Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Procesamiento de Lenguaje Natural con Deep Learning para el Análisis de Perfiles Profesionales en Consejos de Administración

Autor

Manuel Astorgano Antón

Tutores

Benjamín Sahelices Fernández

Fernando Tejerina Gaite

Resumen

El gobierno corporativo ha ganado atención en el campo de las finanzas durante varias décadas. Juega un papel clave en la toma de decisiones de inversión o de negocio. Una de sus características más importantes es la independencia del consejo de administración debido a su importante función de control e influencia en diversas variables ligadas al rendimiento y creación de riqueza.

Recientemente, los consejos de administración han sido criticados por su complacencia e incapacidad de prevenir crisis en la empresa. Debido a esto, se han ampliado las perspectivas del análisis de los consejos y ahora se incluyen características como la edad, el género, la nacionalidad o la educación. Esta última no puede medirse de forma cuantitativa, ya que no se puede recopilar con datos estructurados, por lo que se ha dejado de lado a pesar de su importancia.

En este trabajo se realiza un análisis en profundidad de la experiencia de los consejos de administración de empresas de la bolsa de Madrid utilizando biografías, Procesamiento del Lenguaje Natural (PLN) y redes neuronales profundas. Primero, se construye un modelo de lenguaje (un modelo capaz de generar texto) en español. Después, se modifica y se convierte en un modelo de regresión mediante transferencia de aprendizaje. El modelo final es capaz de, dada una biografía de un consejero, proporcionar una medida de varios perfiles profesionales.

Se utilizan dos arquitecturas que generan dos modelos distintos: redes neuronales recurrentes LSTM y *transformers*. A cada uno se le realiza un análisis de interpretabilidad mediante SHAP para obtener las palabras más importantes de las entradas. Los archivos de donde provienen las biografías se obtienen usando *web scraping* y se estructuran mediante un proceso ETL.

Finalmente, se realiza una comparación de las dos arquitecturas que nos lleva a elegir el modelo con transformadores por su mejor rendimiento e interpretabilidad.

Abstract

Corporate governance has been the subject of attention in the financial field of study for several decades. It plays a key role in making investment or business decisions. One of its most important features is the independece of boards of directors due to its importance on carrying out their control function and their influence on various company variables linked to performance and value creation.

Recently, these boards of directors have been criticized for their complacency and inability to prevent corporate crisis. The effect of this criticism has widened the perspective of the board analysis, which now includes new characteristics such as age, gender, nationality or education. The latter cannot be measured quantitatively as it does not qualify as structured data, so it was left behind despite its importance.

In this project, an in-depth analysis of the education of directive boards from the stock market of Madrid is conducted using biographies, Natural Language Processing (NLP) and deep neural networks. First, a spanish language model (i.e. a neural net capable of generating text) is built. Once the previous task is completed, the language model is modified and converted into a regression model using transfer learning. This final model is capable of, given a director biography, outputting a measure of several professional profiles.

Two different architectures which provide two different models are used: recurrent LSTM neural networks and state of the art transformers. An interpretability analysis using SHAP is performed to obtain the key words of the inputs. The files containing the biographies used to build these models are extracted using web scraping and later structured in an ETL process.

Finally, a comparison of the two architectures is given leading us to choose the transformer model thanks to its better performance and interpretability.

Agradecimientos

A Benjamín y a Fernando por su ayuda y dedicación durante el desarrollo del proyecto.
A los profesores del Grado en Ingeniería Informática que durante estos años me han enseñado las y transmitido el interés en la informática.
A mi familia y amigos por su ayuda y apoyo durante estos años.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Descripción del problema

El gobierno corporativo ha ganado atención durante los últimos años tanto desde la perspectiva académica como legal con la proliferación de prácticas de buen gobierno y otros tipos de regulaciones débiles (aquellas que no prohíben determinadas acciones siempre y cuando se sigan procedimientos preestablecidos o se suministre cierta información).

Una de las características más controvertidas en este área es la independencia del consejo de administración, ya que se considera esencial para llevar a cabo sus funciones de dirección además de su influencia en otros aspectos de la empresa relacionados con el rendimiento y la creación de riqueza. En este contexto, suelen analizarse variables como el porcentaje de consejeros independientes, el tamaño del consejo, la separación de las figuras de presidente y CEO (*Chief Executive Officer* o jefe ejecutivo de operaciones) o el número de reuniones. Todo lo anterior se localiza en lo que se denomina «diversidad estructural del consejo».

Sin embargo, en los últimos años los consejos están siendo criticados por su excesiva complacencia e incapacidad de evitar las crisis de la empresa. Esto contribuye a que se analicen de forma más amplia, explorando nuevas perspectivas. Así, tanto en círculos académicos como reguladores, se ha centrado la atención en las características de los consejos de administración que puedan influir en la efectividad de los procesos de toma de decisiones. Estas características incluyen, entre otras, la edad, la educación, el género o la nacionalidad de los consejeros.

Muchas de las variables analizadas anteriormente se pueden medir de forma cuantitativa, lo que facilita su análisis. No obstante, otros aspectos como el perfil profesional se han dejado de lado debido a la dificultad de obtención y organización de la información.

Con el incremento de la potencia de procesamiento en los ordenadores y la disponibilidad de grandes cantidades de información, es posible aplicar modelos de *machine learning* a la investigación financiera [1].

En este trabajo se construirán modelos de aprendizaje profundo con redes neuronales en el campo del PLN o Procesamiento del Lenguaje Natural (NLP o *Natural Language Processing*) que permitirán evaluar los perfiles profesionales de los consejos de administración de forma automática. Primero se extraerá la información necesaria con técnicas de *web scraping*. Después se utilizarán tecnologías actuales como PyTorch y la computación en la nube mediante GPUs para crear dos modelos con arquitecturas diferentes y evaluar su rendimiento.

1.2. Objetivos

Realizar un análisis de un perfil profesional es costoso. La obtención de los datos y su posterior procesamiento requieren mucho tiempo. La automatización de procesos y el procesamiento de lenguaje natural han demostrado ser de utilidad cuando el número de perfiles a analizar se cuentan por miles. Los objetivos de este trabajo son:

- Automatizar el análisis de perfiles profesionales de consejeros de compañías que cotizan en el Mercado Continuo de la Bolsa de Madrid.
- Estudiar cómo las redes neuronales profundas aplicadas sobre los perfiles permiten hacer inferencias sobre variables de interés.
- Aprender cómo funcionan los modelos de lenguaje con arquitecturas modernas o de *state of the art*.

1.3. Estructura de la memoria

La memoria del trabajo se organiza según el siguiente esquema. El capítulo 2 describe los fundamentos de las redes neuronales y las arquitecturas que se van a utilizar. El capítulo 3 introduce la teoría financiera necesaria para comprender el análisis de consejeros además de las técnicas y herramientas informáticas empleadas para la realización del proyecto. El capítulo 4 presenta el proceso de planificación del trabajo y una estimación del presupuesto del mismo. A continuación, en el capítulo 5 se profundiza en la construcción de un modelo de lenguaje en español, desde la obtención de los datos hasta los resultados y su evaluación. En el capítulo 6 se modifica el modelo de lenguaje anterior con nuevos conjuntos de datos y cambios en la arquitectura para que se ajuste al objetivo inicial del trabajo. El capítulo 7 muestra los

resultados finales de ambos modelos, su interpretabilidad dado un conjunto de ejemplo y una comparativa de ambas arquitecturas. Finalmente, en el capítulo 8 se presentan las conclusiones del proyecto y el posible trabajo futuro.

Capítulo 2

Redes neuronales

En esta sección se explican los términos y componentes esenciales que forman una red neuronal. Una red neuronal es un modelo de inteligencia artificial formado por neuronas. Dichas neuronas están conectadas entre sí y pueden transmitir señales entre ellas. Las señales de las conexiones son números reales y la salida de cada neurona se calcula utilizando una función no lineal de la suma de sus entradas. Habitualmente las neuronas tienen pesos y umbrales que se ajustan a medida que transcurre el proceso de aprendizaje. Los pesos aumentan o disminuyen la fuerza de la señal de su conexión, mientras que los umbrales hacen que la señal se propague siempre y cuando la señal supere dicho umbral.

A partir de esta idea, es posible alimentar una red neuronal con datos en un proceso denominado «entrenamiento» de forma que los pesos y umbrales se ajusten para predecir un valor de interés a partir de los datos.

2.1. Perceptrón simple

El perceptrón simple es un tipo de neurona artificial desarrollado por Frank Rosenblatt [2] entre 1950 y 1960. Se inspira en el trabajo anterior de Warren McCulloch y Walter Pitts [3], quienes propusieron el primer modelo computacional de una neurona. Dicho modelo intenta «imitar» el comportamiento de una neurona biológica, agregando n entradas binarias y proporcionando una salida binaria.
Las entradas x de la neurona son binarias.

$$x_i \in \{0, 1\}, i = 1...n \tag{2.1.1}$$

Las salida y de la neurona también es binaria.

$$y \in \{0, 1\} \tag{2.1.2}$$

Finalmente, el modelo cuenta con un parámetro θ que actúa como umbral. La función de activación de la neurona es la siguiente:

$$\begin{aligned} g(x_1, ..., x_n) = g(\mathbf{x}) &= \sum_{i=1}^{n} x_i \\ y = f(g(\mathbf{x})) &= \begin{cases} 1 & \text{si } g(\mathbf{x}) > \theta \\ 0 & \text{si } g(\mathbf{x}) \leq \theta \end{cases} \end{aligned} \tag{2.1.3}$$

Lo único que hace $g(\mathbf{x})$ es una agregación simple. Si esta agregación es mayor que el umbral, la neurona se activará.
El modelo de McCulloch-Pitts puede representar puertas lógicas siempre y cuando su representación sea linealmente separable, es decir, exista un plano en el que todas las entradas que producen un 1 estén separadas de las entradas que producen un 0. De esta forma, `AND`, `OR`, `NOR` y `NOT` son adecuadas. Sin embargo, `XOR` es la primera función no representable por el perceptrón.

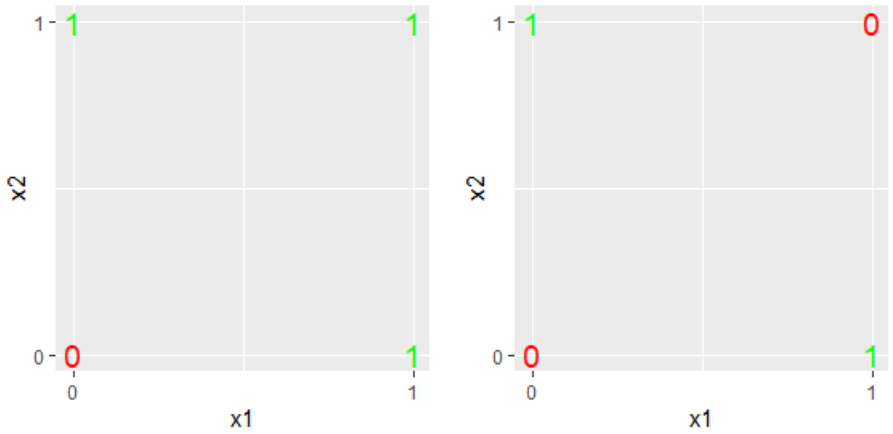


Figura 2.1.1: Separabilidad lineal de `OR` y `XOR`.

Con las imágenes de la Figura 2.1.1 es sencillo comprobar que la función `XOR` no es representable por esta neurona artificial.
Basándose en esta idea, Rosenblatt desarrolló el perceptrón simple [2]. La principal mejora respecto a sus predecesores es la inclusión de pesos, w_i, para modelar la importancia de las entradas a la hora de calcular la salida. Ahora, la salida de la neurona está determinada por la

suma ponderada y el valor umbral θ, lo que permite resolver problemas más complejos.

$$g(x_1, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i \tag{2.1.4}$$

Donde la salida y se calcula de la misma manera que en 2.1.3. Normalmente, el valor umbral se sustituye por el *bias* en la ecuación ($b = -\theta$), reescribiendo y.

$$y = f(g(\mathbf{x})) = \begin{cases} 1 & \text{si } g(\mathbf{x}) + b > 0 \\ 0 & \text{si } g(\mathbf{x}) + b \leq 0 \end{cases} \tag{2.1.5}$$

Observamos la estructura del perceptrón en la Figura 2.1.2.

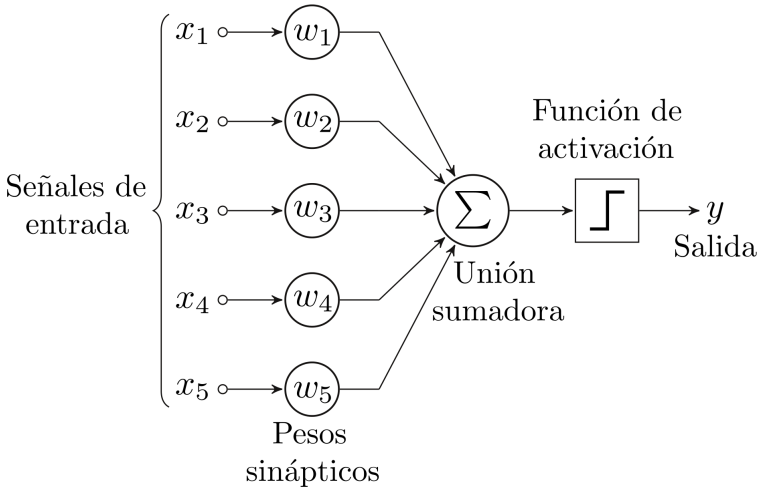


Figura 2.1.2: Perceptrón simple. Fuente: [4].

Cuando realizamos la suma de las entradas ponderadas, se evalúa a una función de activación generando la salida binaria.

Normalmente, las neuronas se agregan en capas. Diferentes capas generan diferentes transformaciones en sus entradas. Las señales viajan a través de la primera capa (capa de entrada) a la última capa (capa de salida), posiblemente atravesando múltiples capas intermedias (capas ocultas).

