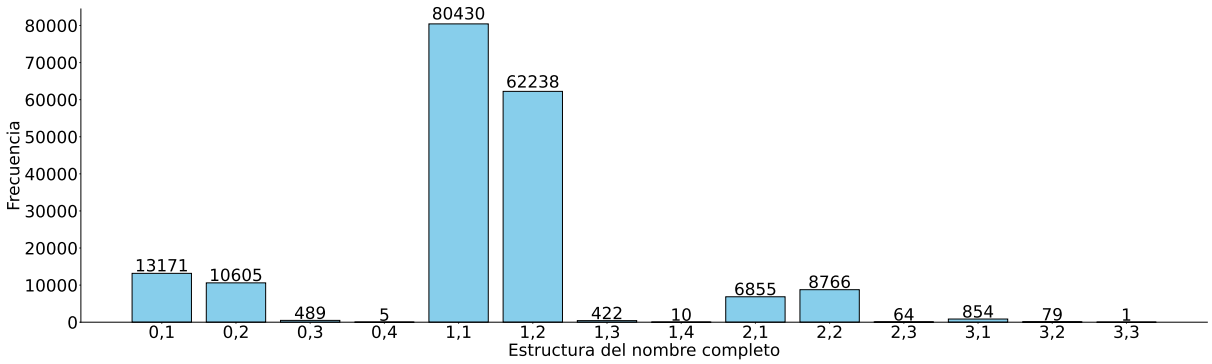


Figura 9: Distribución de las diferentes estructuras en los nombres regularizados de las personas. Una estructura es la pareja (número de nombres, número de apellidos).

Unit" o incluso acrónimos como "CSIC".

Tipo	Frecuencia
Universidad	65687
Departamento	45125
Instituto	23573
Escuela	5327
Facultad	9033
Hospital	26336
Centro	11240
Otros	106111

Tabla 5: Distribución de los diferentes tipos de filiación en los datos a analizar extraídos en el proceso de regularización.

Comentar también que en el 94.0 % de los casos se tiene la ciudad de la filiación. A nivel de provincia, disponemos de ella en el 96.4 % de los casos. Si disponemos de la ciudad y/o de la provincia, también conocemos la comunidad autónoma.

3.2. Unificación de autores

En este apartado se comentan los modelos probados, cómo se han entrenado y qué resultados han sacado.

3.2.1. Construcción de un conjunto de entrenamiento sintético

En el caso de las personas, debido al gran volumen de datos de entrada y la variabilidad que los caracteriza, es muy difícil encontrar casos reales "de calidad", esto es, casos difíciles donde poder valorar si los algoritmos funcionan bien o no. Si cogemos dos personas aleatorias, es muy probable que sean claramente distintas, una se puede llamar "Pedro Gutierrez" y la otra "Marta Godes", trabajar en filiaciones diferentes, y estar en ciudades diferentes. Dicho de otro modo, coger un dataset aleatorio en el que se produzcan suficientes casos de unificación implicaría que éste fuera demasiado grande para manejarlo de forma manual. Es por ello que se recurre a la generación de un dataset sintético, esto es, lo creamos nosotros.

La ventaja de un dataset sintético es que podemos generar tantos registros como queramos, de manera que haya suficiente volumen de datos para hacer estadística y medir el funcionamiento del modelo.

La estrategia para generar este dataset sintético es hacer ternas de personas. La primera, llamada ancla (*anchor* en inglés), es la persona de referencia, la segunda es una persona con una pequeña modificación de forma que se pueda considerar que es la misma persona que el ancla (un caso positivo) y la tercera es una persona con una modificación tal que ya no se puedan considerar la misma persona que con el ancla (caso negativo). El motivo de este enfoque es conseguir hacer casos difíciles, pues a partir de la misma persona podemos conseguir ejemplos negativos que no sean tan obvios como si escogiéramos dos personas al azar.

De esta forma, se han desarrollado diferentes criterios para generar estos datos, intentando plasmar los que utilizaría un humano para hacer esta labor.

Para generar los casos positivos, se admite una y solo una de las siguientes modificaciones:

- No cambiamos nada.
- Añadir inicial. Tomamos el nombre de la persona ancla y le añadimos una inicial. "José Ramírez" pasa a "José L. Ramírez".
- Añadir nombre. Añadimos un nombre a la persona ancla. "José Ramírez" pasa a "José Luís Ramírez"
- Reducimos el nombre. Esto es, cogemos el nombre y lo convertimos a una inicial "David Pérez" pasa a "D. Pérez".
- Eliminamos el último nombre (en caso de que tenga más de uno). "Francisco José" pasa a "Francisco".
- Eliminamos el último apellido (en caso de que tenga más de uno). "Pedro Martínez Lor" pasa a "Pedro Martínez".
- Pasamos el primer apellido a último nombre. Esto es útil para el algoritmo en C el cual utiliza los nombres regularizados. Para las redes neuronales este cambio será como no hacer nada.
- Cambiar el lugar. Cambiamos ciudad, provincia o comunidad autónoma aleatoriamente de la persona.
- Reducimos el lugar. Esto es, si tenemos informado hasta la ciudad, por ejemplo, dejamos la ciudad desinformada. Se toma ciudad, provincia o comunidad autónoma aleatoriamente.
- Cambiar filiación. Cambiamos aleatoriamente el nivel 0, 1 ó 2 de la persona.
- Reducimos filiación. Igual que con el lugar, si sabemos una filiación de una persona hasta nivel 2 (departamento), lo dejamos hasta nivel 0 (universidad) por ejemplo. Se toma aleatoriamente el nivel que se elimina.

Los casos negativos se pueden formar de dos maneras. Una de ellas es haciendo una variación en el nombre con los cambios anteriores y una variación en el lugar o filiación con los cambios anteriores también. La otra manera es hacer uno de estos cambios

- Cambiar nombre. “Pedro de la Rosa” pasa a “Antonio de la Rosa”
- Cambiar iniciales (en el caso que no tenga nombre). “P. Bonilla” pasa a ”A. Bonilla”
- Cambiar nombre pero mantener inicial. “Antonio López” pasa a “Ángel López”
- Cambiar apellido. “Fernando Alonso” pasa a “Fernando Cuquerella”
- Cambiar último apellido (si hay más de uno). “Rosa Grau Malrás” pasa a “Rosa Grau Arenas”

De esta forma, se han tomado 40.000 personas aleatorias de las disponibles en nuestros datos tras regularizar y se han hecho 5 ejemplos con cada una de ellas, por lo que se dispone 200.000 tripletas, lo que es un total de 200.000 casos positivos y 200.000 casos negativos. Se ha dividido este conjunto en aproximadamente un 30 % para validar, y el resto para entrenar. Al hacer esta división se ha tenido cuidado de agrupar todas las tripletas que provienen de la misma persona ancla para que todas estén en el mismo conjunto, o validación o entrenamiento.

3.2.2. Predicciones del dataset sintético sobre el modelo de C inicial

En primer lugar, analizaremos el funcionamiento del modelo en C inicial explicado en 2.4.2 Unificación de autores con C, con unos parámetros que venían de antemano y no se le ha realizado ningún entrenamiento, sino que ha sido revisión humana la decisión de dichos parámetros. Para ello se calculan las distancias entre las personas con el modelo en C con los parámetros por defecto.

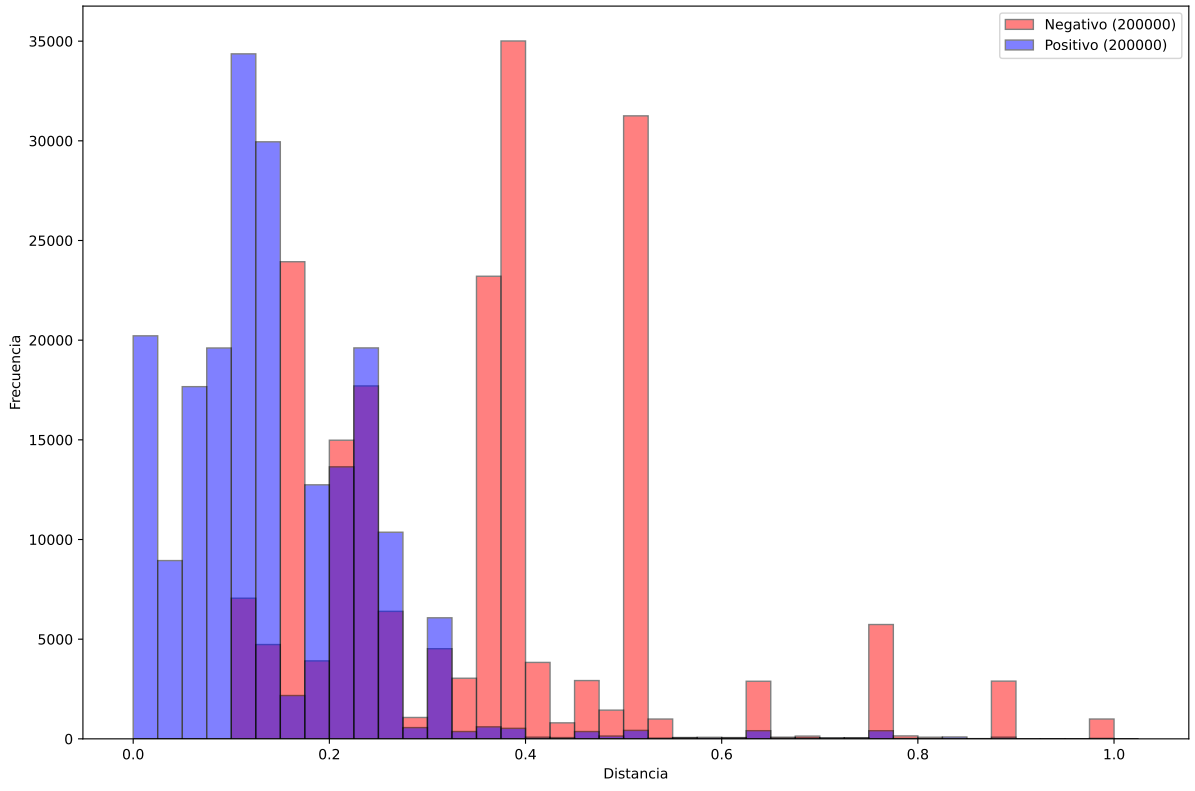


Figura 10: Distribución de distancias sobre todo el dataset sintético utilizando el modelo de C.

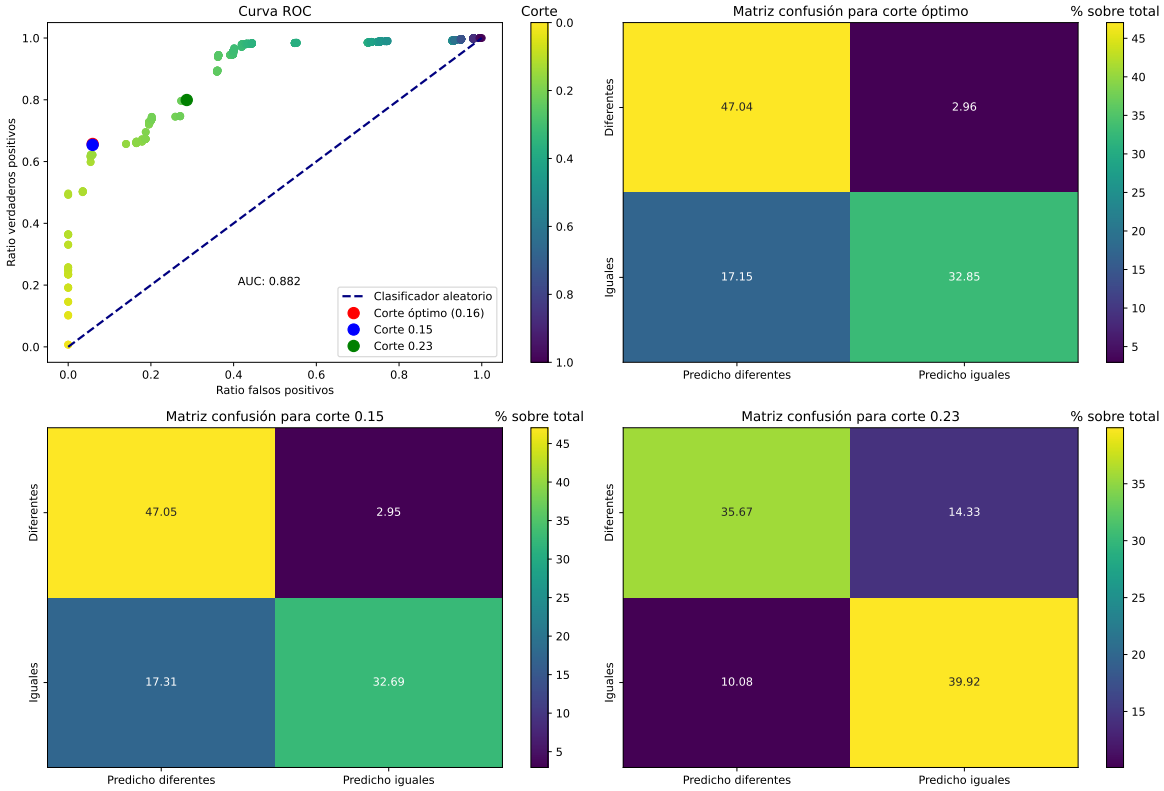


Figura 11: Curva ROC para el modelo de C inicial junto con diferentes matrices de confusión. En la esquina superior izquierda la curva ROC. En la esquina superior derecha la matriz de confusión para un corte de 0.16, el cual es el óptimo. Este corte óptimo se toma como el punto para el cual la diferencia entre el ratio de verdaderos positivos y falsos negativos es máxima. En la esquina inferior izquierda la matriz de confusión para un corte de 0.15. En la esquina inferior derecha la matriz de confusión para un corte de 0.23. El corte refleja el umbral sobre el cual dos personas se consideran la misma, de esta forma, si la distancia entre dos personas es menor al umbral, se consideran la misma persona.

