Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
(Mención Computación)

Procesado de lenguaje natural mediante Redes Neuronales Recurrentes.

Autor:

Esther Cuervo Fernández

Tutores:

Teodoro Calonge Cano

Joaquín Adiego Rodríguez

Agradecimientos

A mis padres, por su constante apoyo y ayuda.
A Raquel y Diego, por hacer estos cuatro años más amenos.

Resumen

Este proyecto se ha realizado como Trabajo de Fin de Grado de Ingeniería Informática con mención en Computación de la Universidad de Valladolid.

El trabajo descrito en esta memoria consiste en la realización de una aplicación que haga uso de Redes Neuronales Recurrentes para clasificar textos periodísticos.

La aplicación se ha desarrollado en su totalidad en Python 3, y es compatible con Windows 7, 8, o 10, y cualquier sistema basado en Linux compatible con Python 3.

Abstract

This work corresponds to the Final Project to obtain the Undergraduate Degree in Computing at the University of Valladolid (Spain).

This document describes a Recurrent Artificial Neural Network applied to journal texts classification.

The software has been fully programmed using Python3, which guarantees the use compatibility on Windows 7, 8, and 10 release, as well as on any linux instalation which has built-in Python3.

Índice

Índice de figuras

Índice de cuadros

Parte I

Introducción

1. Motivación

El procesado de lenguaje natural o NLP, del inglés *Natural Language Processing*, es un campo de la inteligencia artificial que engloba toda interacción entre ordenadores y lenguajes denominados naturales. Un lenguaje natural es aquel desarrollado sin premeditación, surgido de su uso con finalidad comunicativa entre seres humanos. Están en contraposición con los planificados, que son creados especificamente siguiendo una serie de reglas, para su uso en lógica, matemáticas, o, como en el caso del Esperanto, para su uso internacional.

Precisamente en esa falta de reglas explícitas y la existencia de contexto es donde surge la dificultad en reconocer y manejar lenguajes naturales automáticamente. Las primeras aproximaciones se realizaron a mediados del siglo XX, motivadas por el éxito en la rotura de códigos secretos realizada durante la Segunda Guerra Mundial. Estos sistemas generalmente buscaban la traducción de textos de un lenguaje a otro, y se basaban en enormes diccionarios bidireccionales con algunas reglas generadas manualmente, que pronto fueron consideradas demasiado complejas para ser definidas explícitamente [Hut05].

Es a principios y mediados de la década de los 90 cuando se comienzan a usar métodos estadísticos de aprendizaje automático en el análisis de lenguaje natural basados en árboles de decisión que generaban reglas estrictas. Sin embargo, la investigación ha llevado al uso de modelos probabilísticos a través de la exploración de grandes corpus de escritos.

Dentro del campo del procesado de lenguaje natural, existen numerosos objetivos, como pueden ser: la traducción de textos, la generación automática, el reconocimiento óptico de caracteres, o el reconocimiento de sentimientos.

Este trabajo se centra en la clasificación de textos, más específicamente en la extracción del tema del que trata un texto complejo escrito con un lenguaje natural no controlado, como son el español y el inglés. Este problema tiene numerosos usos prácticos, como clasificado de libros, marketing, extracción de temas en textos legales, médicos, etc.

El planteamiento general de este trabajo consiste en considerar los textos como una secuencia de palabras de longitud arbitraria, sobre las que se aplicarán diferentes redes neuronales recurrentes para su clasificación. Estas redes se engloban dentro del campo denominado *Deep Learning*, formado por mode-

los inspirados por sistemas biológicos. Modelos de *Deep Learning* se han usado previamente para clasificar texto (véase [Kow+18]), y las Redes Neuronales Recurrentes se han utilizado para otras tareas relacionadas con NLP (véase [Mes+13]), pero el uso explícito de Redes Neuronales Recurrentes para clasificación está poco explorado.

Respecto a aplicaciones de procesado de lenguaje natural de uso comercial, existen opciones gratuitas, como las herramientas ofrecidas por *The Standford Natural Language Processing Group*, que consisten en paquetes desarrollados en Java, y opciones de pago como las API *MonkeyLearn* o *Natural Language Understanding* de *Ambiverse*. Mientras que estas y otras aplicaciones son utilizadas por varias instituciones para tareas de análisis de lenguaje natural, este proyecto busca explotar específicamente la efectividad de las Redes Neuronales Recurrentes.

2. Objetivos

El objetivo principal del proyecto es elaborar una aplicación que permita el entrenamiento de modelos complejos de Aprendizaje Automático, a usuarios con conocimientos básicos sobre Redes Neuronales, que puedan utilizarse para clasificar textos.

Por tanto, la aplicación debe contar con tres funcionalidades esenciales:

1. Almacenamiento y gestión tanto de textos, como de modelos de Aprendizaje Automático.
2. Procesado de textos complejos para su utilización por redes neuronales.
3. Implementación de dos tipos de Redes Neuronales Recurrentes, las denominadas Elman y Jordan.

Debido a la metodología basada en prueba-error y la constante evolución del campo de la Inteligencia Artificial, se ha buscado una mayor independencia y modularidad entre estas funcionalidades, para facilitar cualquier cambio tanto en el procesado NLP, como en el tipo de clasificador utilizado.

Otro de los objetivos es obtener resultados tras la utilización de la aplicación sobre un conjunto de documentos, que permitan una primera valoración sobre la eficacia de las Redes Neuronales Recurrentes en el análisis de Lenguaje Natural.

Parte II

Fundamento teórico

A continuación se hará una breve introducción sobre la teoría en la que se asienta el presente trabajo.

3. Redes Neuronales

Una red neuronal es un sistema computacional, que se apoya en modelos matemáticos basados en los procesos que ocurren en el cerebro humano. Su origen se remonta al trabajo pionero de los investigadores Warren McCulloch y Walter Pitts en su artículo de 1943, *A Logical Calculus of Ideas Immanent in Nervous Activity*, donde proponen que el sistema nervioso humano se puede representar mediante modelos matemáticos [MP43].

Este trabajo es expandido por Frank Rosenblatt en 1958, con la invención del perceptrón simple, el primer modelo de aprendizaje supervisado. Para una explicación detallada del funcionamiento de este modelo, véase el capítulo 1 del libro de Simon Haykin, *Neural Networks and Learning Machines* [Hay09].

Sin embargo, este modelo estaba limitado a problemas linealmente separables, como demostraba en 1969 Marvin Minsky y Seymour Papert en su libro *Perceptrons* [MP69]. Como solución a esta limitación, surge el Perceptrón Multicapa, o MLP, del inglés *Multilayer Perceptron*, que introduce una o más capas ocultas, que permiten aproximar problemas no linealmente separables. Para más información sobre el fucnionamiento del perceptrón multicapa véase el capítulo 4 de *Neural Networks and Learning Machines* [Hay09].

4. Redes Neuronales Recurrentes

Mientras que los perceptrones multicapa son capaces de resolver diversidad de problemas, no son capaces de modelar uno de los factores clave en muchas tareas de aprendizaje: el tiempo. En nuestro caso particular, el análisis de lenguaje natural, la temporalidad surge mediante secuencias de palabras.

Los sistemas que tienen en cuenta el tiempo en su modelado se conocen como sistemas dinámicos. Para más información sobre estos sistemas tratados con Redes Neuronales véase el capítulo 13 de *Neural Networks and Learning Machines* [Hay09].

Existen varias maneras de introducir dinamismo en una red neuronal. En el caso de nuestra aplicación, se parte de la arquitectura tradicional de un MLP, y se introduce realimentación global, es decir, se involucra a varias capas del perceptrón. Esta realimentación lleva la salida de una capa en el instante t a la entrada de otra capa en el instante $t + 1$.

En este trabajo se hace uso de dos tipos de Redes Neuronales Recurrentes:

4.1. Red Elman

La Red Elman, también conocida como *simple recurrent network* es descita por Jeffrey Elman en 1990, en su artículo *Finding structure in time* [Elm90], que exploraba, precisamente, el uso de recurrencia en el aprendizaje del lenguaje.

Esta red consiste en introducir una copia de la capa oculta de un perceptrón multicapa en un instante t del entrenamiento y utilizarla como entrada a la capa oculta en un instante $t + 1$.

Se describe la arquitectura básica de esta red en la siguiente ilustración incluida en *Finding structure in time* [Elm90].

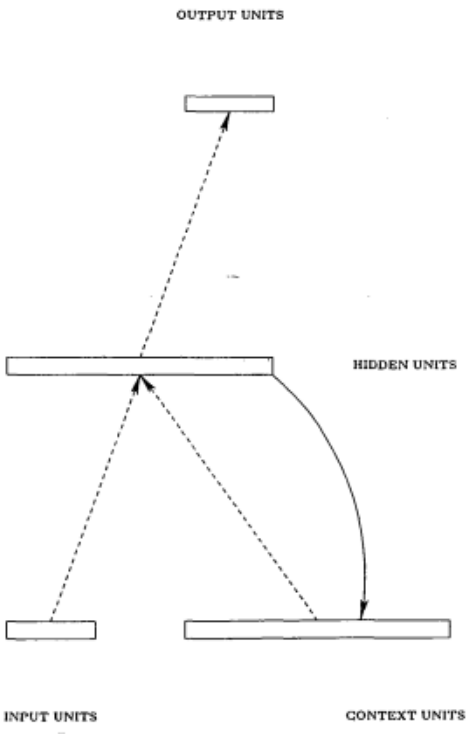

Figura 1: Arquitectura de la red Elman.

4.2. Red Jordan

La Red Jordan es descrita por Michael I. Jordan en 1986, en su informe *Serial order: a parallel distributed processing approach* [Jor86].

Esta red sigue el mismo principio de realimentación entre capas, pero la copiada es la capa de salida.

Se incluye a continuación una figura de la arquitectura básica de la red, incluida en *Serial order: a parallel distributed processing approach* [Jor86].

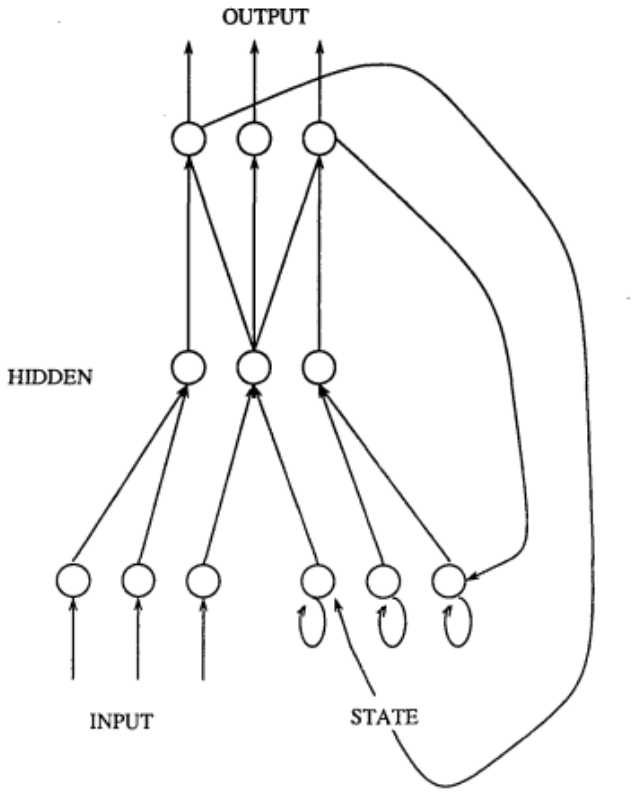

Figura 2: Arquitectura de la red Jordan.

5. Formulaciones

Al igual que en el perceptrón multicapa, las redes recurrentes cuentan con dos fases, la fase hacia delante, en la cual la entrada es introducida en la red, y la fase hacia atrás, en la que se calcula el error cometido por la red y se modifican los pesos.

Existen dos métodos principales de aprendizaje supervisado, el denominado

Batch learning y *On-line learning*, que habitualmente se traduce por aprendizaje episódico y continuo respectivamente. En el primero se actualizan los pesos después de que la red haya observado todos los ejemplos de entrenamiento. En *On-line learning* se actualizan los pesos después de cada ejemplo. Para más información sobre estos métodos véase la sección 4.3 de *Neural Networks and Learning Machines* [Hay09].

En esta aplicación se ha utilizado *On-line learning*, ya que es más simple de implementar y se ha comprobado que es más eficaz con problemas dificiles.

Ya que las redes Elman y Jordan son relativamente similares en su arquitectura, las fórmulas son también similares. Por ello, se desarrollan las fórmulas solo para la red Elman, tras lo cual se indican las diferencias con la de Jordan.

5.1. Fase hacia delante

Designando como:

- N: el número de neuronas de entrada.
- H: el número de neuronas en la capa oculta.
- S: el número de neuronas en la capa de salida.
- w_{ij}: el peso de la conexión que va de la neurona i a la neurona j.
- θ_j: el término umbral de la neurona j.
- $\sigma(x)$: la función de activación, en esta aplicación es la función sigmoide $\frac{1}{1+e^{-x}}$.
- $\overline{x}(t)$: el vector de entrada en el instante t. $[x_1, x_2, ..., x_N]$.
- $\overline{h}(t)$: la salida de la capa oculta en el instante t. $[h_1, h_2, ..., h_H]$.
- $\overline{y}(t)$: la salida de la capa de salida en el instante t. $[y_1, y_2, ..., y_S]$.

Debido a que la aplicación se basa en el clasificado de secuencias, es importante indicar que cada instancia de entrenamiento estará formada por un número arbitrario de vectores $\overline{x}$ de entrada. Cada instancia se puede representar con una matriz como esta:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{t1} & x_{t2} & x_{t3} & \dots & x_{tN} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{\tau 1} & x_{\tau 2} & x_{\tau 3} & \dots & x_{\tau N} \end{bmatrix} \qquad \begin{array}{lccccc} \overline{x}(1) = & [x_{11} & x_{12} & x_{13} & \dots & x_{1N}] \\ & \vdots & \vdots & \vdots & \vdots & \vdots \\ \overline{x}(t) = & [x_{t1} & x_{t2} & x_{t3} & \dots & x_{tN}] \\ & \vdots & \vdots & \vdots & \vdots & \vdots \\ \overline{x}(\tau) = & [x_{\tau 1} & x_{\tau 2} & x_{\tau 3} & \dots & x_{\tau N}] \end{array}$$

Siendo τ el número de elementos en la instancia representada.

Capa oculta La capa oculta recibe como entrada el vector de entrada, y la salida de la capa oculta en el instante anterior.

La salida de la neurona oculta j en el instante $1 < t \leq \tau$ es:

$$h_j(t) = \sigma(u_j) \tag{1}$$

$$u_j = \sum_{i=1}^{N} w_{ij} x_i(t) + \sum_{z=1}^{H} w_{zj} h_z(t-1) + \theta_j \tag{2}$$

En el instante inicial $t = 1$, no existe salida de la capa oculta para el instante anterior, por lo que la fórmula es:

$$h_j(1) = \sigma(u_j) \tag{3}$$

$$u_j = \sum_{i=1}^{N} w_{ij} x_i(1) + \theta_j \tag{4}$$

Capa salida La capa de salida recibe como entrada la respuesta de la capa oculta.

La salida de la neurona de salida k en el instante $1 \leq t \geq \tau$ es:

$$y_k(t) = \sigma(u_k) \tag{5}$$

$$u_k = \sum_{j=1}^{H} w_{jk} h_j(t) + \theta_k \tag{6}$$

Como se trata de una tarea de clasificación, la salida de la red *net* se corresponde con la clase asignada a la neurona de salida con mayor valor. Aunque *net* existe para todo elemento t de la secuencia, sólo tiene sentido al final del procesado de la secuencia completa, es decir para $t = \tau$. Si contamos con un

vector *clases* que contiene la clase correspondiente a cada neurona de salida, $argmax(y)$ sería la función que devuelve la posición del máximo en el vector y:

$$net(\tau) = clases[argmax(\overline{y}(\tau))] \tag{7}$$

5.2. Fase hacia atrás

En la fase hacia atrás se aplica el algoritmo de *Backpropagation*. Este algoritmo calcula el gradiente del error en la capa de salida, y lo propaga hacia capas anteriores. Este gradiente se basa en la derivada parcial del error respecto a cada peso de la red.

La función de coste utilizada es la del error cuadrático. Siendo d la salida deseada de la última capa, que en el caso de la clasificación se traduce en un vector $\overline{d}$ con valor $0,9$ en el índice correspondiente a la clase real y $0,1$ en el resto de posiciones. Estos dos valores numéricos sustituyen al 1 y 0 respectivamente, justificado por el problema de saturación de la función sigmoide. De esta manera, el error obtenido para una instancia tras la fase hacia adelante es:

$$Error = C(\overline{d} - \overline{y}(\tau))^2 = \frac{1}{2}(\overline{d} - \overline{y}(\tau))^2 \tag{8}$$

C es una constante arbitraria que, en el caso de este algoritmo, tiene valor $\frac{1}{2}$, para poder simplificar el exponente a la hora de derivar.

Capa de salida Se calcula la variación del peso w_{jk} de la conexion entre la neurona de la capa oculta j y la neurona de la capa de salida k.

$$\Delta w_{jk} = -\frac{\partial Error}{\partial w_{jk}} = -\frac{\partial Error}{\partial y_k(\tau)} \frac{\partial y_k(\tau)}{\partial w_{jk}} \tag{9}$$

$$-\frac{\partial Error}{\partial y_k(\tau)} = -\frac{\partial \frac{1}{2}(d_k - y_k(\tau))^2}{\partial y_k(\tau)} = d_k - y_k(\tau) \tag{10}$$

$$\frac{\partial y_k(\tau)}{\partial w_{jk}} = \frac{\partial \sigma(u_k)}{\partial w_{jk}} = \sigma'(u_k) h_j(\tau) \tag{11}$$

$$\Delta w_{jk} = (d_k - y_k(\tau))\sigma'(u_k) h_j(\tau) \tag{12}$$

La derivada de la función sigmoide es $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, y $\sigma(u_k) = y_k(\tau)$, por lo que queda:

$$\Delta w_{jk} = (d_k - y_k(\tau)) y_k(\tau)(1 - y_k(\tau)) h_j(\tau) \tag{13}$$

Y agrupando aquellos términos que sólo dependen de la neurona de la capa de salida:

$$\Delta w_{jk} = \delta_k h_j(\tau) \tag{14}$$

El término bias θ_k también sufre una variación, que siguiendo el mismo procedimiento es:

$$\Delta\theta_k = -\frac{\partial Error}{\partial \theta_k} = -\frac{\partial Error}{\partial y_k(\tau)}\frac{\partial y_k(\tau)}{\partial \theta_k} \tag{15}$$

$$\vdots \tag{16}$$

$$\frac{\partial y_k(\tau)}{\partial \theta_k} = \frac{\partial \sigma(u_k)}{\partial \theta_k} = \sigma'(u_k) \tag{17}$$

$$\Delta\theta_k = (d_k - y_k(\tau))\sigma'(u_k) = \delta_k \tag{18}$$

Capa oculta En la capa oculta existen dos grupos de pesos, aquellos que conectan el vector de entrada a la red con la capa oculta, y los que conectan la salida de la capa oculta en el instante anterior.

Para el peso w_{ij} de las conexiones con la entrada:

$$\Delta w_{ij} = -\frac{\partial Error}{\partial w_{ij}} = -\frac{\partial Error}{\partial h_j(\tau)}\frac{\partial h_j(\tau)}{\partial w_{ij}} \tag{19}$$

Para obtener $-\frac{\partial Error}{\partial h_j(\tau)}$ podemos considerar $Error$ como una función de las salidas de las neuronas en la capa de salida que reciben $h_j(\tau)$ como entrada. Por tanto:

$$-\frac{\partial Error}{\partial h_j(\tau)} = -\frac{\partial E(y_1(\tau), y_2(\tau), ..., y_S(\tau))}{\partial h_j(\tau)} \tag{20}$$

Ya que todas las variables de la función $E(\overline{y}(\tau))$ son dependientes de $h_j(\tau)$ podemos calcular la derivada total respecto a $h_j(\tau)$:

$$-\frac{dE(\overline{y}(\tau))}{dh_j(\tau)} = -(\frac{\partial E(\overline{y}(\tau))}{\partial y_1(\tau)}\frac{\partial y_1(\tau)}{\partial h_j(\tau)} + ... + \frac{\partial E(\overline{y}(\tau))}{\partial y_S(\tau)}\frac{\partial y_S(\tau)}{\partial h_j(\tau)}) \tag{21}$$

$$= \sum_{k=1}^{S} -\frac{\partial Error}{\partial y_k(\tau)}\frac{\partial y_k(\tau)}{\partial h_j(\tau)} = \sum_{k=1}^{S}(d - y_k(\tau))\sigma'(u_k)w_{jk} \tag{22}$$

$$= \sum_{k=1}^{S}\delta_k w_{jk} \tag{23}$$

$$\frac{\partial h_j(\tau)}{\partial w_{ij}} = \sigma'(u_j)x_i(\tau) \tag{24}$$

$$\Delta w_{ij} = \sum_{k=1}^{S}\delta_k w_{jk}\sigma'(u_j)x_i(\tau) \tag{25}$$

Para los pesos de las conexiones de realimentación:

$$\Delta w_{zj} = -\frac{\partial Error}{\partial w_{zj}} = -\frac{\partial Error}{\partial h_j(\tau)}\frac{\partial h_j(\tau)}{\partial w_{zj}} \tag{26}$$

$$\vdots \tag{27}$$

$$\frac{\partial h_j(\tau)}{\partial w_{zj}} = \frac{\partial \sigma(u_j)}{\partial w_{zj}} = \sigma'(u_j)h_z(\tau - 1) \tag{28}$$

$$\Delta w_{zj} = \sum_{k=1}^{S}\delta_k w_{jk}\sigma'(u_j)h_z(\tau - 1) \tag{29}$$

De manera análoga, el término umbral θ_j de las neuronas de la capa oculta varía de la siguiente forma:

$$\Delta\theta_j = -\frac{\partial Error}{\partial \theta_j} = -\frac{\partial Error}{\partial h_j(\tau)}\frac{\partial h_j(\tau)}{\partial \theta_j} \tag{30}$$

$$\vdots \tag{31}$$

$$\frac{\partial h_j(\tau)}{\partial \theta_j} = \frac{\partial \sigma(u_j)}{\partial \theta_j} = \sigma'(u_j) \tag{32}$$

$$\Delta\theta_j = \sum_{k=1}^{S}\delta_k w_{jk}\sigma'(u_j) \tag{33}$$

Por tanto, los pesos obtenidos tras realizar la fase hacia atrás para la instancia p es:

$$w_{ij} = \alpha\Delta w_{ij} + \gamma\Delta^{p-1}w_{ij} \tag{34}$$

Siendo α el denominado factor de aprendizaje, $\Delta^{p-1}w_{ij}$ el término de inercia o término momento, que representa la variación del peso producida por la

instancia anterior, y γ el correspondiente factor de inercia. Esto último es introducido para evitar caer en mínimos locales y que el aprendizaje quede atrapado en uno de ellos.

5.3. Jordan

En el caso de la red Jordan, cambia la fase hacia delante:

$$h_j(t) = \sigma(u_j) \tag{35}$$

$$u_j = \sum_{i=1}^{N} w_{ij} x_i(t) + \sum_{z=1}^{S} w_{zj} y_z(t-1) + \theta_j \tag{36}$$

$$h_j(1) = \sigma(u_j) \tag{37}$$

$$u_j = \sum_{i=1}^{N} w_{ij} x_i(1) + \theta_j \tag{38}$$

Y el delta del peso w_{zj}:

$$\Delta w_{zj} = \sum_{k=1}^{S} \delta_k w_{jk} \sigma'(u_j) y_z(\tau - 1) \tag{39}$$

El resto de ecuaciones son análogas a las de Elman.

6. Selección de modelos

Mientras que, en términos simples, el objetivo del aprendizaje supervisado es el de entrenar un modelo con una serie de instancias de ejemplo, el verdadero objetivo es que la red neuronal se adapte también a entradas que nunca ha visto. Esto se consigue con dos conjuntos disjuntos, uno de entrenamiento y otro de validación.