2.2. Funciones de activación

La función de activación en una red neuronal define la salida de una neurona dado un conjunto de entradas. Cada neurona tiene asignada una función de activación, perteneciente a uno de los dos tipos principales: lineal o no lineal. Las funciones no lineales permiten modelar problemas más complejos ya que solucionan problemas presentes en las lineales [5]. Se trata de uno de los hiperparámetros que hay que elegir a la hora de confeccionar una red neuronal.

2.2.1. Función lineal

Toma como entrada la suma ponderada de la Ecuación 2.1.4 y genera una salida proporcional. Se trata de una recta. En el caso de que $m = 1$, recibe el nombre de función identidad.

$$f(x) = mx \tag{2.2.1}$$

La función de activación lineal apenas se usa ya que presenta dos problemas:

- El algoritmo de retropropagación no se puede aplicar, ya que la derivada de $f(x)$ es constante. No tiene relación con la entrada.
- Una red neuronal con n capas y función de activación lineal es equivalente a una red con solo una capa. Como cada capa conforma una combinación lineal, la última capa es una combinación lineal de la primera.

2.2.2. Función signo

Se trata de la función usada en el perceptrón simple de la Figura 2.1.2. Transforma la entrada en una salida binaria en función del signo.

$$f(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases} \tag{2.2.2}$$

Tiene la mala propiedad de que pequeños cambios en la entrada pueden modificar la salida, ya que no presenta un cambio gradual.

2.2.3. Función sigmoide o logística

Esta función toma como entrada cualquier valor real y proporciona otro valor en el rango $[0, 1]$. Cuanto más positiva sea la entrada, más cercana a 1 será la salida y viceversa. Dichas salidas

pueden interpretarse como probabilidades, por lo que suele usarse en la capa de salida ya que es sencilla de interpretar.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2.3}$$

Presenta algunas desventajas como un alto coste computacional y pequeñas variaciones en la salida ante grandes cambios en la entrada. Esta última puede ocasionar lo que se conoce como el desvanecimiento del gradiente.

2.2.4. Función tangente hiperbólica

Similar a la función logística, presenta una forma parecida pero toma valores en el rango $[-1, 1]$.

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.2.4}$$

Las ventajas de usar esta función de activación son dos. La primera es que se encuentra centrada en el cero, por lo que podemos fijar la salida como negativa, positiva o neutral. La segunda es que suele usarse en las capas ocultas ya que la media de sus valores de salida son cero o muy cercanos a cero. Esto facilita el centrado de los datos y el aprendizaje para la siguiente capa.
Sin embargo, cuenta con el mismo problema que la función sigmoide: el desvanecimiento del gradiente

2.2.5. Función rectificador lineal

Normalmente conocida como ReLU (*Rectified Linear Unit*). Es una de las funciones de activación más usadas.

$$f(x) = max(0, x) \tag{2.2.5}$$

Destaca frente al resto ya que es rápida tanto a la hora de entrenar como a la hora de evaluar. Presenta valores en el intervalo $[0, \infty]$. Su principal desventaja aparece cuando su entrada se aproxima a 0. Esto provoca que la retropropagación del error no sea efectiva y dificulta el aprendizaje del modelo.
Se propone como solución al anterior problema la función de activación *Leaky* ReLU.

$$f(x) = \begin{cases} x & \text{si } x > 0 \\ 0{,}01x & \text{si } x \leq 0 \end{cases} \tag{2.2.6}$$

De esta forma, para valores de entrada negativos la pendiente es positiva haciendo posible la retropropagación del error. No obstante, sigue presentando algunas limitaciones como la

inconsistencia de las predicciones para entradas negativas o el alto tiempo de aprendizaje debido al pequeño valor del gradiente para entradas negativas.

2.2.6. Función *softmax*

Se basa en la función sigmoide y suele ser la opción a elegir en la última capa de problemas de multiclasificación. Genera la probabilidad relativa de pertenecer a cada clase, de forma que la suma de las salidas sea 1.

$$f_i(z_i) = \frac{e^{z_i}}{\sum_n^{i=0} e^{z_i}} \tag{2.2.7}$$

Aplica la función exponencial a cada elemento z_i del vector $\mathbf{z}$ de tamaño n, normalizando los valores dividiendo por la suma de todas las exponenciales.

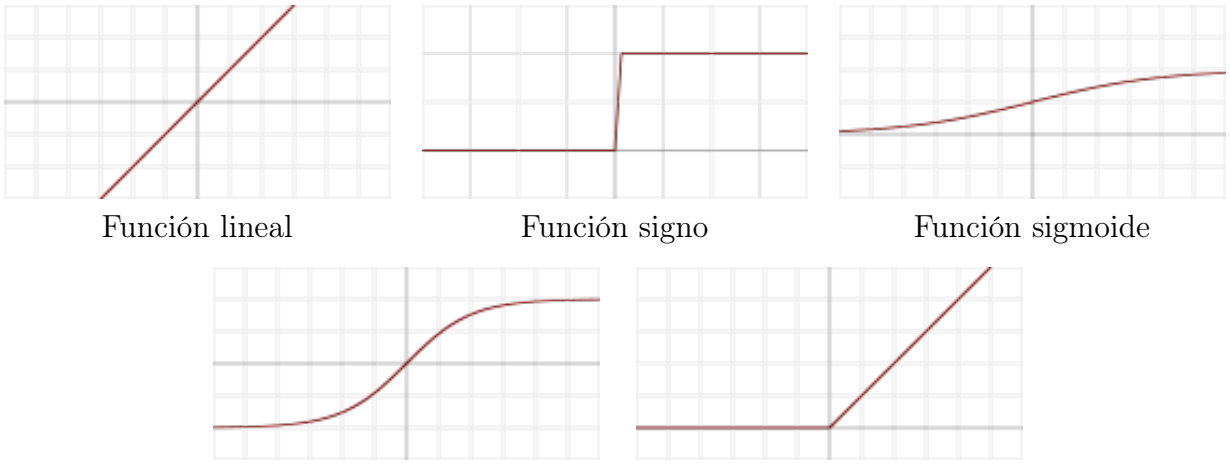


Figura 2.2.1: Funciones de activación

Existen más funciones de activación aparte de las anteriores, normalmente restricciones de los casos generales. Observamos las que hemos explicado anteriormente en la Figura 2.2.1.

2.3. Entrenamiento

Como se ha mencionado en la Ecuación 2.1.4, los pesos y *bias* asociados a cada neurona son esenciales para obtener un modelo con buenos resultados. Para una misma entrada, dos neuronas con distintos pesos pueden generar salidas distintas. La etapa de entrenamiento de una red neuronal consisten en calcular los valores adecuados para estos parámetros de cada neurona de

forma que las predicciones del modelo sean lo más cercanas a la realidad posible. Lo anterior se traduce en minimizar el error de generalización del clasificador.

Nos referiremos a las entradas de entrenamiento como x y a las salidas deseadas como y. Buscamos un algoritmo que permita encontrar tanto los pesos como los *bias* de forma que la red aproxime y para todos los valores de entrada x. Para cuantificar el progreso en esta tarea se define una función de pérdida o coste. Como el error de clasificación toma valores en $\{0, 1\}$, lo que no conforma una función suave, se utiliza una función de pérdida que sí lo sea para evaluar el modelo durante el entrenamiento.

2.3.1. Función de pérdida

Una función de pérdida o coste calcula la distancia entre la salida actual del algoritmo y la salida esperada. Permite evaluar lo bien que dicho algoritmo modela los datos [6]. Existen dos grupos de funciones de pérdida dependiendo del tipo de problema: clasificación (valores discretos) o regresión (valores continuos). Las funciones de coste que destacamos son las siguientes:

- Error cuadrático medio (*mean squared error*, MSE). Utilizada en problemas de regresión, valores continuos.
  $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Error absoluto medio (*mean absolute error*, MAE). Utilizada en problemas de regresión, valores continuos.
  $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- Entropía cruzada binaria (*binary cross entropy*). Utilizada en problemas de clasificación binarias, dos clases.
  $BCE = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$
- Entropía cruzada categórica (*categorical crossentropy*). Utilizada en problemas de clasificación con varias clases.
  $CCE = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$

Inicialmente, se utilizaba el error cuadrático medio en todos los problemas de redes neuronales. Fue a partir de 1990 cuando la entropía cruzada comenzó a sustituir al MSE, generando resultados mucho mejores en problemas de clasificación [7].

Existen más funciones de pérdida, pero en este trabajo nos limitaremos a las que se han descrito ya que se ajustan a la naturaleza del problema (entropía cruzada para generar textos y MSE para regresión).

2.3.2. Descenso del gradiente

El descenso del gradiente es una manera de minimizar una función objetivo C (en nuestro caso, la función de coste) parametrizada por los parámetros del modelo (pesos w_k y bias b_l de la red) en la dirección opuesta al gradiente de la función objetivo. Introducimos otro hiperparámetro al modelo: la velocidad de aprendizaje o *learning rate* η, que indica la longitud de los pasos que tomamos hasta alcanzar un mínimo local. Se sigue la dirección de la pendiente en la superficie creada por la función objetivo hasta llegar a un mínimo [8].

$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \tag{2.3.1}$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l} \tag{2.3.2}$$

Esta idea presenta algunos problemas que dificultan aplicarla directamente a una red neuronal. El principal es que para obtener el gradiente de la función de coste ∇C hay que calcular el gradiente de cada instancia de entrenamiento ∇C_x. Normalmente contamos con una gran cantidad de datos, por lo que este procedimiento puede llevar mucho tiempo y el aprendizaje es demasiado lento. Por ello, se desarrolla la idea del descenso del gradiente estocástico (*stochastic gradient descent*, SGD).

Esta variante del algoritmo se emplea en la gran mayoría de problemas de inteligencia artificial. Introduce el concepto de *mini-batch* o mini-lote de m muestras de entrenamiento seleccionadas aleatoriamente del set de datos original $(x_1, ..., x_m)$.

$$\nabla C = \frac{1}{n} \sum_{i=1}^{n} \nabla C_x \approx \frac{1}{m} \sum_{i=1}^{m} \nabla C_{x_i} \tag{2.3.3}$$

Donde el primer término es el gradiente sobre el conjunto de datos de tamaño n y el segundo, sobre el *mini-batch*. Conectando esta idea con la actualización de pesos y bias tenemos:

$$w_k \to w'_k = w_k - \frac{\eta}{m} \frac{\partial C_{x_i}}{\partial w_k} \tag{2.3.4}$$

$$b_l \to b'_l = b_l - \frac{\eta}{m} \frac{\partial C_{x_i}}{\partial b_l} \tag{2.3.5}$$

Cuando se acaba con un lote, se toma otra muestra sin reemplazamiento del conjunto de entradas y se repite el proceso. El procedimiento termina cuando se agotan las entradas y se dice que se ha completado una época (*epoch*) de entrenamiento.

En la práctica, se introducen algunos pequeños cambios interesantes en el algoritmo:

- Modificar la tasa de aprendizaje durante el entrenamiento. A medida que se entrena el modelo, se reduce gradualmente para obtener un mejor comportamiento. Sin embargo, esta modificación no se adapta al conjunto de datos y tiene que definirse de antemano [9].
- Introducir el término inercia (*momentum*). Como si se lanzase una pelota por una cuesta, permite acelerar el SGD en la dirección correcta, disminuir las oscilaciones y escapar de mínimos locales [10].

2.3.3. Retropropagación

El algoritmo de retropropagación o *backpropagation* es un algoritmo para calcular de forma rápida los gradientes de la sección anterior. Se introdujo en 1970 pero no se empezó a poner en práctica hasta que se publicó un estudio en 1986 [11]. La solución descrita mejoraba notablemente el rendimiento de las redes neuronales, permitiéndoles resolver problemas anteriormente inabordables. Hoy en día, el algoritmo de la retropropagación se utiliza en todos los modelos de redes neuronales.

La base de este método se encuentra en una expresión concreta de la derivada parcial de la función de coste C ($\frac{\partial C}{\partial w}$ y $\frac{\partial C}{\partial b}$) respecto a cualquier peso w o bias b de la red. La nomenclatura para explicar el algoritmo es la siguiente:

- w^l es la matriz de pesos de las neuronas que conectan con la capa l. Cada elemento w^l_{jk} es el peso de la neurona k-ésima en la capa $l-1$ que conecta con la neurona j-ésima en la capa l.
- b^l es el bias o sesgo de la neurona j-ésima en la capa l. Cada elemento b^l_j es el bias de la neurona j.
- a^l es el vector de activaciones de la capa l. Cada elemento a^l_j es la activación de la neurona j.
- z^l es la suma ponderada de las neuronas de la capa l. $z_l = w^l a^{l-1} + b^l$.
- δ^l es el vector de errores asociado a la capa l. Cada elemento δ^l_j es el error de la neurona j, definido como $\delta^l_j = \frac{\partial C}{\partial z^l_j}$.

De esta forma, tenemos una ecuación en forma vectorizada para a^l dentro de la función de activación f con la suma ponderada z^l.

$$a^l = f(z^l) \tag{2.3.6}$$

El algoritmo de retropropagación permite observar cómo los cambios sobre los pesos y bias de la red afectan a la función de coste C. Para ello, hay que calcular δ^l para cada capa y relacionar los errores con las derivadas parciales

$$\frac{\partial C}{\partial w_{jk}^l} \text{ y } \frac{\partial C}{\partial b_j^l} \tag{2.3.7}$$

Backpropagation se basa en cuatro fórmulas fundamentales que calculan tanto el error δ^l como el gradiente de la función de pérdida.

1. Ecuación para el error de la capa de salida δ^L.

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} f'(z_j^L) \tag{2.3.8}$$

   Donde el primer término mide cómo de rápido cambia el coste en función de la entrada j, mientras que el segundo mide cómo de rápido cambia la función de activación f en función del error.