En la Figura 10 se muestra la distribución de las distancias obtenidas con este modelo con las 200.000 tripletas mientras que en la Figura 11 se muestra la curva ROC para este modelo de C con los parámetros por defecto. Este modelo arroja un AUC de 0.882. En la Tabla 6 se dejan las métricas de precision, recall y F_1 score. Podemos ver que este modelo, usando el corte óptimo, deja bastantes casos sin unificar que sí debería unificar, pero, por contra, las personas que son distintas sí es capaz de discernirlas con claridad, por lo que, aunque deje personas sin unificar es un modelo que no sobreunifica. Esto es muy importante para este problema, sobretodo en etapas tempranas de la cadena. La sobreunificación conlleva a *súper personas* que acumulan muchas entradas bajo un mismo identificador. Hay casos difíciles que en etapas tempranas del proceso no se quieren unificar, como por ejemplo, dos autores pueden ser "A. Tarancón" de la Universidad de Zaragoza, localizado en Zaragoza ciudad y otro "A. Tarancón" de la Universidad de Barcelona localizado en Barcelona. Con esta información es difícil discernir si estos dos autores son el mismo o no, un humano podría pensar que sí, que únicamente es una persona que se ha movido de centro. Otra opción, en este caso la real, ya que es un caso que tenemos controlado, es que sean personas distintas, pues en verdad la primera persona es "Alfonso Tarancón" y la

segunda “Alberto Tarancón”. Si en una primera pasada se unifica erróneamente a estas dos personas, se construye un autor con nombre “A. Tarancon” y filiaciones tanto la Universidad de Zaragoza como la de Barcelona. Esto hace, por la forma acumulativa en la que está diseñado el algoritmo, que si posteriormente vienen otras dos personas con el nombre bien, esto es “Alfonso Tarancón” de la Universidad de Zaragoza y “Alberto Tarancón” de la Universidad de Barcelona, ambas sean próximas a esta persona unificada con nombre “A. Tarancón” y podríamos terminar con una persona con nombres “A. Tarancón”, “Alfonso Tarancón” y “Alberto Tarancón”, lo cual sería erróneo.

Se presentan también las matrices de confusión con cortes 0.15 y 0.23. El corte de 0.15 es el corte que se utilizaba para este modelo hasta la fecha, podemos ver cómo el corte utilizado finalmente está muy próximo y tiene unas métricas muy parecidas al óptimo que hemos obtenido con el dataset sintético, por lo que podemos validar que el corte que se estaba usando hasta la fecha era el mejor para este modelo.

El corte de 0.23 es un corte que se usa en etapas posteriores del proceso. Aquí hemos expuesto el algoritmo de unificación, pero no es el único en el que se usa este modelo. En etapas posteriores, una vez se ha hecho una primera unificación de personas, resultan N personas unificadas con varios artículos en ellas, diferentes nombres, diferentes localizaciones y diferentes filiaciones. En una etapa posterior pueden llegar M personas nuevas, estas personas nuevas se comparan con las personas unificadas previamente, y si estamos hablando que comparamos entre años de producción científica a nivel nacional, lo más probable es que la gran mayoría de personas nuevas que lleguen ya estén en las personas unificadas previamente, por lo que se sube el umbral bajo esta hipótesis. De nuevo, la finalidad de este dataset es tener una métrica sobre estos cortes que son usados en producción para conocer su buen funcionamiento.

Corte	Precision	Recall	F_1 score
0.15	0.731	0.941	0.823
0.16	0.733	0.941	0.824
0.23	0.780	0.713	0.745

Tabla 6: Métricas de precisión, recall y F_1 score para los diferentes cortes

3.2.3. Entrenamiento sobre el modelo de C con un modelo de regresión lineal

Lo primero que se planteó fue intentar mejorar el modelo ya existente que estaba hecho en C para obtener unos mejores resultados. Para ello, se adaptó el código para que devolviese tanto la distancia total como la distancia entre nombres, lugares y filiaciones. El modelo de regresión lineal propuesto es

$$d = \beta_0 + \beta_{\text{nombre}} d_{\text{nombre}} + \beta_{\text{lugar}} d_{\text{lugar}} + \beta_{\text{fil}} d_{\text{fil}} + \beta_{\text{flag fil}} I_{\text{flag fil}} \tag{3}$$

donde β_i con $i \in \{0, \text{nombre, lugar, fil, flag fil}\}$ son los coeficientes del modelo, d_i con $i \in \{\text{nombre, lugar, fil}\}$ son las distancias e $I_{\text{flag fil}}$ es una variable binaria que toma valor 1 cuando la persona tiene la filiación informada y 0 cuando no.

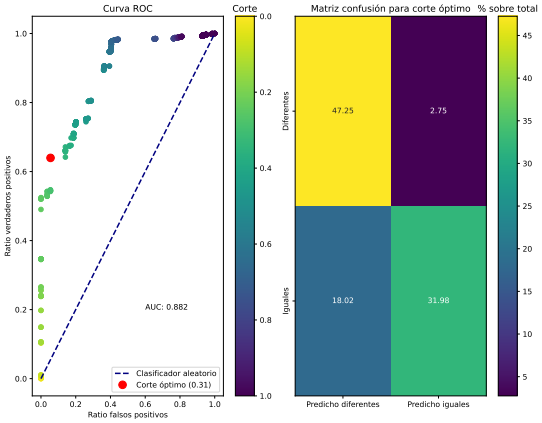
La introducción de la variable binaria es necesaria para contemplar esos casos en los que no tenemos informada la filiación. En tales casos se pone $d_{\text{fil}} = 0$ e $I_{\text{flag fil}} = 1$, de esta forma, cuando comparamos información entre personas que no tienen filiación, añadimos el término $\beta_{\text{flag fil}}$ como penalización haciendo que suba la distancia entre personas.

Variable	Coeficiente	Desv. Est.	p-valor
β_0	-0.1432	0.011	7×10^{-38}
β_{nombre}	1.3586	0.004	0
β_{lugar}	0.5157	0.002	0
β_{fil}	0.6276	0.020	1×10^{-213}
$\beta_{\text{flag fil}}$	0.1836	0.011	4×10^{-61}

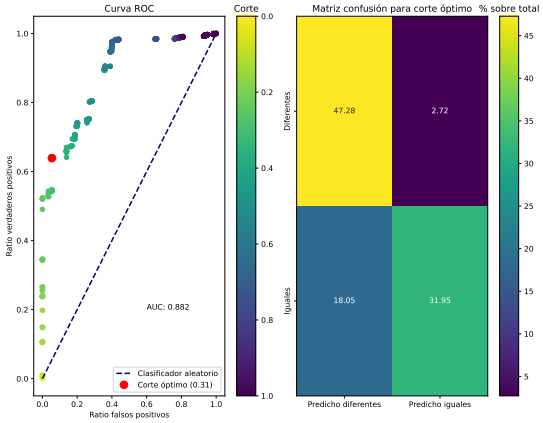
Tabla 7: Valores obtenidos para los coeficientes β_i con $i \in \{0, \text{nombre, lugar, fil, flag fil}\}$ junto con su desviación estándar y su p-valor para la regresión lineal.

En la Tabla 7 se muestran los coeficientes obtenidos tras entrenar. Los p-valores mostrados están asociados al test de Wald, el cual evalúa si dichos coeficientes son distintos de 0. Su hipótesis nula es, por tanto, que dicho coeficiente $\beta_i = 0$. Como todos los p-valores son prácticamente 0, se rechaza la hipótesis nula de que los coeficientes $\beta_i = 0$ en todos los casos y confirmamos que son estadísticamente significativos todos.

Nótese que todos los coeficientes son positivos, lo cual tiene sentido, pues a mayor distancia, mayor se espera que sea la distancia total. También ocurre con el coeficiente asociado a si la filiación está desinformada o no. Si no hay filiación, hay un término constante de 0.18 en la distancia. También se ve cómo el coeficiente asociado a la distancia del nombre es el que tiene mayor peso. Esto es esperable también, pues para determinar si dos personas son la misma o no lo más importante es que el nombre sea muy parecido, si no, no son la misma.



(a) Curva ROC y matriz de confusión para el corte óptimo para el conjunto de entrenamiento del dataset sintético generado para el modelo de regresión lineal. El corte óptimo es calculado como la mayor diferencia entre el ratio de verdaderos positivos y el ratio de falsos positivos.



(b) Curva ROC y matriz de confusión para el corte óptimo para el conjunto de validación del dataset sintético generado para el modelo de regresión lineal. El corte óptimo es calculado como la mayor diferencia entre el ratio de verdaderos positivos y el ratio de falsos positivos.

Figura 12: Curvas ROC y matrices de confusión para los subconjuntos de entrenamiento y validación para el modelo de regresión lineal.

En la Figura 12 mostramos las curva ROC y matrices de confusión para el conjunto de entrenamiento, Figura 12a, y el conjunto de validación, Figura 12b. El área bajo la curva es de 0.882 en ambos casos y podemos ver que los resultados entre entrenamiento y validación son muy parecidos, lo que nos indica que no hay sobreajuste.

Conjunto	Corte óptimo	AUC	Precision	Recall	F_1score
Entrenamiento	0.31	0.882	0.695	0.982	0.814
Validación	0.31	0.882	0.694	0.983	0.814

Tabla 8: Métricas sobre la matriz de confusión para el conjunto de entrenamiento y validación del modelo de regresión lineal.

En la Tabla 8 se muestran el corte óptimo, área bajo la curva, precision, recall y F_1 score para el conjunto de entrenamiento y validación. Podemos ver como todas las métricas son idénticas entre conjuntos, lo que indica que no hay sobreajuste. Comparándolo con el modelo anterior, aunque el F_1 score es parecido al corte óptimo para el otro modelo, vemos cómo la precision baja mientras que el recall sube.

3.2.4. Entrenamiento sobre el modelo de C con un modelo de regresión logística

Otra manera de enfocar este problema es como un problema de clasificación, queremos determinar si dos personas son la misma o no, lo que realmente es una variable de salida binaria. Se decidió hacer una regresión lineal debido a que el código en C está preparado para hacer una media ponderada, que se puede ver como una regresión lineal. En este apartado le daremos el enfoque propio de un problema de clasificación binaria. De esta forma, se ajusta un modelo de regresión logística el cual se formula como

$$\log\left(\frac{P(Y=1|\vec{X})}{P(Y=0|\vec{X})}\right) = \beta_0 + \beta_{\text{nombre}} d_{\text{nombre}} + \beta_{\text{lugar}} d_{\text{lugar}} + \beta_{\text{fil}} d_{\text{fil}} + \beta_{\text{flag fil}} I_{\text{flag fil}} \tag{4}$$

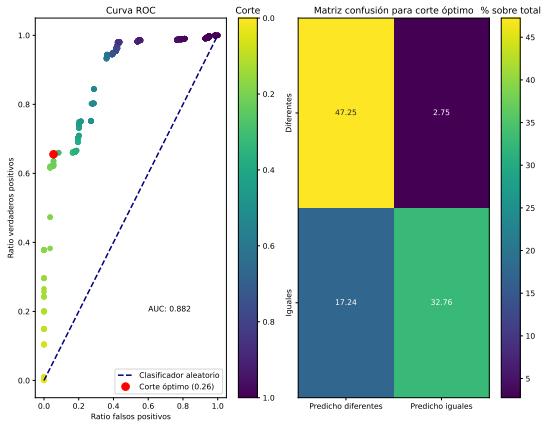
donde $\vec{X} = (d_{\text{nombre}}, d_{\text{lugar}}, d_{\text{fil}}, I_{\text{flag fil}})$ son las mismas variables que las empleadas en la regresión lineal e Y es la variable que determina si dos personas son diferentes ($Y = 1$) o son la misma ($Y = 0$). Hacer hincapié en que $Y = 1$ significa que son distintas, pues, por analogía con el modelo anterior, $Y = 1$ sería como tener distancia $d = 1$, lo que significa que las personas son distintas.

Variable	Coeficiente	Desv. Est.	p-valor
β_0	-4.5542	0.090	0
β_{nombre}	11.29	0.047	0
β_{lugar}	3.25	0.018	0
β_{fil}	4.37	0.161	3×10^{-162}
$\beta_{\text{flag fil}}$	1.42	0.089	5×10^{-57}

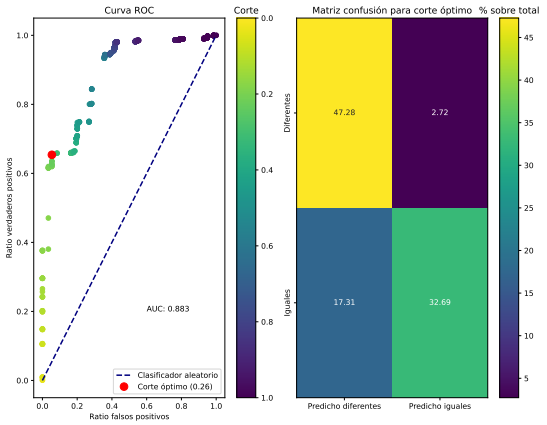
Tabla 9: Valores obtenidos para los coeficientes β_i con $i \in \{0$, nombre, lugar, fil, flag fil$\}$ junto con su desviación estándar y su p-valor para la regresión logística.