Este último se usa para evaluar la eficiencia entre experimentos, donde varía alguno de los parámetros, como el número de neuronas en la capa oculta, o los factores de aprendizaje o momento. Pero también sirve para comparar las prestaciones entre diferentes técnicas de Aprendizaje Automático aplicadas al mismo problema.

Existen varios métodos de selección del conjunto de validación, pero en esta aplicación se implementan dos:

Holdout El holdout es un método en el cual se divide el conjunto utilizado en dos según una proporción dada (tradicionalmente $\frac{1}{3}$ para validación y el resto para entrenamiento). Esta elección es aleatoria, pero se debe preservar la distribución de clases. Esto se conoce bajo el nombre de estratificación. Con ello se consigue reproducir la casuística del experimento global en ambos conjuntos.

Validación cruzada La validación cruzada genera k particiones aleatorias, y entrena k modelos con los mismos hiperparámetros, con $\frac{1}{k}$ de los ejemplos para validación y el resto para entrenamiento, elegidos tal que cada ejemplo es utilizado para validación exactamente una vez. Los resultados obtenidos en los k modelos se promedian, lo que otorga una mejor aproximación a la calidad del modelo. Al igual que en holdout, estas particiones se realizan estratificando por clases.

Parte III

Procesado de los textos

En esta sección se introduce el conjunto de textos utilizados para la obtención de resultados, y se explica la transformación a la que se les ha sometido antes de introducirlos a una red neuronal recurrente.

7. Corpus EFE

El conjunto de textos utilizados ha sido proporcionado por D. Joaquín Adiego, profesor del Departamento de Informática de la Universidad de Valladolid, adscrito a la Escuela de Ingeniería Informática de Valladolid. Se compone de todas las noticias publicadas por la agencia EFE durante el año 1994. Se han obtenido mediante 365 ficheros estructurados, uno por día del año. Cada uno de estos archivos tiene una serie de noticias publicadas en ese día.

Se ha realizado un análisis inicial de dichos ficheros, obteniendo los siguientes datos:

- Hay 36 clases posibles.
- Los ficheros contienen 215.738 documentos.
- El corpus contiene 364.868 palabras diferentes.
- La longitud media de cada documento es de 323,76 palabras.

Un análisis más en profundidad de la distribución de documentos en clases produce los siguientes gráficos:

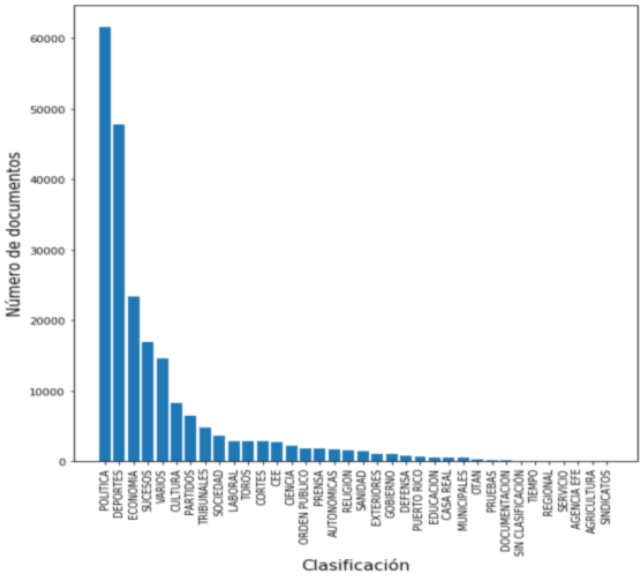

Figura 3: Diagrama de barras que muestra el número de documentos por clase en el corpus EFE.

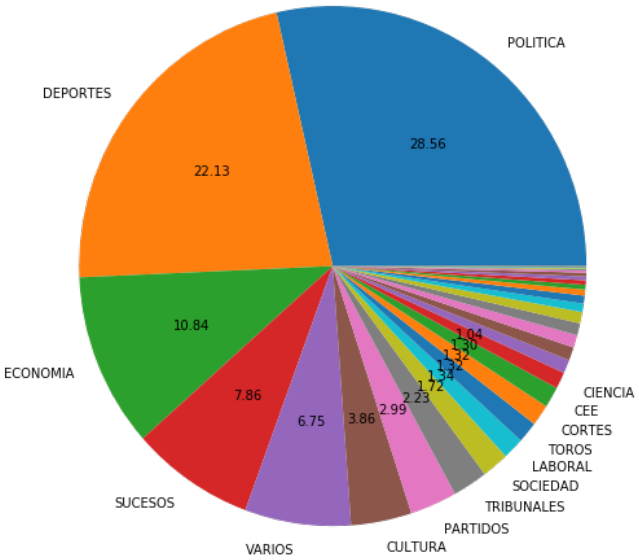

Figura 4: Diagrama de porciones que muestra el porcentaje de documentos por clase en el corpus EFE.

En estos gráficos se aprecia la disparidad entre clases, con dos de ellas mayoritarias: POLITICA y DEPORTES. Le siguen ECONOMIA, SUCESOS, y VARIOS como clases pequeñas pero aún representativas, y el resto con frecuencias de aparición en el corpus muy bajas.

Por tanto, y para simplificar el corpus, se han seleccionado los documentos clasificados con las categorías antes mencionadas, con la excepción de VARIOS, al resultar una etiqueta demasiado poco específica, que se ha sustituido por la siguiente con más documentos: CULTURA.

Tras esto, se obtienen las siguientes características:

- Hay 5 clases posibles.
- El corpus se compone de 158.060 documentos.
- El corpus contiene 304.958 palabras diferentes.
- La longitud media de cada documento es de 316,11 palabras.

A este corpus se le denominará corpus EFE durante el resto del documento.

8. Corpus EFE reducido

Debido a la complejidad del corpus EFE, se ha obtenido también uno reducido, en el cual se utiliza como contenido del documento el título de la noticia. Esto supone una enorme reducción de la complejidad, pero es útil para obtener datos concretos con el poder de computación del que es capaz un ordenador personal tradicional.

Este corpus tiene las siguientes características:

- Hay 5 clases posibles.
- El corpus contiene 158.060 documentos.
- El corpus contiene 53.512 palabras diferentes.
- La longitud media de cada documento es de 10,29 palabras.

9. Metodología

El principal objetivo del procesado de textos es convertir textos de longitud arbitraria en vectores numéricos. Un importante beneficio de las redes neuronales recurrentes es que no es necesario que estos vectores numéricos tengan una

longitud fija, ya que las secuencias introducidas a la red neuronal pueden tener un tamaño arbitrariamente largo.

9.1. Limpiado

El primer paso a la hora de procesar el texto es eliminar símbolos no alfanuméricos, como son los puntos, comas, signos de exclamación, etc. así como la conversión de todo el texto a minúsculas.

En cuanto a los números, se sustituyen por tokens únicos, diferenciando entre números enteros, números reales, y fechas con formato $[0-9]\{2\}/[0-9]\{2\}/[0-9]\{4\}$. Una primera aproximación para tratar los tokens sería eliminarlos, sin embargo se ha considerado que pueden aportar información en la clasificación. Así, un análisis inicial del corpus EFE revela que la frecuencia de estos tokens cambia con la clase:

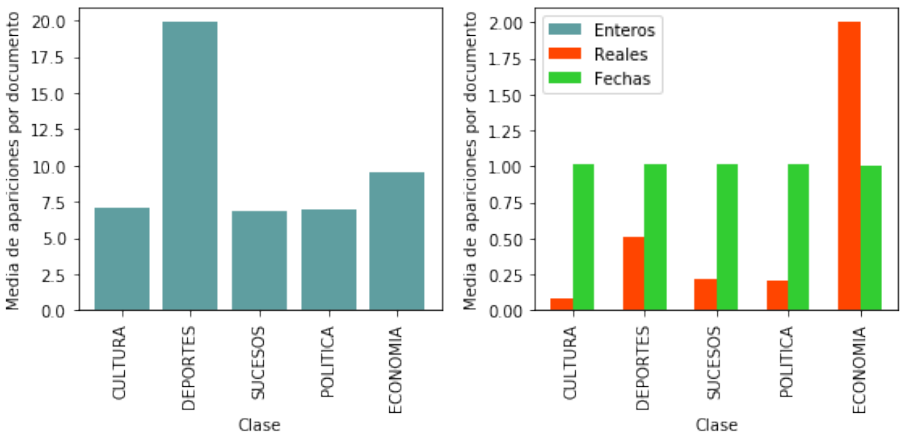

Figura 5: Diagrama de barras indicando la media de apariciones de los tres tipos de token por documento en el corpus EFE en cada clase

Como se puede observar, mientras que todas las clases tienen el mismo número medio de fechas, las noticias de deporte contienen muchos más enteros, mientras que las de economía resaltan por la cantidad de números reales.

9.2. Tokenizado

El segundo paso es el tokenizado, proceso que convierte un texto en un vector, separándolo por algún criterio. El criterio elegido en nuestro caso es separar por palabras, ya que la consideraremos la unidad de aprendizaje más pequeña en

nuestro modelo. Por tanto, tras el tokenizado obtendremos un vector de palabras por cada documento.

9.3. Palabras vacías

El tercer paso es el de eliminar las denominadas palabras vacías. Estas palabras son muy frecuentes en el lenguaje, pero no aportan ninguna información. En el siguiente gráfico se pueden apreciar las 50 palabras más frecuentes en corpus EFE, las marcadas en rojo son las consideradas por la librería *nltk* como palabras vacías para el español.

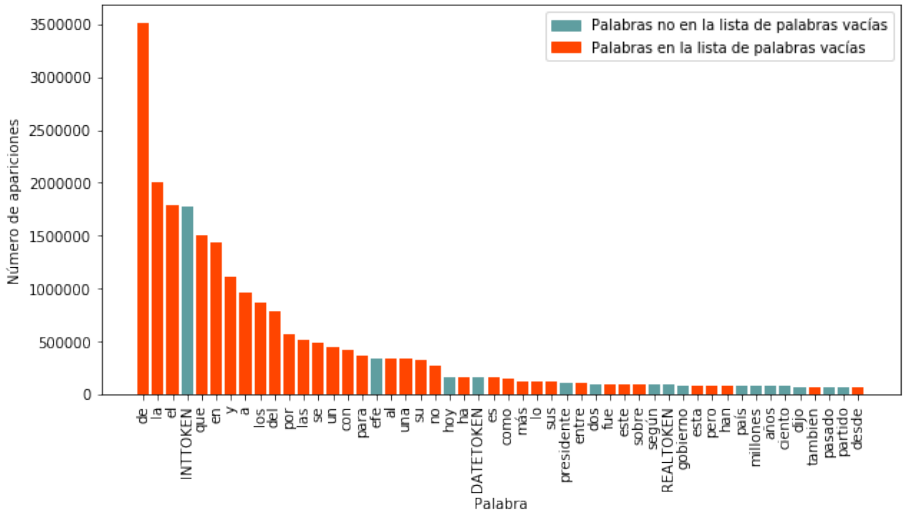

Figura 6: Diagrama de barras indicando la frecuencia de las 50 palabras más comunes en el corpus EFE.

Tras eliminar las palabras vacías, queda el siguiente gráfico:

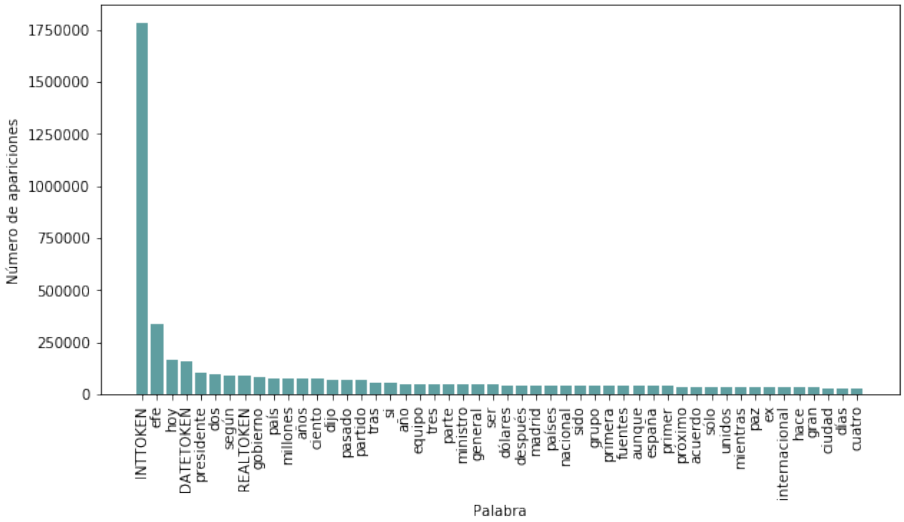

Figura 7: Diagrama de barras indicando la frecuencia de las 50 palabras más comunes en el corpus EFE una vez eliminadas las palabras vacías.

Como podemos observar, aún hay ciertas palabras que, debido a la naturaleza del corpus, probablemente no aporten ninguna información al entrenamiento, como puede ser la palabra *efe*, o, como habíamos mostrado en la Figura 5, *DATETOKEN*. Por ello, se permite incluir una lista de palabras vacías extra, que se eliminan del conjunto de entrenamiento, para dar libertad al usuario de adaptar la aplicación a las particularidades de su corpus.

9.4. Stemming

El siguiente proceso que se aplica a los textos es el stemming, que consiste en aplicar una heurística para reducir las palabras a su raíz. Este proceso busca normalizar palabras a una base común, reduciendo así la complejidad del corpus. En esta aplicación se utiliza el algoritmo Snowball, incluido en el paquete nltk. El funcionamiento de este algoritmo para el español se detalla en la página web oficial del algoritmo [Por].

Tras el stemming, el corpus tiene 179.520 palabras diferentes, frente a las 304.673 obtenidas tras eliminar palabras vacías, lo que implica una reducción de la complejidad del corpus de un 58,9 %.

9.5. Vocabulario

El último paso es la obtención de una asociación uno a uno entre palabras y números enteros, que denominaremos el vocabulario del modelo. Este vocabulario se calcula sobre el conjunto de entrenamiento durante el aprendizaje.

Se otorga al usuario tres opciones para el orden en el que se asignan números a las palabras en los textos, ya que la cercanía entre números puede producir una relación entre palabras en la red.

- El primero es por orden de aparición, en el que se itera por todas las palabras del corpus en el orden en el que aparecen, manteniendo un contador, que se asigna e incrementa cada vez que se encuentra una palabra que no haya aparecido ya en el vocabulario.
- El segundo es por orden aleatorio.
- El tercero es por orden lexicográfico.

El vocabulario obtenido se utiliza para convertir cada instancia del corpus en un vector numérico. Esto mismo se utiliza para convertir los conjuntos de validación y cualquier documento que se quiera clasificar utilizando el modelo. En el caso de las palabras no incluidas en el vocabulario, se cambian por un token único, que indica que la palabra es desconocida para el modelo.

Para tratar estos tokens, es posible asignarles un valor, o eliminarlos. En esta aplicación se ha optado por quitarlos, ya que no se considera que otorguen información útil a la red.

9.6. Normalizado

Para la introducción de los textos en la red es necesario convertir el rango del vocabulario, que contiene números enteros desde 0 hasta ∞, a reales en el rango 0,1 a 0,9.

Este normalizado se realiza principalmente por la naturaleza de la función de activación elegida.

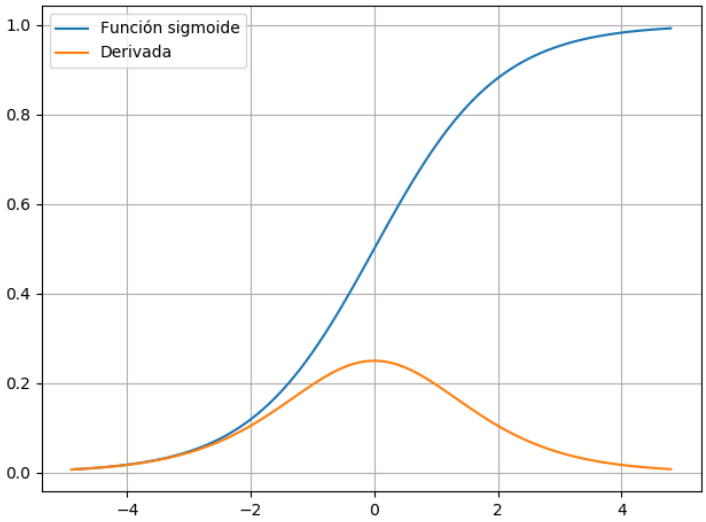

Figura 8: Función sigmoide y su derivada

Los valores $0,1$ y $0,9$ se utilizan para evitar la saturación de la función sigmoide [Has95].

La fórmula utilizada para normalizar la componente x_i del vector $\overline{x}$ es:

$$x_i' = 0{,}1 + (0{,}9 - 0{,}1)\frac{x_i - min(\overline{x})}{max(\overline{x}) - min(\overline{x})} \tag{40}$$

Parte IV

Aplicación

Se procede a describir los procesos de análisis y diseño de la aplicación, así como una sección en la que se explican decisiones tomadas en la implementación.

10. Análisis

10.1. Requisitos funcionales

ID	Nombre	Descripción
RF01	Importación de ficheros SGML	El sistema debe permitir cargar documentos con contenido, título, y clasificación desde ficheros estructurados con extensión .sgml
RF02	Importación de ficheros planos	El sistema debe permitir acceder al contenido de documentos desde ficheros de texto plano.
RF03	Editado atributos documentos	El sistema debe permitir modificar el título y clasificación de los documentos tras su importación.
RF04	Elimina documentos	El sistema debe permitir borrar documentos.
RF05	Visualización de documentos	El sistema debe permitir visualizar los documentos en el sistema, su contenido, título, y clasificación.
RF06	Entrenamiento de redes	El sistema debe permitir la utilización de documentos con clasificación en el sistema para el entrenamiento de Redes Neuronales Recurrentes.
RF07	Guardado de modelos	El sistema debe permitir almacenar el modelo resultante del entrenamiento.

RF08	Entrenamiento posterior	El sistema debe permitir continuar entrenando un modelo un número extra de iteraciones.
RF09	Clasificación de documentos	El sistema debe permitir utilizar los modelos almacenados para la clasificación de documentos en el sistema.

Cuadro 1: Tabla de requisitos funcionales

10.2. Requisitos no funcionales

ID	Nombre	Descripción
RN01	Entorno de uso	La aplicación debe funcionar en ordenadores con cualquier sistema operativo que soporte Python 3.6
RN02	Facilidad de uso	El sistema debe ser usable por personas con conocimiento básico sobre Redes Neuronales.
RN03	Lenguaje de programación	La aplicación será desarrollada en Python 3.6.
RN04	Sistema de almacenamiento	La aplicación usará una base de datos basada en SQLite como sistema de almacenamiento.
RN05	Librerías utilizadas	La aplicación utilizará librerías con licencias de uso libre disponibles en el índice de paquetes de Python PyPI

Cuadro 2: Tabla de requisitos no funcionales

10.3. Requisitos de información

ID	Nombre	Descripción
RI01	Documentos	El sistema debe almacenar de los documentos: su contenido y, si existe, su título y clasificación.
RI02	Modelos	El sistema debe almacenar de los modelos: el tipo de red utilizada, las etiquetas de las clases posibles, el número de neuronas, los pesos de las conexiones de la red, la asociación entre palabra y valor utilizado para convertir los textos a valores numéricos, y el valor mínimo y máximo necesario para normalizar dichos valores.

Cuadro 3: Tabla de requisitos de información

10.4. Casos de uso

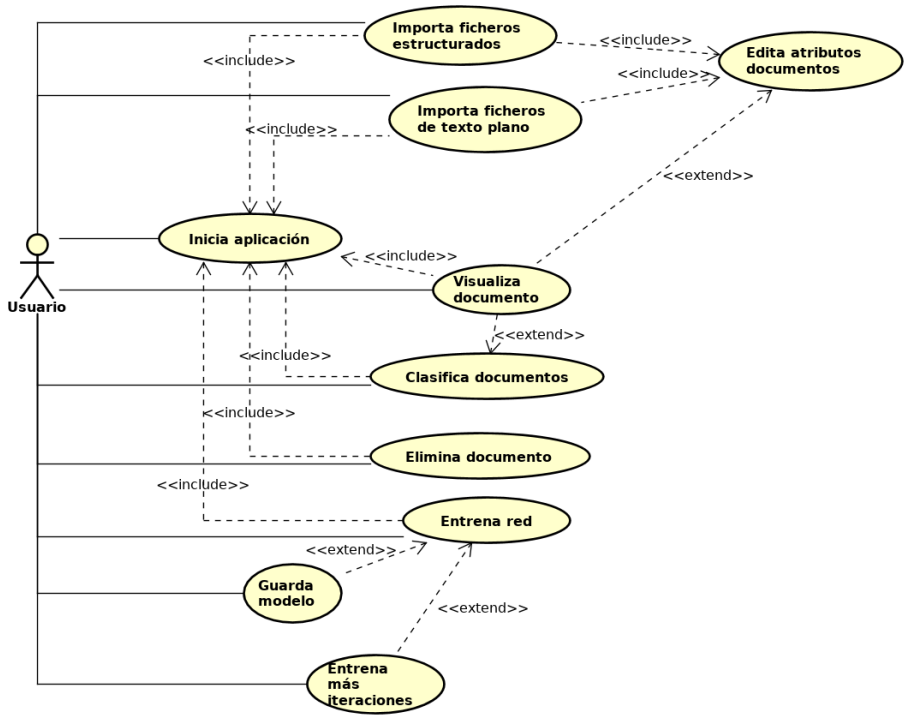

Figura 9: Diagrama de casos de uso

10.4.1. CU00: Inicia aplicación

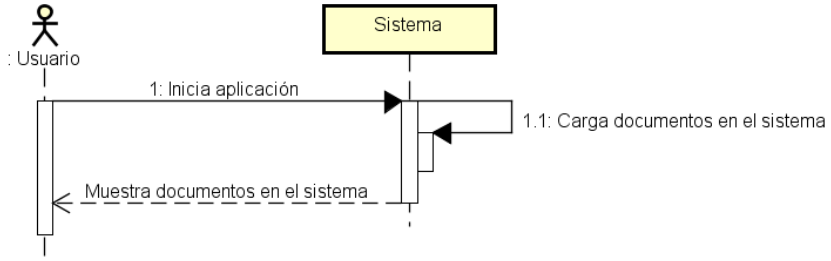

Figura 10: Diagrama de Secuencia de CU00: Inicia aplicación

Nombre	CU00 - Inicia aplicación.
Resumen	El sistema muestra la lista de documentos almacenados al iniciar la aplicación.
Actor	Usuario.
Precondición	
Postcondición	El usuario se encuentra en la pantalla principal.
Flujo principal	1. El usuario lanza la aplicación. 2. El sistema carga los documentos almacenados en el sistema y se los muestra al usuario.
Flujo alternativo	2. No existen documentos en el sistema. El sistema no muestra ningún documento.
Excepciones	

Cuadro 4: Descripción de CU00: Inicia aplicación

10.4.2. CU01: Importa ficheros estructurados

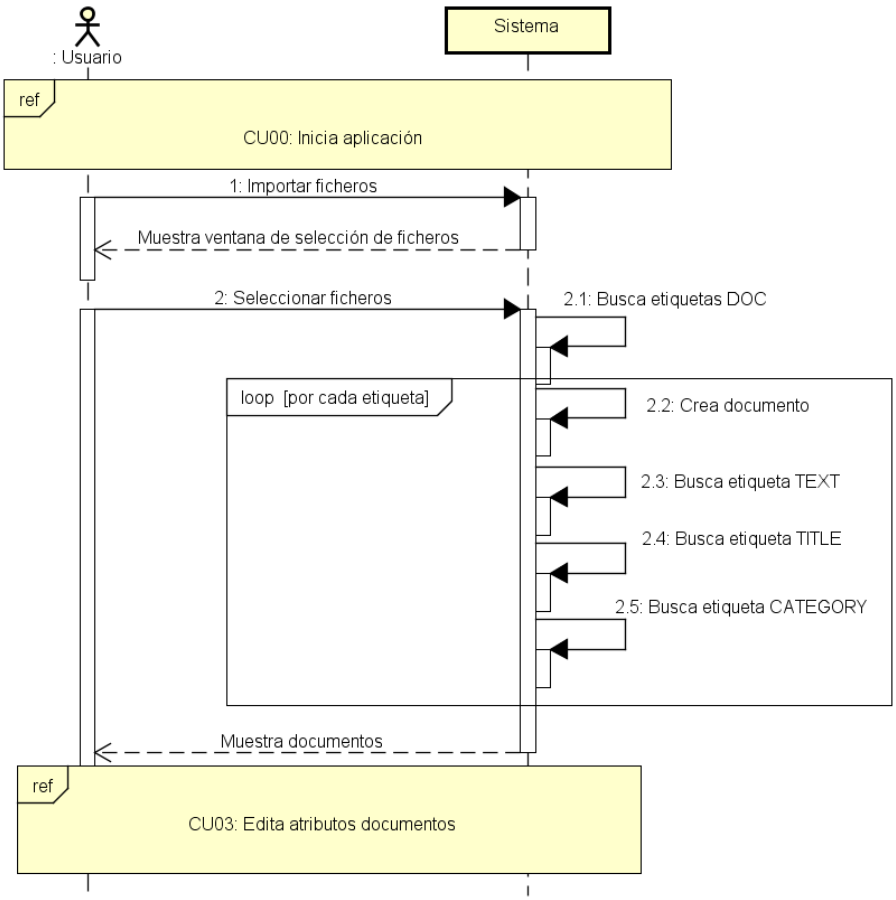

Figura 11: Diagrama de secuencia de CU01: Importa ficheros estructurados

Nombre	CU01 - Importa ficheros estructurados.
Resumen	El sistema debe permitir importar documentos con contenido, título, y clasificación desde ficheros estructurados con extensión .sgml.
Actor	Usuario.
Precondición	El sistema ha completado el caso de uso *CU00*.
Postcondición	Se leen el contenido, título, y clasificación de los documentos que se han encontrado en los ficheros.
Flujo principal	1. El usuario selecciona la opción de importar fichero. 2. El sistema muestra una ventana de selección de ficheros. 3. El usuario selecciona uno o más ficheros con extensión *.sgml* para importar. 4. El sistema busca las etiquetas DOC, y crea un documento por cada una. 5. El sistema busca las etiquetas de TEXT anidadas dentro de las DOC, y almacena el contenido en el cuerpo del documento. 6. El sistema busca las etiquetas TITLE anidadas dentro de las DOC, y almacena el contenido en el título del documento. 7. El sistema busca las etiquetas CATEGORY anidadas dentro de las DOC, y almacena el contenido en la clasificación del documento. 8. El sistema muestra la lista de documentos obtenida y ejecuta el caso de uso *CU03: Edita atributos documentos*.
Flujo alternativo	6a. No se encuentra la etiqueta TITLE. No se almacena ningún título en el documento. 7a. No se encuentra la etiqueta CATEGORY. No se almacena ninguna clasificación en el documento.
Excepciones	3a. El usuario cancela la importación. El caso de uso termina sin efecto. 3b. El usuario no selecciona ningún fichero. El caso de uso termina sin efecto. 4, 5. No se encuentran las etiquetas indicadas. Se muestra un mensaje de error y el caso de uso termina sin efecto.
Observaciones	Este caso de uso no guarda los documentos en el almacenamiento persistente.