2. Ecuación para el error δ^l en función de la capa siguiente (δ^{l+1}).

$$\delta^l = ((w^{l+1})^t \delta^{l+1}) \cdot f'(z^l) \tag{2.3.9}$$

   Suponiendo que conocemos el error de la capa $l+1$, al trasponer la matriz de pesos estamos propagando el error hacia atrás (de ahí *backpropagation*) y midiendo el error en la salida de la capa l. Se multiplica por la función de activación obteniendo el error en la entrada ponderada hacia la capa l.

3. Ecuación de la derivada del coste en función del bias.

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.3.10}$$

4. Ecuación de la derivada del coste en función del peso.

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{2.3.11}$$

Las dos últimas ecuaciones nos dan el gradiente de la función de coste.

2.3.4. Optimizadores

Un optimizador es un método iterativo de reducción de la función de pérdida, normalmente basado en el descenso del gradiente. Existen varios algoritmos (siendo SGD uno de ellos) y cada uno se comporta de forma distinta dependiendo de la función a optimizar.

- Descenso del gradiente estocástico (SGD). Explicado anteriormente, suele incluirse con el término inercia [12], basado en la fórmula de inercia de Nesterov [13].
- RMSprop. Su objetivo es obtener la media móvil de los cuadrados de los gradientes y dividir el gradiente por la raíz de la media móvil anterior para obtener una estimación de la varianza [14].
- Adam (*Adaptive Moment Estimation*). Se basa en la estimación de momentos de primer y segundo orden de los gradientes (media y varianza). Es un método computacionalmente eficiente y que consume poca memoria, lo que hace que sea adecuado para problemas con muchos datos o parámetros [15].

La elección del optimizador suele basarse en el tiempo de entrenamiento. Adam es el algoritmo que converge más rápido entre los usados actualmente. Sin embargo, no es el que mejor generaliza. SGD es más lento pero ofrece los mejores resultados con una elección adecuada de *momentum* [16]. Con el tiempo se han desarrollado otros optimizadores basados en los anteriores que ofrecen un balance entre tiempo y calidad de generalización. En el caso de AdamW, se ofrece un entrenamiento rápido a la vez que compite con la calidad de SGD [17]. En el ámbito del procesamiento de lenguaje natural, debido a que los modelos son costosos de entrenar, suele recurrirse a optimizadores como Adam o similar.

2.4. Mejoras del modelo

Tras el entrenamiento, es posible que aparezcan problemas de rendimiento o simplemente sea posible introducir mejoras en el comportamiento de la red. El principal problema de un modelo generalmente es el sobreajuste. Existen también otros relativos a las redes neuronales, siendo el desvanecimiento del gradiente el más importante.

2.4.1. Sobreajuste

El sobreajuste u *overfitting* ocurre cuando se genera un modelo que obtiene resultados muy buenos para el conjunto de datos con el que se ha entrenado pero falla al predecir sobre observaciones fuera de dicho conjunto. La esencia del sobreajuste se basa en que se ha extraído la

varianza residual de los datos (como el ruido) y se ha incluido en la estructura del modelo. Se trata de la incapacidad de generalización, que es justo el objetivo de un sistema de inteligencia artificial.

Por el contrario, el infraajuste (aunque sea menos común ya que es fácil detectarlo) ocurre cuando el modelo no puede aprender la estructura subyacente a los datos de entrenamiento. En el caso de las redes neuronales suele remediarse añadiendo más complejidad, aumentando el número de neuronas.

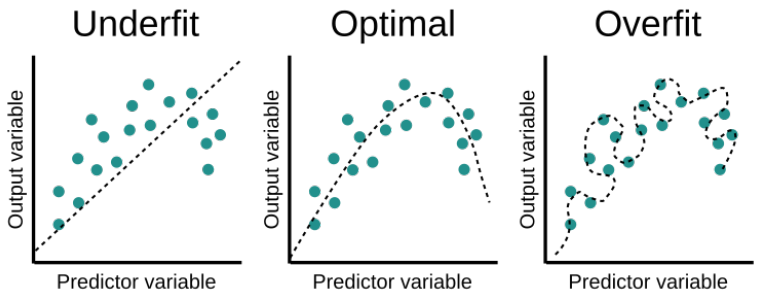


Figura 2.4.1: Ejemplo de infraajuste (izquierda), ajuste óptimo (centro) y sobreajuste (derecha). Fuente: [18].

Algunas técnicas para remediar el sobreajuste en redes neuronales son:

- Parada temprana (*early stopping*). Si se entrena durante demasiado tiempo, nos arriesgamos a que la red aprenda el ruido del conjunto de entrenamiento. Para evitarlo, es recomendable dejar de entrenar en la época donde el comportamiento sobre el conjunto de test empeore.

- Regularización. Esta técnica reduce la complejidad del modelo añadiendo una penalización a la función de coste. Existen dos técnicas de regularización: L1 y L2. La primera busca minimizar el valor absoluto de los parámetros sumando a la función de coste $\lambda \sum_{i=1}^{n} |\theta_i|$. La segunda minimiza el la suma cuadrada de los parámetros con $\lambda \sum_{i=1}^{n} \theta_i^2$. En estas fórmulas, θ son los parámetros del modelo y λ es la intensidad de regularización (los parámetros se acercan a 0 cuando crece λ).
  L1 es robusta a valores atípicos y tiende a generar un modelo más sencillo, mientras que L2 es capaz de aprender patrones más complejos a costa de ser más sensible a *outliers*.

- *Dropout*. Técnica sencilla que se basa en eliminar al azar (con una probabilidad prefijada) neuronas en la red durante el proceso de entrenamiento. Eliminar diferentes conjuntos de neuronas equivale a entrenar redes neuronales diferentes. Estas redes sobreajustan de

maneras distintas, por lo que el efecto del *dropout* reducirá el sobreajuste (ver Figura 2.4.2).

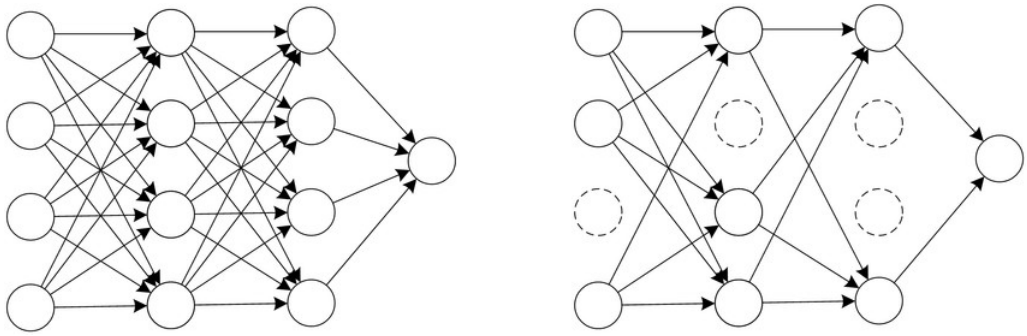


Figura 2.4.2: Ejemplo de *dropout*. Fuente: [19]

- Aumento del conjunto de datos. Esta opción es sencilla pero normalmente muy complicada en la práctica. Es raro disponer de más datos aparte de los que ya se tiene, por lo que se recurre a aumentar artificialmente el conjunto de datos (*data augmentation*). En redes neuronales basadas en imágenes es posible realizar rotaciones, cambios de resolución o recortes. En tareas de NLP (*Natural Language Processing* o procesamiento del lenguaje natural), como veremos a continuación en la Sección 2.5, no es tan directo y debe de hacerse con cuidado para no modificar el significado del texto.

La estrategia más usada para evitar el sobreajuste tanto en redes neuronales como en cualquier modelo predictivo es la simplificación del modelo. En nuestro caso, consiste en reducir las capas de la red o disminuir el número de neuronas por capa.

2.4.2. Desvanecimiento del gradiente

El desvanecimiento del gradiente es un problema propio de las redes neuronales que es propenso a aparecer en el ámbito del aprendizaje profundo debido a errores de redondeo. Cuando se entrena una red mediante un método de gradiente y retropropagación, cada uno de los pesos recibe una actualización proporcional a la derivada parcial de la función de coste con respecto al peso actual. El problema aparece cuando el gradiente es demasiado pequeño, evitando que el peso modifique su valor. En los casos extremos, la red puede parar de entrenar. Este fenómeno suele darse cuando las funciones de activación tienen gradientes que toman valores en rangos pequeños como $[0, 1]$ (sigmoide o tangente hiperbólica entre otras) [20].

De la misma forma, cuando se usan funciones cuyos gradientes toman valores grandes, aparece la explosión del gradiente. Existen algunas soluciones para este problema:

- Reducir las capas de la red.
- Inicialización *ad hoc* de los pesos. Normalmente se inicializan aleatoriamente. Empleando más tiempo es posible encontrar una combinación que no provoque el desvanecimiento.
- Otras funciones de activación cuyas derivadas no sean demasiado pequeñas, como ReLU.
- *Batch normalization*. Consiste en normalizar la entrada de las funciones de activación para que la salida no alcance los extremos de la misma. De esta forma, la derivada no tomará un valor bajo.

2.5. Aumento de datos

En tareas de procesamiento de lenguaje natural es muy común disponer de una gran cantidad de datos. Para obtener un modelo de lenguaje obtener grandes textos como Wikipedia o libros en el ámbito de estudio son suficientes para entrenar una red capaz de generar texto. Sin embargo, cuando nuestro objetivo es clasificar reseñas de películas, detectar *spam* o etiquetar biografías es complicado encontrar conjuntos de datos etiquetados. Normalmente se dispone de unas pocas observaciones y en los últimos años se ha recurrido a los aumentos del conjunto de datos de forma artificial.

Nos referimos al aumento de datos o *data augmentation* como las estrategias para incrementar la diversidad en el conjunto de entrenamiento sin recoger explícitamente nuevos datos [21]. Comparado con el procesamiento de imágenes, estas técnicas aplicadas al NLP resultan novedosas debido a que la naturaleza discreta del lenguaje y sus reglas gramaticales introducen ruido que dificulta generar modelos robustos.

Un aumento de datos ideal ha de ser sencillo de implementar y mejorar el rendimiento de la red entrenada. Los métodos disponibles hoy en día obligan a renunciar a una de las dos características. Dependiendo del modelo hay que buscar una técnica que mantenga un balance entre rendimiento y sencillez. A continuación veremos algunas técnicas.

2.5.1. Aumento de datos base

El aumento de datos base o *Easy Data Augmentation* [22] utiliza cuatro operaciones básicas sobre el texto que previenen el sobreajuste y ayudan a entrenar modelos más robustos.

- Sustitución de sinónimos. Consiste en escoger un número aleatorio de palabras que no sean vacías o *stopwords* (artículos, pronombres, preposiciones) y reemplazarlas por un

sinónimo también elegido al azar.

- Inserción aleatoria. Encontrar sinónimos de palabras no vacías e insertarlos en posiciones aleatorias en la oración. Por ejemplo, tomando los sinónimos de técnica y aumento:

```
(1) Esta sección explica una técnica de aumento de
    datos en NLP.
(2) Esta sección explica estrategia una técnica de
    aumento de datos incremento en NLP.
```

  Un lector humano tendría dificultad para entenderlo, pero es una forma de aclarar el contexto para una red neuronal debido a que se incluyen nuevas palabras que ayudan a afianzar el contexto de la oración a costa de perder algo de coherencia.

- Intercambio aleatorio. Se eligen dos palabras aleatorias en la oración y se intercambian sus posiciones.

- Borrado aleatorio. Eliminar cada palabra de la oración con una probabilidad fijada.

2.5.2. Traducción inversa

Con la mejora de los traductores automáticos, este método se ha popularizado durante los últimos años. La traducción inversa o *back translation* consiste en traducir el texto original a otro idioma, tomar el resultado y traducirlo de vuelta al idioma original, duplicando de forma efectiva nuestros datos [23]. Un ejemplo sencillo con el traductor de Google:

1. (ES) Me he comprado un ordenador.
2. (FR) J'ai acheté un ordinateur.
3. (ES) Compré una computadora.

2.5.3. Modelos generadores

Finalmente, una de las técnicas más actuales consiste en emplear modelos de lenguaje de gran tamaño como BERT, RoBERTa o GPT-3. Se les alimenta con el texto del que se dispone inicialmente y generan nuevas oraciones. Es importante prestar atención a los resultados que proporcionan puesto que abusar de esta técnica puede hacer que se pierda el significado del texto original y conduzca a un resultado peor que el inicial.

Existen muchas otras estrategias para realizar *data augmentation*, las que se han explicado cubren desde métodos sencillos hasta grandes modelos generadores. Es común que dependiendo del problema que nos encontremos unos métodos de aumento de datos sean mejores que otros, por lo que es recomendable probar varios y elegir el que mejor funcione. Además, es muy importante no utilizar los datos aumentados como conjunto de validación para evitar el sobreajuste.

Finalmente, hay que destacar que aplicar los métodos anteriores en español dificulta la generación de nuevos datos debido a que muchas palabras tienen persona, género y número.

2.6. Procesamiento del lenguaje natural

El procesamiento de lenguaje natural o PLN ha pasado por tres épocas desde su invención en 1950 [24]. Inicialmente se hablaba de PLN simbólico (1950-1990), después de PLN estadístico (1950-2010) con la introducción de algoritmos de *machine learning* y la mejora en potencia computacional. Actualmente la técnica más usada es el PLN neuronal debido a que se obtienen muy buenos resultados en muchas tareas de lenguaje natural como la modelización de lenguajes o el *parsing* (identificación de estructuras sintácticas).

Debido a que los métodos estadísticos requerían técnicas de ingeniería de datos muy elaboradas y al aumento de disponibilidad de GPUs para paralelizar el trabajo, la gran mayoría de los modelos de lenguaje natural se basan en redes neuronales artificiales.