En la Tabla 9 se muestran los coeficientes obtenidos para el modelo de regresión logística junto con sus desviaciones estándar y su p-valor. De nuevo, todos los coeficientes son estadísticamente significativos y diferentes a 0. Además, vemos cómo todos los coeficientes son positivos, lo que tiene sentido, pues un aumento en la distancia, o tener desinformada la filiación, implica que la probabilidad de ser diferentes es mayor. También se ve cómo, de nuevo, el coeficiente asociado al nombre es mucho mayor que los demás. La explicación es la misma que en el caso anterior, la

distancia entre nombres juega un papel fundamental en determinar si dos personas son la misma o no.



(a) Curva ROC y matriz de confusión para el corte óptimo para el conjunto de entrenamiento del dataset sintético generado para el modelo de regresión logística. El corte óptimo es calculado como la mayor diferencia entre el ratio de verdaderos positivos y el ratio de falsos positivos.



(b) Curva ROC y matriz de confusión para el corte óptimo para el conjunto de validación del dataset sintético generado para el modelo de regresión logística. El corte óptimo es calculado como la mayor diferencia entre el ratio de verdaderos positivos y el ratio de falsos positivos.

Figura 13: Curvas ROC y matrices de confusión para los subconjuntos de entrenamiento y validación para el modelo de regresión logísitca.

En la Figura 13 mostramos las curva ROC y matrices de confusión para el conjunto de entrenamiento, Figura 13a, y para el conjunto de validación, Figura 13b. El área bajo la curva es de 0.882 y 0.883 respectivamente y podemos ver que los resultados entre entrenamiento y validación son muy parecidos, lo que nos indica que no hay sobreajuste igual que en el caso anterior.

Conjunto	Corte óptimo	AUC	Precision	Recall	F_1score
Entrenamiento	0.26	0.882	0.706	0.947	0.809
Validación	0.26	0.883	0.706	0.949	0.810

Tabla 10: Métricas sobre la matriz de confusión para el conjunto de entrenamiento y validación del modelo de regresión logística.

Observamos en la Tabla 10 que las métricas obtenidas son muy parecidas a las del modelo de regresión lineal y el modelo de C original, por lo que tampoco se ve una mejora clara sobre el modelo base de C.

3.2.5. Ajuste fino sobre red neuronal utilizando una Contrastive Loss

En los siguientes apartados vamos a cambiar a modelos mucho más grandes de redes neuronales. En todos utilizaremos un modelo de Sentence Transformer[4], en particular uno llamado "distiluse-base-multilingual-cased-v1"[8]. La arquitectura y funcionamiento del mismo se detalla en 2.4.3 Unificación de autores con redes neuronales.

[8]Link al modelo: https://huggingface.co/sentence-transformers/distiluse-base-multilingual-cased-v1

Utilizaremos el mismo dataset que los utilizados en los modelos de regresión lineal y logística. En el apartado 2.4 Unificación de autores explicamos la información disponible para cada persona. El modelo desarrollado en C utiliza identificadores en lugar de texto para los lugares y las filiaciones, sin embargo, para estos modelos utilizaremos el texto correspondiente. A partir del nombre regularizado se reconstruye el nombre en forma de texto de forma que siempre sea el nombre seguido de los apellidos, esto es, aunque la persona firmara originalmente un artículo como "Ruiz, E.", el nombre pasará a ser "e ruiz". La estrategia en este caso es condensar al máximo toda la información en un vector semántico de 512 componentes. Para ello, dada una persona con diferentes nombres, lugares y filiaciones, se toma el nombre más frecuente y se construye una frase para esa persona con

"nombre [placed in lugares] [working in filiaciones]."

nombre es el nombre más frecuente para esa persona. Si la persona tienes lugares asociados, se concatena el nombre con la cadena "placed in lugares" donde lugares es una cadena que se forma concatenando ciudad, provincia, comunidad autónoma y país. Si la persona tiene filiaciones asociadas, se concatena la cadena "working in filiaciones" donde filiaciones es la cadena formada concatenando la filiación de nivel 0, nivel 1 y nivel 2. En caso que dicha persona tenga varios lugares o filiaciones, se concatenan las mismas separándolas mediante comas.

Para verlo con ejemplos y que se entienda mejor, sea una persona llamada "Elisabet Llaurado", con ciudad Reus y que trabaja en la Facultad de Medicina y Ciencias de la Salud de la Universidad de Rovira i Virgili, su frase queda

"elisabet llaurado placed in reus tarragona cataluna espana working in univ rovira virgili fac medicina ciencias salud"

Para una persona llamada "Roser García Armengol", con ciudad Badalona y Barcelona y sin filiación, la frase queda

"roser garcia armengol placed in badalona barcelona cataluna espana, barcelona barcelona cataluna espana"

En el primer modelo utilizaremos una Contrastive Loss[5] o pérdida de contraste, esta pérdida es ampliamente utilizada en el entrenamiento de modelos como Sentence Transformers, diseñados para generar embeddings significativos para oraciones. El objetivo principal de la Contrastive Loss en este contexto es asegurar que las personas iguales (parejas positivas) tengan vectores semánticos cercanos, mientras que las personas diferentes (parejas negativas) tengan vectores semánticos más distantes.

Para entrenar con esta pérdida, se utilizan pares positivos y negativos de personas. La frase asociada a cada persona del par pasa por la red neuronal de Sentence Transformer para generar su vector semántico, en nuestro caso de 512 componentes. Sean $\vec{u}$ y $\vec{v}$ los vectores semánticos de dos personas, entonces la función de pérdida para la Contrastive Loss, L_C, se calcula mediante

$$L_C = (1-y)\ \frac{1}{2}\big(D(\vec{u},\vec{v})\big)^2 + y\frac{1}{2}(\text{máx}(0, m - D(\vec{u},\vec{v})))^2 \tag{5}$$

donde la $D(\vec{u},\vec{v})$ es la distancia entre ambos vectores calculada como uno menos la similaridad coseno

$$D(\vec{u},\vec{v}) = 1 - \cos(\vec{u},\vec{v}), \tag{6}$$

y es una variable binaria que toma valor $y = 1$ si la pareja es negativa e $y = 0$ si la pareja es positiva, y m es un hiperparámetro llamado margen, el cual tomaremos como $m = 0.5$, que modula cuánto de separados tienen que estar los casos negativos. Viendo la fórmula de pérdida, vemos que para las parejas positivas, $y = 0$, el segundo término es nulo y la función de pérdida aumenta conforme aumenta la distancia entre pares positivos. Por contra, para parejas negativas $y = 1$ y el primer término es nulo. En este caso la función de pérdida aumenta siempre que la distancia entre la pareja esté por debajo del margen m, en otro caso es nula.

De esta forma, con esta pérdida forzamos a que personas similares tengan vectores semánticos similares, mientras que personas diferentes tengan vectores semánticos a distancia m como mínimo en el mejor de los casos.

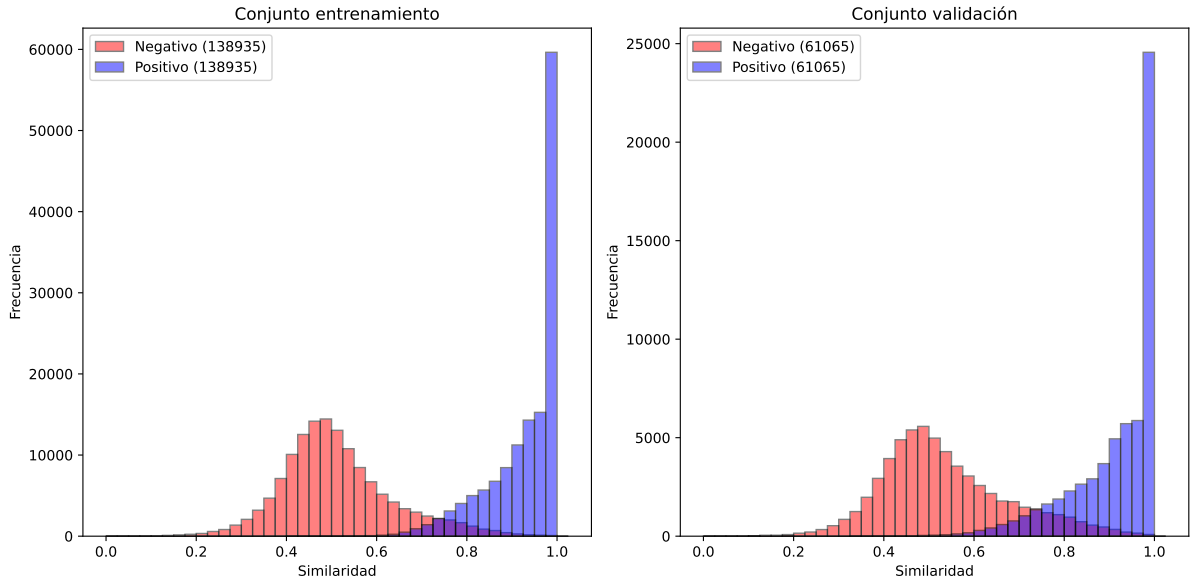
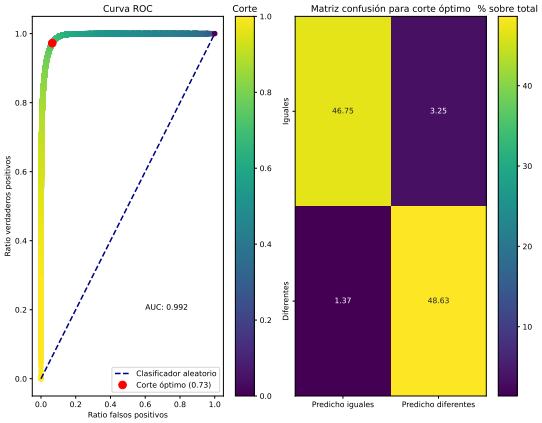


Figura 14: Distribución de similaridades sobre el dataset sintético utilizando el modelo de Sentence Transformers utilizando una Contrastive Loss. A la izquierda para el conjunto de entrenamiento y a la derecha para el conjunto de validación.

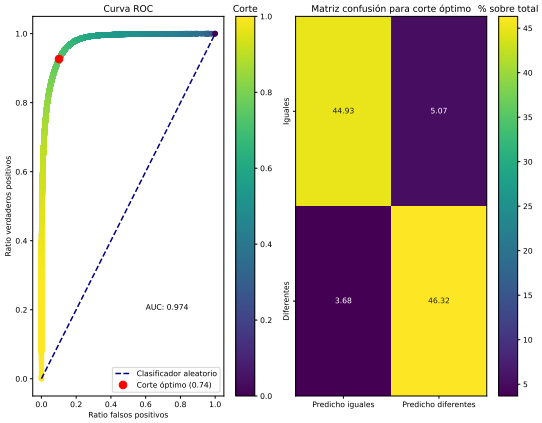
Para el entrenamiento se toma el margen de 0.5, se entrenan dos épocas y se utiliza un batch de 16. En la Figura 14 se muestra la distribución de distancias para el conjunto de entrenamiento y el de validación. Podemos ver, en contraste con la Figura 10 donde presentamos la distribución de distancias con el modelo de C inicial, como las distribuciones están más diferenciadas y agrupadas sobre un valor central.

Se ve en la Figura 15 el resultado del entrenamiento y cómo quedan las curvas ROC y matrices de confusión para los cortes óptimos para el subconjunto de entrenamiento, Figura 15a, y para el subconjunto de validación, Figura 15b.

Las métricas sobre estos modelos mejoran respecto a los basados en C. En la Tabla 11 recogemos las métricas para el conjunto de entrenamiento y validación. Se observa cómo todas las métricas tienen un pequeño aumento, indicando un mejor resultado del entrenamiento. El resultado entre entrenamiento y validación no se ve tan parecido como ocurría en los modelos basados en C, lo que podría indicar un cierto sobreajuste.



(a) Curva ROC y matriz de confusión para el corte óptimo para el conjunto de entrenamiento del dataset sintético generado para el modelo de Sentence Transformers utilizando una Contrastive Loss. El corte óptimo es calculado como la mayor diferencia entre el ratio de verdaderos positivos y el ratio de falsos positivos.



(b) Curva ROC y matriz de confusión para el corte óptimo para el conjunto de validación del dataset sintético generado para el modelo de Sentence Transformers utilizando una Contrastive Loss. El corte óptimo es calculado como la mayor diferencia entre el ratio de verdaderos positivos y el ratio de falsos positivos.

Figura 15: Curvas ROC y matrices de confusión para los subconjuntos de entrenamiento y validación para el modelo de Sentence Transformers utilizando una Contrastive Loss.