Cuadro 5: Descripción de CU01: Importa ficheros estructurados

10.4.3. CU02: Importa ficheros de texto plano

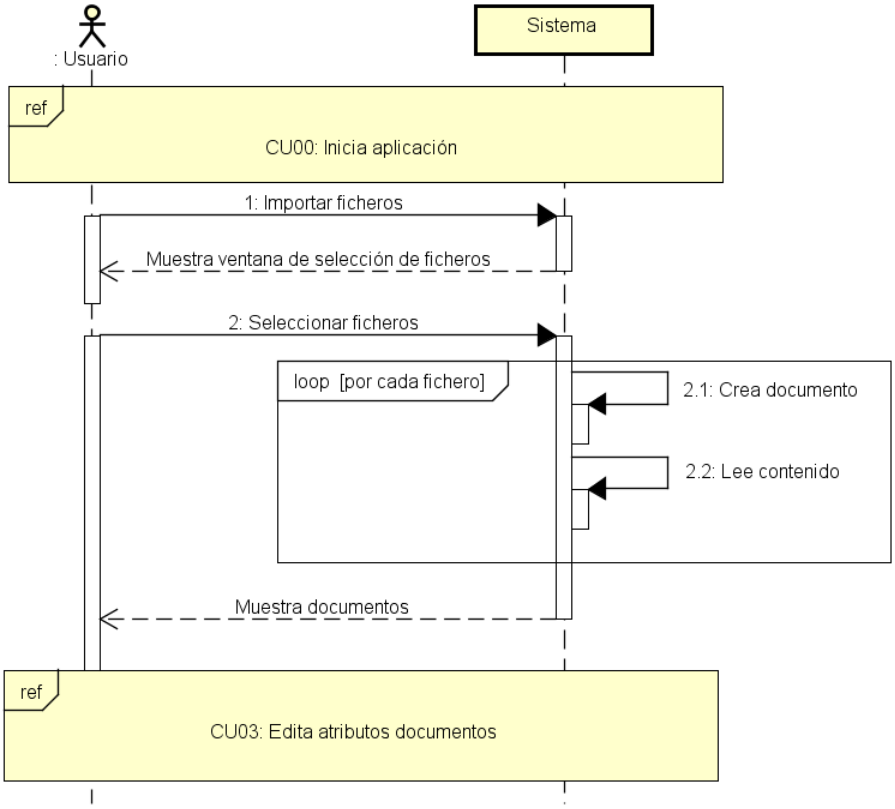

Figura 12: Diagrama de secuencia de CU02: Importa ficheros de texto plano

Nombre	CU02 - Importa ficheros de texto plano.
Resumen	El sistema debe permitir importar el contenido de documentos desde ficheros de texto plano.
Actor	Usuario.
Precondición	El sistema ha completado el caso de uso *CU00*.
Postcondición	Se leen el contenido de los ficheros seleccionados.
Flujo principal	1. El usuario selecciona la opción de importar fichero. 2. El sistema muestra una ventana de selección de ficheros. 3. El usuario selecciona uno o más ficheros con extensión arbitraria para importar. 5. El sistema crea un documento por cada fichero seleccionado. 4. El sistema lee el contenido de cada fichero al cuerpo del documento correspondiente. 5. El sistema muestra la lista de documentos obtenida y ejecuta el caso de uso *CU03: Edita atributos documentos*.
Flujo alternativo	
Excepciones	3a. El usuario cancela la importación. El caso de uso termina sin efecto. 3b. El usuario no selecciona ningún fichero. El caso de uso termina sin efecto. 4. No se puede leer el fichero. Se muestra un mensaje de error y el caso de uso termina sin efecto.
Observaciones	Este caso de uso no guarda los documentos en el almacenamiento persistente.

Cuadro 6: Descripción de CU02: Importa ficheros de texto plano

10.4.4. CU03: Edita atributos documentos

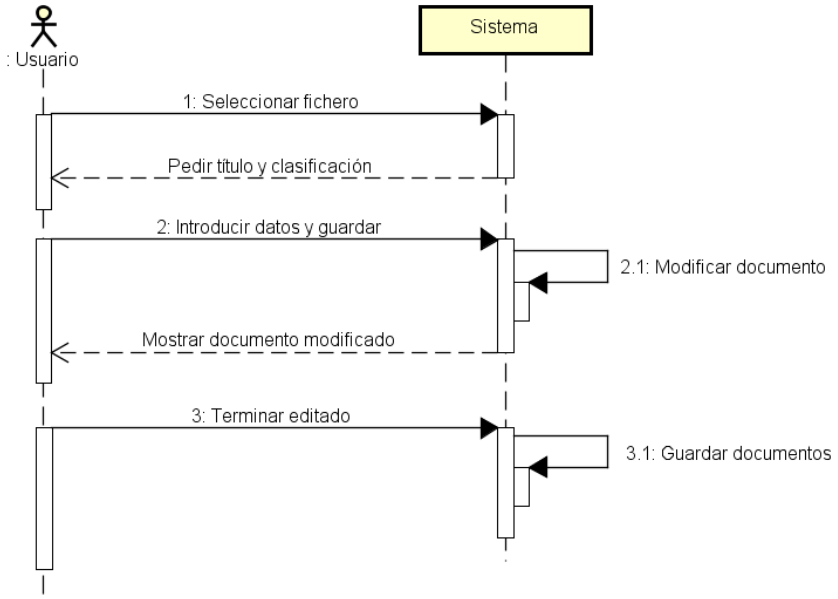

Figura 13: Diagrama de Secuencia de CU03: Edita atributos documentos

Nombre	CU03 - Edita atributos documentos.
Resumen	El sistema debe permitir editar el título y clasificación de los documentos tras su importación.
Actor	Usuario.
Precondición	El sistema ha completado el caso de uso *CU01* o *CU02*.
Postcondición	Los documentos se registran en el sistema con los títulos y clasificaciones introducidos.
Flujo principal	1. El usuario selecciona un documento de la lista de documentos obtenidos del caso de uso *CU01* o *CU02*. 2. El sistema permite introducir un nuevo título o clasificación para el documento seleccionado. 3. El usuario introduce un título o clasificación y pulsa el botón de guardado. 4. El sistema modifica el documento seleccionado con el título o clasificación introducido y lo muestra. 5. El usuario indica que ha terminado el editado de atributos. 6. El sistema guarda los documentos en el almacenamiento persistente y el caso de uso termina.
Flujo alternativo	1a. El usuario seleciona visualizar un documento. Se inicia el caso de uso *CU05: Visualiza documento*. 5a. El usuario selecciona otro documento. El caso de uso regresa al paso 2.
Excepciones	1, 3, 5. El usuario cancela el editado de documentos. El caso de uso termina sin efecto.

Cuadro 7: Descripción de CU03: Edita atributos documentos

10.4.5. CU04: Elimina documentos

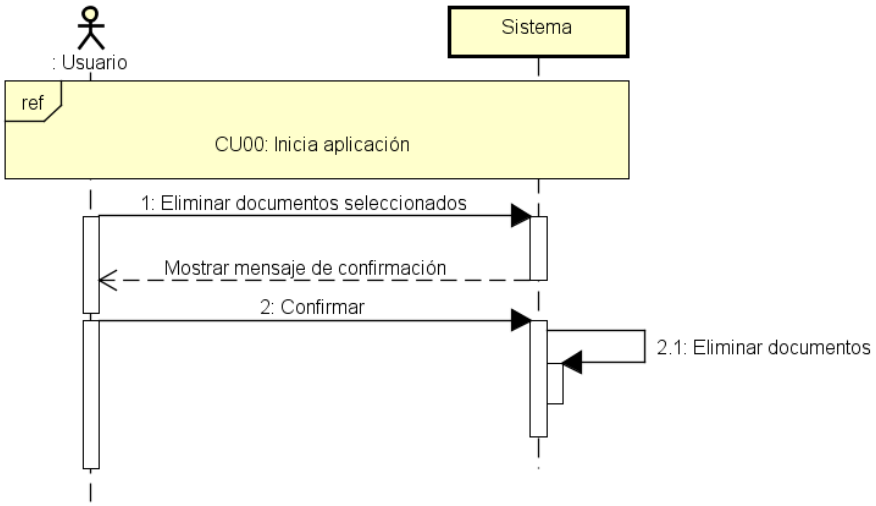

Figura 14: Diagrama de Secuencia de CU04: Elimina documentos

Nombre	CU04 - Elimina documentos.
Resumen	El sistema debe permitir eliminar documentos.
Actor	Usuario.
Precondición	Existen documentos almacenados en el sistema. El sistema ha completado el caso de uso *CU00*.
Postcondición	Los documentos seleccionados se eliminan del sistema.
Flujo principal	1. El usuario selecciona los documentos que quiere eliminar de la lista de documentos en el sistema, y selecciona la opción de eliminar documentos. 2. El sistema muestra un mensaje de confirmación. 3. El usuario confirma la eliminación. 4. El sistema elimina los documentos seleccionados del sistema y el caso de uso termina.
Flujo alternativo	
Excepciones	3. El usuario cancela la eliminación. El caso de uso termina sin efecto.

Cuadro 8: Descripción de CU04: Elimina documentos

10.4.6. CU05: Visualiza documento

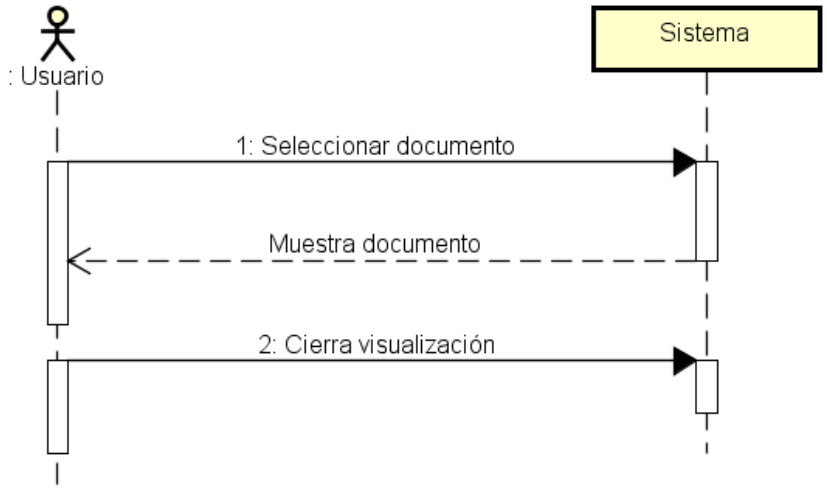

Figura 15: Diagrama de Secuencia de CU05: Visualiza documento

Nombre	CU05 - Visualiza documento.
Resumen	El sistema debe permitir visualizar los documentos en el sistema, su contenido, título, y clasificación.
Actor	Usuario.
Precondición	Existen documentos almacenados en el sistema. El usuario está en la pantalla principal, en la pantalla de editado de atributos de los documentos, o en la pantalla de visualización de resultados de la clasificación.
Postcondición	
Flujo principal	1. El usuario selecciona el documento que quiere visualizar de la lista de documentos en el sistema. 2. El sistema muestra una pantalla con el contenido, título, y clasificación del documento. 3. El usuario cierra la pantalla de visualización y el caso de uso termina.
Flujo alternativo	1a. El usuario selecciona el documento que quiere visualizar de la lista de documentos clasificados durante el caso de uso *CU07: Visualiza resultado clasificación* y el caso de uso pasa al paso 2. 1b. El usuario selecciona le documento que quiere visualizar de la lista de documentos durante el caso de uso *CU03: Edita atributos documentos* y el caso de uso pasa al paso 2.
Excepciones	

Cuadro 9: Descripción de CU05: Elimina documentos

10.4.7. CU06: Entrena red

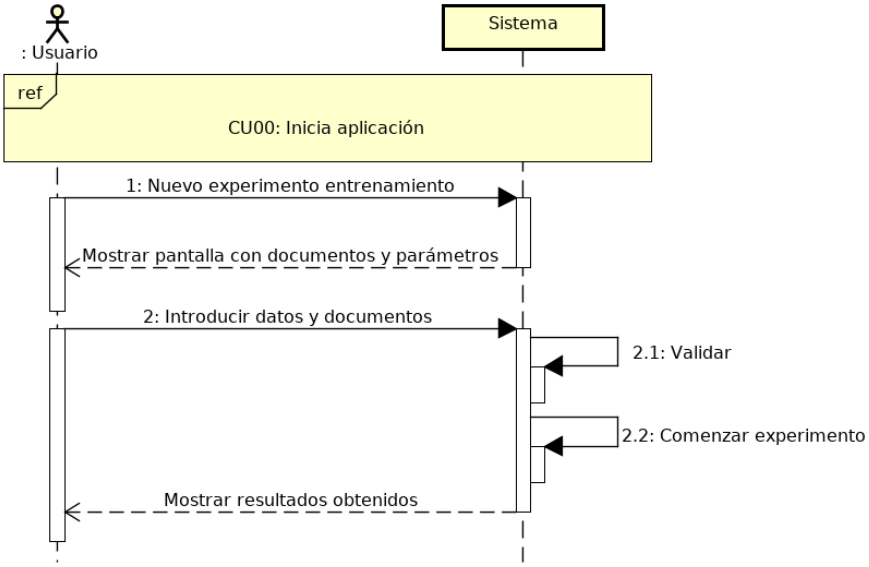

Figura 16: Diagrama de secuencia de CU06: Entrena red

Nombre	CU06 - Entrena red.
Resumen	El sistema debe permitir la utilización de documentos con clasificación en el sistema para el entrenamiento de Redes Neuronales Recurrentes.
Actor	Usuario.
Precondición	Existen documentos con clasificación almacenados en el sistema. El sistema ha completado el caso de uso *CU00*.
Postcondición	
Flujo principal	1. El usuario selecciona la opción de Nuevo Experimento de Entrenamiento. 2. El sistema muestra una pantalla con la lista de documentos con clasificación y los distintos parámetros para el experimento. 3. El usuario selecciona los documentos a utilizar en el entrenamiento, e introduce los datos. 4. El sistema comprueba que los datos introducidos sean válidos. 5. El sistema procesa los documentos y comienza el entrenamiento. 6. El experimento termina y el sistema muestra los resultados obtenidos. 7. El usuario cierra la pantalla de visualización de resultados y el caso de uso termina.
Flujo alternativo	4a. El sistema comprueba que alguno de los datos introducidos no es válido. Muestra un mensaje de error y regresa al paso 2. 7a. El usuario selecciona la opción de guardar modelo. Se inicia el caso de uso *CU07: Guarda Modelo*. 7b. El usuario selecciona la opción de entrenar más iteraciones. Se inicia el caso de uso *CU08: Entrena más iteraciones*.
Excepciones	3, 5. El usuario cancela el entrenamiento. El caso de uso termina sin efecto.

Cuadro 10: Descripción de CU06: Entrena red

10.4.8. CU07: Guarda Modelo

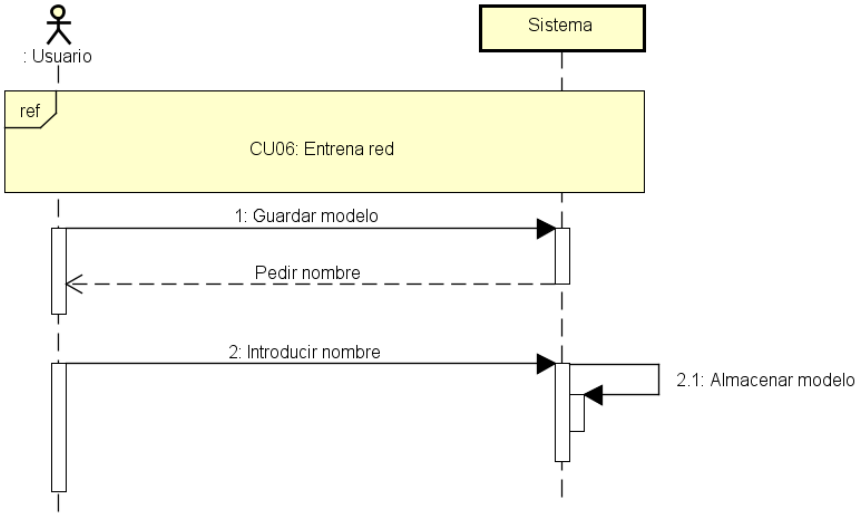

Figura 17: Diagrama de secuencia de CU07: Guarda Modelo

Nombre	CU07 - Guarda Modelo.
Resumen	El sistema debe permitir almacenar el modelo resultante del entrenamiento.
Actor	Usuario.
Precondición	Se ha ejecutado el flujo alternativo 7a del caso de uso *CU06- Entrena red*.
Postcondición	El modelo resultado del entrenamiento se registra en el sistema con el nombre indicado.
Flujo principal	1. El sistema solicita al usuario introducir un nombre para el modelo. 2. El usuario introduce un nombre. 3. El sistema guarda el modelo en el almacenamiento persistente y el caso de uso termina.
Flujo alternativo	
Excepciones	2. El usuario cancela el guardado. El caso de uso termina sin efecto.

Cuadro 11: Descripción de CU07: Guarda Modelo.

10.4.9. CU08: Entrena más iteraciones

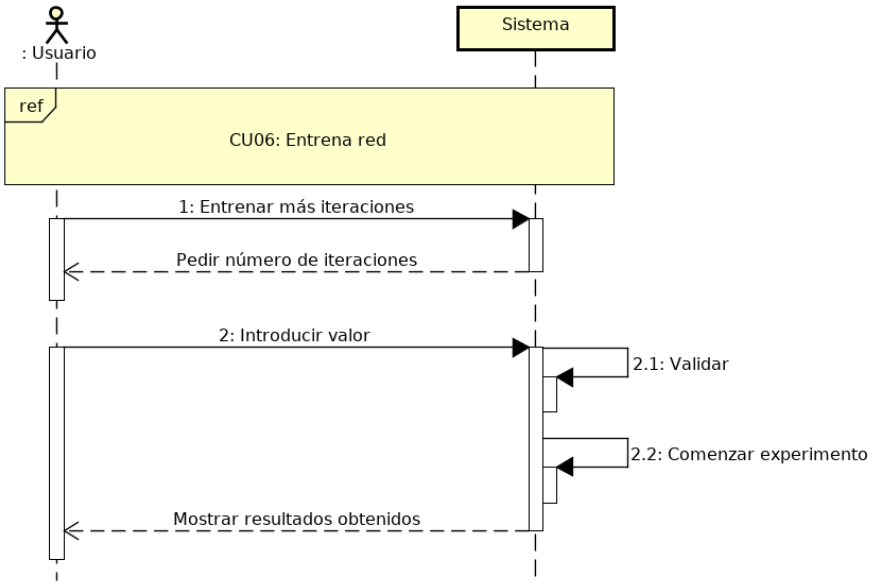

Figura 18: Diagrama de secuencia de CU08: Entrena más iteraciones.

Nombre	CU08 - Entrena más iteraciones.
Resumen	El sistema debe permitir continuar entrenando un modelo un número extra de iteraciones.
Actor	Usuario.
Precondición	Se ha ejecutado el flujo alternativo 7b del caso de uso *CU06- Entrena red*.
Postcondición	
Flujo principal	1. El sistema solicita al usuario introducir el número de iteraciones extras a realizar. 2. El usuario introduce un valor. 3. El sistema comprueba que el valor es válido y reanuda el entrenamiento. 4. El experimento termina y el sistema muestra los nuevos resultados obtenidos. 5. El usuario cierra la pantalla de visualización de resultados y el caso de uso termina.
Flujo alternativo	3a. El sistema comprueba que el valor introducir no es válido. Muestra un mensaje de error y regresa al paso 1. 5a. El usuario selecciona la opción de guardar modelo. Se inicia el caso de uso *CU07: Guarda Modelo*. 5b. El usuario selecciona la opción de entrenar más iteraciones. Regresa al paso 1.
Excepciones	2, 3. El usuario cancela el entrenamiento de más iteraciones. El caso de uso termina sin efecto.

Cuadro 12: Descripción de CU08: Entrena más iteraciones

10.4.10. CU09: Clasifica documentos

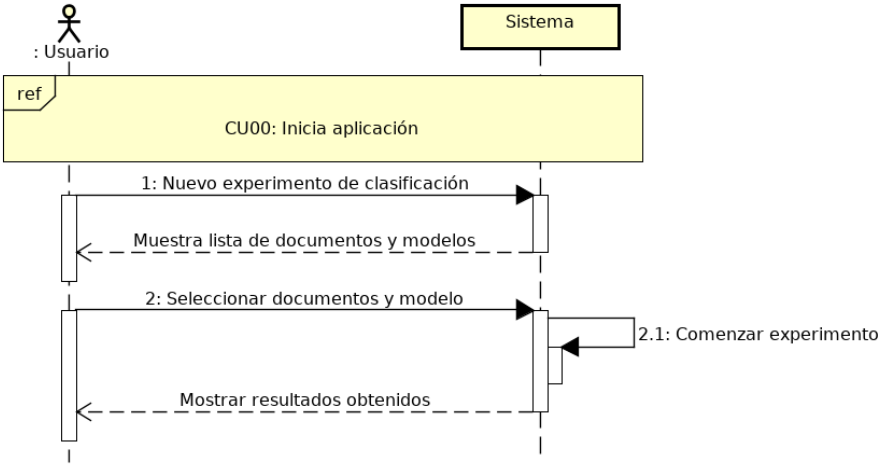

Figura 19: Diagrama de secuencia para CU09: Clasifica documentos.