El modelo de lenguaje es la parte más importante de una red para PLN. Se trata de una configuración de pesos y bias concreta que permite a la red «entender» un idioma. Al alimentar a la red con una cadena de texto, ésta intentará predecir la siguiente palabra, que deberá ser sintácticamente correcta si el modelo ha sido adecuadamente entrenado. A partir de aquí, es posible adaptar el funcionamiento de la red para otras tareas de clasificación o regresión. A continuación, veremos cómo se construye un modelo de lenguaje.
Partiendo de que una red neuronal acepta como entrada una variable categórica que se usa como variable independiente (en nuestro caso, una secuencia de palabras):

1. Se crea una lista de todos los niveles de esa variable categórica. En PLN, esta lista se denomina vocabulario o *vocab*.
2. Cada nivel de la variable se reemplaza por su índice en el vocabulario.
3. Se crea una matriz de *embeddings*. Tendrá una fila por cada elemento del vocabulario.

4. Esta matriz será la primera capa de la red neuronal. Se trata de una forma rápida y eficiente de recibir las entradas ya que se introducen los índices creados en el paso 2 [25].

Aplicando esta idea a un texto completo, se concatenan todos nuestros documentos en una gran secuencia de texto que separamos en palabras (lo que genera una lista de *tokens*). Como en un modelo de inteligencia artificial habitual, contamos con variables dependientes e independientes. La variable independiente será la cadena de palabras desde la primera hasta la penúltima y la variable dependiente, la cadena desde la segunda hasta la última.

El vocabulario estará formado por una lista de palabras que pertenecen al idioma que queremos modelizar. Normalmente se utilizan entre 30000 y 60000 palabras diferentes. En las siguientes secciones procedemos a explicar los pasos descritos anteriormente con más detalle.

2.6.1. Tokenización

El paso de convertir una gran secuencia de texto en una lista de palabras o *tokens* no es una tarea sencilla. La puntuación, las palabras compuestas, las mayúsculas y otros símbolos deben ser tratados con cuidado para evitar romper la estructura del texto y mantener su significado. Dependiendo del idioma con el que queramos entrenar al modelo, encontramos tres estrategias para tokenizar:

- Palabras completas. Consiste en separar las frases por espacios, palabras o partes con significado completo. Esta última varía para cada idioma, donde se aplican reglas específicas para separar las partes del significado cuando no hay espacios (como por ejemplo convertir «pasarlo» en «pasar lo»).
- Sub-palabras. Dividir las palabras en sub-palabras basándose en las ocurrencias más comunes. Por ejemplo, «ordenador» se podría tokenizar como «ordena dor».
- Caracteres. Separar una frase en caracteres individuales.

La tokenización que mejores resultados da en español es la primera, ya que ofrece un balance entre sencillez y calidad de representación del idioma. Un ejemplo de un texto tokenizado con el método de palabras completas:

```
Vamos a pasarlo bien ordenando todos estos ficheros.
'Vamos', 'a', 'pasar', '##lo', 'bien', 'ordena', '##n', '##do',
'todos', 'estos', 'ficheros', '.'
```

El tokenizador puede variar dependiendo de la arquitectura de la red. Aquí observamos que utiliza el símbolo «##» para indicar que la palabra anterior formaba parte de la actual. En las

siguientes secciones se explican los tokens especiales que emplea cada arquitectura utilizada para indicar sucesos como el comienzo de una oración, una letra mayúscula o una palabra desconocida.

2.6.2. Numericalización

La numericalización es el proceso de traducir los tokens a números enteros. Esto es equivalente a categorizar una variable a gran escala. Primero, se crea una lista de todos los niveles posibles de la variable categórica *vocab*. Segundo, se reemplaza cada nivel con su índice en el vocabulario.

Para que el procedimiento sea computacionalmente factible, se introducen algunas condiciones que afectan mínimamente al modelo de lenguaje final pero que aceleran mucho el proceso de numericalización:

- Frecuencia mínima de aparición. Reemplazar cualquier palabra que aparezca menos de n veces en el texto por el token desconocido.
- Tamaño máximo del vocabulario. Reemplazar cualquier palabra que no se encuentre entre las m más comunes en el texto por el token desconocido.

Estos parámetros se fijan de forma arbitraria y dependen del problema, pero su valor habitual suelen estar entre 1 y 3 para n y 30000 y 60000 para m, como se ha mencionado en la Sección 2.6. De esta forma, se agrupan muchas palabras distintas que tienen muy poca influencia en el texto en una sola observación (token desconocido), reduciendo considerablemente el tamaño de la matriz de *embeddings*.

2.6.3. Creación de la variable dependiente

Para poder entrenar una red en el ámbito del lenguaje, hay que asegurarse de que las entradas tengan el mismo tamaño. Cuando se utilizan imágenes, se ajusta su resolución a un formato estándar. En el caso del texto, es algo más complicado ya que no podemos recortar o alargar las oraciones como queramos. Queremos lotes o *batches* (pequeños subconjuntos de frases) del mismo tamaño y que estén ordenados para que el modelo pueda predecir eficientemente cuál es la siguiente palabra. Por eso, cada lote debe retomar el texto donde lo dejó el anterior.

El tamaño de los lotes o *batch size* es uno de los hiperparámetros más importantes en una red neuronal. Por ejemplo, si contamos con una entrada de 1000 tokens y elegimos un *batch size* de 10, tendremos 10 entradas con 100 tokens cada una. Se trata de un compromiso entre velocidad de entrenamiento, calidad de ajuste y consumo de recursos. Un tamaño de lote grande significa

más consumo de memoria y mayor rapidez de entrenamiento de la red, ya que se procesa una gran cantidad de datos en paralelo. Si elegimos un *batch size* pequeño, el entrenamiento será más lento, se consumirá menos memoria y la calidad de ajuste será mejor [26]. Normalmente, se elige un tamaño de lote entre 2 y 32, considerando 64 como un tamaño grande.

Finalmente, la variable dependiente (y) se crea tomando la independiente (X) y desplazando un token una ventana que abarca un número de tokens igual al tamaño del lote, como observamos en el siguiente ejemplo simplificado de tamaño 9:

```
X: Mis amigos y yo intentamos reservar entradas para el
y: amigos y yo intentamos reservar entradas para el cine
```

2.6.4. Modelo de lenguaje

Una vez se dispone del conjunto de entrenamiento (variable dependiente e independiente) es posible entrenar el modelo de lenguaje. Un modelo de lenguaje es una distribución de probabilidad sobre un grupo de palabras. En la práctica, nos da la probabilidad de que una palabra en una secuencia de palabras sea válida. Por validez entendemos que la oración que se construye se parezca a la forma de escribir de las personas. Si el modelo está correctamente entrenado dará resultados coherentes y gramaticalmente correctos. Los dos problemas que aparecen en un modelo de lenguaje son:

- Distribución de probabilidades. Al haber muchas combinaciones de palabras distintas que no aparecen, su probabilidad acaba siendo la misma debido a que no es posible contar con todas ellas durante el entrenamiento.
- Contexto. Las palabras anteriores y posteriores a la que queremos predecir afectan a su distribución de probabilidad.

Existen dos tipos principales de modelos de lenguaje: métodos probabilísticos que aprenden las probabilidades de aparición de las palabras basándose en sus apariciones en el texto y los basados en redes neuronales. Nos centraremos en el segundo ya que es el que se va a utilizar.

Las redes neuronales resuelven el problema de la distribución de probabilidades por la forma en la que se codifican las entradas (matriz de *embeddings*). De esta forma el modelo de lenguaje es una función (como todas las redes neuronales) que dado un texto de entrada es capaz de predecir las probabilidades de la siguiente palabra. En este trabajo, el problema del contexto ha abordado con dos arquitecturas distintas: redes neuronales recurrentes y transformadores o *transformers*.

2.7. Redes neuronales recurrentes

Las redes neuronales recurrentes incorporan el concepto de «secuencia» al aprendizaje. Esto les permite aprender el concepto del tiempo haciendo uso de un estado interno o memoria para procesar secuencias de entradas [27]. El concepto de memoria secuencial es intuitivo: recitar el abecedario es sencillo, pero para hacerlo al revés es necesario haberlo practicado antes. Esta arquitectura es aplicable a cualquier tipo de datos, pero explicaremos su funcionamiento en el ámbito del procesamiento del lenguaje natural.

De una forma similar, las redes recurrentes incorporan este concepto mediante un bucle que alimenta información anterior a las capas ocultas 2.7.1. El bucle en cuestión no deja de ser una serie de capas conectadas que toman como entradas los elementos de la secuencia, como se ve en la Figura 2.7.2.

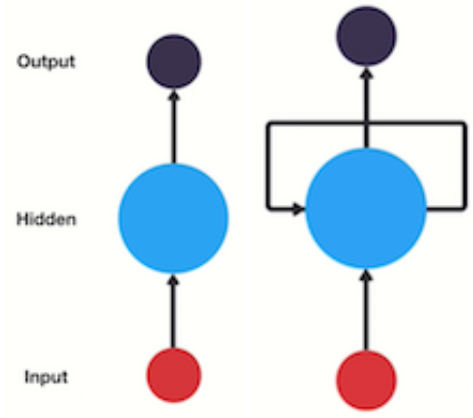


Figura 2.7.1: Red hacia delante (izquierda) y red recurrente (derecha). Fuente: [28].

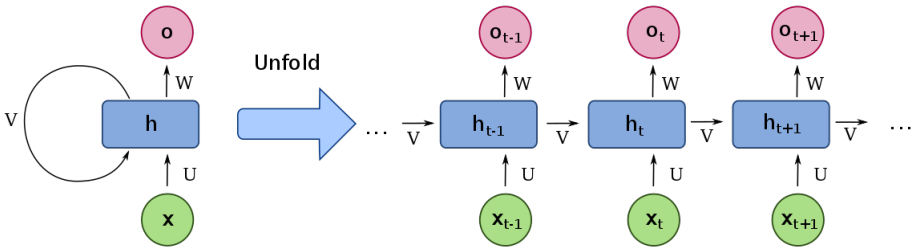


Figura 2.7.2: Desdoblamiento de la capa recurrente. Fuente: [29].

A medida que se recorren las entradas x_i, se produce una salida o_i y un estado oculto h_i que se envía a la siguiente capa. De esta forma, cuando se introduce una secuencia de n elementos

en una red de n capas, se produce una salida en el elemento n que contiene la información de todos los anteriores. La capa oculta tiene dos tareas principales:

- Mantener la información adecuada para la capa de salida y así predecir el token correcto.
- Retener en su «memoria» todo lo que ha ocurrido en la secuencia.

Cuando se aplican las redes recurrentes al análisis de textos aparecen algunas dificultades. El primer problema de las ideas anteriores es la inviabilidad de crear tantas capas como tokens. Si disponemos de un documento con 10000 tokens, necesitaríamos una red de 10000 capas. Cuando se llegase a la última palabra, necesitaríamos calcular todas las derivadas hasta la primera capa (según se ha visto en el algoritmo de retropropagación). A día de hoy, esto es imposible computacionalmente. Para solucionar este problema aparece la retropropagación en el tiempo truncada o *truncated backpropagation through time*. Consiste en entrenar la red con una secuencia de tamaño k_1 y después aplicar el algoritmo de retropropagación durante k_2 instantes de tiempo. De nuevo, es un compromiso entre tiempo de entrenamiento, calidad de ajuste y complejidad del problema. Un k_1 grande implica entrenar durante más tiempo, mientras que un k_2 grande resulta en desvanecimiento del gradiente [30].

Aumentar la capacidad de modelizar un problema en una red recurrente implica añadir más capas recurrentes. Esto significa que las activaciones de una capa recurrente alimentan una siguiente capa recurrente. Sin embargo, esto rara vez da resultados aceptables ya que vuelve a aparecer el problema del desvanecimiento o explosión del gradiente (ver Sección 2.4.2). En el caso de las redes recurrentes, suele ocurrir el primero y la información modelizada en los instantes de tiempo más alejados se pierde porque el gradiente propagado hacia esas capas es nulo.

Para remediar el desvanecimiento, se ha propuesto la utilización de dos tipos de capas: unidades recurrentes cerradas (*gated recurrent units, GRUs*) y capas de gran memoria a corto plazo (*long short-term memory*, LSTM). Las diferencias son pequeñas, pero en este trabajo se ha usado un modelo con capas LSTM debido a que dan mejores resultados en conjuntos de datos grandes a costa de un entrenamiento más lento.

2.7.1. Long Short-Term Memory

La arquitectura LSTM fue introducida en 1997 [31]. Cuenta con dos estados ocultos a diferencia de una red recurrente tradicional, donde el estado oculto es la salida de la red en el instante anterior.

Si consideramos la oración "Estrenaron una buena película, por lo que mis amigos y yo queremos

ir al cine." es importante que el modelo sea capaz de retener la palabra «película» para predecir «cine». En la práctica, las redes recurrentes son malas a la hora de retener información que aparece en los inicios de oración. Debido a esto, se incluye un nuevo estado oculto en las LSTM: el estado celda o *cell state*. Dicho estado se encargará de mantener la memoria a corto plazo mientras que el estado oculto de la RNN se centrará en predecir el siguiente token.

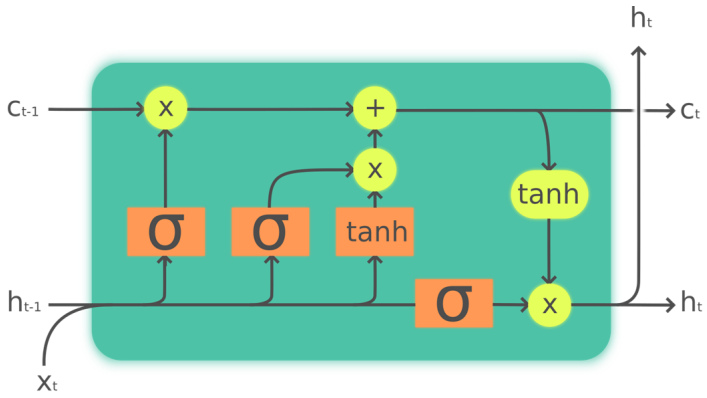


Figura 2.7.3: Estructura interna de una LSTM. Fuente: [32].

En la Figura 2.7.3 observamos las siguientes partes:

- Entrada x_t.
- Estado oculto anterior y actual h_{t-1} y h_t respectivamente.
- Estado celda anterior y actual c_{t-1} y c_t respectivamente.
- Capas o puertas en naranja con funciones de activación sigmoide σ y tangente hiperbólica *tanh* vistas en la Sección 2.2.
- Operaciones elemento a elemento en amarillo (suma, multiplicación y tangente hiperbólica).