Conjunto	Corte óptimo	AUC	Precision	Recall	F_1score
Entrenamiento	0.73	0.992	0.973	0.933	0.953
Validación	0.73	0.974	0.934	0.887	0.910

Tabla 11: Métricas sobre la matriz de confusión para el conjunto de entrenamiento y validación de la red neuronal de Sentence Transformer entrenado con la Contrastive Loss.

3.2.6. Ajuste fino sobre red neuronal utilizando una Triplet Loss

En este segundo ajuste, utilizaremos el mismo modelo que el anterior, un Sentence Transformer, pero en este caso utilizaremos una función de pérdida diferente llamada Triplet Loss[6] o pérdida por ternas. La Triplet Loss es otra función de pérdida comúnmente utilizada en modelos como Sentence Transformers para aprender representaciones semánticas de oraciones. A diferencia de la Contrastive Loss, que trabaja con pares de ejemplos (positivos y negativos), la Triplet Loss trabaja con ternas de ejemplos, un ancla (persona de referencia), un caso positivo (persona similar al ancla) y un caso negativo (persona diferente al ancla).

El objetivo de la Triplet Loss es que la distancia entre el ancla y el caso positivo sea menor que la distancia entre el ancla y el caso negativo. En este contexto, se explica también por qué decidimos generar un dataset sintético de esta forma, ya que de esta forma somos capaces de agrupar ternas difíciles que el modelo aprende a diferenciar mejor que si le proporcionamos un caso positivo y uno negativo por separado como veremos al comparar resultados entre modelos. En la Figura 16 se muestra, de manera visual, cuál es el objetivo de esta función de pérdida. Conforme vamos entrenando el modelo, el caso positivo se acerca al ancla y el caso negativo se aleja, teniendo así el ancla y el caso positivo un vector semántico más parecido que el ancla y el caso negativo.

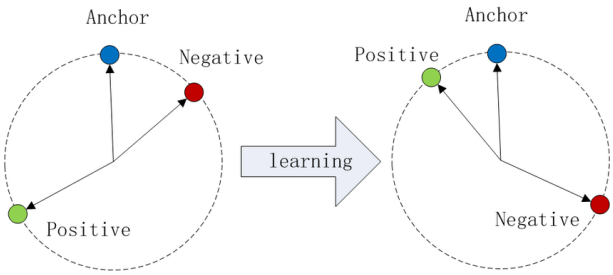


Figura 16: Esquema de funcionamiento de la Triplet Loss.

La fórmula para la función de pérdida Triplet Loss L_T es

$$L_T = \text{máx}\left[D\left(\vec{a}, \vec{p}\right) - D\left(\vec{a}, \vec{n}\right) + m, 0\right] \tag{7}$$

donde $\vec{a}$ es el vector semántico de la persona de referencia o ancla, $\vec{p}$ es el vector semántico de la persona para el caso positivo, $\vec{n}$ es el vector semántico de la persona para el caso negativo, $D(\vec{u}, \vec{v})$ es la distancia definida igual que en el modelo anterior como 1 menos la similaridad coseno y m, de nuevo, es un hiperparámetro llamado margen que modula la diferencia mínima entre la distancia entre los vectores semánticos del positivo y negativo respecto al ancla para que la pérdida sea cero, si la distancia es mayor que este margen entonces el función de pérdida aumenta.

Para este modelo presentamos un ajuste de este hiperparámetro para que se vea cómo afecta al entrenamiento. Para ello pondremos diferentes valores de este hiperparámetro y veremos su efecto sobre la distribución de similaridades y las métricas.

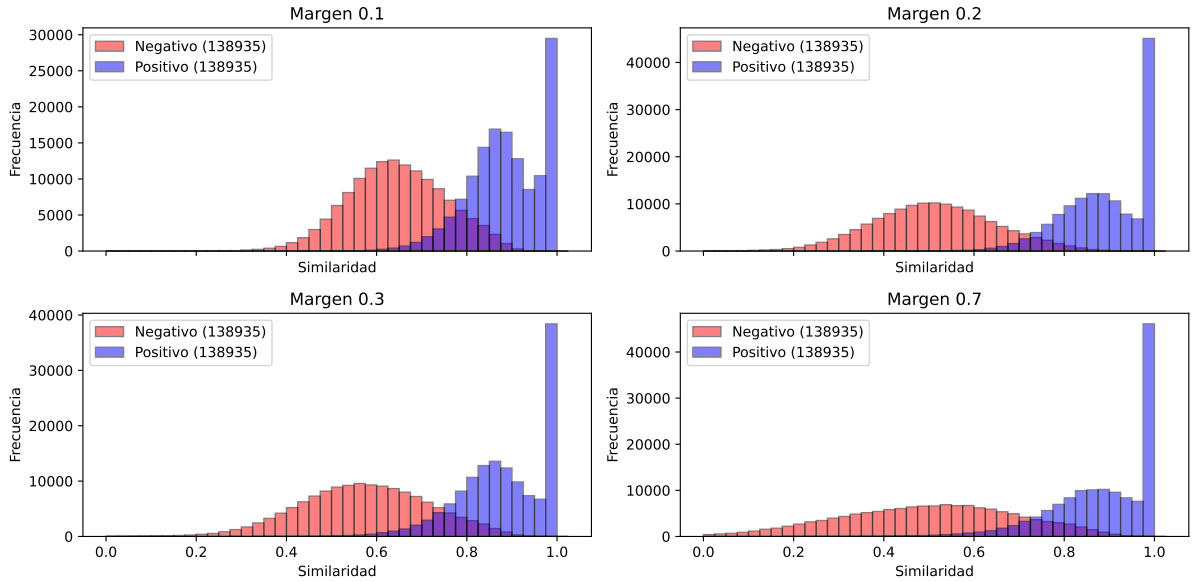


Figura 17: Comparativa de las distribuciones de la similaridad para el conjunto de entrenamiento usando diferentes valores del hiperparámetro de margen m de la función de pérdida Triplet Loss.

En la Figura 17 se muestran las distribuciones de similariades sobre el conjunto de entrenamiento entrenando el modelo fijando el valor del margen m a 0.1, 0.2, 0.3 y 0.7 respectivamente. De manera visual, se ve el efecto de este hiperparámetro, se observa cómo al aumentar el margen, el valor medio de las distribuciones para los casos positivos y negativos se separa, mientras que

la dispersión de los mismos aumenta. En la Tabla 12 podemos ver esto numéricamente, donde vemos cómo la desviación estándar aumenta al aumentar el valor del margen m, especialmente en la distribución de los casos negativos. A su vez, se ve cómo aumenta también la diferencia entre los valores medios de la distribución de casos positivos y casos negativos conforme aumenta m.

Margen m	0.1	0.2	0.3	0.7
Casos positivos	0.89 ± 0.08	0.89 ± 0.09	0.88 ± 0.10	0.89 ± 0.11
Casos negativos	0.64 ± 0.11	0.51 ± 0.14	0.57 ± 0.14	0.50 ± 0.20

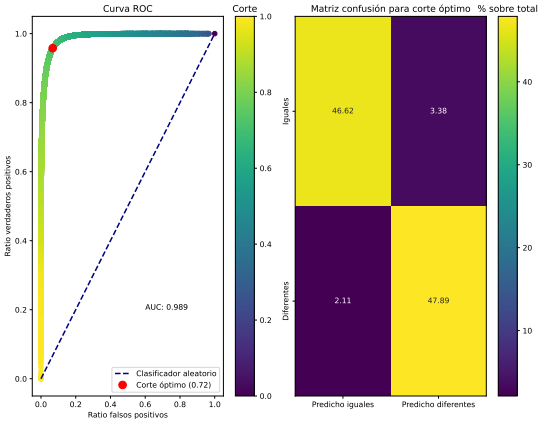
Tabla 12: Media y desviación estándar de las distribuciones de similaridades presentadas en la Figura 17 para el conjunto de entrenamiento con diferentes márgenes m.

En la Tabla 13 vemos las métricas de estos modelos con los diferentes márgenes tanto para el conjunto de entrenamiento como el de validación. Se puede ver cómo una mala elección de este hiperparámetro conduce a un peor resultado en las métricas del modelo. Si nos fijamos en el F_1 score, que es la media armónica de precision y recall, observamos como $m = 0.2$ sería la mejor elección para nuestro problema. Una elección demasiado grande o pequeña de este parámetro implica peores métricas. Esto se debe al solapamiento entre ambas distribuciones, para valores demasiado pequeños de m, aunque la desviación estándar disminuya, las medias de las distribuciones son más cercanas haciendo que haya un mayor solapamiento entre las distribuciones. Por contra, con un valor de m alto, las medias entre las distribuciones se separan más, pero como también aumenta la desviación estándar, resulta en que sigue aumentando el solapamiento entre las distribuciones. Es en un punto medio donde se compensan estos efectos que encontramos el valor óptimo de m.

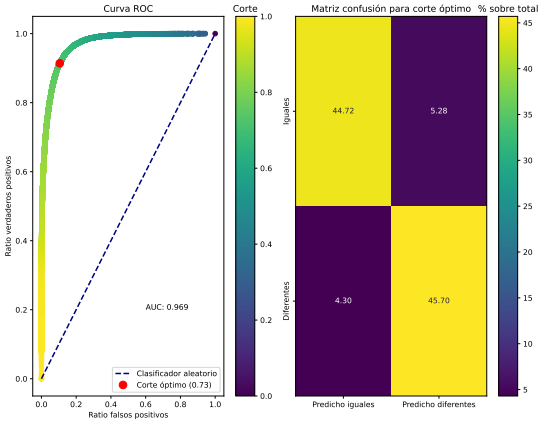
Margen m	Conjunto	Corte	AUC	Precision	Recall	F_1score
0.1	entrenamiento	0.73	0.961	0.954	0.776	0.856
	validación	0.73	0.952	0.946	0.767	0.847
0.2	entrenamiento	0.73	0.989	0.949	0.940	0.945
	validación	0.73	0.969	0.909	0.898	0.903
0.3	entrenamiento	0.73	0.963	0.925	0.861	0.892
	validación	0.73	0.951	0.908	0.847	0.877
0.7	entrenamiento	0.73	0.962	0.905	0.879	0.892
	validación	0.73	0.947	0.882	0.859	0.870

Tabla 13: Métricas obtenidas para el mismo dataset de entrenamiento pero con diferentes valores del hiperparámetro de margen m usando la función de pérdida Triplet Loss.

Finalmente, tomamos como margen $m = 0.2$ y analizamos los resultados sobre este modelo. En la Figura 18 mostramos las curvas ROC y matrices de confusión para el conjunto de entrenamiento y validación respectivamente. Tanto en las gráficas como en la Tabla 14 vemos un comportamiento parecido al modelo anterior con la Contrastive Loss, por lo que a priori no hay ninguna mejora sobre el anterior.



(a) Curva ROC y matriz de confusión para el corte óptimo para el conjunto de entrenamiento del dataset sintético generado para el modelo de Sentence Transformers utilizando una Triplet Loss. El corte óptimo es calculado como la mayor diferencia entre el ratio de verdaderos positivos y el ratio de falsos positivos.



(b) Curva ROC y matriz de confusión para el corte óptimo para el conjunto de validación del dataset sintético generado para el modelo de Sentence Transformers utilizando una Triplet Loss. El corte óptimo es calculado como la mayor diferencia entre el ratio de verdaderos positivos y el ratio de falsos positivos.

Figura 18: Curvas ROC y matrices de confusión para los subconjuntos de entrenamiento y validación para el modelo de Sentence Transformers utilizando una Triplet Loss.

Conjunto	Corte óptimo	AUC	Precision	Recall	F_1score
Entrenamiento	0.73	0.989	0.949	0.940	0.945
Validación	0.73	0.969	0.909	0.898	0.903

Tabla 14: Métricas sobre la matriz de confusión para el conjunto de entrenamiento y validación de la red neuronal de Sentence Transformer entrenado con la Triplet Loss.

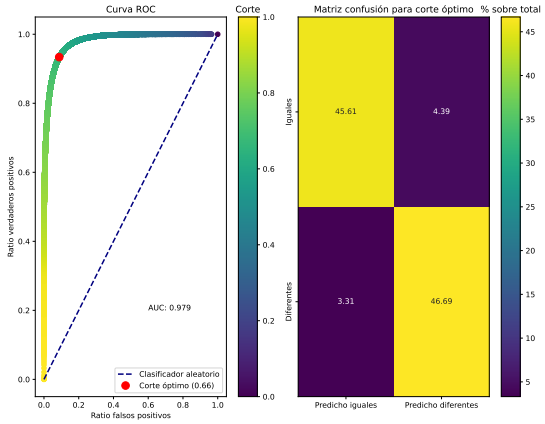
3.2.7. Ajuste fino sobre red neuronal utilizando una Multiple Negatives Ranking Loss

Como última prueba de entrenamiento, probamos otra función de pérdida llamada Multiple Negatives Ranking Loss[7]. Esta función de pérdida es una variante de la pérdida de ranking, diseñada para situaciones en las que se cuenta con un par de oraciones (una ancla y una positiva) y se desea maximizar la similitud entre el par positivo mientras se minimiza la similitud con todas las demás oraciones en el batch, tratándolas como negativas. La finalidad del entrenamiento es que el modelo que genera los vectores semánticos de personas genere representaciones que agrupen personas similares cerca unas de otras y separe personas diferentes en el espacio de embeddings. Esta función de pérdida facilita este aprendizaje, aprovechando el hecho de que en un batch de entrenamiento, los ejemplos que no son el par positivo pueden actuar como negativos.