Nombre	CU09 - Clasifica documentos.
Resumen	El sistema debe permitir utilizar los modelos almacenados para la clasificación de documentos en el sistema.
Actor	Usuario.
Precondición	Existen documentos y modelos almacenados en el sistema. El sistema ha completado el caso de uso *CU00*.
Postcondición	
Flujo principal	1. El usuario selecciona la opción de Nuevo Experimento de Clasificación. 2. El sistema muestra la lista de documentos y modelos en el sistema. 3. El usuario selecciona los documentos y el modelo a utilizar. 4. El sistema procesa los documentos seleccionados y comienza la clasificación. 5. La clasificación termina y el sistema muestra los resultados obtenidos. 6. El usuario cierra la pantalla de visualización de resultados y el caso de uso termina.
Flujo alternativo	6a. El usuario selecciona uno de los ficheros para visualizar. Se inicia el caso de uso *CU05: Visualiza documento*.
Excepciones	3,4. El usuario cancela la clasificación. El caso de uso termina sin efecto.

Cuadro 13: Descripción de CU09: Clasifica documentos

10.5. Modelo de Dominio

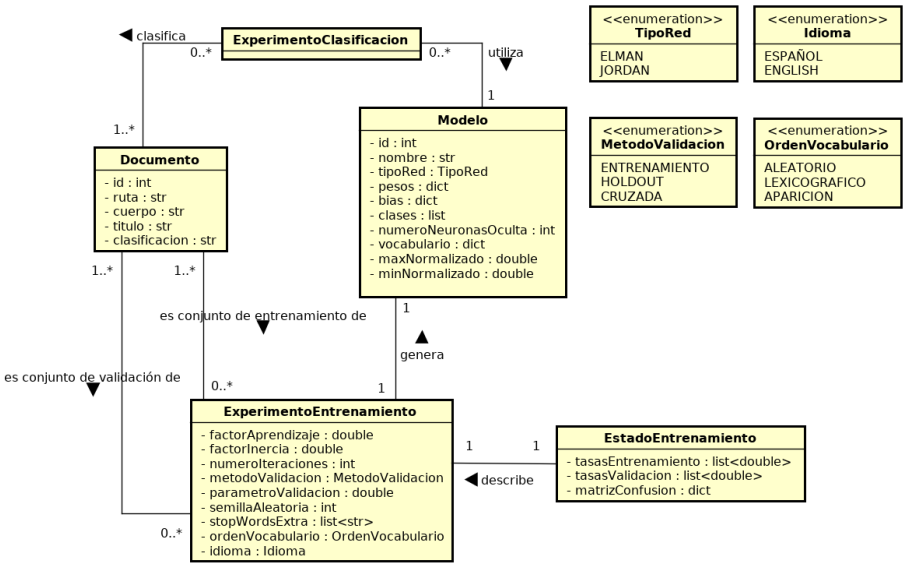

Figura 20: Diagrama de dominio.

11. Diseño

11.1. Arquitectura

Se pasa a describir la arquitectura elegida para la aplicación, así como los diagramas detallados de la descomposición de la aplicación, y los diagramas de secuencia detallados para cada caso de uso.

11.1.1. Patrón MVC Pasivo.

Se ha elegido el patrón Modelo-Vista-Controlador porque es una manera sencilla de separar los datos y la lógica de negocio de su presentación al usuario. El esquema del patrón MVC usado es el siguiente:

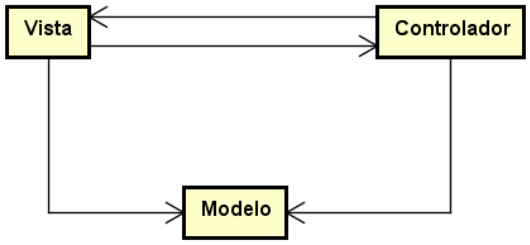

Figura 21: Descripción del patrón MVC

Los elementos de este patrón y sus funciones son:

- **Vista**. Parte de la aplicación que muestra información al usuario, utiliza el Modelo para transmitir y recibir información con el Controlador.
- **Controlador**. Parte de la aplicación que responde a los eventos generados por el usuario a través de la vista, implementando los casos de uso de la aplicación.
- **Modelo**. Representa los datos de la aplicación.

Este patrón se denomina MVC Pasivo porque el Controlador manipula el Modelo y manda a la Vista su actualización. Por tanto el Modelo es independiente de la Vista y del Controlador, y solamente representa los datos, no se encarga de notificar cambios.

Se pasa a definir el bucle básico de funcionamiento de este patrón:

1. El **usuario** interactúa con la **vista** mediante los elementos clásicos de una interfaz, como son botones, campos de texto, enlaces, etc. la **vista** encapsula los datos recibidos en el **modelo**.

2. El **controlador** recibe el evento resultante de la interacción del **usuario** con la **vista**, accede a los elementos del **modelo** apropiados para responder a la petición del **usuario** y los modifica, notificando a la **vista** al terminar.

3. El **usuario** ve el resultado de su acción en la **vista** y esta queda a la espera de la próxima interacción con el usuario.

11.1.2. Patrón DataAccessObject

Para implementar la permanencia de la aplicación se ha utilizado una variación del patrón DataAccessObject o DAO. En el patrón original, se utiliza una interfaz DAO que adapta cualquier tipo de almacenamiento a un uso genérico por el resto de la aplicación.

En esta aplicación no se utiliza una interfaz, sino una clase normal, que se encarga de encapsular los datos obtenidos de las consultas realizadas por las clases internas DocumentoDAO y ModeloDAO en objetos del Modelo.

Esto permite que cualquier cambio en el método de almacenamiento afecte solo a las clases internas del paquete, pudiendo mantener DAO como punto de acceso al paquete por el resto de la aplicación.

11.1.3. Decomposition Style

El diagrama *Decomposition Style* permite mostrar cómo se descompone la aplicación en subsistemas, paquetes, y clases.

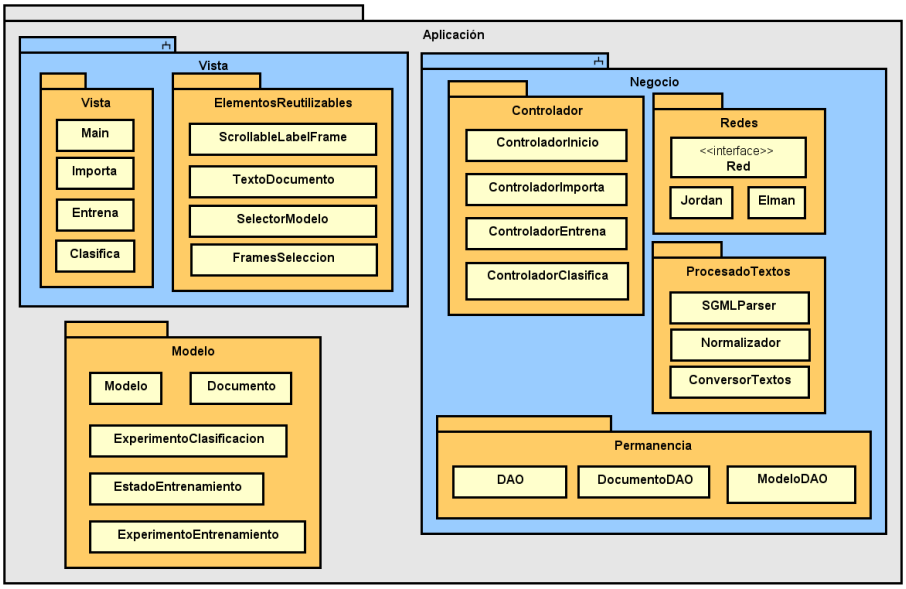

Figura 22: Diagrama Decomposition Style de la aplicación

11.1.4. Uses Style

Un diagrama *Uses Style* permite mostrar las dependencias entre módulos. Las relaciones representan que un módulo *utiliza* a otro.

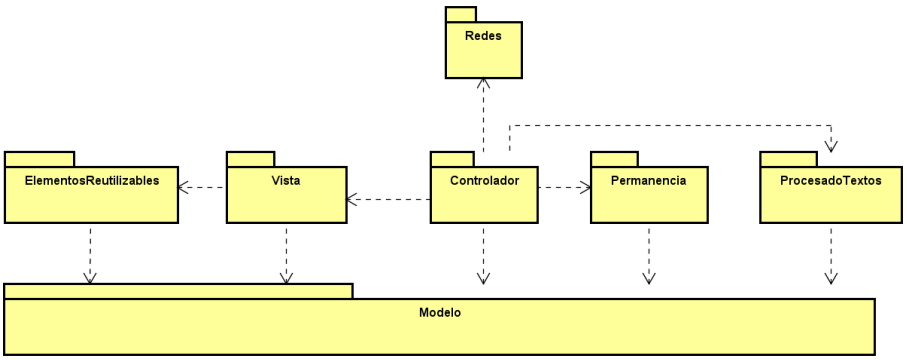

Figura 23: Diagrama Uses Style de la aplicación.

11.1.5. Generalization Style

Un diagrama *Generalization Style* expresa relaciones de generalización y especificidad entre clases.

En el caso de nuestra aplicación, solo existe una relación de este tipo, la que forman la interfaz *Red*, y las dos clases que la implementan, *Elman* y *Jordan*:

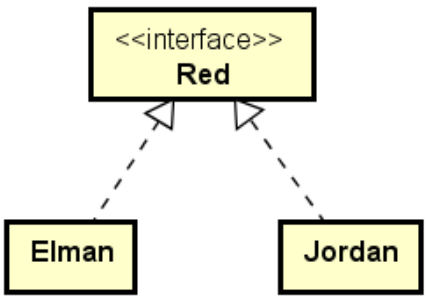

Figura 24: Diagrama Generalization Style de las clases Elman y Jordan, y la interfaz Red.

Este diseño permite añadir nuevos tipos de red neuronal a la aplicación facilmente, siempre que implementen dicha interfaz.

11.2. Descripción de las clases

En esta sección se pasa a hablar en profundidad de las clases en las que se organiza la aplicación.

11.2.1. Paquete Controlador

El paquete Controlador se encarga de la unión entre la lógica de negocio y la vista. Contiene las siguientes clases:

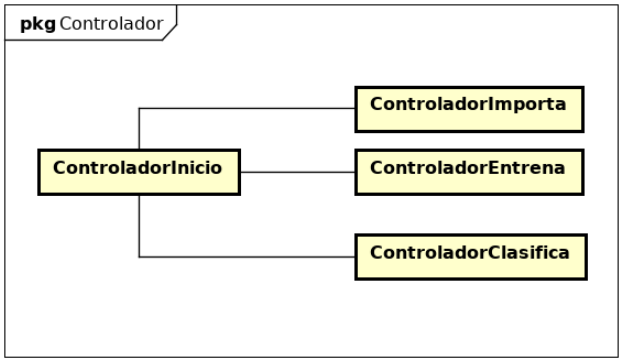

Figura 25: Diagrama de las relaciones entre clases en el paquete Controlador

ControladorInicio Implementa los casos de uso CU00 - Inicia la aplicación y CU04 - Elimina documentos.

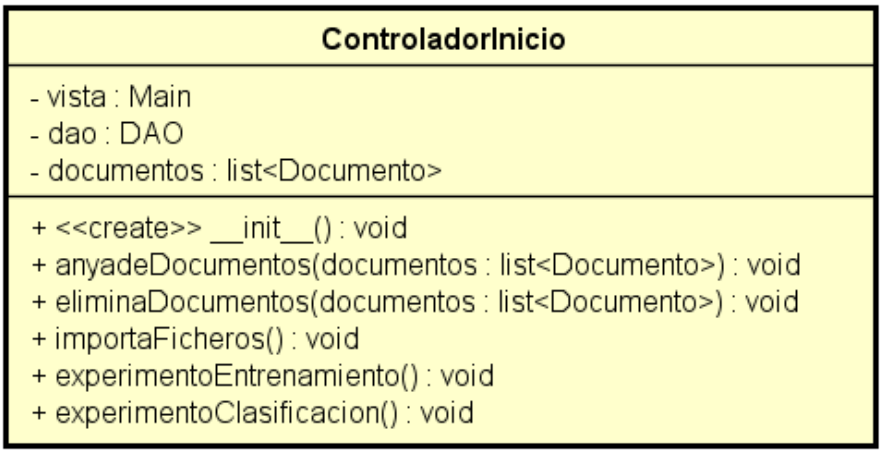

Figura 26: Diagrama de la clase ControladorInicio en el paquete Controlador

ControladorImporta Implementa los casos de uso CU01 - Importa ficheros estructurados, CU02 - Importa ficheros de texto plano y CU03 - Edita atributos documentos.

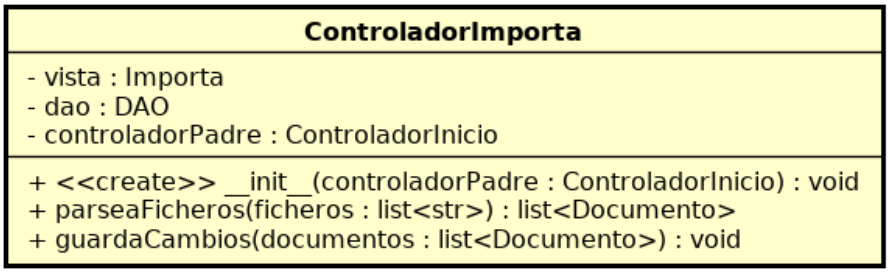

Figura 27: Diagrama de la clase ControladorImporta en el paquete Controlador

ControladorEntrena Implementa los casos de uso CU06 - Entrena red, CU07 - Guarda modelo y CU08 - Entrena más epocas.

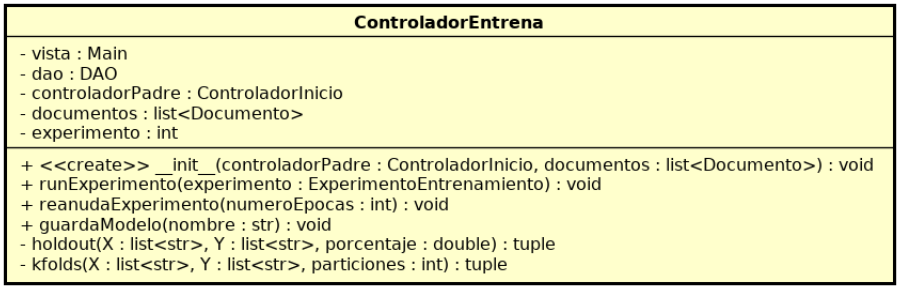

Figura 28: Diagrama de la clase ControladorEntrena en el paquete Controlador

ControladorClasifica Implementa el caso de uso CU09 - Clasifica documentos.

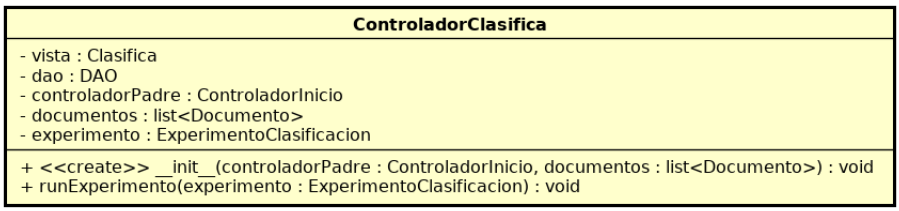

Figura 29: Diagrama de la clase ControladorClasifica en el paquete Controlador

11.2.2. Paquete Vista

El paquete Vista contiene las clases de la interfaz gráfica que permiten interactuar al usuario con la aplicación.

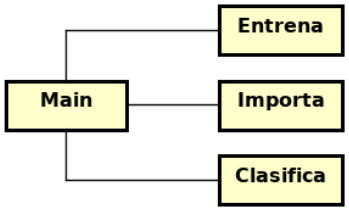

Figura 30: Diagrama de las relaciones entre clases en el paquete Vista

Main La clase Main implementa la pantalla principal, con una lista de los documentos en el sistema, la pantalla de contenido del fichero, y un menú que permite acceder al resto de funcionalidad. Esta pantalla se mantiene visible durante toda la aplicación.

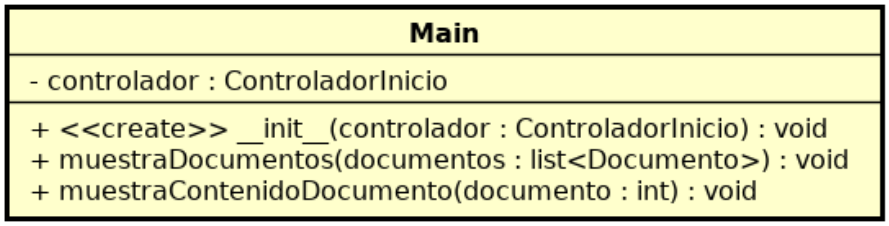

Figura 31: Diagrama de la clase Main en el paquete Vista

Importa La clase Importa implementa una ventana de selección de ficheros del sistema, la pantalla que muestra los documentos resultados del parseo de los ficheros y permite editar su título y clasificación, y la pantalla de contenido del documento.

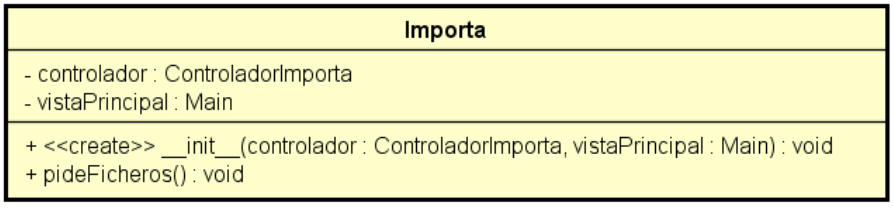

Figura 32: Diagrama de la clase Importa en el paquete Vista

Entrena La clase Entrena implementa la ventana de selección de documentos y parámetros para un experimento de entrenamiento, y la pantalla de resultado final.

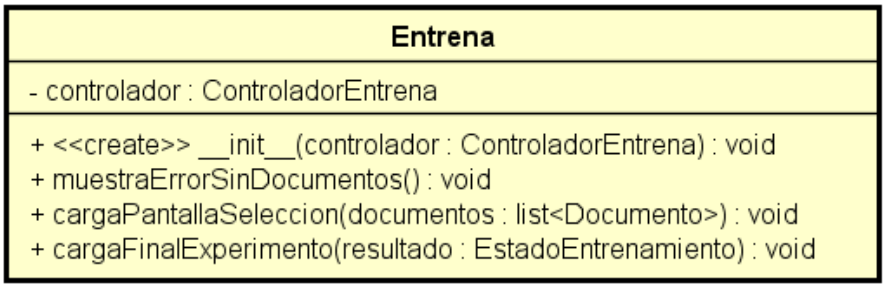

Figura 33: Diagrama de la clase Entrena en el paquete Vista

Clasifica La clase Clasifica implementa la ventana de selección de documentos y modelo para un experimento de clasificación, la ventana de resultado, y una ventana que muestra el contenido de un documento.

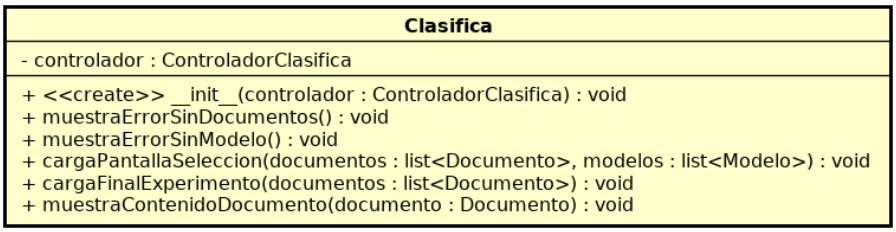

Figura 34: Diagrama de la clase Clasifica en el paquete Vista

11.2.3. Paquete ElementosReutilizables

Este paquete contiene clases que se utilizan por la vista, formadas por varios elementos gráficos. Su objetivo es evitar la repetición de código.

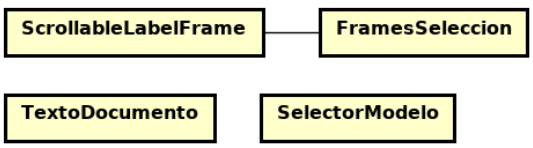

Figura 35: Diagrama de las relaciones entre clases en el paquete ElementosReutilizables

ScrollableLabelFrame Esta clase implementa una lista que muestra detalles de los documentos, y genera una barra de scroll cuando la lista es demasiado larga para caber en pantalla.

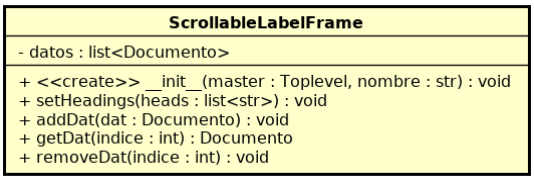

Figura 36: Diagrama de la clase ScrollableLabelFrame

TextoDocumento Esta clase implementa una ventana con campos para el título, clasificación, y contenido de un documento.

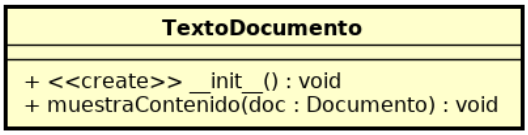

Figura 37: Diagrama de la clase TextoDocumento

SelectorModelo Esta clase implementa una lista de modelos y una lista de atributos que se actualiza al seleccionar un modelo.

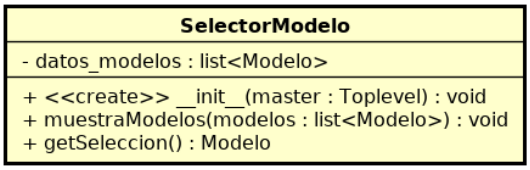

Figura 38: Diagrama de la clase SelectorModelo

FramesSeleccion Esta clase implementa dos instancias de ScrollableLabel-Frame, una de las cuales actúa como lista de documentos no seleccionados, y la otra como lista de documentos seleccionados, y permite pasar elementos de una a otra.

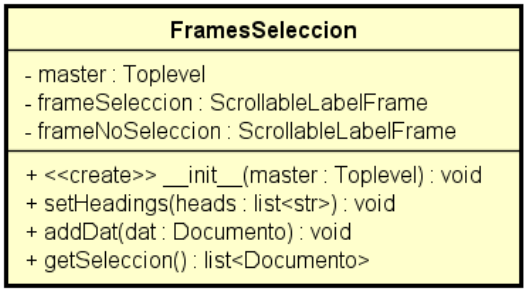

Figura 39: Diagrama de la clase FramesSeleccion

11.2.4. Paquete Modelo

Este paquete implementa los objetos que definen el dominio de la aplicación.

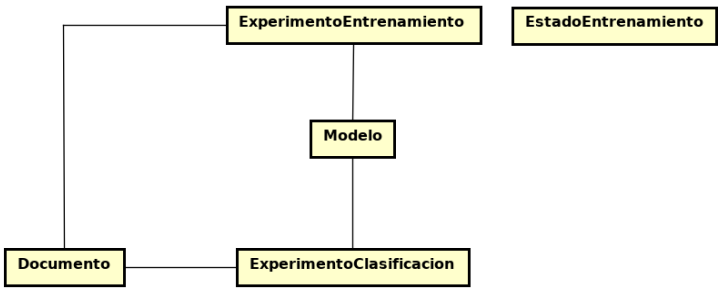

Figura 40: Diagrama de las relaciones entre clases en el paquete Modelo

Documento La clase Documento representa un documento de texto con cuerpo, título, clasificación, un identificador único, y la ruta del fichero del que se ha obtenido.