Sabiendo esto, el procedimiento interno de la celda es el siguiente:

1. Se unen x_t y h_{t-1} apilándolos en un tensor o matriz.
2. La primera puerta (de izquierda a derecha), denominada puerta de olvido o *forget gate* es una capa lineal cuyas salidas estarán en el rango $[0, 1]$. Se multiplica este resultado por c_{t-1} para decidir qué información retener (valores cercanos a 1) y cuál olvidar (cercanos a 0).
3. La segunda puerta es la puerta de entrada o *input gate*, que se combina con la tercera

(puerta-celda o *cell gate*) para actualizar el estado celda c_t. La segunda puerta reemplaza información que la primera ha eliminado de forma análoga (valores de 0 a 1) mientras que la tercera puerta determina el valor de la información reemplazada en el rango $[-1, 1]$. Finalmente, se suma el resultado al estado celda c_t.

4. La última puerta se llama puerta de salida (*output gate*). Toma los datos del estado celda c_t y decide qué información usar para generar el estado oculto h_t de forma análoga al punto anterior.

Las redes LSTM proporcionan una buena solución al problema del desvanecimiento del gradiente y son capaces de modelar razonablemente bien el contexto de las oraciones. Sin embargo, son bastante propensas al sobreajuste (ver Sección 2.4.1). Una solución es el aumento de datos artificial explicado en la Sección 2.5. No es una técnica lo suficientemente fiable a día de hoy, por lo que no debe ser nuestra primera opción. Existen técnicas de regularización generales (*drop-out*) y específicas de las LSTM (regularización de activación y regularización de activación temporal) que generan un nuevo modelo más resistente al sobreajuste: AWD-LSTM. Estas mejoras se introdujeron inicialmente en [33] y serán el primer modelo utilizado en este trabajo.

La regularización de activación busca reducir los valores finales de las activaciones en vez de los pesos. De esta forma, se evitan probabilidades para los tokens demasiado altas. La penalización consiste en sumar a la función de pérdida la media de los cuadrados de las activaciones.

La regularización de activación temporal se relaciona con la predicción de palabras: las salidas de la red deberían tener sentido cuando se leen en orden. Por eso, se añade una penalización a la función de pérdida para que la diferencia entre dos activaciones consecutivas sea la menor posible.

2.8. Transformadores

Un transformador (*transformer*) es un modelo de aprendizaje profundo introducido en 2017 por el equipo de inteligencia artificial de Google [34]. Con el paso del tiempo ha ido sustituyendo a las redes recurrentes LSTM ya que presentan mejores propiedades a la hora de entrenar y generar predicciones. Las principales características que diferencian a los transformadores de las redes recurrentes son:

- Introducción del mecanismo de atención, que permite tanto dar contexto a las palabras de una secuencia como procesar dicha secuencia en el cualquier orden. La función de atención permite valorar las conexiones entre las palabras. Por ejemplo: “El perro no caminaba por

el campo porque estaba cansado". «Cansado» generaría un valor de atención alto con «perro» pero no con «campo».

- Aumento de la capacidad de paralelizar el entrenamiento, lo que permite entrenar con conjuntos de datos más grandes y generar modelos preentrenados como BERT o GPT que se pueden adaptar a multitud de problemas.

2.8.1. Arquitectura

Un transformador se basa en una estructura codificador-decodificador (*encoder-decoder*). El codificador traduce una entrada formada por una secuencia de símbolos $\mathbf{x} = (x_1, ..., x_n)$ a una secuencia de valores continuos $\mathbf{z} = (z_1, ..., z_n)$. Dado $\mathbf{z}$, el decodificador genera una secuencia de salida $\mathbf{y} = (y_1, ..., y_m)$ de símbolos, uno en cada iteración. En cada paso, el modelo consume los símbolos anteriores como entradas adicionales a medida que se genera el texto.

Generalmente, se sigue la arquitectura de la Figura 2.8.1, donde aparece el codificador a la izquierda y el decodificador a la derecha.

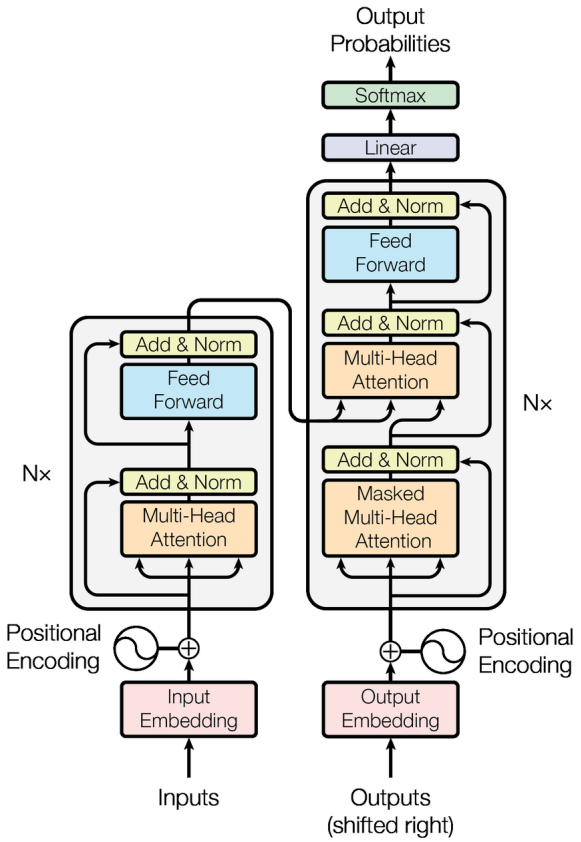


Figura 2.8.1: Arquitectura de un transformador. Fuente: [34].

- Codificador. Se compone de N capas idénticas. Cada una tiene dos sub-capas. La primera es una capa de atención y la segunda una capa *fully-connected* hacia adelante. Las dos sub-capas se conectan mediante una suma con la entrada y su posterior normalización: $Norm(x + subcapa(x))$.
- Decodificador. De nuevo, se compone de otras N capas idénticas. Como se observa en la Figura 2.8.1, tiene las mismas sub-capas que el codificador. Se le añade una capa de antención enmascarada para evitar que entradas anteriores «observen» entradas posteriores. Observamos que las salidas se desplazan hacia la derecha una posición como se explica en la Sección 2.6.

A continuación se explica el funcionamiento de la función de atención de un transformador. De forma análoga a un sistema de búsqueda como el buscador de vídeos de YouTube, el motor mapea la consulta del usuario a un conjunto de claves como pueden ser el título del vídeo, su descripción o puntuación (variables o características). Finalmente se presentan los vídeos que mejor se ajusten a las variables en forma de valores. Un mecanismo o función de atención se basa en mapear una consulta q y un conjunto de pares clave-valor $k : v$ a una salida. La consulta, las claves, los valores y las salida son vectores que se organizan en matrices (Figura 2.8.2) donde cada fila es un vector. La consulta y las claves tienen tamaño d_k mientras que los valores son de tamaño d_v.

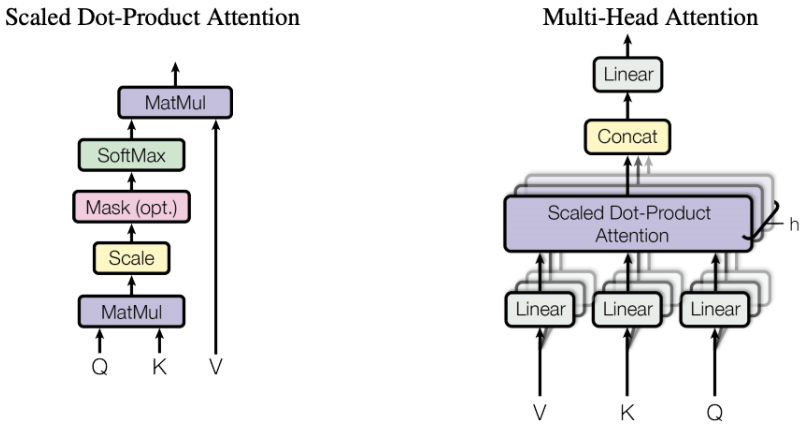


Figura 2.8.2: Interior de una capa de atención (izquierda) y capa de atención en paralelo (derecha). Fuente: [34].

En cada capa de atención (izquierda en la Figura 2.8.2), el transformador tiene tres matrices de pesos: los pesos de las consultas W_Q, los pesos de las claves W_K y los pesos de los valores W_V. Cada token i de la matriz de *embeddings* x_i se multiplica con cada una de las matrices

para producir un vector de consulta $q_i = x_i W_Q$, un vector de claves $k_i = x_i W_K$ y un vector de valores $v_i = x_i W_V$. A continuación, los pesos de la atención se calculan utilizando los vectores consulta y clave: el peso de atención a_{ij} desde el token (palabra) i hasta j es $q_i \cdot k_j$. Se dividen por $\sqrt{d_k}$ para estabilizar los gradientes durante el entrenamiento y se normalizan los pesos con la función softmax. Las matrices W_Q y W_K son distintas y permiten que la atención no sea simétrica: si una palabra i tiene un valor de atención alto hacia la palabra j ($q_i \cdot k_j$ grande), no significa que la palabra j tenga una atención alta hacia i. Multiplicar por el vector v_i nos permite asignar un *embedding* al resultado obtenido, es decir, obtener el token del valor de atención correspondiente. Finalmente, las consultas, claves y valores se organizan en matrices para aprovechar las optimizaciones en el cálculo de matrices.

$$Atencion(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.8.1}$$

En la arquitectura final del transformador, se concatenan varias capas de atención en paralelo (derecha en Figura 2.8.2) para poder modelizar los valores de atención de las palabras desde varias perspectivas. Por último, las salidas del decodificador pasan por una capa lineal y una función softmax que devuelve las probabilidades de cada palabra del vocabulario, igual que en el modelo LSTM.

En este trabajo se utilizará una arquitectura basada en transformadores para compararla con la LSTM.

Capítulo 3

Contexto

En esta sección se profundiza en los conceptos necesarios para entender el análisis de consejeros (contexto financiero) y en las herramientas que se han utilizado para ello (contexto tecnológico).

3.1. Contexto financiero

3.1.1. Órganos de gobierno de las sociedades anónimas

Las empresas necesitan dividirse en órganos especializados para asegurar su buen funcionamiento. Estos órganos se encargan de funciones de administración y gestión de la compañía. Los dos más importantes son la Junta General de Accionistas y el Consejo de Administración [35].

Junta General de Accionistas

Es el órgano de la sociedad integrado por todos sus accionistas. Periódicamente sus miembros se reúnen para decidir sobre asuntos de su competencia. Entre dichos asuntos destacan la valoración de la gestión de la sociedad, la aprobación de cuentas, aumentos o reducciones de capital, modificaciones de los estatutos sociales o la emisión de obligaciones. También se encarga del nombramiento, ratificación y cese tanto de los miembros del Consejo de Aministración como de los auditores de cuentas. Finalmente, la Junta General toma decisiones sobre la adquisición de acciones propias, la política de remuneración de los consejeros, la transformación o la adquisición de activos esenciales para la sociedad.

La Junta General de Accionistas puede ser ordinaria o extraordinaria. La primera está obligada a reunirse dentro de los seis primeros meses de cada ejercicio o año contable. En ella se debe tratar

como mínimo la aprobación de las cuentas del ejercicio anterior, la distribución de beneficios y la gestión del Consejo de Administración. Todas las demás reuniones se consideran extraordinarias.

El acceso a la Junta General de Accionistas se permite a todos los accionistas con acciones inscritas a su nombre. No obstante, los estatutos de algunas sociedades suelen establecer un número mínimo de acciones (no superior a 1000) para poder asistir. Aquellos accionistas que no lo alcancen pueden agruparse eligiendo un representante.

La participación de los accionistas en la Junta es esencial para el funcionamiento de la empresa, ya que les corresponde decidir sobre cuestiones relevantes en las que ni siquiera el Consejo de Administración tiene poder de decisión. Habitualmente, el peso del voto de un accionista es directamente proporcional a su porcentaje sobre el total de acciones. Debido a esto, se recomienda a los pequeños accionistas la opción de asociarse para obtener una voz mayor en la Junta.

Consejo de Administración

El Consejo de Administración de una sociedad cotizada se constituye a partir de tres clases de consejeros:

- Ejecutivos. Desempeñan funciones de alta dirección o son empleados de la sociedad o de otras sociedades del mismo grupo.
- Dominicales. Se designan debido a su condición de accionistas o a propuesta de un accionista importante. Su cometido es conseguir que la la gestión de la empresa sea correcta y la inversión de recursos sea rentable.
- Independientes. Se caracterizan por llevar a cabo sus funciones sin verse influenciados por el equipo gestor o los accionistas de referencia. Su elección gira en torno a su capacitación profesional, solvencia y experiencia.

Las funciones del Consejo de Administración se centran en la gestión, dirección, administración y representación de la sociedad. Dichas funciones se pueden resumir en:

- Aprobar el plan estratégico, los objetivos de gestión y presupuesto anuales, la política de inversiones y de financiación, la política de responsabilidad social corporativa y la política de dividendos.
- Determinar tanto la política de control y gestión de riesgos como la supervisión de los sistemas de información y control.
- Determinar la política de gobierno corporativo de la sociedad.

- Convocar la Junta General de Accionistas y ejecutar sus decisiones.
- Formular las cuentas anuales, el informe de gestión y la propuesta de aplicación del resultado.
- Redactar el Informe Anual de Gobierno Corporativo. Éstos serán los documentos que se extraigan y analicen con la red neuronal.
- Nombramiento y destitución de directivos que dependen del Consejo.
- Aprobar inversiones u operaciones de alto coste, carácter estratégico o riesgo fiscal siempre y cuando su aprobación no corresponda a la Junta.
- Definir la estructura del grupo de sociedades del que la sociedad sea entidad dominante.
- Aprobar las operaciones que la sociedad realice con consejeros o accionistas con una participación significativa.