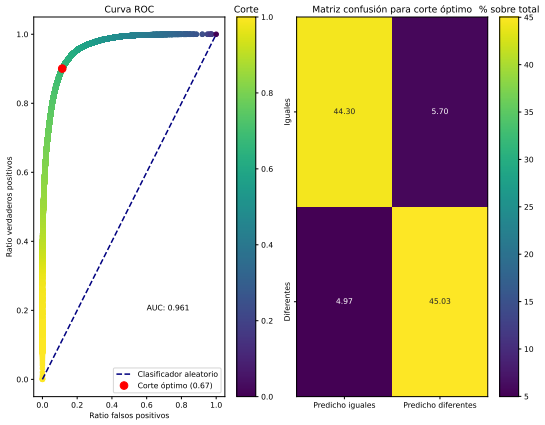
La fórmula para la función de pérdida Multiple Negatives Ranking Loss L_{MNR} es

$$L_{MNR} = -\log\left(\frac{\exp\left(S(\vec{a},\vec{p})\right)}{\sum_{j=1}^{N-1}\exp\left(S(\vec{a},\vec{v}_j)\right)}\right) \tag{8}$$

donde $\vec{a}$ es el vector semántico de la persona ancla, $\vec{p}$ es el vector semántico de la persona positiva, $\vec{v}_j$ es el vector semántico de las demás personas en el batch, consideradas negativas, con $j = 1, ..., N-1$, N es el tamaño del batch y $S(\vec{u},\vec{v})$ es la similaridad coseno de los vectores $\vec{u}$ y $\vec{v}$.



(a) Curva ROC y matriz de confusión para el corte óptimo para el conjunto de entrenamiento del dataset sintético generado para el modelo de Sentence Transformers utilizando una Multiple Negatives Ranking Loss. El corte óptimo es calculado como la mayor diferencia entre el ratio de verdaderos positivos y el ratio de falsos positivos.



(b) Curva ROC y matriz de confusión para el corte óptimo para el conjunto de validación del dataset sintético generado para el modelo de Sentence Transformers utilizando una Multiple Negatives Ranking Loss. El corte óptimo es calculado como la mayor diferencia entre el ratio de verdaderos positivos y el ratio de falsos positivos.

Figura 19: Curvas ROC y matrices de confusión para los subconjuntos de entrenamiento y validación para el modelo de Sentence Transformers utilizando una Multiple Negatives Ranking Loss.

En la Figura 19 vemos el resultado del entrenamiento con esta función de pérdida y en la Tabla 15 las métricas asociadas al entrenamiento. A diferencia de los dos modelos anteriores, en este el corte queda en 0.66, aunque las métricas quedan parecidas a las de los demás modelos.

Conjunto	Corte óptimo	AUC	Precision	Recall	F_1score
Entrenamiento	0.66	0.979	0.936	0.907	0.921
Validación	0.67	0.961	0.902	0.882	0.892

Tabla 15: Métricas sobre la matriz de confusión para el conjunto de entrenamiento y validación de la red neuronal de Sentence Transformer entrenado con la Multiple Negatives Ranking Loss.

3.2.8. Comparación entre modelos con un conjunto de personas real

En este apartado abordaremos una comparativa más exhaustiva sobre la diferencia en la eficiencia de estos modelos. En primer lugar, reunimos las métricas mostradas en los apartados anteriores en la Tabla 16 a modo de resumen. Recuérdese que en los tres primeros modelos probados, basados en C, se calculan distancias mientras que en los tres modelos posteriores hechos con Sentence Transformer se calculan similaridades. Se nota un salto en las métricas entre los modelos basados en C y distancias y los modelos de redes neuronales basados en Sentence Transformer y similitudes, siendo las redes neuronales ligeramente mejor sobre los modelos basados en C.

Modelo	Conjunto	Corte óptimo	AUC	Precision	Recall	F_1score
Modelo de C	Todo	0.16	0.882	0.733	0.941	0.824
Regresión Lineal	Entrenamiento	0.31	0.882	0.695	0.982	0.814
	Validación	0.31	0.882	0.694	0.983	0.814
Regresión Logística	Entrenamiento	0.26	0.882	0.706	0.947	0.809
	Validación	0.26	0.883	0.706	0.949	0.810
Contrastive Loss	Entrenamiento	0.73	0.992	0.973	0.933	0.953
	Validación	0.74	0.974	0.934	0.887	0.910
Triplet Loss	Entrenamiento	0.72	0.989	0.949	0.940	0.945
	Validación	0.73	0.969	0.909	0.898	0.903
MNR Loss	Entrenamiento	0.66	0.979	0.936	0.907	0.921
	Validación	0.67	0.961	0.902	0.882	0.892

Tabla 16: Métricas de los diferentes modelos probados sobre el dataset sintético generado. Los tres primeros modelos son los que basan en el modelo de C y los tres últimos los modelos de Sentence Transformers con las diferentes funciones de pérdida empleadas.

Sin embargo, someteremos a estos modelos a otro test para compararlos. Sobre los modelos de regresión logística y lineal no se profundizó más, pues no presentaban una mejora sobre el modelo de base de C. Para los modelos de redes neuronales, sabiendo que el modelo de C está probado sobre casos reales y es un punto de referencia, se decidió probarlos sobre un conjunto de personas reales. Para ello, se tomaron todos los ORCID de las 183989 personas españolas presentes en nuestros datos. El ORCID, del inglés *Open Researcher and Contributor ID*, es un código alfanumérico que identifica a un investigador. Es un identificador no único para un investigador, pues el mismo investigador puede tener varios ORCID, aunque sí es cierto que dos investigadores no pueden compartir ORCID. Se filtraron los ORCID que se repitiesen en al menos dos personas distintas y se filtraron las personas que tuviesen alguno de esos ORCID. De esta forma, de las 183989 personas que tenemos en los datos descargados, quedan 82404 personas que tienen al menos un ORCID que se repita al menos dos veces entre todas las personas. Agrupando estas 82404 personas por ORCID, resultaban en 18837 personas unificadas únicas. Esto significa que, en media, cada persona unificada consta de 4.4 personas que se han unificado.

Por consiguiente, se aplicó el algoritmo desarrollado para la unificación de personas, tanto con el modelo en C como con los tres modelos de redes neuronales de Sentence Transformers. En la Tabla 17 se muestran los resultados obtenidos por cada modelo. Se observa que el modelo que se acerca más al número de personas final es el modelo entrenado con la función de pérdida Multiple

Modelo	Nº personas	ARI	NMI	Homogeneidad	FMI
C	22513	0.949	0.992	0.999	0.950
CL	10820	0.129	0.899	0.833	0.247
TL	16698	0.733	0.973	0.959	0.749
MNRL	18826	0.926	0.990	0.988	0.927

Tabla 17: Métricas obtenidas al comparar los resultados de unificación de personas por los diferentes modelos con las unificaciones esperadas por ORCID. El primer modelo es el modelo de C, sin regresión lineal ni logística, los tres siguientes hacen referencia a las redes neuronales de Sentence Transformer entrenadas con las funciones de pérdida de Contrastive Loss (CL), Triplet Loss (TL) y Multiple Negatives Ranking Loss (MNRL) respectivamente.

Negatives Ranking Loss, mientras que los otros modelos tienden a sobreunificar, sobretodo el modelo entrenado con la función de pérdida Contrastive Loss. Los modelos con mejores métricas son el modelo de C y el modelo entrenado con la función de pérdida Multiple Ranking Loss. Se observa cómo el modelo de C unifica menos, quedando más personas al final del proceso. Esto es motivo del corte puesto a 0.16. Como se ha explicado ya, este modelo es sensible a la etapa de unificación que nos encontremos, teniendo que ser conservadores en la primera pasada a costa de unificar lo "evidente" y así ganar información de cada persona conforme se le atribuyen diferentes artículos con diferente información. Esto también se ve en la homogeneidad, donde vemos que los clústeres formados con este modelo realmente son de personas que tienen que estar en el mismo clúster. Por otro lado, el modelo con la función de pérdida MNRL se acerca mucho al número de personas esperado por el ORCID, manteniendo unas métricas en los demás indicadores muy parecidas al modelo de C.

Además, otro aspecto importante es la velocidad del proceso de unificación. Unificar estas 82404 personas de entrada con el modelo de C ha tardado 292 minutos, casi 5 horas con una CPU Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz. Esto es debido a que este proceso de unificación es un proceso cuadrático donde conforme se van analizando personas en el algoritmo y se va poblando la lista de salida de personas unificadas, cada vez hay que hacer más comparaciones. Para tener una idea, las 10000 primeras personas el algoritmo tarda 8 minutos, para las 20000 personas tarda 30 minutos, más del doble debido a la componente cuadrática. En este sentido, el uso de redes neuronales junto con GPUs ayuda y mucho a optimizar en tiempo este algoritmo. Unificar estas 82404 personas con las redes neuronales es cuestión de 5-10 minutos en una GPU NVIDIA GeForce RTX 3060. El algoritmo es el mismo, pero aquí la ventaja es que toda la información de cada persona se almacena en un vector semántico de 512 componentes, los cuales se precalculan antes de empezar el algoritmo, de forma que hacer una comparación entre dos personas resulta en hacer únicamente una similaridad coseno de dos vectores de 512 componentes, mientras que en el modelo en C, para sacar la distancia de nombre, lugar y filiación, se recorren todas las combinaciones de nombre, lugares y filiaciones respectivamente, y en cada comparación, en el caso de nombres hay que calcular distancias de Levenshtein, lo que es costoso computacionalmente.

Como conclusión, debido a que el modelo de C ha probado su eficiencia en otros proyectos, no se tiene por qué descartar, ya que es un modelo más interpretable que ayuda a entender el peso que tiene cada variable en la unificación de dos personas. Se pueden unir las virtudes de ambos modelos, sacando candidatos a unificar con el modelo de red neuronal, para que así, la

gran mayoría de comparaciones necesarias en el algoritmo las haga la red neuronal, y calculando una distancia final con el modelo de C para terminar de determinar si hay que unificar la pareja candidata o no.

3.3. Identificación de filiaciones

3.3.1. Creación de un dataset manual

Para poder comprobar las mejoras sobre los algoritmos, se tomaron 55 participaciones y se etiquetaron a mano. Recordamos que el concepto de participación es una cadena completa del tipo "Univ Girona, Fac Sci, Dept Biol" la cual la dividimos por comas generando tres filiaciones, en este caso, "Univ Girona", "Fac Sci" y "Dept Biol". De esta forma, de estas 55 participaciones resultan 176 filiaciones. Cada una de estas filiaciones no se tiene por qué corresponder con una filiación del maestro, pues puede ser algo que directamente no está en el mismo, como por ejemplo una cadena tipo "unidad dermatologia", o que tenga varias filiaciones en la cadena, como por ejemplo "univ complutense madrid csic", donde tenemos tanto la Universidad Complutense de Madrid como el CSIC (Consejo Superior de Investigaciones Científicas). Los primeros casos, en los que la filiación no se encuentre en el maestro, se etiquetarán como "Nada" y esperamos que el algoritmo no asigne ninguna filiación del maestro a dicha filiación de entrada. En los segundos casos, en los que hay más de una filiación en la cadena de texto, se espera que el algoritmo identifique cada uno de ellos. Por este motivo, aunque en la entrada tendremos 176 filiaciones, tenemos etiquetadas 194 filiaciones de salida como buenas.

Tipo de filiación	Frecuencia en el conjunto de entrada
Universidad	49
Departamento	23
Instituto	20
Escuela	2
Facultad	18
Hospital	18
Centro	9
Otros	37

Tabla 18: Distribución por tipo de las filiaciones de entrada en el proceso

En la Tabla 18 se recogen los tipos extraídos de la regularización para estas 176 filiaciones.

3.3.2. Identificación de filiaciones con similaridad por distancia de edición

En el apartado 2.5.1 Identificación de filiaciones con similaridad por distancia de edición se explica el algoritmo utilizado para la identificación de filiaciones de manera más extensa. En este apartado vamos a explicar tres mejoras sobre el algoritmo y cómo han afectado a los resultados. además de un ajuste del mismo con el parámetro que tiene dicho algoritmo.

El algoritmo, para cada una de las participaciones, recorre las filiaciones que hay en la participación y las compara contra el maestro de filiaciones. En las primeras versiones del algoritmo, se hacía una búsqueda por cada filiación, buscando en primer lugar acrónimos y seguidamente comparando los nombres regularizados de la filiación de entrada y las del maestro. Para ir más

rápido y reducir el número de comparaciones, se hacen filtros al maestro, bien sea por lugares geográficos o por el tipo de filiación. El algoritmo tiene un único parámetro que es el corte c. Este corte es tal que si la distancia entre dos filiaciones es menor al mismo, se consideran la misma filiación. La distancia es calculada con una modificación de la distancia de Levenshtein "por palabras" más adaptada al contexto de este problema, pues las cadenas a comparar son largas.

En la primera versión del algoritmo no se estaba usando una información muy interesante que disponemos en el maestro, que son las relaciones entre las filiaciones. Por ello, se actualizó el algoritmo para hacer dos pasadas sobre todas las filiaciones de una participación. En la primera pasada se trata de identificar filiaciones de nivel 0 como universidades y hospitales. La estrategia es la misma, buscar por acrónimos y por distancia para esta primera pasada.