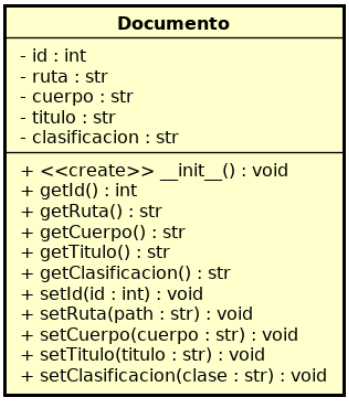

Figura 41: Diagrama de la clase Documento en el paquete Modelo

Modelo La clase Modelo representa el conjunto de informacion necesaria para recrear una red neuronal para clasificado de textos.

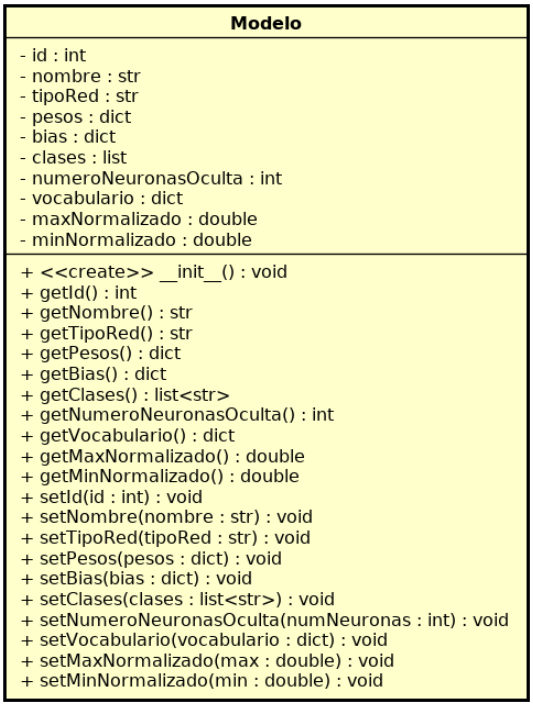

Figura 42: Diagrama de la clase Modelo en el paquete Modelo

ExperimentoEntrenamiento La clase ExperimentoEntrenamiento representa los datos necesarios para crear un experimento cuya finalidad es conseguir un Modelo a partir de Documentos.

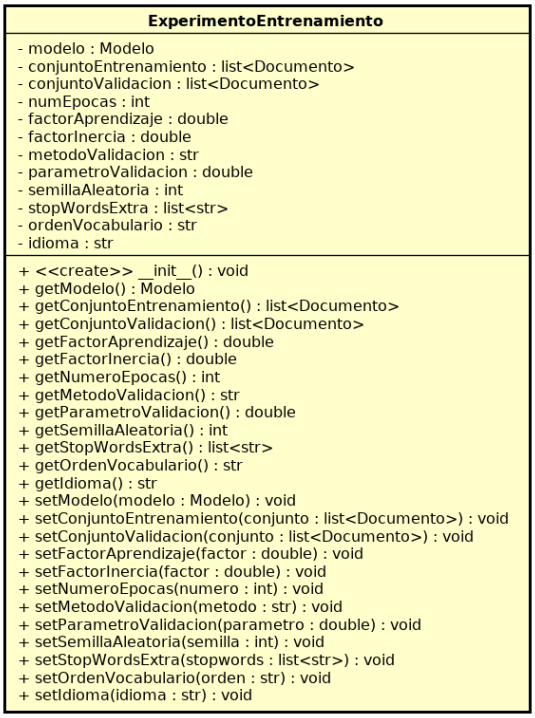

Figura 43: Diagrama de la clase ExperimentoEntrenamiento en el paquete Modelo

EstadoEntrenamiento La clase EstadoEntrenamiento representa los datos necesarios para informar al usuario del resultado de un experimento de entrenamiento.

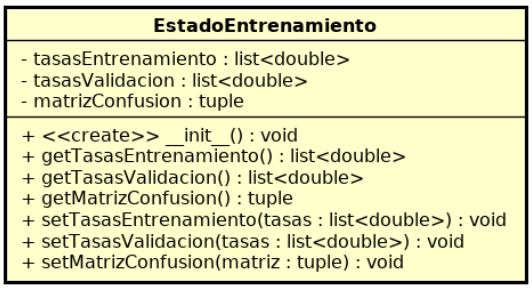

Figura 44: Diagrama de la clase EstadoEntrenamiento en el paquete Modelo

ExperimentoClasificacion La clase ExperimentoClasificacion representa los datos necesarios para obtener la clase de un conjunto de Documentos a partir de un Modelo.

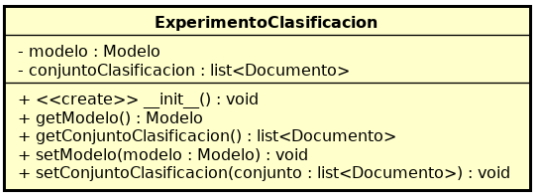

Figura 45: Diagrama de la clase ExperimentoClasificacion en el paquete Modelo

11.2.5. Paquete Permanencia

Este paquete implementa el almacenamiento persistente de la aplicación.

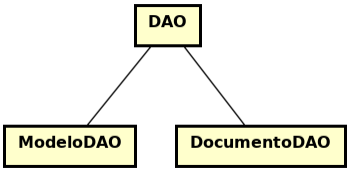

Figura 46: Diagrama de las relaciones entre clases en el paquete Permanencia

DAO La clase DAO implementa la conexión con la base de datos y convierte los datos crudos obtenidos de la base de datos a clases del Dominio de la aplicación.

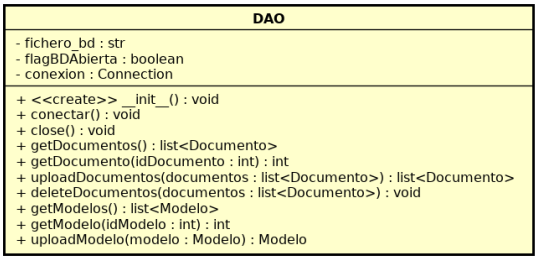

Figura 47: Diagrama de la clase DAO en el paquete Permanencia

DocumentoDAO La clase DocumentoDAO implementa las lecturas de la base de datos de la tabla DOCUMENTO.

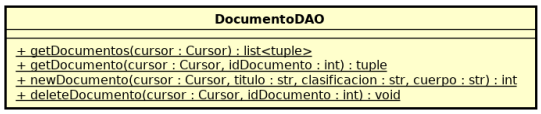

Figura 48: Diagrama de la clase DocumentoDAO en el paquete Permanencia

ModeloDAO La clase ModeloDAO implementa las lecturas de la base de datos de la tabla MODELO, PESO, VOCABULARIO, CLASE, y TIPORED.

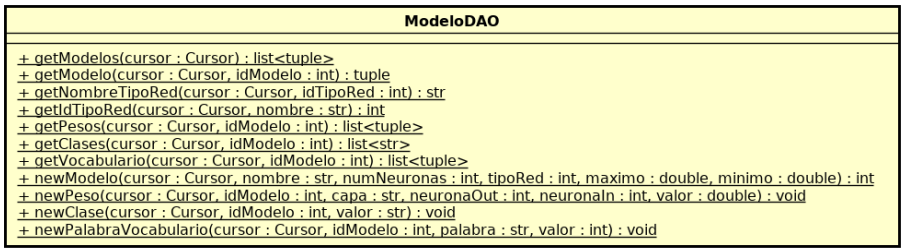

Figura 49: Diagrama de la clase ModeloDAO en el paquete Permanencia

11.2.6. Paquete ProcesadoTextos

Este paquete implementa todas las funcionalidades necesarias para extraer documentos de ficheros y convertirlos a un formato introducible a una red neuronal.

Figura 50: Diagrama de las clases del paquete ProcesadoTextos

SGMLParser Esta clase extrae documentos de ficheros estructurados.

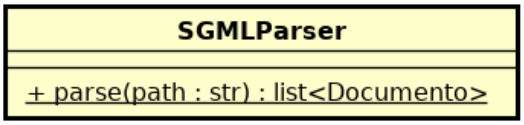

Figura 51: Diagrama de la clase SGMLParser en el paquete ProcesadoTextos

ConversorTextos Esta clase implementa un conversor de listas de textos complejos en un vector numérico, realizando una previa reducción de la complejidad. Véase la sección 9.

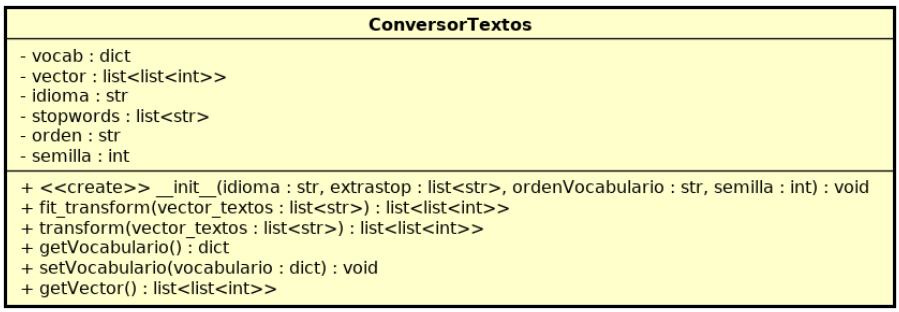

Figura 52: Diagrama de la clase ConversorTextos en el paquete ProcesadoTextos

Normalizador Esta clase implementa un conversor de listas de vectores numéricos en un rango 0-∞ al rango 0,1-0,9. Véase la sección 9.6.

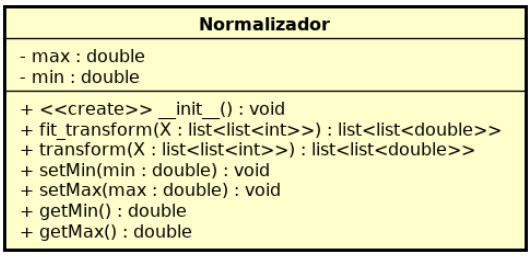

Figura 53: Diagrama de la clase Normalizador en el paquete ProcesadoTextos

11.2.7. Paquete Redes

Este paquete implementa redes neuronales recurrentes de tipos Elman y Jordan, y establece una interfaz para las redes del sistema.

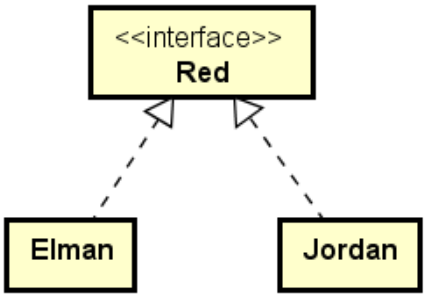

Figura 54: Diagrama de las relaciones entre clases en el paquete Redes.

Red Interfaz que debe ser implementada por todas las redes neuronales que se quieran incorporar en el sistema.

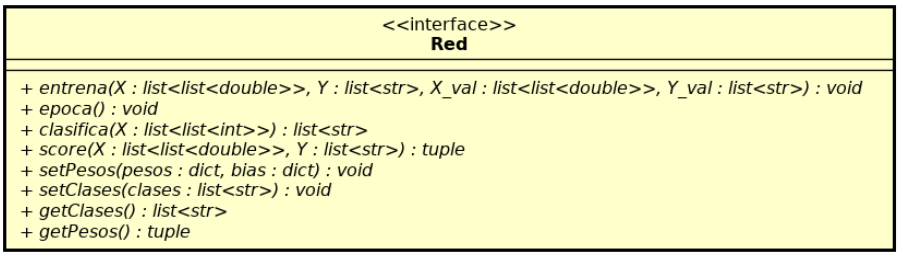

Figura 55: Diagrama de la interfaz Red del paquete Redes.

Elman Clase que implementa la red neuronal recurrente Elman. Véase la sección 4.1.

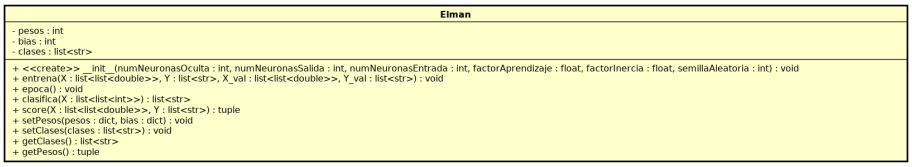

Figura 56: Diagrama de la clase Elman del paquete Redes.

Jordan Clase que implementa la red neuronal recurrente Jordan. Véase la sección 4.2.

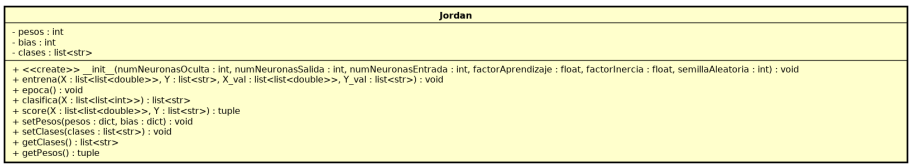

Figura 57: Diagrama de la clase Jordan del paquete Redes.

11.3. Diagramas de Secuencia

11.3.1. CU00: Inicia aplicación

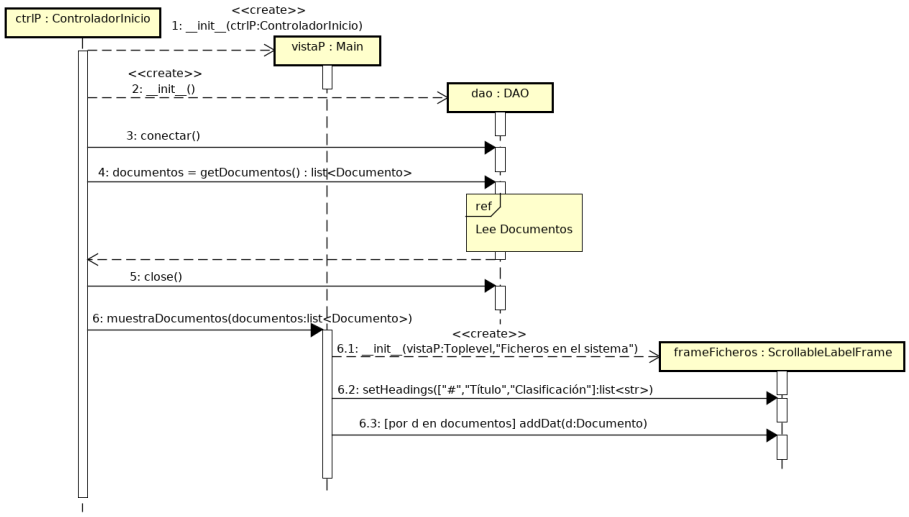

Figura 58: Diagrama de secuencia en diseño de CU00: Inicia aplicación

El diagrama referenciado se encuentra en el *Anexo 1*: Figura 73.

11.3.2. CU01: Importa ficheros estructurados

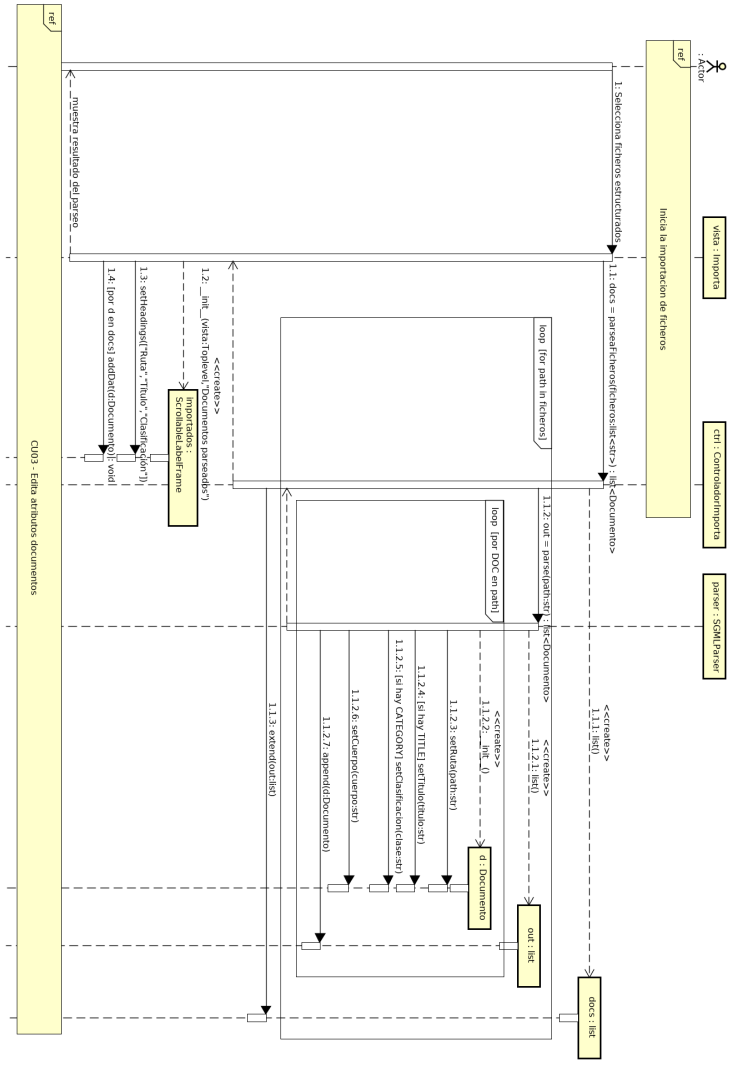

Figura 59: Diagrama de secuencia en diseño de CU01: Importa ficheros estructurados

El diagrama referenciado se encuentra en el *Anexo 1*: Figura 74.

11.3.3. CU02: Importa ficheros de texto plano

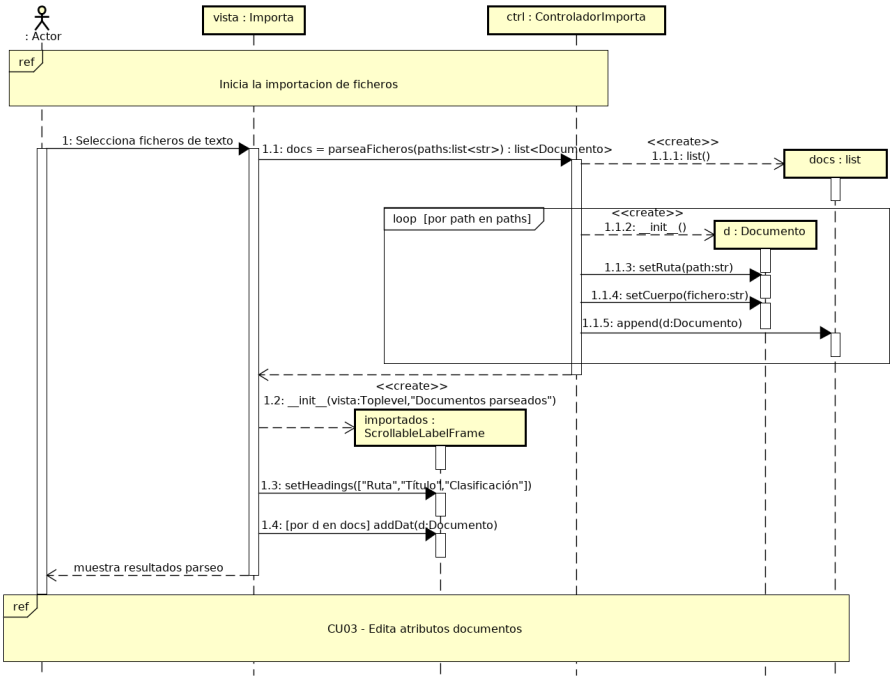

Figura 60: Diagrama de secuencia en diseño de CU02: Importa ficheros de texto plano

El diagrama referenciado se encuentra en el *Anexo 1*: Figura 74.

11.3.4. CU03: Edita atributos documentos

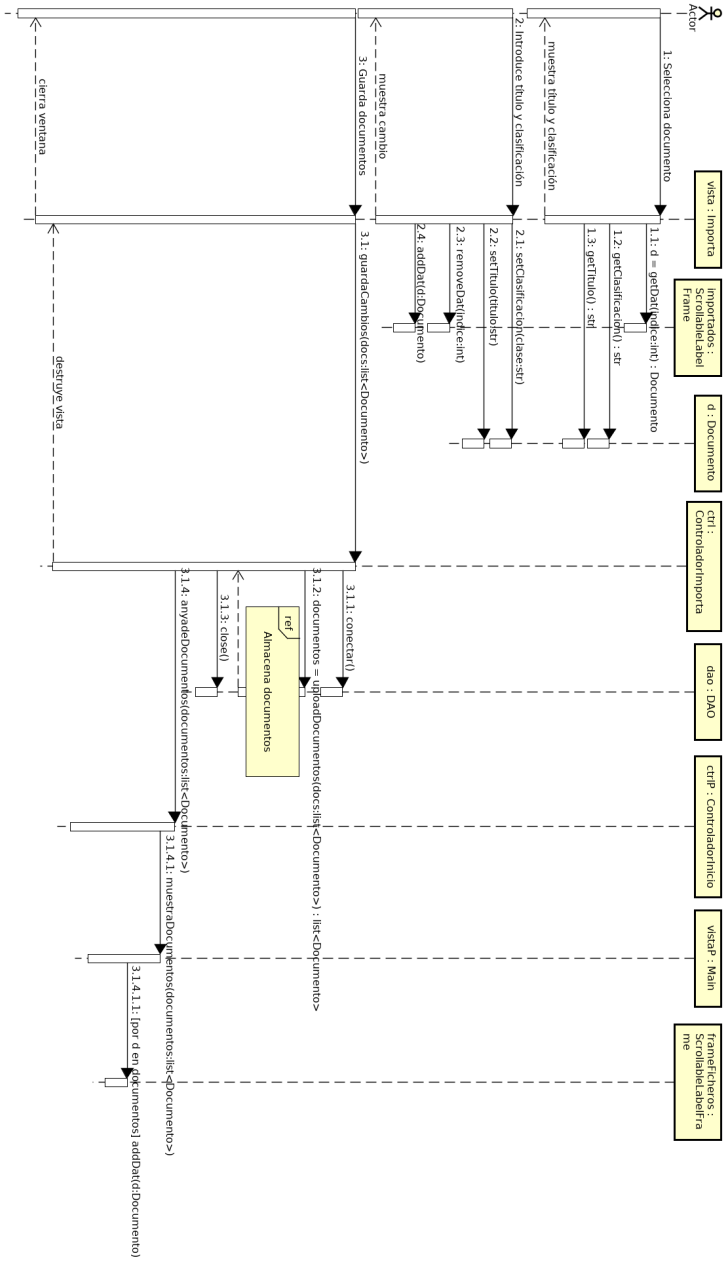

Figura 61: Diagrama de secuencia en diseño de CU03: Edita atributos documentos

El diagrama referenciado se encuentra en el *Anexo 1*: Figura 75.