3.1.2. Información y composición de los consejos de administración

Como se ha dicho anteriormente, el Consejo de Administración tiene asignadas las competencias de gestión y representación de la sociedad. Se recomienda, según el Código de Buen Gobierno, que el consejo realice sus funciones con el propósito de conseguir un negocio rentable y sostenible a largo plazo, promoviendo su continuidad y maximización del valor económico de la empresa [36]. Cada año, la Comisión Nacional del Mercado de Valores redacta el Informe Anual de Gobierno Corporativo (IAGC), que recoge información sobre las empresas que cotizan en el Mercado Continuo de la Bolsa de Madrid. Se explica con más detalle en la Sección 3.1.3.

Según el IAGC, a finales de 2020, había 1243 consejeros en los órganos de administración de las sociedades cotizadas españolas, siendo 435 de compañías del IBEX 35. El tamaño de un consejo de administración varía en función de la capitalización bursátil de la sociedad. El tamaño medio de los consejos de administración del IBEX 35 ronda en torno a 13 consejeros, 10 para las empresas con más de 500 M € de capital y 8 para las empresas con menos de 500 M € de capital.

De nuevo según las recomendaciones del Código de Buen Gobierno, el porcentaje de consejeros dominicales sobre el total de consejeros no ejecutivos no debería ser mayor que la proporción existente entre el capital de la sociedad representado por dichos consejeros y el resto del capital. En cuanto a los consejeros independientes, se recomienda que representen al menos la mitad del total de consejeros. A parte del reparto de consejeros, en el informe de gobierno corporativo se recogen también otras variables de interés entre las que destacan:

- Antigüedad de los consejeros. La antigüedad ha ido aumentando con los años hasta alcanzar 7.4 años de media en 2020.
- Diversidad de género. Debido a la insuficiente cantidad de mujeres en los consejos, se intenta promover objetivos que la aumente. Desde 2017 a 2020, el porcentaje de mujeres ha pasado del 19 % al 26 %.
- Edad de los consejeros. Introducida en 2018, la edad media de los consejeros es de 60.5 años en 2020.
- Participación en varios consejos. Es común encontrar consejeros que forman parte de más de un consejo de administración. En 2020, un total de 1128 personas ocupaban 1243 puestos de consejeros.
- Motivos de cese de consejeros. La explicación más frecuente de cese se relaciona con motivos personales, profesionales o con el buen gobierno.
- Experiencia y habilidades. No se definen explícitamente, pero en los informes se incluyen breves biografías de los consejeros que explican su experiencia laboral, estudios y especialización.

Los cargos individuales más importantes dentro del consejo de administración son los siguientes:

- Presidente del consejo. Normalmente, si el presidente tiene la condición de consejero ejecutivo, es necesario nombrar un consejero coordinador entre los consejeros independientes. Tienen funciones como el mantener contactos entre inversiones y accionistas centrándose en materia de gobierno corporativo o coordinación del plan de sucesión del presidente.
- Vicepresidente del consejo. Se designa un vicepresidente para cada una de las tres categorías de consejeros: ejecutivo, dominical e independiente. Durante los últimos años, se han reducido las sociedades que contaban con vicepresidente.
- Secretario del consejo. Su función es velar por que las actuaciones del consejo sean conformes a las leyes y sus normas de desarrollo, así como con los eatatutos y reglamentos internos de la sociedad. Facilita el buen desarrollo de las sesiones, asiste al presidente para que los consejeros reciban información importante, maneja la documentación y deja constancia de las sesiones en los libros.

3.1.3. Datos sobre la muestra

En este apartado se explica la información referente a la muestra y sus fuentes. Un análisis orientado a la informática de su obtención, procesamiento y resultado se detalla en la Sección 6.1.3.

Los datos principales de este trabajo se han obtenido de la Cominsión Nacional del Mercado de Valores o CNMV. Se trata de un organismo regulador independiente fundado en 1988 que se encarga de la supervisión de los mercados de valores en España. Su objetivo es velar por la transparencia de estos mercados y la correcta formación de precios en los mismos, además de la protección de los inversores.

La CNMV compila anualmente los informes de Gobierno Corporativo que redacta cada compañía. Dichos informes contienen un análisis de la importancia de las buenas prácticas de gobierno corporativo para aumentar la eficacia económica y reforzar la confianza de los inversores. Las empresas deben rellenar un documento que contiene la siguiente información:

1. Estructura de la propiedad. Datos relativos a la sociedad, su capital y acciones.
2. Estructura de la administración de la sociedad. Detalles sobre el consejo de administración, sus integrantes y características. En esta sección aparecen las biografías con los perfiles profesionales que se van a analizar.
3. Operaciones vinculadas. Aquellas que se realizan entre personas físicas o jurídicas entre las cuales existe un determinado grado de vinculación entre ellas.
4. Sistemas de control de riesgos. Describe la política de riesgos de la sociedad junto con la materialización de dichos riesgos.
5. Junta general. Contiene información sobre la Junta General de Accionistas y sus diferencias con el régimen previsto en la Ley de Sociedades Anónimas.
6. Seguimiento de las recomendaciones de gobierno corporativo. Se específica el grado de seguimiento respecto a las recomendaciones del Código de Buen Gobierno.
7. Otras informaciones de interés. Aquí se incluye cualquier principio o aspecto relevante relativo a las prácticas de gobierno corporativo aplicado por la sociedad que no haya sido abordado en el informe.

En este trabajo se extraen 1835 informes, que se procesan para obtener un conjunto etiquetado. En dicho conjunto aparecen 896 consejeros independientes que pertenecen a 134 empresas distintas en un total de 1028 biografías. Los informes datan desde 2003 hasta 2020 y conforme

se avanza en el tiempo la CNMV recoge información adicional de las empresas, modificando ligeramente la disposición de los datos en el documento. Las biografías se etiquetan siguiendo los perfiles profesionales descritos en el apartado siguiente, donde cada perfil toma un valor en el rango $[0, 1]$.

3.1.4. Diversidad de los consejos

Durante los últimos años, se ha buscado una mayor diversidad en los consejos de administración de las empresas. Directivos con diferentes perfiles de negocio y socio-económicos proveen un mayor rango de perspectivas al consejo a la hora de monitorizar y aconsejar. Sin embargo, este enfoque se centra, en su mayor parte, en la diversidad étnica y de género. La diversidad de habilidades del consejo no ha recibido demasiada atención.

El perfil profesional de un consejero es un componente vital para el consejo, ya que mejora enormemente su capacidad de monitorización y asesoramiento. Un consejo con una gran diversidad de habilidades contará con más puntos de vista y variedad de talento. Directivos con diferentes perfiles profesionales también aportarán una base de conocimiento más amplia junto con un rango de perspectivas y habilidades de resolución de problemas [37].

Como se ha comentado en la introducción, el análisis de los perfiles profesionales de los altos cargos de una compañía son importantes debido a su influencia en la toma de decisiones. Las variables más complicadas de evaluar son aquellas que son cualitativas como la experiencia en diferentes campos.

Se definen seis perfiles profesionales que resultan interesantes a la hora de analizar un consejero:

- Financiero. Se refiere a las personas con experiencia en el sector financiero, ya sea en instituciones como bancos, compañías de inversiones o la bolsa en general. Miembros del consejo expertos en esta disciplina son muy influyentes en decisiones de inversión o financiación.
- Ejecutivo/Consultor. Consejeros que han mantenido o mantienen posiciones de gestión en otras compañías o han llevado a cabo tareas importantes de asesoramiento. Esta cualidad ayuda a mejorar el valor de la firma con la experiencia en la industria.
- Auditor/Contable/Fiscal. Como su nombre indica, se trata de consejeros con experiencia como auditores, contables o en materias fiscales. Empresas con miembros en el consejo con estas cualidades han mostrado ser más cautelosas a la hora de tomar decisiones.
- Legal. Abogados o expertos legales entran en esta categoría. La presencia de consejeros

con estas características se asocia con una mayor calidad de la información financiera.

- Político. Se refiere a los miembros que mantienen o han mantenido cargos públicos de diferentes tipos, especialmente puestos políticos. Esto permite extender las relaciones de la empresa para un mayor beneficio, aunque también se ha observado que la toma de decisiones no es óptima.
- Académico. Consejeros con experiencia académica. Estas personas ofrecen un rol importante a través de asesoramiento o supervisión, lo que implica un mayor rendimiento en inversiones o en I+D.

Cuando una empresa busca consejeros independientes, suele buscar aquellos que mejor satisfagan sus necesidades. Normalmente, un consejero independiente supervisa y asesora al consejo. La empresa les asignará sus cargos teniendo en cuenta sus habilidades para llevar a cabo las anteriores funciones, basándose en su perfil profesional.

En secciones posteriores se explica la obtención de un conjunto de datos que contenga información sobre los atributos de los perfiles profesionales. Dicho conjunto contiene biografías de consejeros con atributos etiquetados por un experto y su predicción será el objetivo final del modelo.

3.2. Contexto tecnológico

3.2.1. Transfer learning

La transferencia de aprendizaje o *transfer learning* es una técnica de aprendizaje profundo que consiste en utilizar un modelo preentrenado, es decir, una red cuyos pesos ya han sido ajustados para una tarea y utilizarlo para resolver un problema distinto para el que fue construido.

Entrenar una red neuronal de gran tamaño es muy costoso. Incluso si se dispone de un *hardware* de alto rendimiento y un *software* eficiente, el consumo energético y los tiempos de entrenamiento en problemas complejos o con muchos datos son muy altos. Un modelo preentrenado en el ámbito de la medicina puede ser difícil de adaptar, pero en el PLN son de gran utilidad ya que el proceso de generación de un modelo de lenguaje se acorta enormemente. Es por eso que en los últimos años se han hecho públicos modelos de lenguaje preentrenados como AWD-LSTM, BERT o GPT. Basta con descargarse sus parámetros e incluirlos en un modelo vacío para obtener un modelo de lenguaje completo.

Sin embargo, no suele ser suficiente con tomar los pesos del modelo anterior para obtener

buenos resultados. Hay que tener en cuenta también el concepto de *fine-tuning*, afinado o entrenamiento específico de una red. Consiste en entrenar un modelo preentrenado durante un número de épocas para que se ajuste a la tarea que queremos realizar. Durante el afinado también se realiza el *freeze* y *unfreeze* (congelamiento y descongelamiento) de las capas del modelo. Este proceso implica no alterar los pesos, especialmente en las primeras capas, tras la etapa de retropropagación para conservar las características aprendidas por la red.

El proceso de afinado modifica los pesos de la red y normalmente añade capas específicas para resolver el nuevo problema. Por ejemplo, si tomamos un modelo de lenguaje entrenado con Wikipedia y queremos clasificar correos como spam o no spam, hemos de reentrenar el modelo preentrenado con datos relacionados con correos, además de añadir las capas necesarias para que realicen la clasificación binaria. El resultado será un modelo capaz tanto de «entender» el idioma con el que fue entrenado como de predecir palabras relacionadas con el spam. Este procedimiento se denomina *Universal Language Model Fine-tuning* (ULMFit) o afinado universal de modelos de lenguaje y se han obtenido excelentes resultados con él [38].

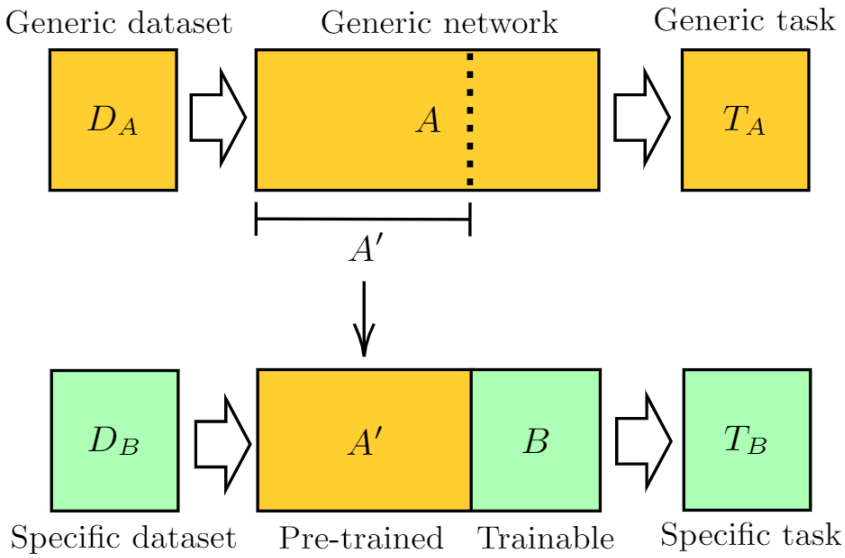


Figura 3.2.1: Esquema de *transfer learning*. Fuente: [39].

En la Figura 3.2.1 podemos ver que D_B sería nuestro conjunto de datos de spam, A' la parte de la red preentrenada y B las capas que añadimos para ajustarnos a la clasificación. Estas capas se denominan *head* o cabeza del modelo.

3.2.2. PyTorch

PyTorch es una de las principales librerías de aprendizaje profundo. Fue inicialmente desarrollada por el equipo de inteligencia artificial de Facebook y es de código abierto. Está diseñada tanto para la investigación como para la producción de aplicaciones, permitiendo crear redes neuronales profundas de forma eficiente gracias a dos características de alto nivel que lo diferencian de su competencia [40]:

- Computación tensorial acelerada por GPU. Permite aprovechar los núcleos CUDA (*Compute Unified Device Architecture*) de la tarjeta gráfica para realizar operaciones en paralelo en una escala mucho mayor a la de una CPU, acortando enormemente los tiempos de entrenamiento [41].
- Redes neuronales construidas sobre un sistema de diferenciación automática. Esto permite obtener el gradiente de la función de pérdida de forma rápida ya que las derivadas se calculan de forma automática con una precisión alta [42].