Lo que se añadió nuevo al algoritmo es hacer una segunda pasada usando la información de la primera. Si detectamos una universidad en la primera pasada nos es de gran ayuda, pues cuando estemos buscando por ejemplo una facultad, ya nos quitamos el problema de que hay varias facultades, escuelas o departamentos con el mismo nombre pero en diferentes universidades, y buscaremos primero entre dichas filiaciones "hijas". De esta forma, este segundo algoritmo usa la jerarquización de las filiaciones del maestro.

La tercera mejora vino a través de indagar en los datos de entrada. Se pudo observar que en ocasiones las filiaciones estaban escritas en inglés, lo que dificulta la comparación por distancia de Levenshtein, "dept ciencias salud" dista mucho de "health science dept". Para solventar estos casos, se añadieron al maestro de filiaciones las traducciones al inglés de los nombres de las filiaciones. Para ello, se utilizó una red neuronal entrenada para la traducción[9], cuyo el modelo de traducción se llama "michaelfeil/ct2fast-opus-mt-es-en", y el tokenizador es "Helsinki-NLP/opus-mt-es-en".

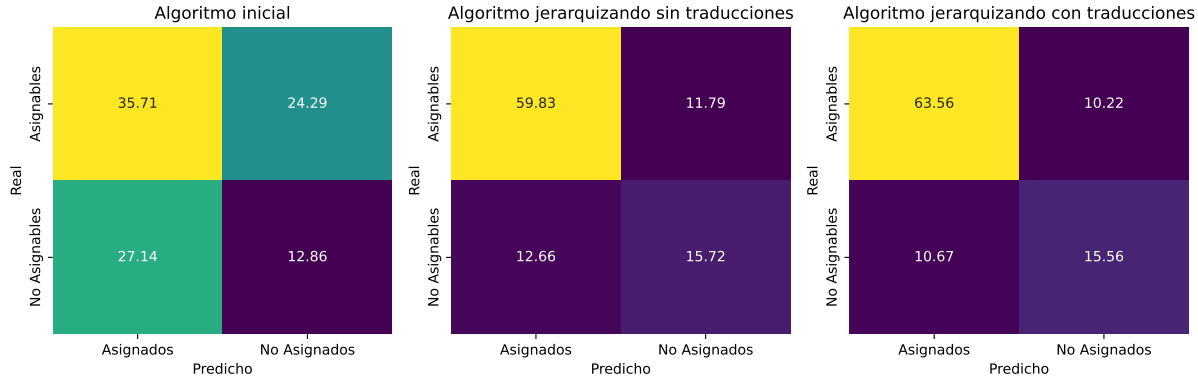


Figura 20: Matrices de confusión para las diferentes versiones del algoritmo. A la izquierda la primera versión del algoritmo, en el centro tras introducir la jerarquización y a la derecha tras añadir las traducciones a las filiaciones del maestro. Todos los algoritmos se han ejecutado con corte $c = 0.15$.

En la Figura 20 se muestran las matrices de confusión obtenidas para el algoritmo inicial, el algoritmo jerarquizando añadiendo esta segunda pasada y el algoritmo jerarquizando tras añadir las traducciones al maestro de filiaciones. Para sacar estas matrices de confusión se han tenido en cuenta los siguientes criterios. Sea A el conjunto de filiaciones etiquetado a mano que se deben

[9]Enlace al repositorio: https://pypi.org/project/hf-hub-ctranslate2/

asignar a una filiación de entrada y B el conjunto de filiaciones que ha asignado el algoritmo a dicha filiación de entrada, entonces los verdaderos positivos son aquellas filiaciones en ambos conjuntos y que no sean "Nada", es decir, $|A \cap B \setminus \{\text{"Nada"}\}|$. Los verdaderos negativos serán los casos donde "Nada" esté en ambos conjuntos $|A \cap B \cap \{\text{"Nada"}\}|$. Los falsos positivos serán elementos que estén en B pero no en A y no sean "Nada", esto es, $|B \setminus \{A \cup \{\text{"Nada"}\}\}|$. Los falsos negativos serán elementos que estén en A pero no en B y no sean "Nada" $|A \setminus \{B \cup \{\text{"Nada"}\}\}|$. Con estos criterios, se muestra en la Tabla 19 la precision, recall y F_1 score para las diferentes versiones del algoritmo. Se ve cómo las mejores métricas las obtiene el algoritmo nuevo con las traducciones.

Algoritmo	Precisión	Recall	F_1score
Inicial	0.568	0.595	0.581
Jerarquizando sin traducciones	0.825	0.835	0.830
Jerarquizando con traducciones	0.856	0.861	0.859

Tabla 19: Resultados de los diferentes algoritmos, todos con corte $c = 0.15$.

El algoritmo tiene un parámetro que es el corte c por debajo del cual dos filiaciones se consideran que son la misma o no. Para determinar el corte óptimo, se presentan las matrices de confusión para tres pruebas en la Figura 21 con cortes de $c \in \{0.1, 0.15, 0.2\}$.

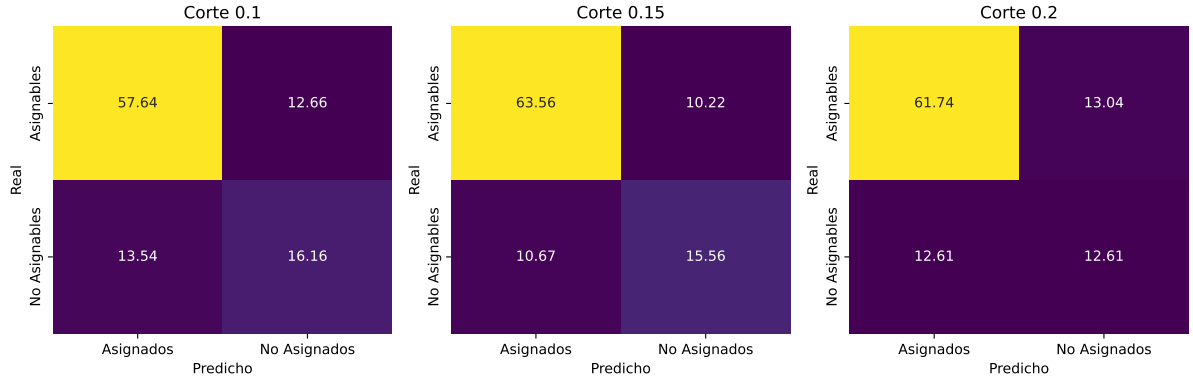


Figura 21: Matrices de confusión para diferentes cortes c con el algoritmo jerarquizando con traducciones. A la izquierda con corte $c = 0.1$, en el centro con corte 0.15 y a la derecha con corte 0.2.

El corte óptimo sucede con $c = 0.15$ como podemos ver tanto en las matrices de confusión como en la Tabla 20.

Corte c	Precisión	Recall	F_1score
0.1	0.810	0.820	0.815
0.15	0.856	0.861	0.859
0.2	0.830	0.826	0.828

Tabla 20: Resultados del algoritmo jerarquizando con traducciones con diferentes cortes c.

En la Figura 22 se muestra un estudio por tipo de los aciertos y fallos del algoritmo jerarquizando con traducciones y corte $c = 0.15$. Se observa cómo todas las universidades de entrada

han sido asignadas correctamente al maestro. Las universidades normalmente es lo más sencillo de detectar, pues su nombre no varía mucho entre español e inglés, tampoco tienen muchas formas de escritura y también se pueden detectar mediante acrónimos. La mejora del algoritmo ha afectado notablemente a departamentos, escuelas y facultades. El uso de la segunda pasada en el algoritmo permite utilizar la información de la universidad, descartando facultades, escuelas y departamentos que tienen el mismo nombre, e incluso lugar, pero son de diferente universidad. Además, con el uso de las traducciones, también se consiguen asignar casos donde la filiación de entrada estaba escrita en inglés.

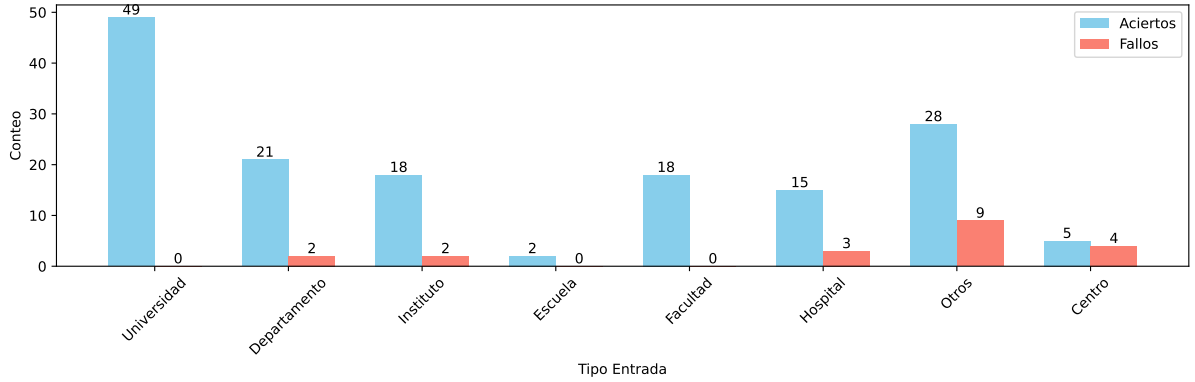


Figura 22: Aciertos y fallos por tipo de entrada usando el algoritmo nuevo con traducciones y corte $c = 0.15$.

Por último, se ha ejecutado este algoritmo sobre los datos descargados de WoS. Recordamos que disponemos de 114042 participaciones en las cuales encontramos 292432 filiaciones a asignar contra el maestro.

En la Tabla 5 se muestran la distribución de estas 292432 filiaciones analizadas en los diferentes tipos y en la Figura 23 se muestra el porcentaje de asignación para cada tipo. El porcentaje de asignación más alto se obtiene para universidades, mientras que el más bajo es para el tipo "Otros". Este bajo porcentaje en el tipo "Otros" es de esperar, pues en dicha categoría hay filiaciones que realmente no lo son, como direcciones, unidades, grupos de investigación y demás casuística. Para el resto de tipos vemos un porcentaje de asignación del 65-80 % excepto por las escuelas, que está en el 47 %.

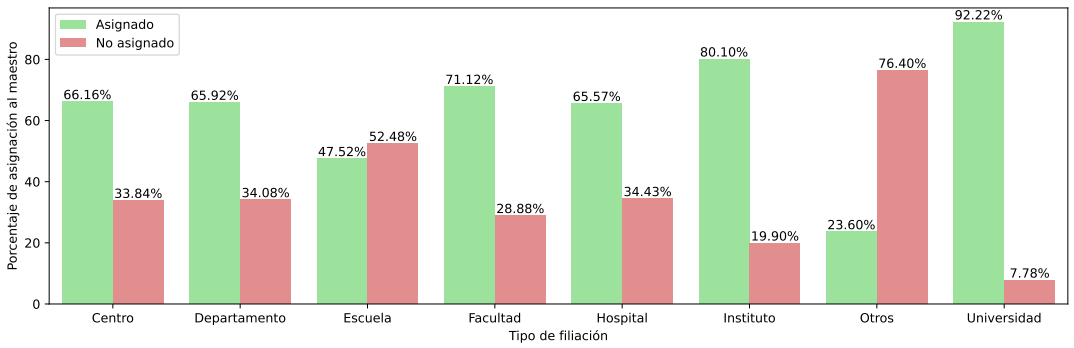


Figura 23: Porcentajes de asignación para los diferentes tipos de filiación analizados.

3.3.3. Identificación de filiaciones con redes neuronales. Creación del algoritmo

Los resultados obtenidos con el algoritmo anterior usando la distancia de edición han dado muy buenos resultados, pero aun así exploramos el uso de utilizar una red neuronal para este problema. Los modelos de redes neuronales probados son los mismos que los usados para la unificación de autores, que son los Sentence Transformers. En el caso de las filiaciones solo tendremos como variable el nombre de la propia filiación con diferentes formas de escritura que podamos tener de la misma. La información como el lugar o la clasificación de la filiación se usará con otros fines como se mostrará más adelante, pero la comparación entre dos filiaciones se hará única y exclusivamente comparando los nombres de las mismas.

Vamos a utilizar el mismo modelo preentrenado usado para personas, "distiluse-base-multilingual-cased-v1", elaborando un algoritmo para identificar las filiaciones del dataset manual contra el maestro de filiaciones. Con la ayuda del dataset etiquetado a mano, mediremos qué algoritmo es mejor manteniendo el mismo modelo.

Recordamos que una participación es un conjunto de filiaciones que aparecen en la misma cadena de texto. Como ejemplo para explicar los diferentes algoritmos desarrollados, tomaremos la participación "univ zaragoza, inst biocomp & fisica sistemas complejos, fac ciencias, dept fis teo", la cual tiene filiaciones "univ zaragoza", "inst biocomp & fisica sistemas complejos", "fac ciencias" y "dept fis teo".