11.3.5. CU04: Elimina documentos

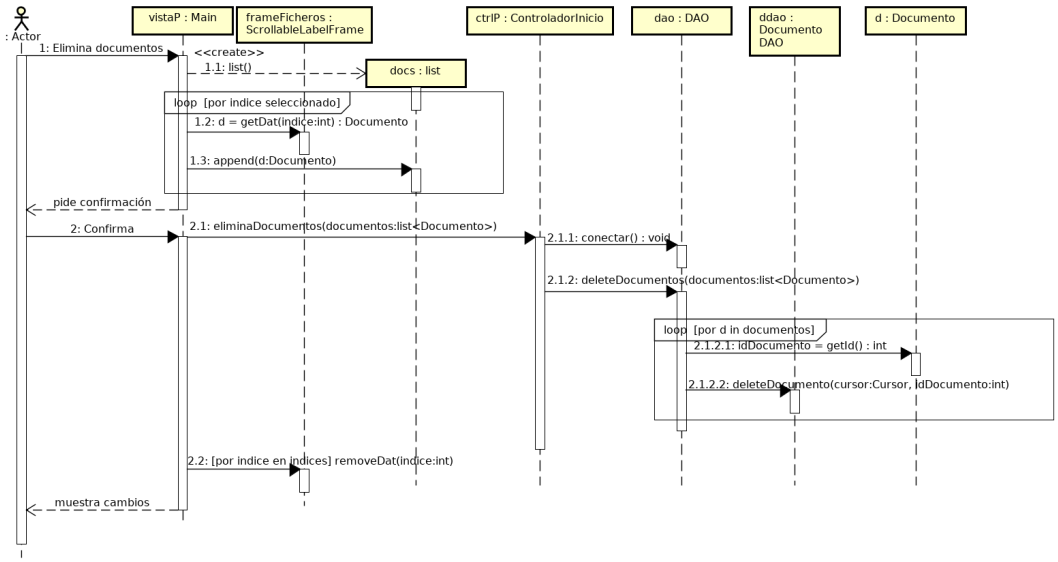

Figura 62: Diagrama de secuencia en diseño de CU04: Elimina documentos

11.3.6. CU05: Visualiza documento

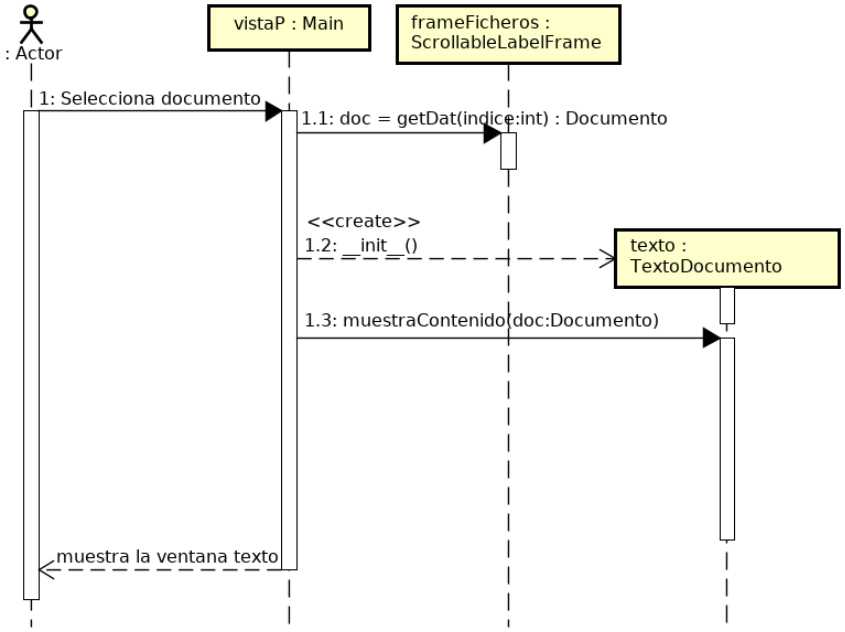

Figura 63: Diagrama de secuencia en diseño de CU05: Visualiza documento

11.3.7. CU06: Entrena red

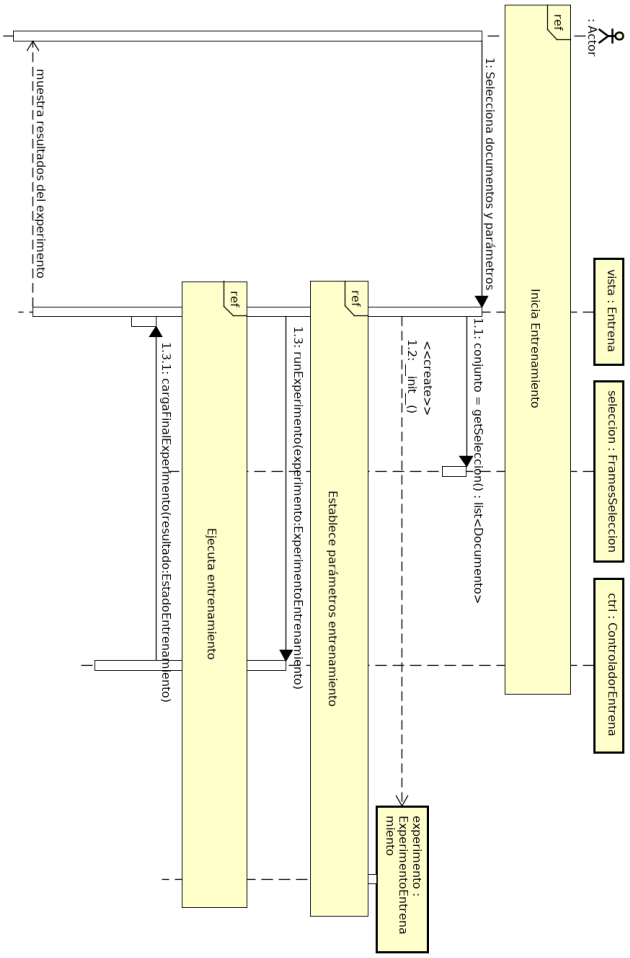

Figura 64: Diagrama de secuencia en diseño de CU06: Entrena red

Los diagramas referenciados se encuentran en el *Anexo 1*: Figura 76 y Figura 77. Debido a su importancia en el caso de uso, Ejecuta entrenamiento se incluye a continuación.

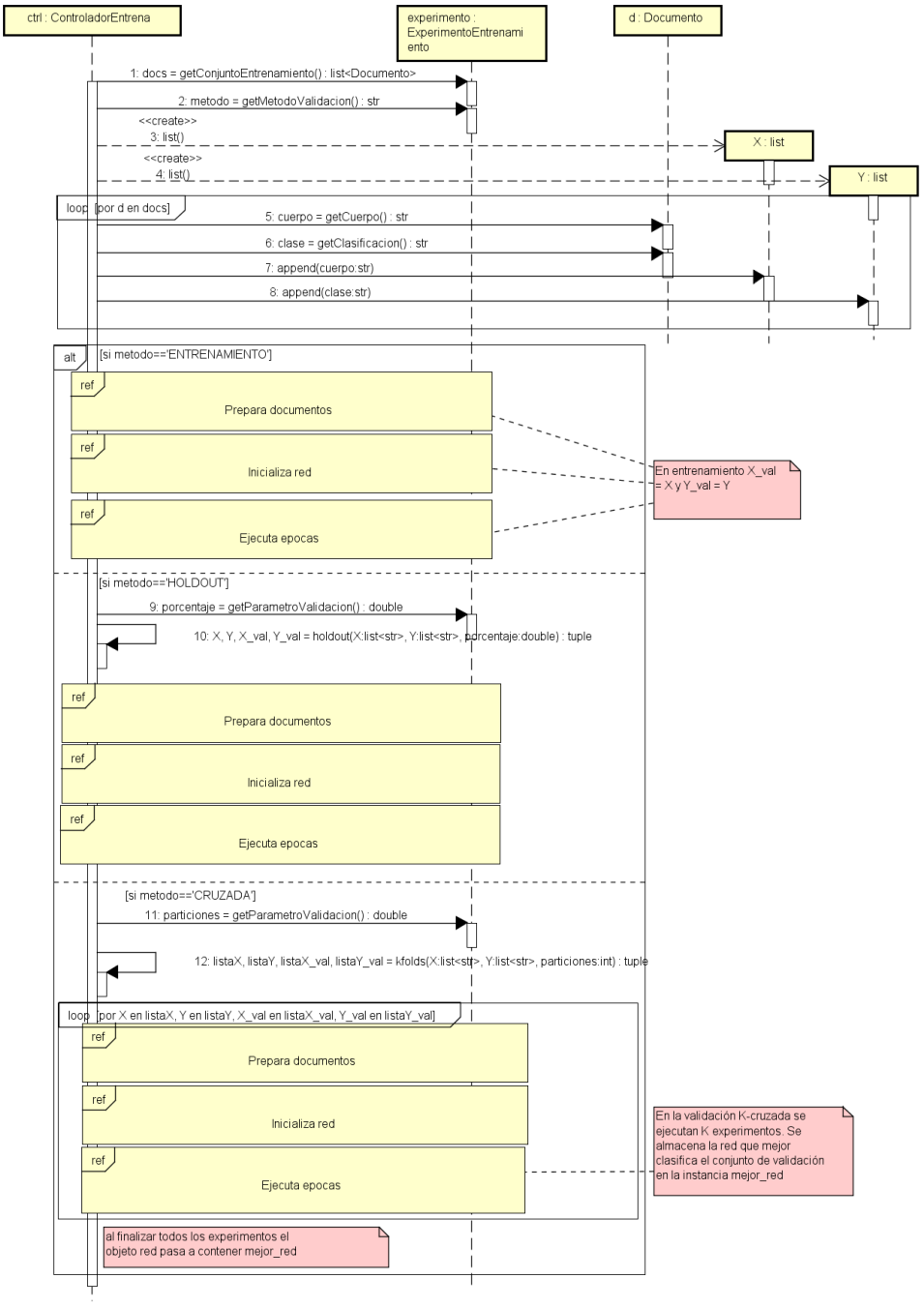

Figura 65: Diagrama de secuencia en diseño de Ejecuta entrenamiento

Los diagramas referenciados se encuentran en el *Anexo 1*: Figura 78, Figura 79 y Figura 80.

11.3.8. CU07: Guarda Modelo

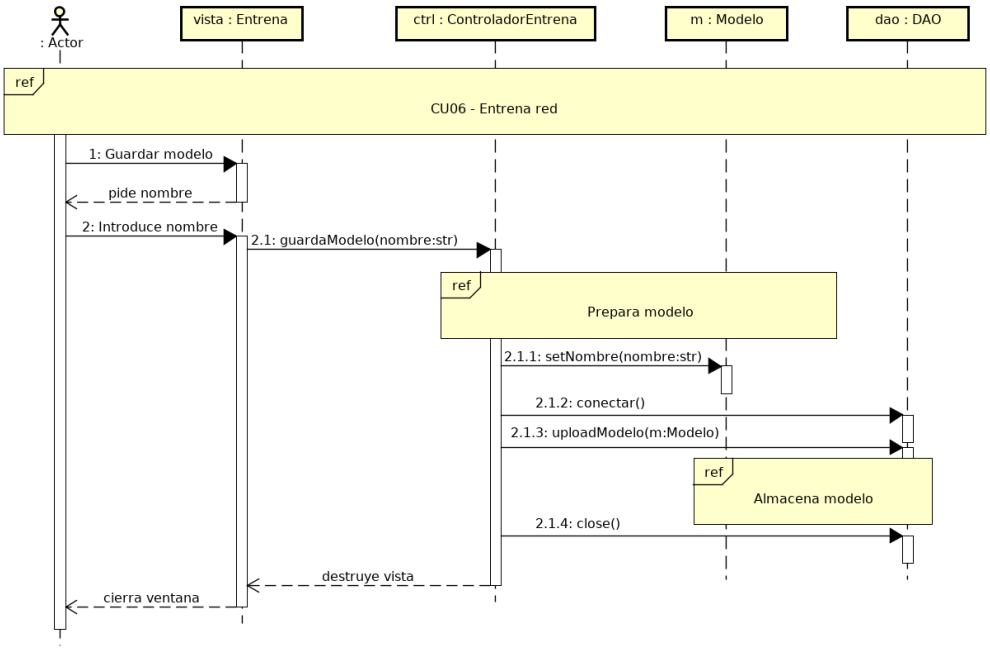

Figura 66: Diagrama de secuencia en diseño de CU07: Guarda Modelo.

Los diagramas referenciados se encuentran en el *Anexo 1*: Figura 81 y Figura 82.

11.3.9. CU08: Entrena más iteraciones

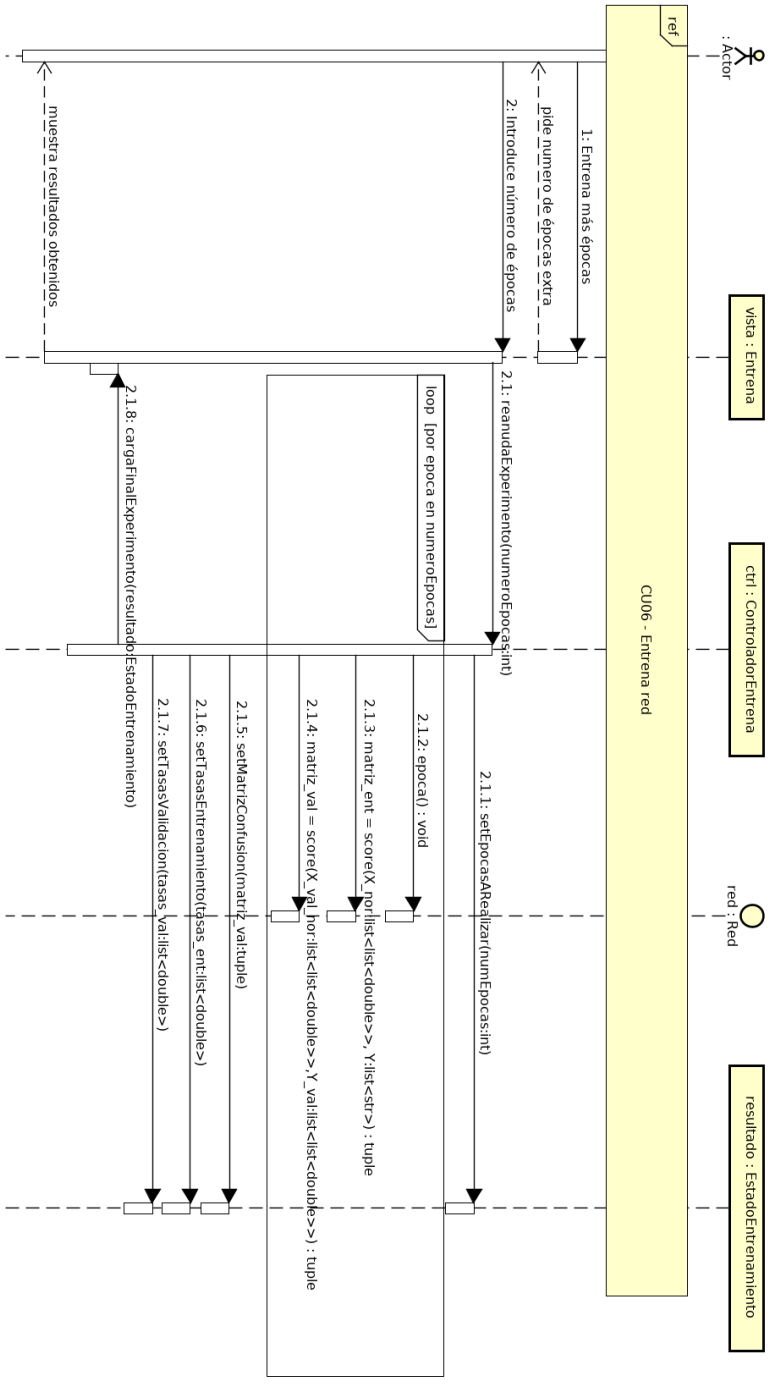

Figura 67: Diagrama de secuencia en diseño de CU08.

11.3.10. CU09: Clasifica documentos

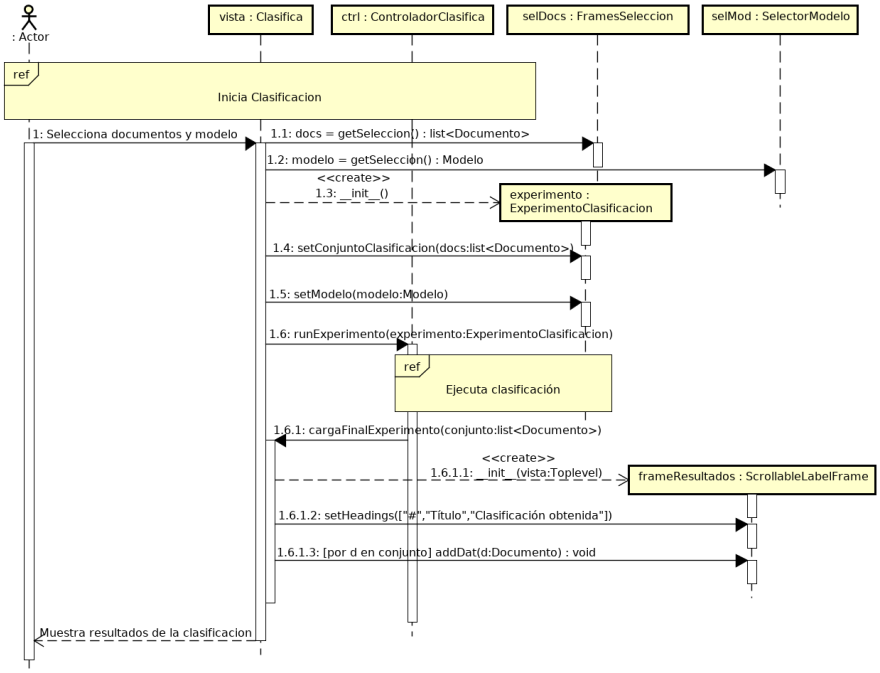

Figura 68: Diagrama de secuencia en diseño de CU09.

El diagrama referenciado se encuentra en el *Anexo 1*: Figura 83. Debido a su importancia en el caso de uso, Ejecuta clasificación se incluye a continuación.

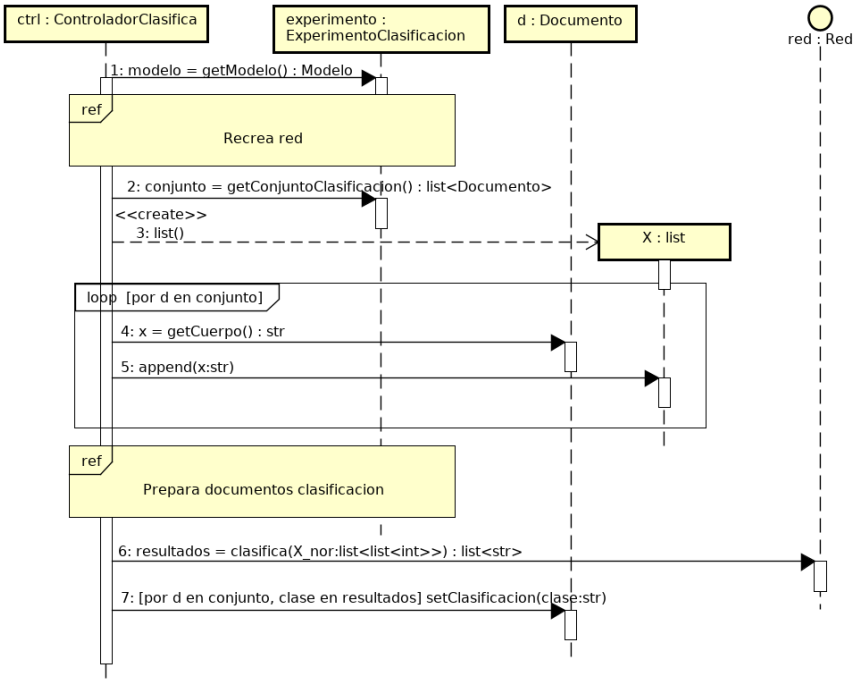

Figura 69: Diagrama de secuencia en diseño de Ejecuta clasificación.

Los diagramas referenciados se encuentran en el *Anexo 1*: Figura 85 y Figura 86.

11.4. Base de Datos

En esta sección se habla de la estructura utilizada para el sistema de almacenamiento persistente de la aplicación.

Se han utilizado los tipos propios de SQLite: INTEGER, REAL, y TEXT, que representan un entero, un real, y una cadena de caracteres de tamaño arbitrario respectivamente.

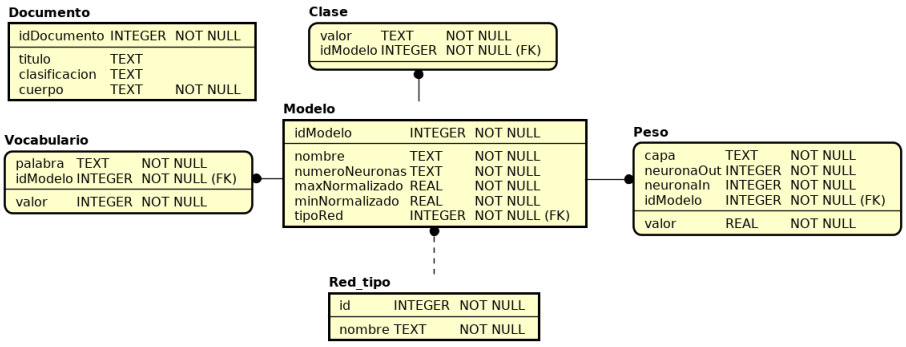

Figura 70: Diagrama Entidad-Relación de la base de datos.

Como se puede apreciar, la representación de **Modelo** está formado por atributos simples, y una serie de atributos que se han separado en tablas por su complejidad. Se pasa a explicarlos:

- **Clase.** Almacena las etiquetas de clase para el modelo. Referencia al modelo al que pertenece. Su clave primaria es la etiqueta de clase y la foreign key del modelo.
- **Vocabulario.** Almacena una asociación palabra-a-entero que permite convertir el texto complejo en vectores numéricos entendibles por la red neuronal. Referencia al modelo que pertenece. Su clave primaria es la palabra y la foreign key del modelo.
- **Peso.** Almacena una posición de la matriz de pesos. Para la facilidad a la hora de almacenar y de extraer la matriz de pesos se cuenta con tres atributos identificadores, a parte de la foreign key del modelo al que pertenece. El atributo *capa* es un TEXT que indica la clave del diccionario que almacena los pesos. El atributo *neuronaOut* indica el segundo subíndice del peso w_{ij}. El atributo *neuronaIn* representa el primer subíndice.

Además **Modelo** referencia a **Red_tipo**, que actúa como enumeration de los tipos posibles de red. Al crear la base de datos se añaden las filas (1,'ELMAN'), (2,'JORDAN') a esta tabla.

La tabla **Documento** no referencia a ninguna otra tabla, ni es referenciado, ya que no es necesario almacenar ninguna relación entre **Documento** y **Modelo**.

12. Implementación

En esta sección se describen ciertas decisiones tomadas a la hora de implementar, así como las tecnologías utilizadas.

12.1. Tecnologías utilizadas

La aplicación se ha desarrollado en su totalidad en *Python 3.6.5*. La razón por la que se ha elegido este lenguaje en particular, es su popularidad en entornos científicos, lo que se traduce en la disponibilidad de paquetes de Aprendizaje Automático. Es compatible con Windows, Linux, y Mac OS, lo que permite un desarrollo multiplataforma por defecto y está registrado bajo licencias que permiten su uso libre.

Una de las desventajas del desarrollo en *Python* es su dificultad de distribución, ya que, al ser un lenguaje no compilado, requiere la instalación de un intérprete. El intérprete de *Python* viene por defecto en la mayoría de distribuciones de Linux, pero no en Windows. Para esta plataforma se ha optado por incluir en la instalación del programa una versión portable conocida como *WinPython*.

Otro inconveniente es la incompatibilidad entre versiones de *Python*, requiriendo esta aplicación un intérprete de *Python 3*. En Windows esto se soluciona con el método de distribución mencionado anteriormente, pero en Linux requiere incluir como requisito de instalación que el intérprete de *Python* esté actualizado a la version 3.

Los paquetes desarrollados por terceros utilizados en esta aplicación son:

- *Tkinter*: Es el más utilizado para realizar interfaces gráficas en Python. *Tkinter* está basado en el lenguaje *Tcl* y su toolkit gráfico *Tk*. Ofrece las ventajas de ser compatible con Windows, Linux, y Mac OS, y su popularidad implica que existe un gran volumen de documentación online. Se distribuye bajo la licencia Python, que permite su uso libre.

- *NumPy*. De este paquete se utilizan sus capacidades para crear y operar sobre arrays multidimensionales de manera más eficiente que los objetos y funciones disponibles por defecto en Python. Esto permite que el módulo

Redes se ejecute con la mayor eficiencia posible. Se distribuye bajo la licencia BSD, que permite su uso libre.

- *Scikit-Learn*: Proporciona numerosas herramientas para el desarrollo de aplicaciones de Aprendizaje Automático. En esta aplicación se utilizan los módulos de selección de modelos, para la separación de los conjuntos de datos en entrenamiento y validación. Se distribuye bajo la licencia BSD, que permite su uso libre.

- *NLTK*: Este paquete es uno de los más utilizados para el análisis de lenguaje natural, en particular, se utilizan sus recursos de stopwords en español e inglés y su implementación del algoritmo SnowballStemmer para dichos idiomas. Se distribuye bajo la licencia Apache License 2.0, que permite su uso libre.

- *Matplotlib*: Proporciona herramientas para realizar gráficos y exportarlos a distintos formatos, así como incrustarlos en *Tkinter*. Se distribuye bajo la licencia Matplotlib, basada en la licencia Python, que permite su uso libre.

- *tabulate*: Permite obtener tablas con alineamiento inteligente a partir de distintas estructuras de datos. Se distribuye bajo la licencia MIT, que permite su uso libre.

- *BeautifulSoup4*: Proporciona parseadores de XML y HTML, que se utilizan para parsear los ficheros estructurados importados a la aplicación. Se distribuye bajo la licencia MIT, que permite su uso libre.

Como ya se ha indicado en los requisitos no funcionales, (véase Cuadro 2), el almacenamiento persistente de la aplicación se ha realizado mediante el motor *SQLite*, de dominio público, a través de la API *SQLite3*, incorporada en *Python 3.6.5* por defecto.

Se ha elegido esta herramienta, porque no requiere conexión a un servidor, sino que utiliza un fichero almacenado en el disco. Esto tiene como ventajas que no requiere una configuración compleja, como la que conlleva un servidor. Como desventajas, la memoria en disco es susceptible a la corrupción por aplicaciones en el cliente, lo cual no ocurre en un servidor.

Otra razón por la que se ha elegido *SQLite* es que funciona en cualquier sistema operativo, y no tiene ninguna dependencia, excepto bibliotecas estándard de C, lo que sumado a la utilización de *Python 3.6.5*, da compatibilidad a la aplicación.

12.2. Paralelismo

El principal cambio respecto al diseño inicial de la aplicación, es el de la inclusión de paralelismo con hilos. Esto se ha añadido debido al funcionamiento de la biblioteca *Tkinter*, utilizada para realizar la interfaz gráfica.

Esta biblioteca bloquea todos los eventos gráficos, cuando un proceso requiere mucho tiempo para completarse, lo que se traduce, en el sistema operativo, detectando erróneamente que la aplicación ha dejado de funcionar.

Esto es especialmente perjudicial en esta aplicación, ya que lo habitual es que el entrenamiento de una red neuronal requiera varias épocas y la aplicación puede estar congelada durante un largo tiempo sin otorgar al usuario ningúna señal de que no debe forzar su cierre.

Por tanto se ha utilizado la biblioteca *threading* incluida en *Python 3* para ejecutar varias funcionalidades en dos hilos, uno reservado para las tareas del Controlador, y otro para la Vista.

La Vista muestra que hay un progreso ejecutándose con una nueva clase, PantallaProgreso, del paquete ElementosReutilizables. El diagrama de esta nueva clase es:

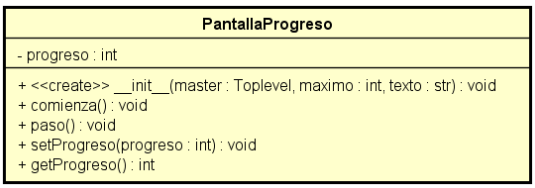

Figura 71: Diagrama de la clase PantallaProgreso en el paquete ElementosReutilizables

Esta clase implementa una pantalla modal con una barra de progreso y un texto.

Los procesos realizados en hilos concurrentes son:

- La carga de documentos al inicio de la aplicación.
- El parseo de ficheros.
- El guardado de documentos en la base de datos.
- El eliminado de documentos.
- El procesamiento de documentos para su utilización por las redes.

- El entrenamiento.

En este último proceso, se ha añadido más que una simple barra de progreso, sino que durante el entrenamiento se muestran además los tamaños, distribución de clases, y las tasas de acierto en los conjuntos de entrenamiento y validación. Esta información se encapsula en una instancia de la clase EstadoEntrenamiento del paquete Modelo. La clase EstadoEntrenamiento pasa a tener el siguiente diagrama:

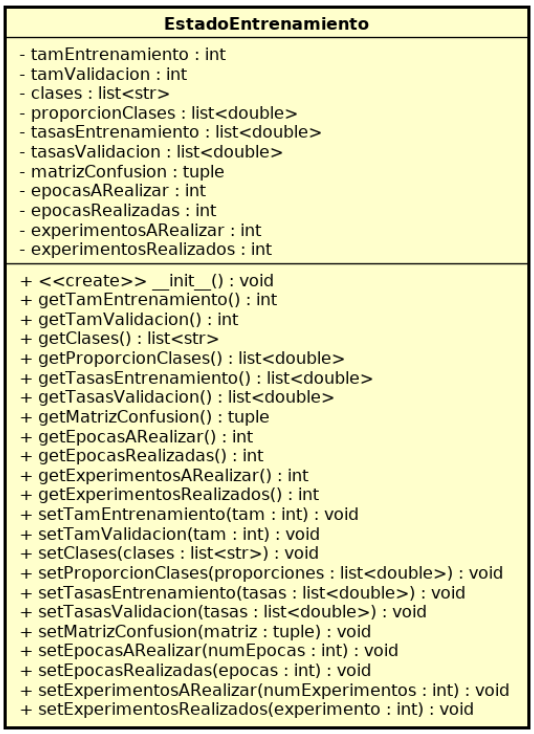

Figura 72: Diagrama de la clase EstadoEntrenamiento en el paquete Modelo en Implementación

Mientras que esto soluciona este primer problema, surge otro, ya que *Tkinter* implementa la seguridad de hilos permitiendo acceder a la Vista solo desde el mismo hilo que la lanzó. Esto implica que no se puede notificar a la Vista directamente de la finalización del proceso ejecutándose en otro hilo, por lo que se cambia el paradigma, para estos procesos, introduciendo colas, clase *Queue* en *Python*, que son compartidas por Controlador y Vista, Controlador actuando como productor y Vista como consumidor.

Los pasos que toma el programa son:

1. La vista recibe la petición de comenzar el proceso por el usuario. Redirige la petición al controlador en un nuevo hilo, acompañado por las colas que consumirá.

2. La vista muestra la pantalla de progreso y hace *polling* a las colas, mostrando los cambios que detecte.

3. El controlador por su parte recibe la petición y ejecuta el proceso, introduciendo los resultados (si procede) en la cola correspondiente. Al terminar el proceso, introduce en la cola un token de finalización.

4. La vista detecta el token y muestra los resultados.

Parte V

Resultados

A continuación se presenta la eficiencia obtenida tras utilizar la aplicación para entrenar varios modelos, empleando el corpus EFE con los títulos de las noticias como cuerpo del documento. Debido al gran tamaño del corpus y las restricciones de tiempo ligadas a la realización del proyecto, se han utilizado sólo los primeros 300 documentos. Utilizar todos los documentos requeriría considerar capas ocultas mucho mayores.

Tipo	Capa oculta	α	γ	Número de épocas	Orden vocabulario	Tasa acierto validación	Tasa acierto entrenamiento
Elman	50	0,8	0,3	2.000	Aparición	32,22 %	94,29 %
Elman	200	0,8	0,6	2.000	Aparición	45,56 %	67,14 %
Elman	200	0,8	0,6	2.000	Aleatorio	26,67 %	94,74 %
Elman	200	0,8	0,6	2.000	Alfabético	24,44 %	83,73 %
Jordan	200	0,8	0,6	3.200	Aparición	31,11 %	50 %
Jordan	100	0,8	0,6	2.000	Aleatorio	27,78 %	48,33 %
Jordan	200	0,8	0,3	2.000	Alfabetico	28,89 %	44,02 %

Cuadro 14: Resultados de los entrenamientos realizados con la aplicación

α representa el factor de aprendizaje, y γ el de inercia.

Como podemos observar, mientras que es posible ajustar el modelo al conjunto de entrenamiento con tasas de acierto cercanas al 100 %, esto no se traduce en un buen resultado en el conjunto de validación, probablemente a causa de un sobreajuste del modelo.

El sobreajuste en un modelo de Aprendizaje Automático se refiere a un fenómeno en el que el modelo aprende el ruido presente en el conjunto de entrenamiento, y es incapaz de generalizar al ver nuevos datos, como son los del conjunto de validación.

Estos resultados podrían ser mejorados aplicando alguno de los cambios de los que se habla en las Conclusiones.

Parte VI

Conclusiones

Este trabajo tiene, como proyecto que culmina los estudios del grado, un carácter integrador de numerosas asignaturas de la carrera. Destacan: asignaturas de planificación y gestión de proyectos, análisis y diseño de software, programación orientada a objetos y las asignaturas relacionadas con Aprendizaje Automático que forman la mención de Computación.

Este proyecto ha supuesto una primera aproximación al procesado del lenguaje natural, un campo de gran interés para mí, pero en el cuál nunca había trabajado con tanta extensión. Mientras que ya había realizado proyectos en Python, nunca habían sido tan ambiciosos, ni habían incluido orientación a objetos, por lo que ha sido una introducción en ese aspecto también.

Aunque ya había realizado una implementación de las Redes Neuronales Recurrentes tratadas para la asignatura de Minería de Datos, este proyecto me ha permitido profundizar más en su funcionamiento, así como entender mejor los retos al desarrollar una aplicación de Aprendizaje Automático desde cero.

La realización del proyecto ha sido relativamente poco problemática, excepto por una cierta decepción en los resultados obtenidos. Sin embargo, es importante señalar que por la propia naturaleza del Aprendizaje Automático, esta aplicación se creó con la idea de que sea sencillo realizar cambios, tanto en el modelo implementado, como en los métodos utilizados para el procesado de texto.

Volviendo a los resultados, la implementación desde cero me ha dado la oportunidad de apreciar muchos detalles que son determinantes en las prestaciones finales del sistema. En particular, tras analizar la dimensionalidad de las entradas y la cantidad de neuronas empleadas, me llevan a pensar que habría que introducir alguna técnica para reducir la complejidad numérica del problema. Por este motivo, en el siguiente apartado se sugirien algunas ideas para conseguir este objetivo pero, sin duda, serían ya materia para otros trabajos, pues su envergadura excedería el tiempo supuestamente dedicado a un TFG.

13. Trabajo futuro

Las sugerencias de posibles funcionalidades y cambios futuros en esta aplicación son:

- La inclusión de métodos de selección de atributos al procesado de textos, que permitan reducir el tamaño de las instancias de entrenamiento y eliminar las palabras que no otorgan información relevante al modelo.
- La utilización de un Mapa Autoorganizado, un modelo estadístico que utiliza entrenamiento no supervisado para convertir un espacio de entrada a otro, denominado mapa. Un Mapa Autoorganizado tiene la función de reducir el tamaño del espacio de entrada, o intentar explotar otro tipo de características del aprendizaje neuronal distinto al que pueden explotar las redes neuronales recurrentes.
- La utilización de otro tipo de Redes Neuronales Recurrentes más avanzadas, como son las BTT (*Backpropagation Through Time*) o LSTM (*Long Short-Term Memory*).
- Aumento del rendimiento mediante algoritmos de Aprendizaje Automático paralelizados, bien a nivel del procesador o en tarjeta gráfica.

Referencias

[MP43] Warren S McCulloch y Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». En: *The bulletin of mathematical biophysics* 5.4 (1943), págs. 115-133.

[MP69] Marvin L. Minsky y Seymour A. Papert. *Perceptrons*. Cambridge, MA, USA: MIT Press, 1969.

[Jor86] MI Jordan. *Serial order: a parallel distributed processing approach. Technical report, June 1985-March 1986*. Inf. téc. California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, 1986.

[Elm90] Jeffrey L Elman. «Finding structure in time». En: *Cognitive science* 14.2 (1990), págs. 179-211.

[Has95] Mohamad H Hassoun. *Fundamentals of artificial neural networks*. MIT press, 1995.

[Hut05] John Hutchins. «The history of machine translation in a nutshell». En: *Retrieved December* 20 (2005), pág. 2009.

[Hay09] Simon S Haykin. *Neural networks and learning machines*. Vol. 3. Pearson Upper Saddle River, NJ, USA: 2009.

[Mes+13] Grégoire Mesnil y col. «Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding.» En: *Interspeech*. 2013, págs. 3771-3775.

[Kow+18] Kamran Kowsari y col. «RMDL: Random Multimodel Deep Learning for Classification». En: *arXiv preprint arXiv:1805.01890* (2018).

[Por] Martin Porter. *Descripción del algoritmo Snowball para stemming en español*. URL: `http://snowballstem.org/algorithms/spanish/stemmer.html`.

Parte VII

Anexo: Diagramas de secuencia referenciados

14. Lee documentos

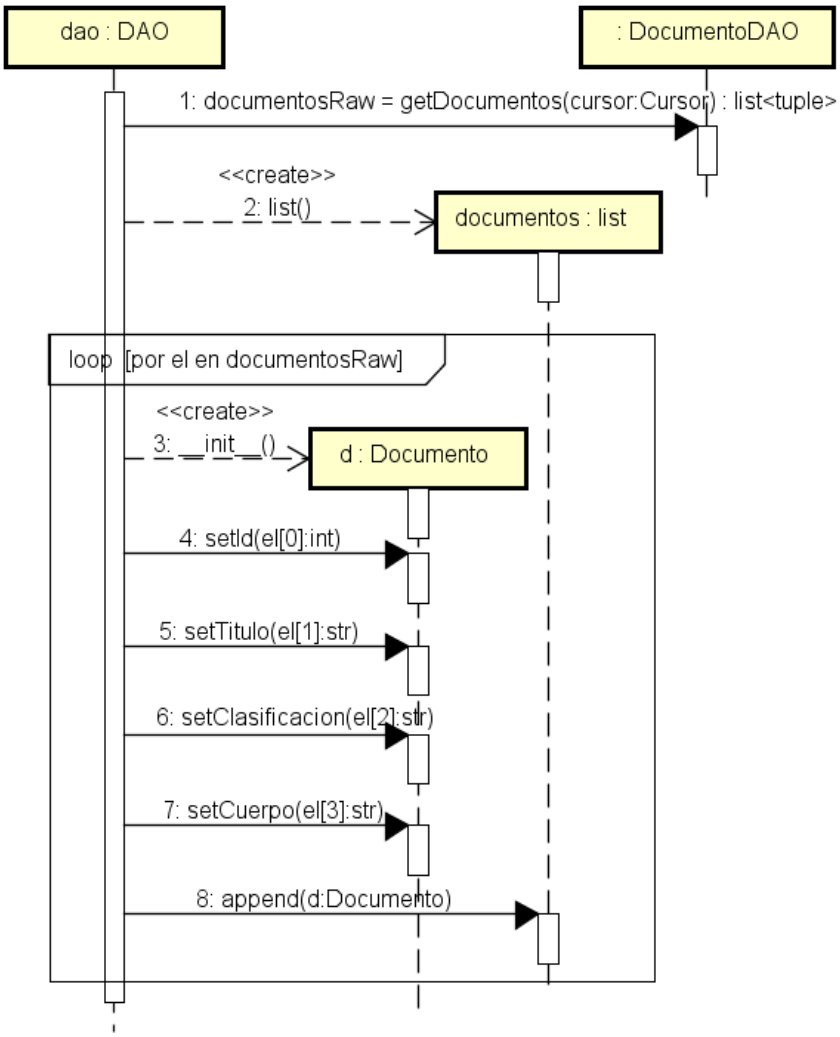

Figura 73: Diagrama referenciado en CU00.

15. Inicia importación de ficheros

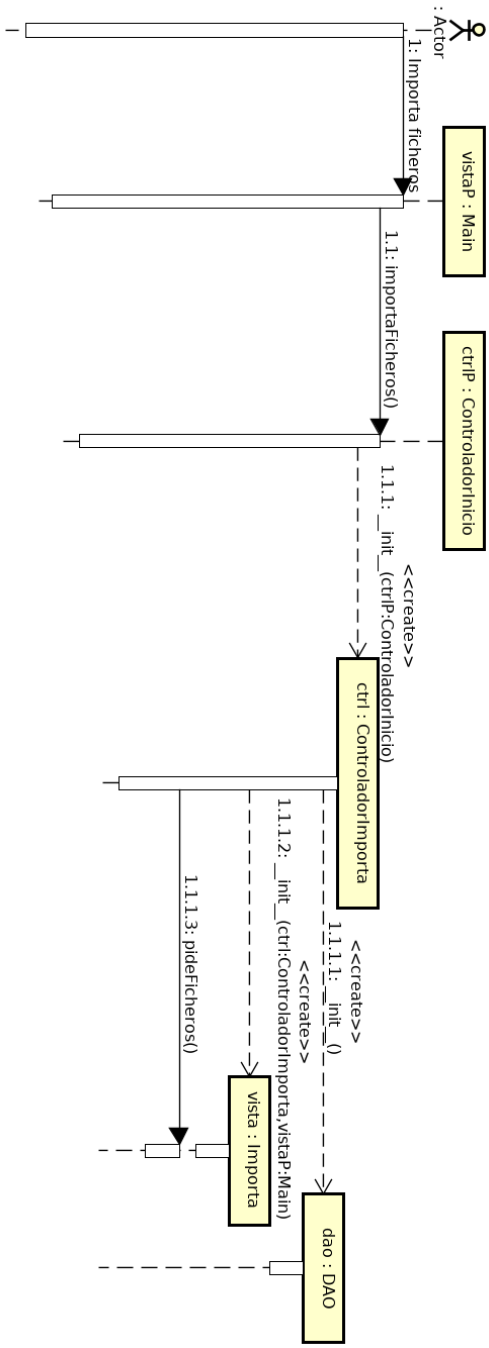

Figura 74: Diagrama referenciado en CU01 y CU02.

16. Almacena documentos

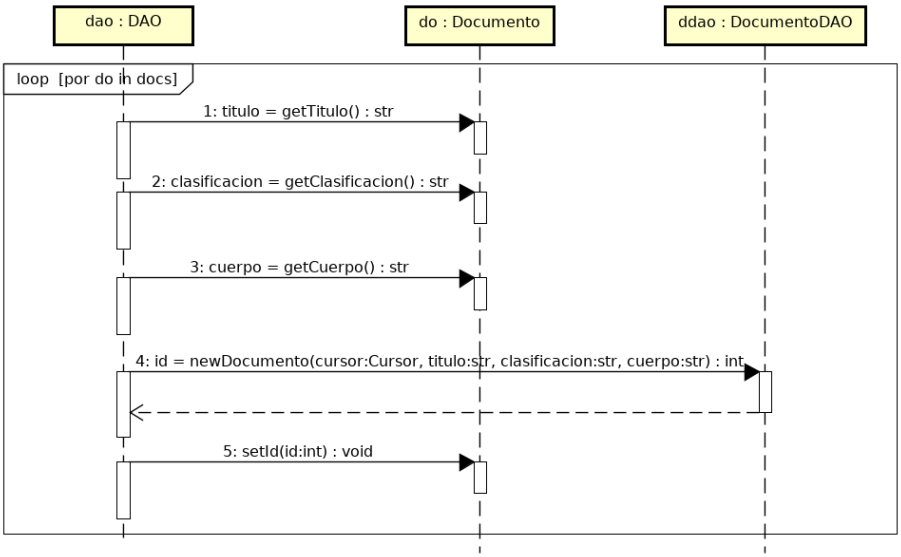

Figura 75: Diagrama referenciado en CU03.

17. Inicia Entrenamiento

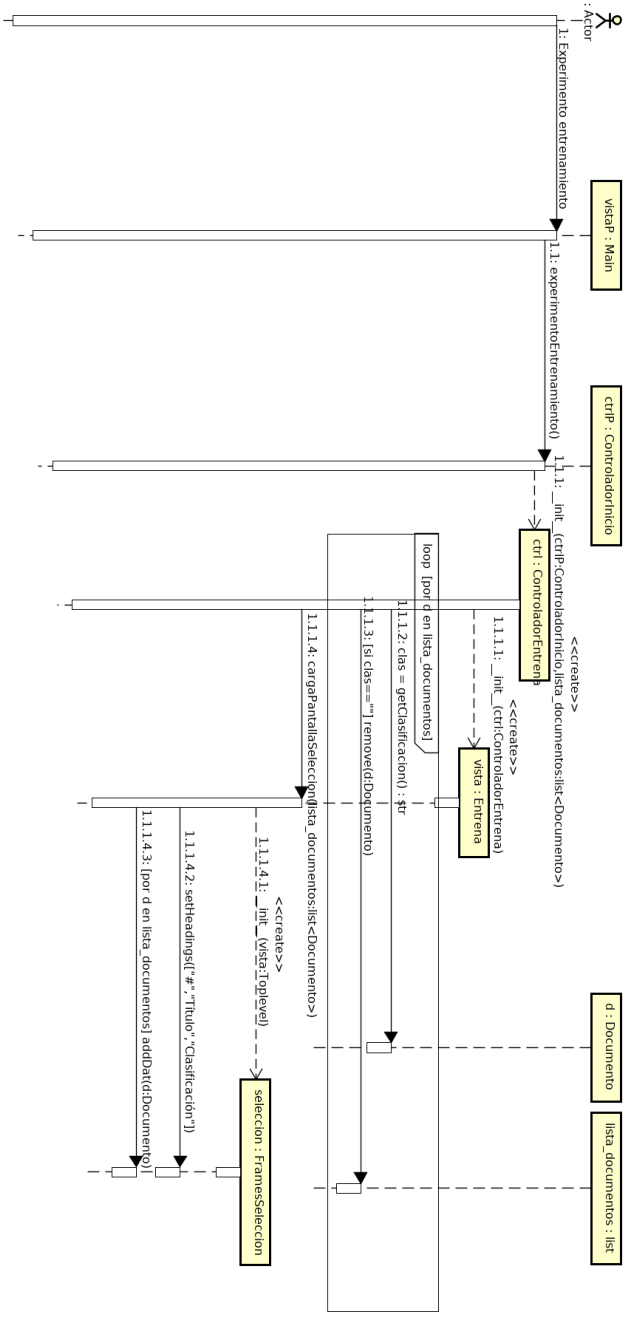

Figura 76: Diagrama referenciado en CU06.

18. Establece parámetros entrenamiento

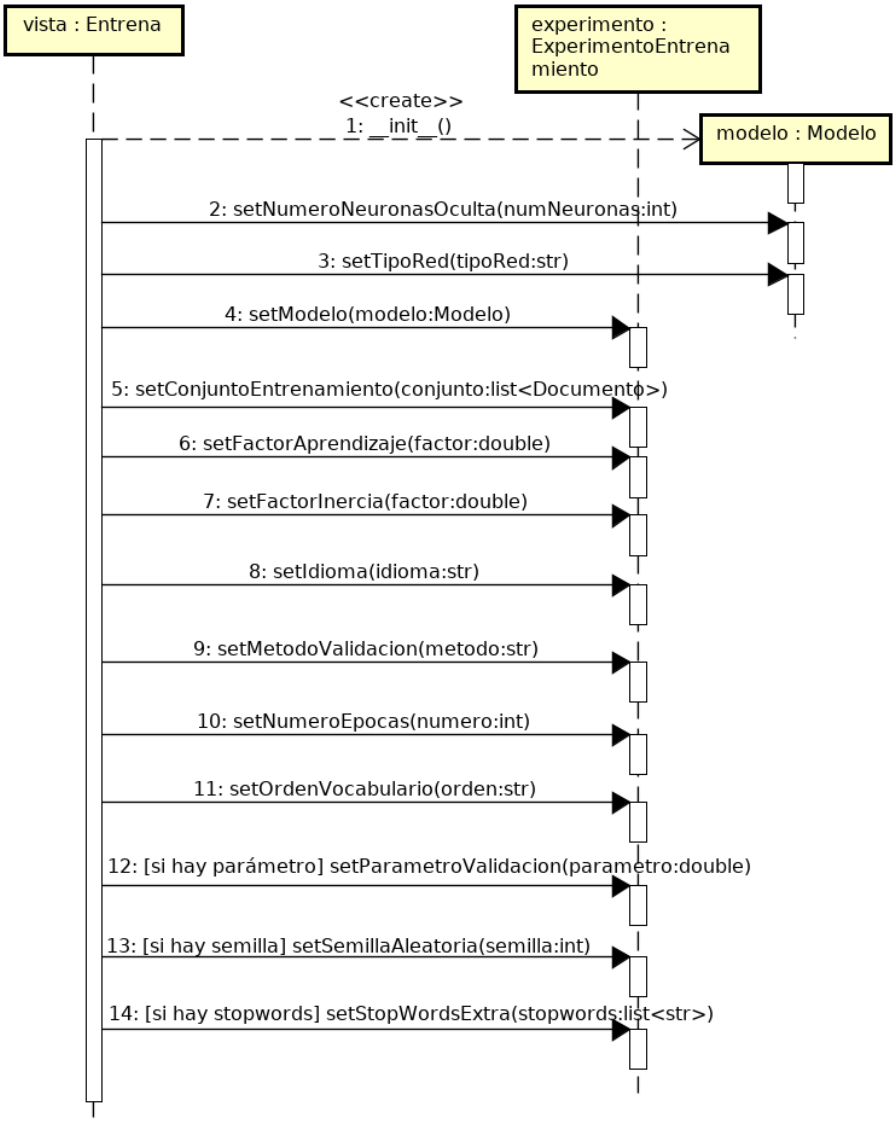

Figura 77: Diagrama referenciado en CU06.