El primer algoritmo desarrollado consiste en comparar toda la participación contra el maestro de filiaciones. Para ello, se toma cada filiación del maestro y se forman frases más largas concatenando los nombres de una filiación con nombres de filiaciones que dependiesen de esta. Por ejemplo, para la Universidad de Zaragoza se forman frases como "Universidad de Zaragoza", "Universidad de Zaragoza, Facultad de Ciencias", etcétera. De esta forma, si la frase del maestro más próxima a la participación de entrada es "Universidad de Zaragoza, Facultad de Ciencias, Departamento de Física Teórica", hemos identificado tres filiaciones del maestro para esa participación. Este es algoritmo llamado "Sin jerarquizar".

En el siguiente algoritmo, llamado "Sin jerarquizar con expansiones", consiste en incorporar expansiones de algunas palabras. Usando la información en la Tabla 2, la cual es utilizada en la regularización para abreviar algunas palabras para una mejor comparación por distancia de edición, expandimos las palabras acortadas a la palabra maestra, pasando todos los "univ" a "universidad" por ejemplo. El motivo de hacer esto es para ayudar a la comparación. Por cómo está preentrenado el modelo, las abreviaturas como "dept", "dpto." o "dept." no son comunes para el tokenizador, lo que hace que al compararlos con palabras como "departamento" la similaridad sea baja. En la Figura 24 se muestra un ejemplo de esto mismo. La similaridad entre "departamento" y "department" es alta, así como la similaridad entre las abreviaturas "dept", "dpto" y "dept". Sin embargo, la similaridad entre estas abreviaturas y "departamento" o "department" es la misma que con la palabra "hospital", que no tiene nada que ver con el resto.

En el tercer algoritmo, llamado "Jerárquico con participaciones", se incorpora el uso de los niveles asociados a las clasificaciones de las filiaciones. Dada una participación, el proceso es como sigue:

1. Búsqueda de nivel 0. Se toman las filiaciones de entrada de la participación y se comparan con las filiaciones de nivel 0 del maestro.

2. Búsqueda de nivel 1. Se toman las filiaciones de entrada de la participación y se comparan

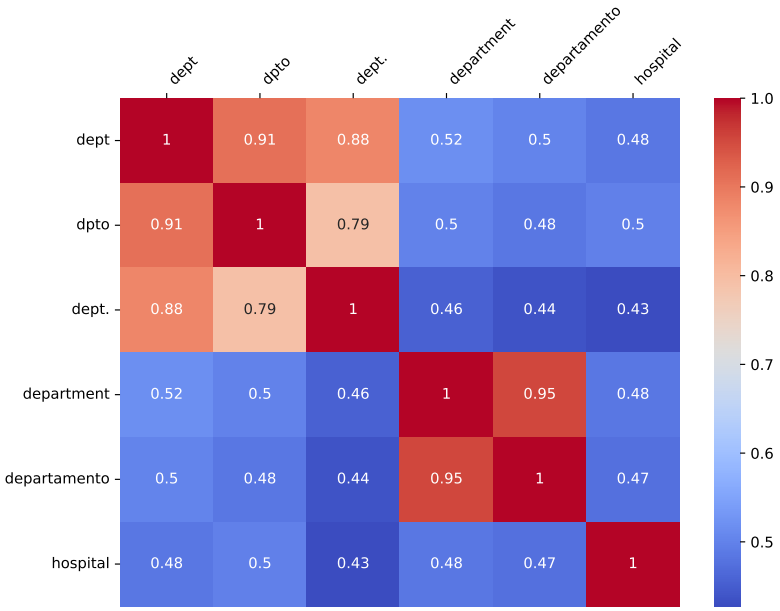


Figura 24: Similaridades entre diferentes palabras con el modelo preentrenado "distiluse-base-multilingual-cased-v1" de Sentence Transformers sin hacer ningún ajuste fino.

con las filiaciones de nivel 1 excepto escuelas y facultades del maestro.

3. Búsqueda por relativos. Se toma la participación de entrada. Para las filiaciones de nivel 0 y 1 encontradas en los anteriores puntos, construimos cadenas completas con sus filiaciones hijas imitando lo que se hace en el algoritmo "Sin jerarquizar con expansiones" para encontrar más filiaciones de nivel 1 y 2.

Para ponerlo en contexto, tomemos la participación de ejemplo nombrada al inicio. Idealmente, en el primer paso, al tomar la filiación de entrada "univ zaragoza", encontraríamos la Universidad de Zaragoza en el maestro. En el segundo paso, al tomar la filiación de entrada "inst biocomp & fisica sistemas complejos", encontraríamos el Instituto de Biocomputación y Física de Sistemas Complejos en el maestro. Para el tercer punto, construiríamos las frases "Universidad de Zaragoza, Facultad de ..., Departamento de ..." con las diferentes facultades y departamentos de la Universidad de Zaragoza en el maestro, y encontraríamos que la participación de entrada "univ zaragoza, inst biocomp & fisica sistemas complejos, fac ciencias, dept fis teo" coincide con "Universidad de Zaragoza, Facultad de Ciencias, Departamento de Física Teórica", por lo que habríamos encontrado la Facultad de Ciencias y el Departamento de Física Teórica de la Universidad de Zaragoza en el maestro.

El algoritmo anterior observamos que no funcionaba muy bien para los niveles 1 y 2, por lo que descartamos la idea del punto 3 de usar toda la participación, usando en su lugar las filiaciones, de ahí el nombre de "Jerárquico con filiaciones". La modificación sobre el algoritmo anterior está en el punto 3, donde ahora se compara cada filiación de la participación de entrada con cada una de las filiaciones hijas de las filiaciones de nivel 0 y 1 encontradas en los puntos anteriores. Así, en el punto 3, idealmente, al buscar "fac ciencias" en las filiaciones hijas de la Universidad de Zaragoza en el maestro, encontraremos la Facultad de Ciencias de la Universidad de Zaragoza, y lo mismo sucederá para el departamento.

La última mejora consiste en la búsqueda de acrónimos en la participación de entrada median-

te expresiones regulares antes del primer paso, identificando así algunas filiaciones del maestro que se usan en el último punto y eliminando dichos acrónimos del texto de la participación de entrada.

En la Tabla 21 se muestran los resultados sobre el conjunto etiquetado a mano para los diferentes algoritmos desarrollados. Se observa cómo cada actualización del algoritmo es una mejora del mismo, siendo los cambios más notorios el uso de la jerarquización, dividiendo el problema en búsqueda de filiaciones por niveles, y la búsqueda de acrónimos.

Algoritmo	Corte óptimo	TP	TN	FP	FN	Precision	Recall	F_1 score
Sin jerarquizar	0.6	30	4	42	71	0.417	0.297	0.347
Sin jerarquizar con expansiones	0.5	36	0	54	65	0.400	0.356	0.377
Jerárquico con participaciones	0.9	38	7	6	63	0.863	0.376	0.524
Jerárquico con filiaciones	0.8	54	6	42	47	0.563	0.535	0.549
Usando acrónimos	0.95	56	8	16	45	0.778	0.554	0.647

Tabla 21: Resultados de diferentes algoritmos en orden cronológico de pruebas sobre el dataset etiquetado a mano. El corte óptimo se ha tomado como el corte con el que el F_1 score es máximo.

3.3.4. Identificación de filiaciones con redes neuronales. Ajuste fino del modelo

Una vez implementadas las mejoras presentadas en el apartado anterior para el algoritmo de identificación de filiaciones, se optó por realizar un ajuste fino del modelo. Para ello, es necesario un conjunto con el que realizar el ajuste fino del modelo. A continuación, se muestran los resultados sobre dos conjuntos de entrenamiento diferentes.

Para generar el primer conjunto de datos, se usó exclusivamente la información disponible en el maestro de filiaciones. Para todas las filiaciones del maestro tenemos al menos dos formas de escritura de la misma, pues contamos con el nombre en español y la traducción en inglés, y en ocasiones tenemos alguna adicional. De igual modo que en el conjunto de entrenamiento sintético generado para autores, para filiaciones también se generaron ternas. Para generar cada una de ellas, se elige una filiación del maestro al azar y se toman, aleatoriamente, dos formas de escrituras del nombre de la filiación. La primera forma de escritura será el ancla y la segunda el caso positivo. Además, con una probabilidad del 20 % se desordenan las palabras del caso positivo. Por ejemplo, si el caso positivo es "univ zaragoza", se podría cambiar a "zaragoza univ". Para escoger un caso negativo, se escoge otra filiación del maestro al azar y se toma una de sus formas de escritura. Para la elección de esta otra filiación, se mantiene el tipo de filiación con una probabilidad del 50 % y el lugar de la filiación con un 50 % también. La idea de esto es que el caso negativo sea parecido al positivo. Por ejemplo, una terna podría estar formada por "univ barcelona" como ancla, "barcelona univ" como caso positivo y "univ autonoma barcelona" como negativo si se mantienen el tipo y el lugar de la filiación para la búsqueda del caso negativo.

Con esta metodología, se generaron alrededor de 100 mil ternas y se dividieron en 30 % para validación y 70 % para el ajuste fino del modelo. Todas las ternas provenientes de la misma

filiación ancla están en el mismo conjunto.

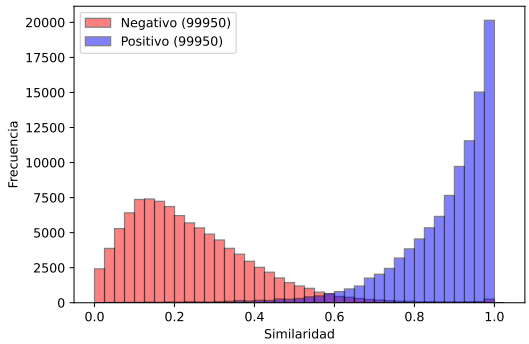


Figura 25: Distribución de similaridades sobre el primer conjunto de entrenamiento sin hacer un ajuste fino al modelo.

Corte óptimo	0.58
AUC	0.990
Precision	0.901
Recall	0.989
F_1 score	0.943

Tabla 22: Métricas sobre el primer conjunto de entrenamiento sin hacer ajuste fino al modelo.

En la Figura 25 se muestra la distribución de similaridades sobre este conjunto generado con el modelo sin realizar ningún ajuste fino. Se puede ver en la Tabla 22 como el modelo sin ajuste fino ya tiene muy buenas métricas.

Para realizar el ajuste fino, se toma la función de pérdida Multiple Negatives Ranking Loss, explicada en el apartado 3.2.7 Ajuste fino sobre red neuronal utilizando una Multiple Negatives Ranking Loss. Se entrena el modelo una época, y se utiliza un tamaño de batch de 16. Observamos cómo la forma de la distribución de similaridades cambia completamente tras el ajuste fino en la Figura 26. Además, las métricas sobre dichas distribuciones son todavía mejores que con el modelo sin entrenar.

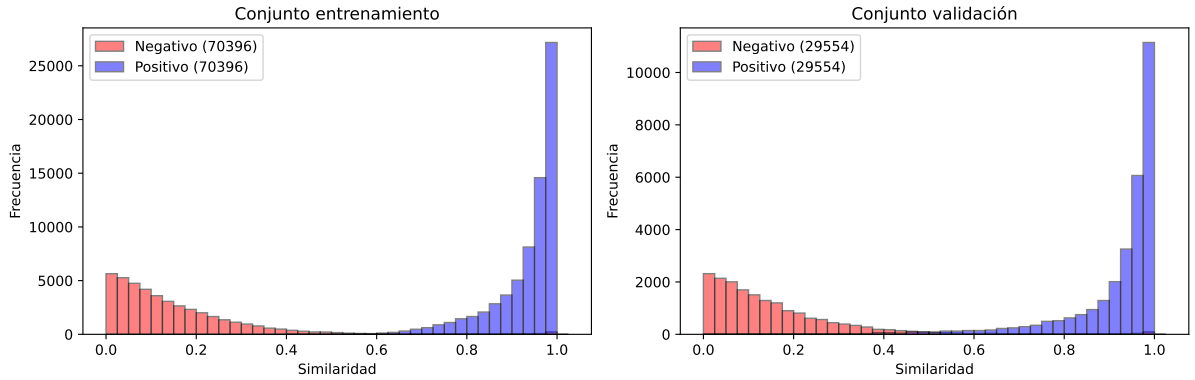


Figura 26: Distribuciones de similaridad en el primer conjunto generado para entrenamiento y validación tras el ajuste fino del modelo.

Conjunto	Corte óptimo	AUC	Precision	Recall	F_1score
Entrenamiento	0.59	0.996	0.972	0.993	0.983
Validación	0.48	0.994	0.931	0.993	0.961

Tabla 23: Métricas sobres los conjuntos de entrenamiento y validación tras el ajuste fino del modelo

Tras realizar este ajuste fino, ponemos a prueba el modelo con el dataset etiquetado a mano.

Para ello ejecutamos el algoritmo con diferentes cortes y vemos el rendimiento en cada uno de esos casos. En la Tabla 24 se muestran los resultados. Guiándonos por el F_1 score, vemos que el mejor corte es 0.8 con un valor del F_1 score de 0.708, lo cual es una mejora sobre el modelo sin el ajuste fino, que tenía un F_1 score de 0.647.