19. Prepara documentos

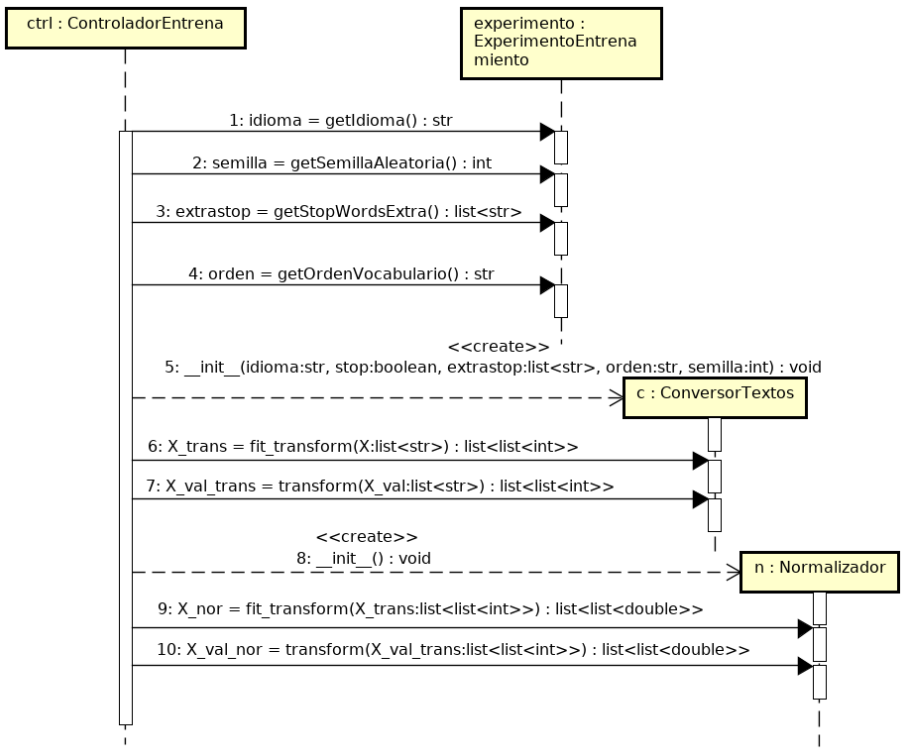

Figura 78: Diagrama referenciado en Ejecuta entrenamiento.

20. Inicializa red

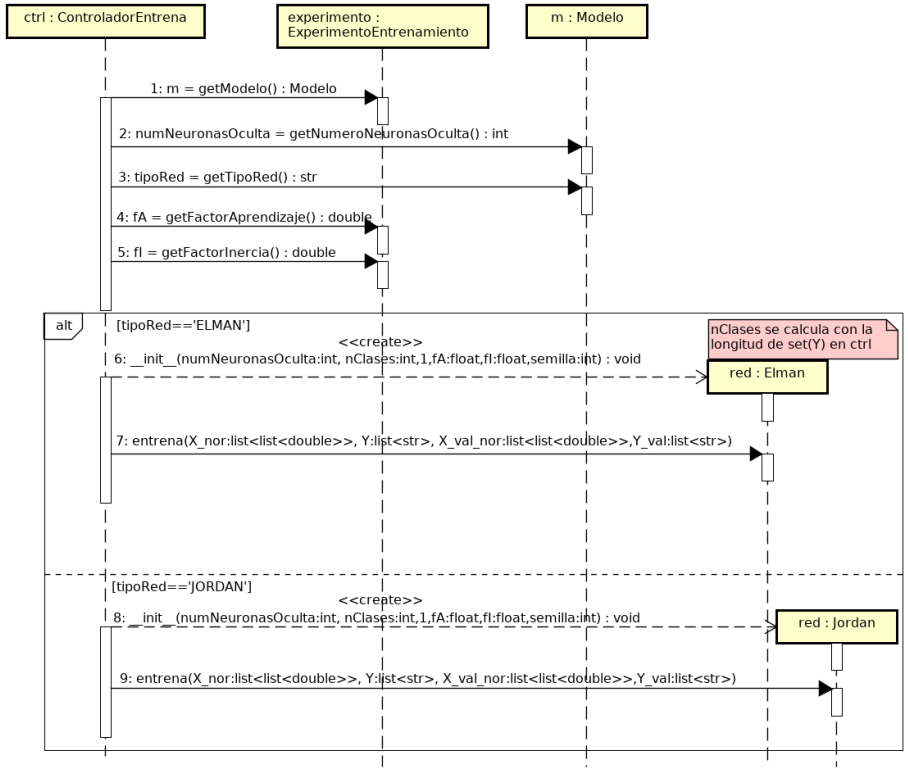

Figura 79: Diagrama referenciado en Ejecuta entrenamiento.

21. Ejecuta iteraciones

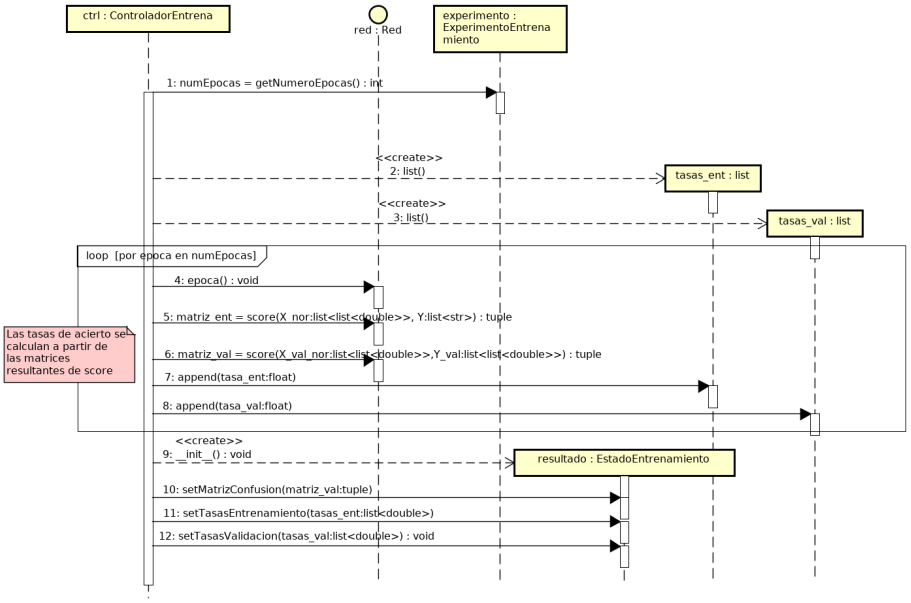

Figura 80: Diagrama referenciado en Ejecuta entrenamiento.

22. Prepara Modelo

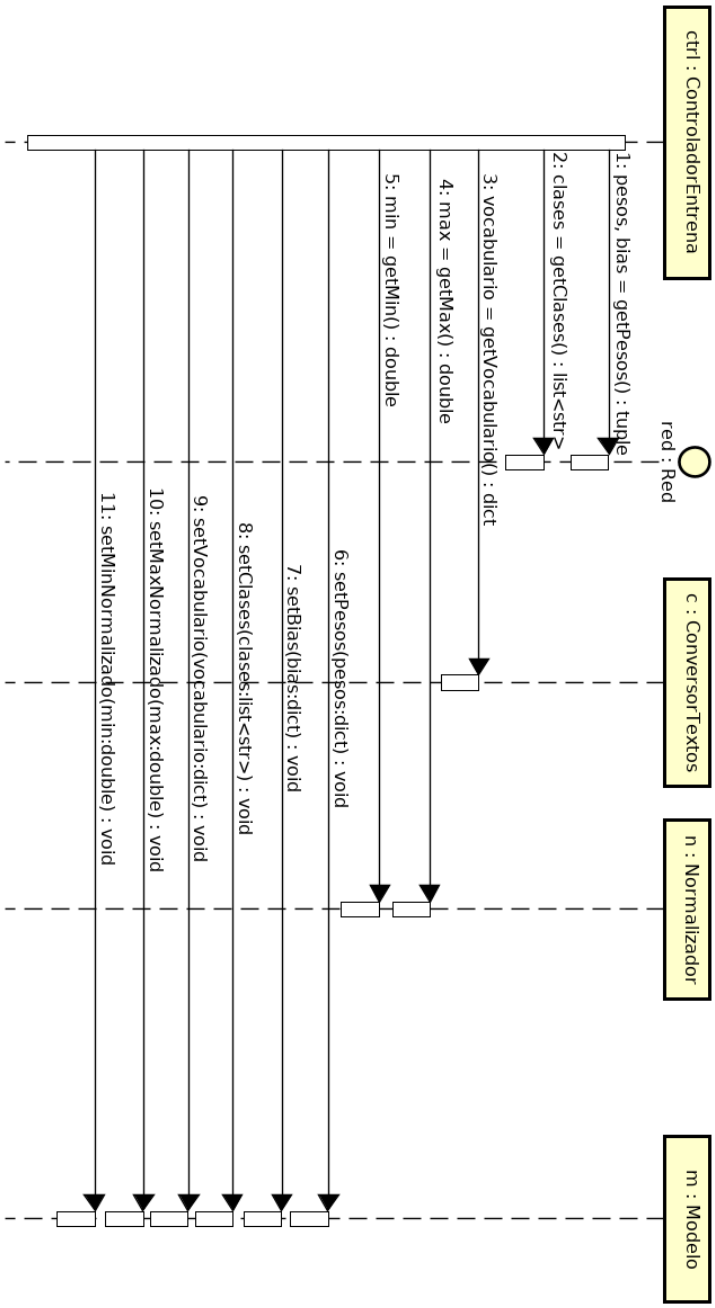

Figura 81: Diagrama referenciado en CU07.

23. Almacena Modelo

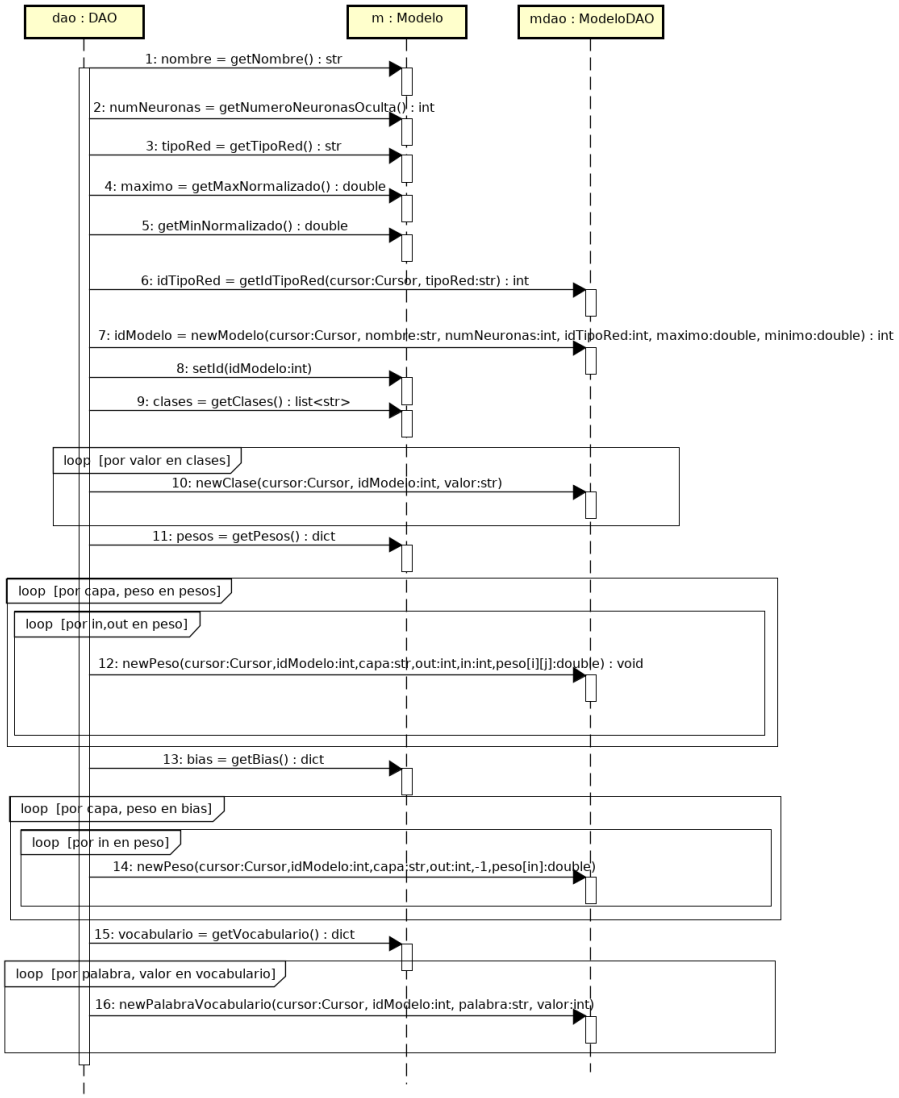

Figura 82: Diagrama referenciado en CU07.

24. Inicia Clasificación

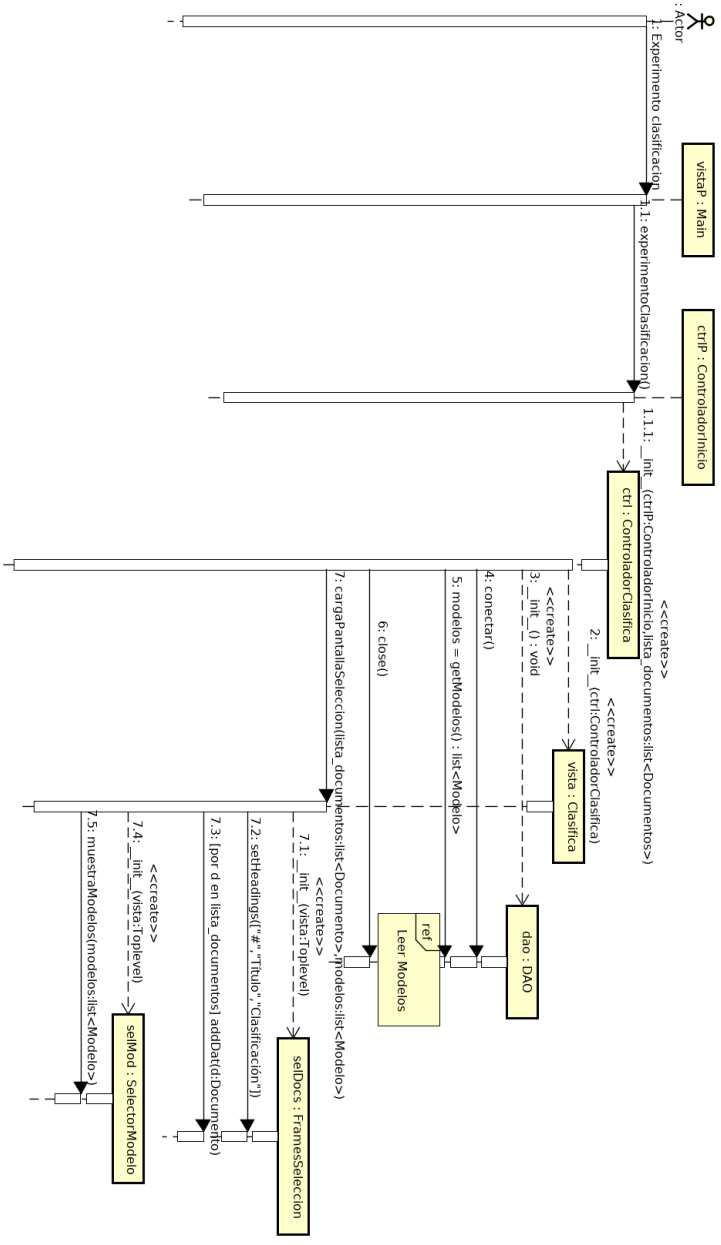

Figura 83: Diagrama referenciado en CU09.

El diagrama referenciado es Figura 84.

25. Leer Modelos

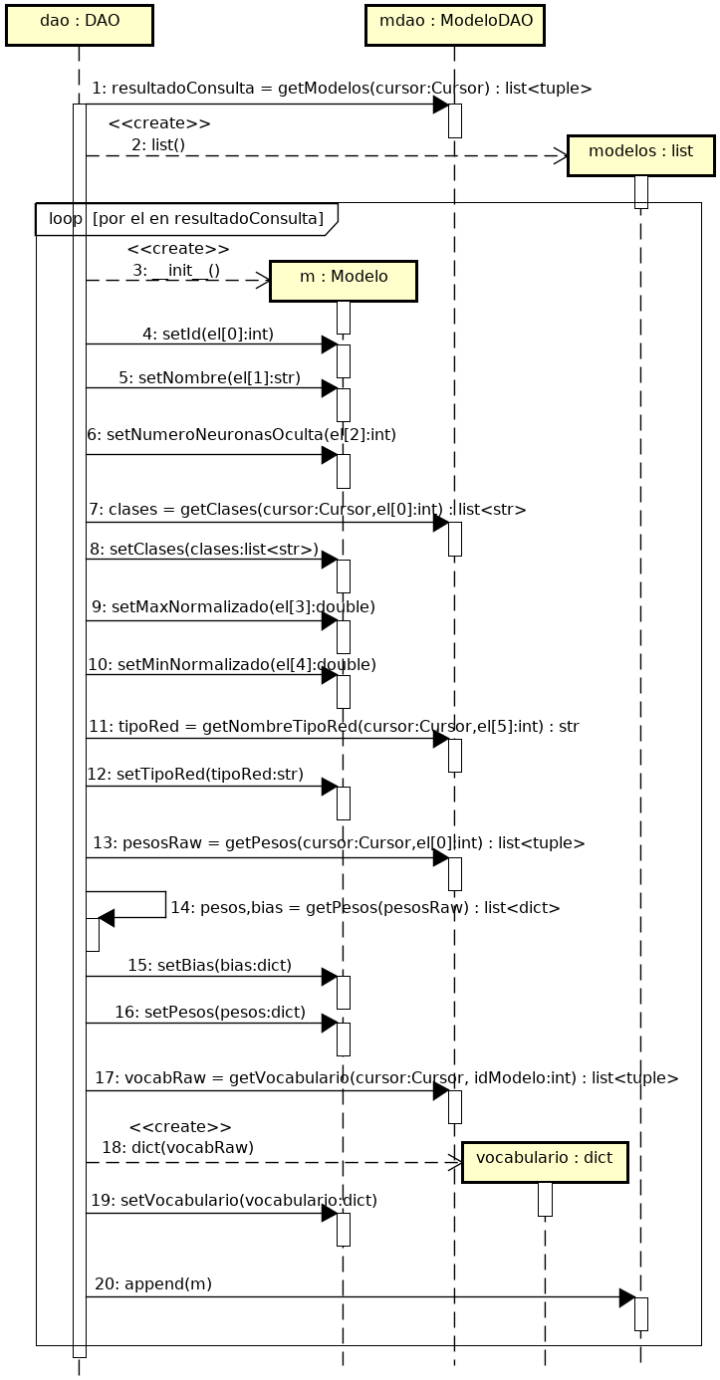

Figura 84: Diagrama referenciado en Inicia Clasificación.

26. Recrea red

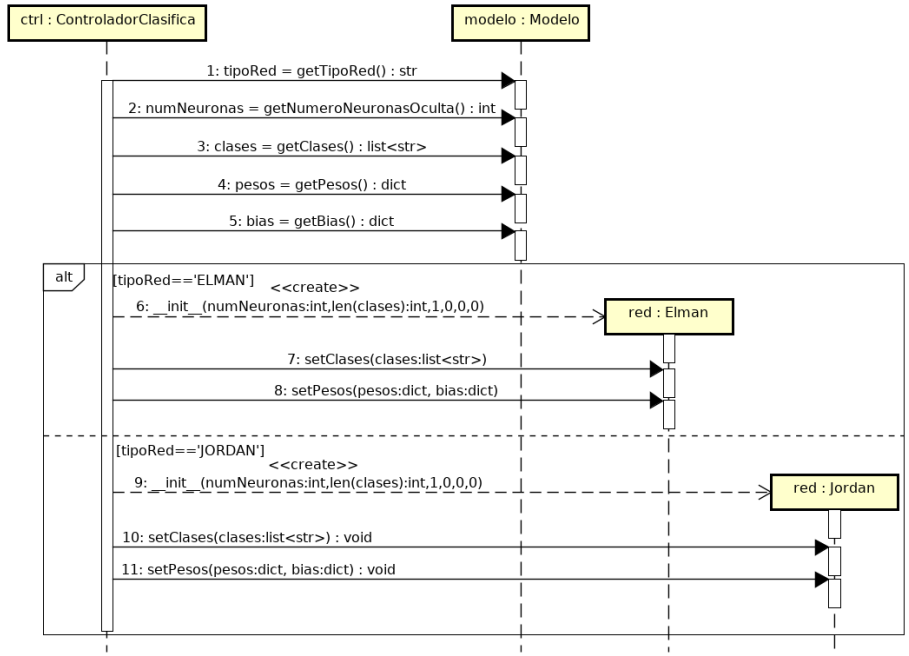

Figura 85: Diagrama referenciado en Ejecuta clasificación.

27. Prepara documentos clasificación

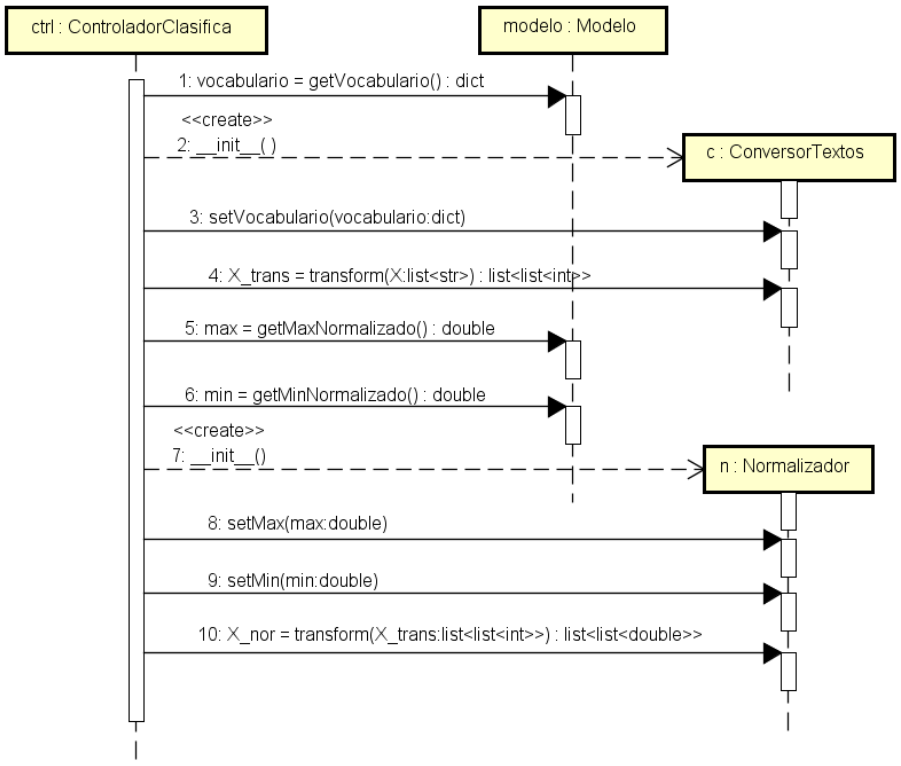

Figura 86: Diagrama referenciado en Ejecuta clasificación.