Corte	TP	TN	FP	FN	Precision	Recall	F_1 score
0.05	78	0	160	23	0.328	0.772	0.460
0.10	78	0	160	23	0.328	0.772	0.460
0.20	78	0	154	23	0.336	0.772	0.468
0.30	78	0	148	23	0.345	0.772	0.477
0.40	78	2	140	23	0.358	0.772	0.489
0.50	76	3	113	25	0.402	0.752	0.524
0.60	75	6	83	26	0.475	0.743	0.579
0.70	73	7	51	28	0.589	0.723	0.649
0.80	69	8	25	32	0.734	0.683	0.708
0.90	59	8	19	42	0.756	0.584	0.659
0.95	55	8	18	46	0.753	0.545	0.632

Tabla 24: Métricas sobre el conjunto etiquetado a mano con diferentes cortes para el modelo con el ajuste fino usando el primer conjunto de entrenamiento.

Analizando la Figura 26, observamos como el conjunto generado no tiene casos que den información, pues el modelo ya es capaz de diferenciarlos bien de base. Por ello, se generó otro conjunto para entrenar el modelo, pero esta vez buscando casos más difíciles. Los nombres que disponemos en el maestro están "correctamente" escritos, es decir, no contienen abreviaciones. Sin embargo, las filiaciones que descargadas de WoS que entran al algoritmo generalmente están abreviadas. En lugar de escribir "Departamento de Psicología" se suelen encontrar situaciones como "psico dept". Las abreviaturas, como se ha mostrado en la Figura 24, son un problema para el tokenizador, pues no sabe de primeras que "dept" y "departamento" son parecidos. Aunque se expanden algunas palabras que tenemos en la Tabla 2, la casuística es mucho más grande. Por ello, se ha decidido generar otro conjunto de entrenamiento, pero esta vez empleando el algoritmo presentado en 3.3.2 Identificación de filiaciones con similaridad por distancia de edición, de forma que encontremos, mediante este algoritmo, formas de escritura de las filiaciones donde habrá abreviaturas y por lo tanto una casuística más variada y compleja.

De esta forma, el procedimiento para generar este segundo conjunto de entrenamiento es parecido al anterior. Se toma una filiación del maestro al azar. El ancla es una forma de escritura de la filiación que proviene del maestro, el caso positivo es una forma de escritura que se ha encontrado gracias al algoritmo de identificación por distancia de edición. El caso negativo se escoge igual que antes, pero en el 90 % de los casos la forma de escritura de la filiación negativa es también una forma de escritura encontrada mediante el algoritmo de identificación de filiaciones.

De esta forma, se ejecutó el algoritmo de identificación de filiaciones mediante distancia de edición sobre los datos descargados del WoS y posteriormente se generó este conjunto nuevo. En este caso se generaron unas 58 mil ternas.

Se observa en la Figura 27 como la distribución de similaridades entre casos positivos y negativos tiene mayor solapamiento con el modelo sin entrenar que con el primer conjunto de datos generado. Esto también se ve en la Tabla 25 donde el F_1 score es de 0.866 para este

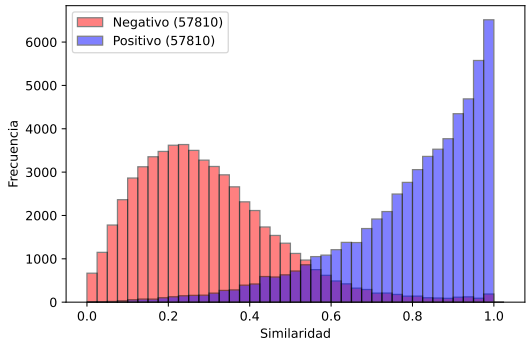


Figura 27: Distribución de similaridades sobre el segundo conjunto de entrenamiento sin hacer un ajuste fino al modelo.

Corte óptimo	0.55
AUC	0.960
Precision	0.779
Recall	0.976
F_1 score	0.866

Tabla 25: Métricas sobre el segundo conjunto de entrenamiento sin hacer ajuste fino al modelo.

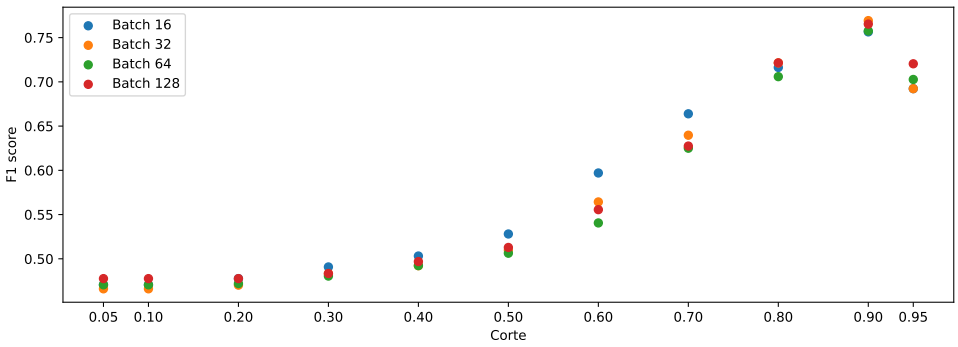


Figura 28: F_1 score obtenido para diferentes cortes sobre el conjunto etiquetado a mano haciendo un ajuste fino al modelo con diferentes tamaños de batch con el segundo conjunto de entrenamiento.

conjunto mientras que para el anterior era de 0.943.

El procedimiento para realizar el ajuste fino es el mismo que con el primer conjunto. En esta ocasión, presentamos también una búsqueda del mejor tamaño de batch de entrenamiento, pues por cómo funciona la función de pérdida Multiple Negatives Ranking Loss, este hiperparámetro es bastante relevante ya que modifica el número de casos negativos a positivos en el entrenamiento.

En este caso, se han realizado diferentes ajustes finos al modelo modificando el tamaño de batch a 16, 32, 64 y 128. Estos modelos, se han testeado sobre el conjunto etiquetado a mano con diferentes cortes. En la Figura 28 se presenta el F_1 score obtenido en cada caso. Para todos los modelos, el mejor corte se encuentra en 0.9, donde el tamaño de batch no influye mucho llegando todos los modelos a un F_1 score similar. Sin embargo, podemos ver en el corte 0.6 como el tamaño de batch sí que afecta bastante al entrenamiento del modelo.

El mayor F_1 score se obtiene haciendo el ajuste fino con un tamaño de batch de 32 y corte en el algoritmo de 0.9, donde el valor del F_1 score es de 0.769. Con este conjunto de entrenamiento hemos conseguido incrementar en 6 centésimas el F_1 score máximo respecto entrenando con el primer conjunto de entrenamiento.

Resulta interesante observar cómo ha quedado la matriz de similariades presentada en la

Figura 24 tras realizar el ajuste fino del modelo. En la Figura 29 podemos observar que tras dicho ajuste, las abreviaturas de las palabras son tan próximas a las palabras como entre ellas. De esta forma, el modelo aprende que las abreviaturas "dept" o "dpto" son parecidas a "department" o "departamento", mientras que la palabra "hospital" ya queda como una palabra diferente al resto.

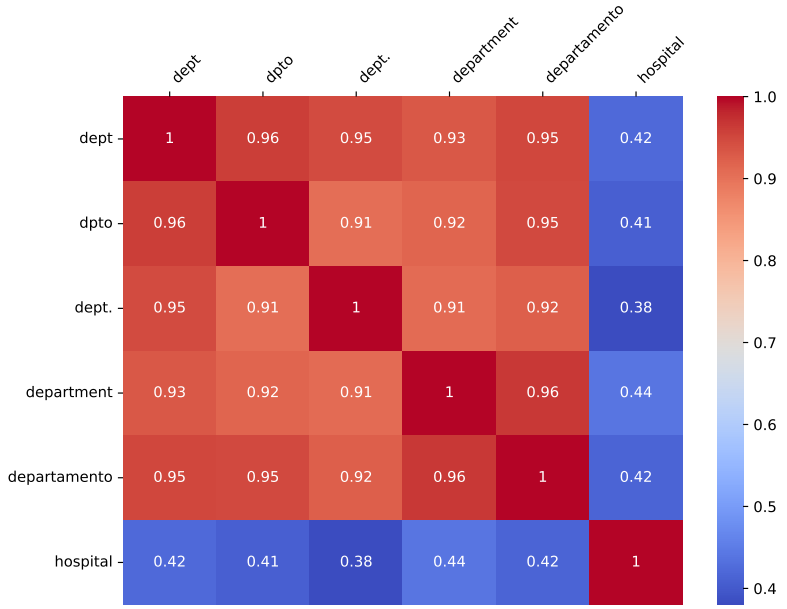


Figura 29: Matriz de similaridades para diferentes palabras usando el modelo con el ajuste fino sobre el segundo conjunto de entrenamiento.

4. Conclusiones

En este trabajo se ha desarrollado una herramienta completa desde la descarga de artículos científicos de Web of Science hasta la creación de unas tablas maestras donde se unifican los autores y se identifican las filiaciones. Para ello se han empleado tanto herramientas propias del *Big Data*, como es PySpark para gestionar el volumen masivo de datos el cual incorpora operaciones SQL y NoSQL, como herramientas de Inteligencia Artificial para los diferentes algoritmos utilizados para la unificación de autores y la identificación de filiaciones contra un maestro.

Se han explorado, tanto en la unificación de autores como la identificación de filiaciones, dos herramientas distintas para resolver el problema. La primera usando una distancia de edición como es la de Levenshtein, y la segunda utilizando modelos de redes neuronales con arquitectura Transformers mucho más complejos que la distancia de edición, y que además se encuentran en el estado del arte actualmente en cuanto a problemas de procesamiento de lenguaje natural.

En cuanto a la unificación de autores, existía un modelo previo en Kampal para comparar dos personas. En este trabajo se ha desarrollado un modelo usando redes neuronales que obtiene unos resultados muy similares, pero además, es mucho más rápido, siendo capaces de reducir el tiempo de procesamiento de un conjunto de entrada de 82404 autores de 5 horas a apenas 5-10 minutos. Esto es gracias a dos factores. El primero es poder condensar toda la información de una persona

en un vector semántico, y el segundo es el uso de GPUs. Realizar una comparación mediante distancias de edición en CPU es mucho más costoso en tiempo que hacer una similaridad coseno entre dos vectores con una GPU ya que el cálculo se hace de manera paralela. Sin embargo, debido a esta condensación de la información, es más difícil capturar algunos detalles como las iniciales de un nombre que con la distancia de edición se captura mejor.

En cuanto a la identificación de filiaciones, se ha desarrollado un algoritmo usando distancias de edición y otro utilizando modelos de redes neuronales. En este caso los resultados obtenidos con el algoritmo basado en distancias edición han sido mejores que los obtenidos con el algoritmo basado en redes neuronales. En el problema de unificación de autores comentamos que hay detalles como las iniciales del nombre que pueden afectar al rendimiento del modelo. En este caso nos encontramos con el problema de que hay muchas más formas diferentes de escritura de una filiación por las posibles abreviaturas de palabras, orden de las mismas, así como por el uso de diferentes idiomas, lo que complica todavía más esta tarea. Como se ha mostrado, estas posibles alteraciones, especialmente las abreviaturas, afectan al tokenizador del modelo.

Como posibles mejoras encontramos el disponer de un mejor conjunto de entrenamiento para ambos problemas. En los dos casos el conjunto de entrenamiento con el que se realiza el ajuste fino a las redes neuronales es generado a partir de casos sintéticos y no de casos reales. Es difícil generar un conjunto de entrenamiento que disponga de toda la casuística y variedad que nos encontramos entre los autores y filiaciones que se descargan de Web of Science, por lo que trabajar en mejorar estos conjuntos de entrenamiento es una línea muy interesante para poder mejorar el resultado de estos modelos. También se pueden estudiar el rendimiento de otros modelos, en este trabajo nos hemos basado en el modelo preentrenado "distiluse-base-multilingual-cased-v1" de Sentence Transformers, pero se podría estudiar el uso de otros modelos también preentrenados en los que quizá el tokenizador funcione mejor para estos problemas. Otra vía de mejora consiste en la utilización de modelos mucho más grandes como puede ser GPT-4, apoyándonos en él para generar estos conjuntos de entrenamiento. Esta técnica se puede ver como un caso particular de Profesor-Alumno, donde la idea es que a partir de un modelo más grande, como es GPT-4, enseñar a un modelo más pequeño, como son los que usamos nosotros. El objetivo de esta técnica es igualar los resultados del Profesor con un modelo más pequeño al tratarse de un problema más concreto.

Referencias

[1] J. A. Molina, D. Iniguez, G. Ruiz, and A. Tarancon, "Leaders among the leaders in economics: a network analysis of the nobel prize laureates," *APPLIED ECONOMICS LETTERS*, vol. 28, no. 7, pp. 584–589, APR 16 2021.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[4] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: http://arxiv.org/abs/1908.10084

[5] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, 2006, pp. 1735–1742.

[6] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2015. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2015.7298682

[7] M. Henderson, R. Al-Rfou, B. Strope, Y. hsuan Sung, L. Lukacs, R. Guo, S. Kumar, B. Miklos, and R. Kurzweil, "Efficient natural language response suggestion for smart reply," 2017. [Online]. Available: https://arxiv.org/abs/1705.00652

[8] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[9] Y. Goldberg and O. Levy, "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method," 2014. [Online]. Available: https://arxiv.org/abs/1402.3722

[10] Y. Sun, Y. Zheng, C. Hao, and H. Qiu, "Nsp-bert: A prompt-based few-shot learner through an original pre-training task–next sentence prediction," 2022. [Online]. Available: https://arxiv.org/abs/2109.03564