PyTorch permite trabajar con tensores. Un tensor es una matriz de múltiples dimensiones que contiene elementos de un solo tipo de datos. Para crearlo, se necesita una matriz, el tipo de datos (puede ser inferido) y el dispositivo donde se almacenará.

```
1 torch.tensor([[1,2,3],[4,5,6],[7,8,9]], dtype=torch.int32, device='cpu')
```

Algunas de las clases de las que haremos uso son *Dataset* y *DataLoader*. Además de otras utilidades para construir modelos.

Dataset

Un *Dataset* es una clase que representa un conjunto de datos, es decir, un mapeado de claves o índices a unos datos. Cuando queremos trabajar con un conjunto de datos propio, tenemos que utilizar esta clase abstracta de PyTorch. Para ello hay que:

- Definir el método `__init__()` que tomará los datos para construir el set así como otros parámetros que sean de interés para el problema.
- Reescribir el método `__getitem__()`, que permite obtener un dato dada su clave.
- Opcionalmente reescribir `__len__()` para obtener el tamaño del conjunto de datos.

Un pequeño ejemplo de *Dataset* que toma como entrada un *DataFrame* de Pandas con dos columnas (X, y):

```python
class ExampleDataset(Dataset):
    def __init__(self, data):
        self.df = data.reset_index()

    def __len__(self):
        return self.df.shape[0]

    def __getitem__(self, index):
        X = self.df.loc[index, 'X']
        y = self.df.loc[index, 'y']
        X = torch.tensor(X)
        y = torch.tensor(y)
        return X, y
```

DataLoader

Un *DataLoader* o cargador de datos utiliza un *Dataset* para generar un iterable sobre el conjunto de datos. Se trata del encargado de «alimentar» a la red neuronal. Los principales argumentos que utilizaremos en este trabajo son:

- `dataset` que hemos construido previamente.
- `batch_size` o tamaño del lote. Se trata del número de observaciones que se van a procesar en paralelo durante el entrenamiento.
- `num_workers` o número de núcleos que se reparten el trabajo a la hora de crear y utilizar el *DataLoader*.

Modelo

La arquitectura de las redes neuronales se define mediante el módulo `torch.nn`, que contiene los diferentes tipos de capas, funciones de pérdida y otras utilidades para construir la red. De nuevo, hay que reescribir una clase abstracta que normalmente cuenta con los siguientes métodos:

- `__init__()`, que se encarga de definir los elementos y los parámetros que conforman la red.
- `forward()`, donde se realiza la computación sobre los datos que proporciona el *DataLoader* y se generan las salidas de la red haciendo uso de los elementos definidos en el punto anterior.

A continuación se define una pequeña red de dos capas a modo de ejemplo:

```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.linear1 = torch.nn.Linear(100, 200)
        self.activation = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(200, 10)
        self.softmax = torch.nn.Softmax()

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        x = self.softmax(x)
        return x
```

Las capas son lineales y cuentan con funciones de activación ReLU y softmax.
Para entrenar el modelo hay que definir la función de pérdida y el optimizador. PyTorch proporciona ambos en los paquetes `torch.nn` y `torch.optim`, respectivamente. Existe una gran variedad de funciones y las más usadas se encuentran disponibles. En este trabajo se utilizarán `Adam` y `SGD` como optimizadores y `CrossEntropyLoss` y `MSELoss` como funciones de pérdida.

Una vez dispongamos de todos los elementos descritos anteriormente, podemos proceder a entrenar el modelo. No existe una función «*fit*» como en otras librerías de inteligencia artificial, por lo que hay que definir un bucle de entrenamiento. En dicho bucle se definen las épocas y se itera sobre el *DataLoader* generando los lotes. Se declara el modelo en estado de entrenamiento mediante el método `train()`, en el cual capas como *dropout* o normalización por lotes se comportan de forma distinta ya que se quiere entrenar. A continuación, los gradientes se fijan a cero con `zero_grad()` sobre el optimizador y se obtienen las predicciones del modelo. Finalmente se calcula el valor de la función de pérdida, se retropropaga el error con `backward()` y se actualizan los parámetros con `step()` sobre el optimizador. A continuación se ve un pequeño código de ejemplo:

```
loss_fn = CrossEntropyLoss()
optimizer = SGD(model.parameters(), lr)
for e in range(n_epoch):
    model.train()
    for X, y in dataloader:
```

```
        optimizer.zero_grad()
        y_pred = model(X)
        loss = loss_fn(y_pred, y)
        train_loss.backward()
        optimizer.step()
```

Una vez termina el entrenamiento, basta con obtener las predicciones de la misma forma que en el código anterior, pero esta vez llamando a `torch.no_grad()` y a `model.eval()` para indicar que los gradientes no han de actualizarse y establecer el modelo en modo «evaluación» respectivamente.

Por defecto, todo lo anterior ocurre en la CPU si no se especifica el dispositivo. Si se va a trabajar con un modelo de gran tamaño, es común mover tanto los datos como el modelo a la GPU para paralelizar las operaciones. Se hace de la siguiente forma:

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)
X.to(device), y.to(device)
```

3.2.3. Fastai

Fastai [43] es una librería de aprendizaje profundo basada en PyTorch que permite abstraer elementos de bajo nivel y facilita la preparación, construcción, entrenamiento y despliegue de modelos de redes neuronales. Es similar a la más conocida Keras de Tensorflow.

Evita la creación de clases y la definición de bucles a la hora de construir redes neuronales, además de ofrecer modelos preentrenados para *transfer learning*. A continuación se explican los elementos más importantes que se han utilizado en este trabajo a modo de comparativa con la sección anterior.

- `TextBlock` permite leer un conjunto de datos tanto de un DataFrame (normalmente Pandas) como de un directorio. Puede considerarse como un punto intermedio entre los datos y la interfaz de Fastai.
- `DataBlock`. Dado un `TextBlock` construye un bloque de datos, similar a un *dataset* de PyTorch. Cuenta con argumentos para indicar las variables dependientes y las independientes.

- `Dataloaders`. Cuando se le proporciona un `DataBlock` construye lo equivalente a un *dataloader* en PyTorch. Sus argumentos son el tamaño del lote y la longitud de la secuencia.
- `language_model_learner`. Construye un modelo de lenguaje a partir de un *dataloader* y una arquitectura que puede ser preentrenada. Aquí definimos el optimizador y la función de pérdida.
- `text_classifier_learner`. Similar al punto anterior, onstruye un clasificador a partir de un *dataloader* y una arquitectura.

Suponiendo que contamos con un DataFrame en Pandas con el texto en una columna llamada «*text*» y queremos construir y entrenar un modelo de lenguaje en Fastai, un ejemplo breve sería así:

```
dataloaders = DataBlock(TextBlock.from_df('text', is_lm=True),
                        get_x=ColReader('text')).dataloaders(df, bs=128, seq_len=72)
learn = language_model_learner(
        dataloaders, AWD_LSTM,
        opt_func=Adam, loss_func=CrossEntropyLoss,
        drop_mult=0.3, pretrained=False,
        metrics=[accuracy])
lr = learn.lr_find()
learn.fit_one_cycle(10, lr.valley)
```

En tan solo cuatro líneas es posible realizar la misma tarea que en el ejemplo de la sección 3.2.2. En el modelo de lenguaje especificamos la arquitectura AWD-LSTM, el *dropout* y que el modelo no es preentrenado junto con la precisión como métrica de rendimiento. Finalmente se fija la función de optimización (Adam) y la de pérdida (entropía cruzada).

Fastai permite encontrar una tasa de aprendizaje adecuada de forma automática con `lr_find()` y realizar el entrenamiento (durante 10 épocas en este caso) con `fit_one_cycle()`. Esta última función realiza también el *freezing* y *unfreezing* de las capas automáticamente.

Si queremos realizar un *fine-tune* sobre un modelo preentrenado, basta con declarar el nombre de nuestros modelos en el argumento `pretrained_fnames` de un `language_model_learner`.

3.2.4. HuggingFace

HuggingFace [44] es una librería de inteligencia artificial centrada en el ámbito del procesamiento de lenguaje natural que ofrece miles de modelos preentrenados basados principalmente en PyTorch.

Su principal proyecto es `transformers`, que pone a disposición de los usuarios modelos de lenguaje natural con arquitectura de transformadores con el propósito de ser afinados para cualquier tipo de tarea.

Utiliza las herramientas descritas en la sección 3.2.2 para proporcionar un modelo y mantener la sintaxis lo más cercana posible a PyTorch. Un ejemplo sencillo de uso sería el siguiente:

```
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=6)
model.classifier = nn.Sequential(nn.Linear(in_features=768, out_features=6, bias=True),
                                 nn.Sigmoid())
```

En el ejemplo anterior tomamos el modelo BERT de Google llamado «*bert-base-uncased*» con 6 etiquetas en la variable dependiente. Modificamos su cabeza (*classifier*) añadiendo una capa lineal y activación sigmoide. De esta forma se dispone de un modelo PyTorch listo para realizar un afinado.
HuggingFace dispone también de otras utilidades como tokenizadores y conjuntos de datos que se utilizarán en este trabajo más adelante.

3.2.5. Kaggle

Kaggle [45] es una comunidad online de científicos de datos y profesionales del aprendizaje automático. Permite a sus usuarios publicar conjuntos de datos y modelos de inteligencia artificial. Además, actúa como una plataforma de publicación de retos en el que dado un conjunto de datos y un límite de tiempo, los usuarios compiten por un ránking basado en los resultados de un modelo de *machine learning* que hayan programado.

La principal utilidad para este trabajo son sus máquinas virtuales. La versión gratis permite subir conjuntos de datos de gran tamaño y utilizarlos desde un ordenador remoto con una GPU válida para aprendizaje profundo. Es posible programar la ejecución de los entrenamientos en «segundo plano» en sesiones de 12 horas.

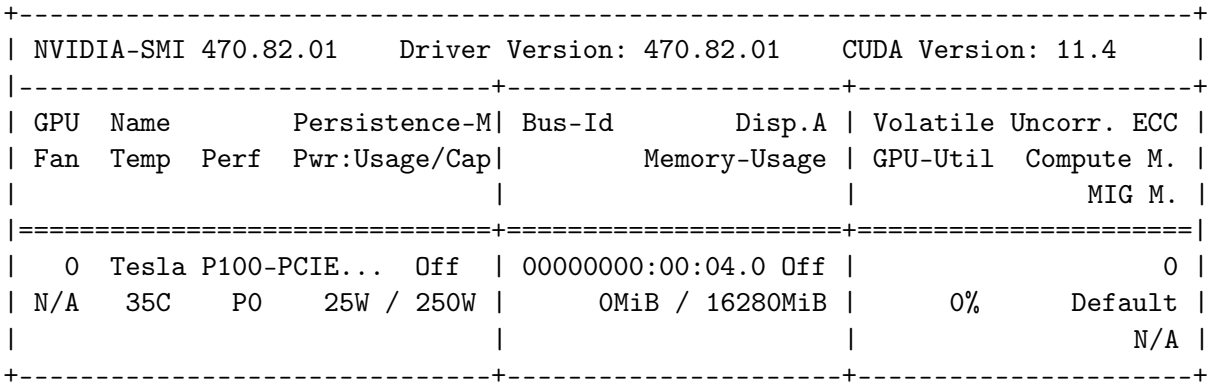


Figura 3.2.2: Resultado del comando `nvidia-smi`.

Una GPU de 16 GB de memoria (Figura 3.2.2) y la posibilidad de entrenar en segundo plano facilitará la realización del trabajo.

Capítulo 4

Planificación

4.1. Tareas

Inicialmente se divide el proyecto en 4 etapas, las cuales se han ampliado con sub actividades a medida que se ha desarrollado el trabajo.

1. Obtención del conjunto de datos principal. El objetivo del trabajo es crear una red neuronal que analice un perfil profesional dada una biografía. Por tanto, el conjunto de datos de los consejeros es esencial para llevar a cabo el proyecto. En esta etapa se aplican conocimientos de *web scraping* y de ingeniería de datos para obtener un conjunto de datos apto para ser etiquetado. Esta primera etapa se compone de:
   - Estudio de la página web.
   - Automatización de descargas.
   - Procesamiento de datos.
2. Lectura de documentación y pruebas. La segunda etapa se centra en el estudio de conceptos necesarios para llevar a cabo el trabajo. Se revisa lo estudiado durante la carrera y se adquieren nuevos conocimientos sobre redes neuronales y nuevas técnicas desarrolladas durante los últimos años. Finalmente, se aprende a utilizar Fastai y PyTorch resolviendo problemas sencillos. Los pasos a seguir en esta etapa son:
   - Lectura de biografía de redes neuronales.
   - Aprendizaje de PyTorch y Fastai.
   - Documentación del trabajo.

3. Modelo de lenguaje. En esta etapa, se crea el modelo de lenguaje para las dos arquitecturas elegidas. Primero, se obtiene el conjunto de datos de Wikipedia para entrenar una red en español desde cero. Después, se procesan libros de finanzas para realizar un *fine tune* sobre el modelo entrenado desde cero y el modelo preentrenado. Finalmente, se obtienen dos modelos de lenguaje capaces de predecir palabras faltantes dado un texto. Las partes definidas para esta fase son:
    - Obtención y preparación de Wikipedia.
    - Entrenamiento con Wikipedia.
    - Procesamiento de libros de finanzas.
    - Entrenamiento con libros.
    - Documentación del trabajo.
4. Modelo de regresión. En la cuarta y última etapa, se toma el modelo de lenguaje anterior y se modifica para que su salida sea una regresión múltiple sobre una serie de atributos. Con los datos obtenidos en la primera etapa se realiza de nuevo un afinado sobre los modelos. Finalmente se evalúan y se comparan sus resultados. Esta última etapa se ha dividido en:
    - Modificación y adaptación del modelo.
    - Entrenamiento con biografías.
    - Documentación del trabajo.

En el Cuadro 4.1.1 se presentan los tiempos estimados en horas para cada subactividad.

Actividad	Subactividad	Tiempo estimado (h)
Obtención de datos	Estudio de la página web	5
	Automatización de descargas	20
	Procesamiento de datos	25
Lectura de documentación y pruebas	Bibliografía de redes neuronales	20
	Aprender PyTorch/Fastai	25
	Documentación	30
Modelo de lenguaje	Obtención y preparación de Wikipedia	10
	Entrenamiento con Wikipedia	40
	Preparación de libros de finanzas	5
	Entrenamiento con libros	25
	Documentación	30
Modelo de regresión	Modificaciones y adaptación del modelo	20
	Entrenamiento	10
	Documentación	35
Total		300

Cuadro 4.1.1: Estimación de tiempo.

En la Figura 4.1.1 se encuentra un diagrama de Gantt que muestra los tiempos ocupados por cada actividad. En la etapa del modelo de regresión se realizan cambios en las arquitecturas de la red para mejorar su rendimiento. Este es el único cambio con respecto a la planificación de etapas inicial.

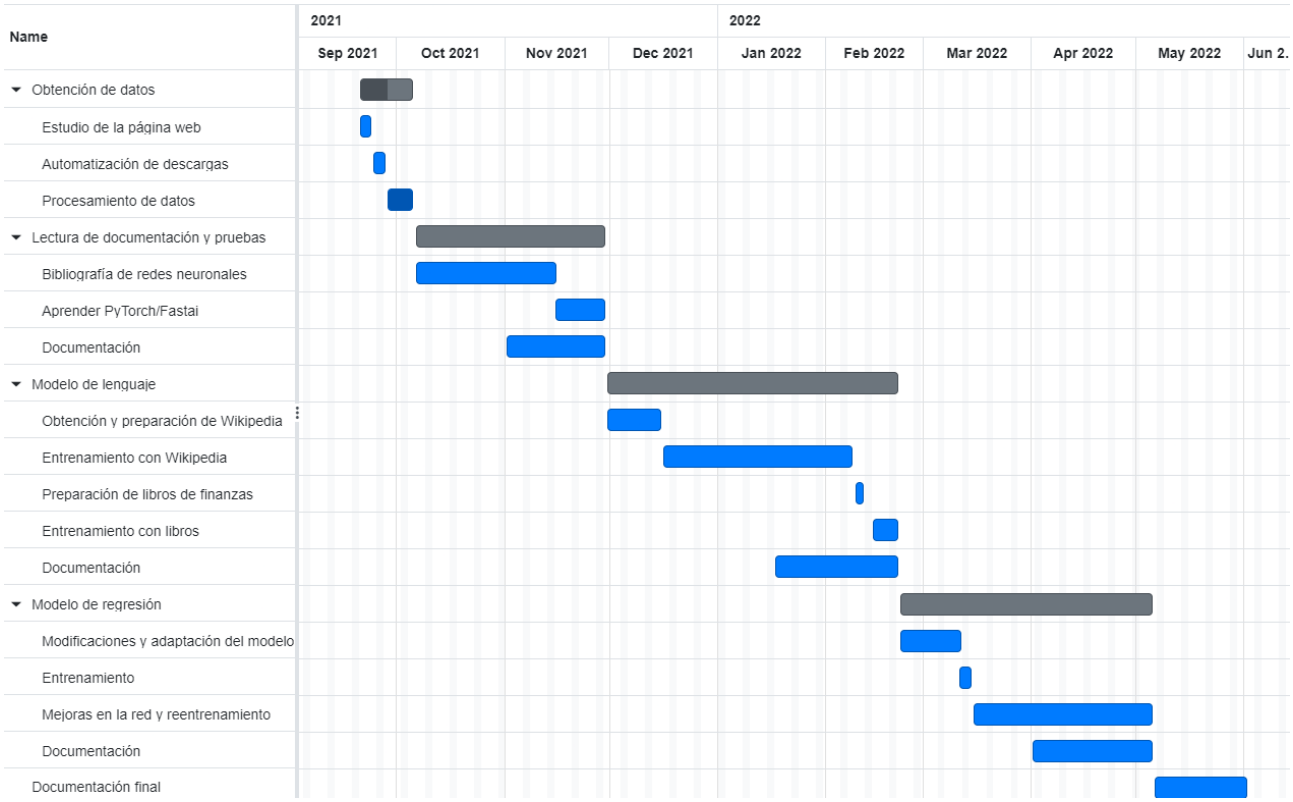


Figura 4.1.1: Diagrama de Gantt para las tareas del proyecto.

4.2. Riesgos y contingencias

Identificar los riesgos, su impacto y probabilidad que pueden afectar al desarrollo del trabajo es importante para prevenirlos. Se utiliza [46] para identificarlos y valorarlos cuantitativamente en el rango de 1 a 10. El análisis se muestra en el Cuadro 4.2.1.

ID	Descripción	Prob.	Impacto	Riesgo
R1	Imposibilidad de automatizar la descarga de datos.	5	8	40
R2	Imposibilidad de procesar los archivos PDF.	6	8	48
R3	Insuficiente limpieza de datos.	2	5	10
R4	Imposibilidad de obtener datos para modelo de lenguaje.	1	8	8
R5	Hardware insuficiente.	3	5	15
R6	Insuficiente tiempo de cómputo.	4	7	28
R7	Incapacidad de construcción de modelos.	5	3	15
R8	Modelo de lenguaje incorrecto.	3	8	24
R9	Mal ajuste del modelo de regresión.	5	5	25
R10	Demasiado tiempo para la documentación.	5	9	45

Cuadro 4.2.1: Riesgos, probabilidad e impacto previsto.

Además de detectar y evaluar los riesgos del trabajo, se ha desarrollado un plan de contingencia para minimizar su impacto en caso de que ocurran. En el Cuadro 4.2.2 se exponen los planes de contingencia para cada riesgo.

ID	**Plan de contingencia**
R1	Herramientas de scraping más complejas o reducir el conjunto de datos manualmente.
R2	Reducir el conjunto de datos manualmente.
R3	Dedicar más horas a proecesar los datos.
R4	Obtener un modelo pre entrenado para ambas arquitecturas.
R5	Utilizar la máquina a disposición del departamento de Informática.
R6	Reducir el tamaño del conjunto de datos.
R7	Dedicar más horas a aprender la tecnología y lectura de bibliografía.
R8	Obtener un modelo pre entrenado para ambas arquitecturas o entrenar durante más tiempo.
R9	Realizar aumento de datos o aumentar la complejidad del modelo.
R10	Resumir la memoria

Cuadro 4.2.2: Planes de contingencia para los riesgos del proyecto.

4.3. Presupuesto

El presupuesto estimado para el proyecto se divide en dos categorías: las horas de trabajo personal o servicios y la amortización de las máquinas de trabajo del proyecto.

Para obtener el presupuesto de trabajo personal, se ha calculado el sueldo medio anual de un trabajador del sector de información y comunicaciones (36700 €) que trabaja 2000 horas al año (8 horas diarias durante 250 días), disponible en [47]. Esto genera un coste de 18.35 € por hora.

El coste de las máquinas de desarrollo se divide en dos máquinas distintas:

- Máquina virtual remota con GPU de alto rendimiento. Se desconocen las especificaciones concretas a excepción de la GPU, que tiene un precio de 3800 €. Se supone una estimación de vida útil de 5 años. Como se ha explicado en la Sección 3.2.5, el *hardware* para la realización del trabajo está disponible de forma gratuita (con limitaciones de tiempo y fines educativos) en máquinas remotas en Kaggle.
- Ordenador portátil. Utilizado para la obtención de datos, desarrollo de modelos y documentación. Tiene un precio de 700 € y una vida útil estimada de 5 años.

Las horas de uso de la máquina virtual remota se estiman en 80 y las del portátil, en 300. Las horas de vida tomando 8 horas diarias de trabajo, 250 días laborales y la vida útil de 5 años se estiman como 10000. Calculamos el porcentaje de horas de uso sobre el total de horas de vida

(0.008 y 0.03 respectivamente). Se aplica al coste de adquisición del *hardware* para obtener el coste de amortización.

Una estimación del presupuesto del proyecto formada por los gastos en suministros (*hardware* en nuestro caso) y los servicios se muestra en el Cuadro 4.3.1. En la última fila se añaden los costes indirectos del trabajo como pueden ser el internet o la luz y que se estiman como un 15 % extra sobre el coste original.

Elemento	**Precio**	**Horas**	**% horas de uso**	**Total (€)**
Desarrollo del modelo	18.35 €/h	300	-	5505
Máquina virtual	3800 €	80	0.021	30.40
Portátil	600 €	300	0.03	18
Total				5553.40
Total + coste indirecto				6386.41

Cuadro 4.3.1: Estimación del presupuesto del proyecto. Fuentes: [48] y [47]

.

Capítulo 5

Modelo de lenguaje

En este capítulo se explica el procedimiento que se ha seguido para crear un modelo de lenguaje en español entrenado en el ámbito de la economía y las finanzas. También se realizará una comparativa de las dos arquitecturas utilizadas: LSTM y Transformadores. La primera se ha construido desde cero mientras que en la segunda se ha utilizado un modelo preentrenado «omitiendo» una etapa del procedimiento. En ambos casos se aplica *transfer learning* (transferencia de aprendizaje).

En la Figura 5.0.1 observamos las etapas de construcción de al red. En este capítulo se tratan los dos primeros pasos. El modelo con transformadores ya ha sido entrenado con el primer paso, por lo que «entiende» español y su construcción comienza desde el segundo paso.

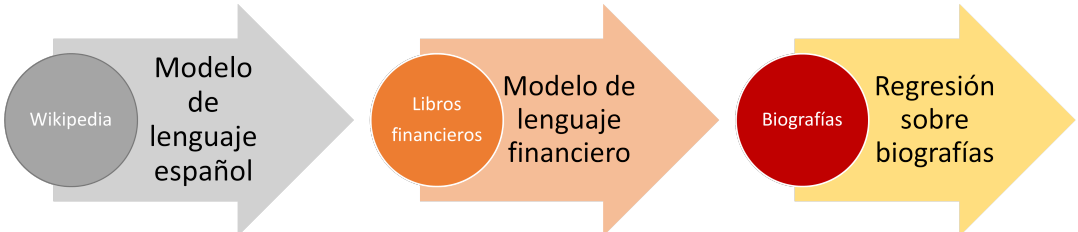


Figura 5.0.1: Etapas de construcción de la red mediante *transfer learning*.

5.1. Conjunto de datos

Se han utilizado dos conjuntos de datos para el modelo de lenguaje: el texto de Wikipedia en español y varios libros de economía y finanzas. En el ámbito del PLN, estos conjuntos de datos se denominan *corpus*. La primera parte de confeccionar un modelo de lenguaje es obtener el texto base con el que entrenar la red. Algunas consideraciones que hay que tener a la hora de elegir el conjunto inicial son [49]:

- Profundidad y precisión. Un corpus completo contiene ejemplos de una gran variedad de palabras y representa de forma equilibrada el idioma, sin contener demasiadas repeticiones.
- Actualidad. El texto ha de adecuarse al problema que se quiere abordar. Un corpus de hace 10 años no dará buenos resultados modelizando un problema de hace 2.
- Variedad. Cuanto mayor sea la variedad de los escritos (diálogos, ficción, revistas, periódicos o textos académicos entre otros) mejor se representa el lenguaje.
- Tamaño. Un corpus de gran tamaño facilita el cumplimiento de las anteriores consideraciones. Se trata de una solución de fuerza bruta, pero la más sencilla de ejecutar.

El objetivo de utilizar dos conjuntos de datos es obtener un modelo de lenguaje capaz de «entender» español y generar textos en el ámbito de las finanzas. Por ello, se utiliza Wikipedia como fuente de español y los libros de economía como fuente de términos financieros.

5.1.1. Wikipedia en español

Para construir el modelo LSTM desde cero necesitaremos un corpus que actúe como base. Para la gran mayoría de tareas de PLN, el mejor texto base es Wikipedia y en este trabajo no será una excepción. Proporciona un corpus extremadamente grande, completo y gratuito. Además, la descarga es relativamente sencilla.

Aproximadamente cada mes, se extrae el contenido de Wikipedia (en varios idiomas) en https://dumps.wikimedia.org/. Accedemos a la dirección en español desde https://dumps.wikimedia.org/eswiki/ y tomamos la correspondiente a la última extracción. En nuestro caso, el 27 de enero de 2022. En ella podemos encontrar la información separada en varias categorías: imágenes, títulos, metadatos, historial de ediciones y los artículos, que será lo que descarguemos.

El archivo comprimido suele tener un tamaño de unos 3 ó 4 GB mientras que el descomprimido, unos 15 ó 20 GB. Trabajar directamente con ficheros tan grandes en un ordenador convencional

es demasiado lento, por lo que primero hay que procesar el fichero fuente:

1. Descomprimir los datos en formato XML en artículos separados. Para ello, existen varias herramientas como `WikiExtractor` [50], que permiten paralelizar el proceso. Esto separa el fichero original en varios ficheros más manejables.

2. Una vez se disponga de archivos de texto separados, se eliminan las etiquetas XML mediante expresiones regulares.

3. Separar las oraciones de cada artículo de modo que haya una por línea. Para esto se utiliza `BlingFire`, un tokenizador que permite paralelizar el trabajo [51].

4. Limpiar el texto. Se eliminan caracteres extraños y se sustituyen símbolos por su significado (e.g. $ por dólar).

Finalmente, se dispone de un conjunto de ficheros de aproximadamente el mismo tamaño (50 KB cada uno) con un total de 78905262 palabras, las cuales 1753991 son distintas para comenzar a construir el modelo de lenguaje.

Podemos visualizar las palabras más frecuentes mediante un gráfico de barras y una nube de palabras, donde el tamaño de cada palabra es directamente proporcional a su frecuencia en las Figuras 5.1.1 y 5.1.2. Comprobamos que hay una gran cantidad de artículos históricos debido a la gran cantidad de palabras relacionadas con el tema: *año*, *historia*, *guerra* o *después* entre otras. El lenguaje utilizado en estos casos suele ser formal, lo que resulta de interés ya que el modelo final va a utilizarse sobre textos con un nivel de formalismo similar.

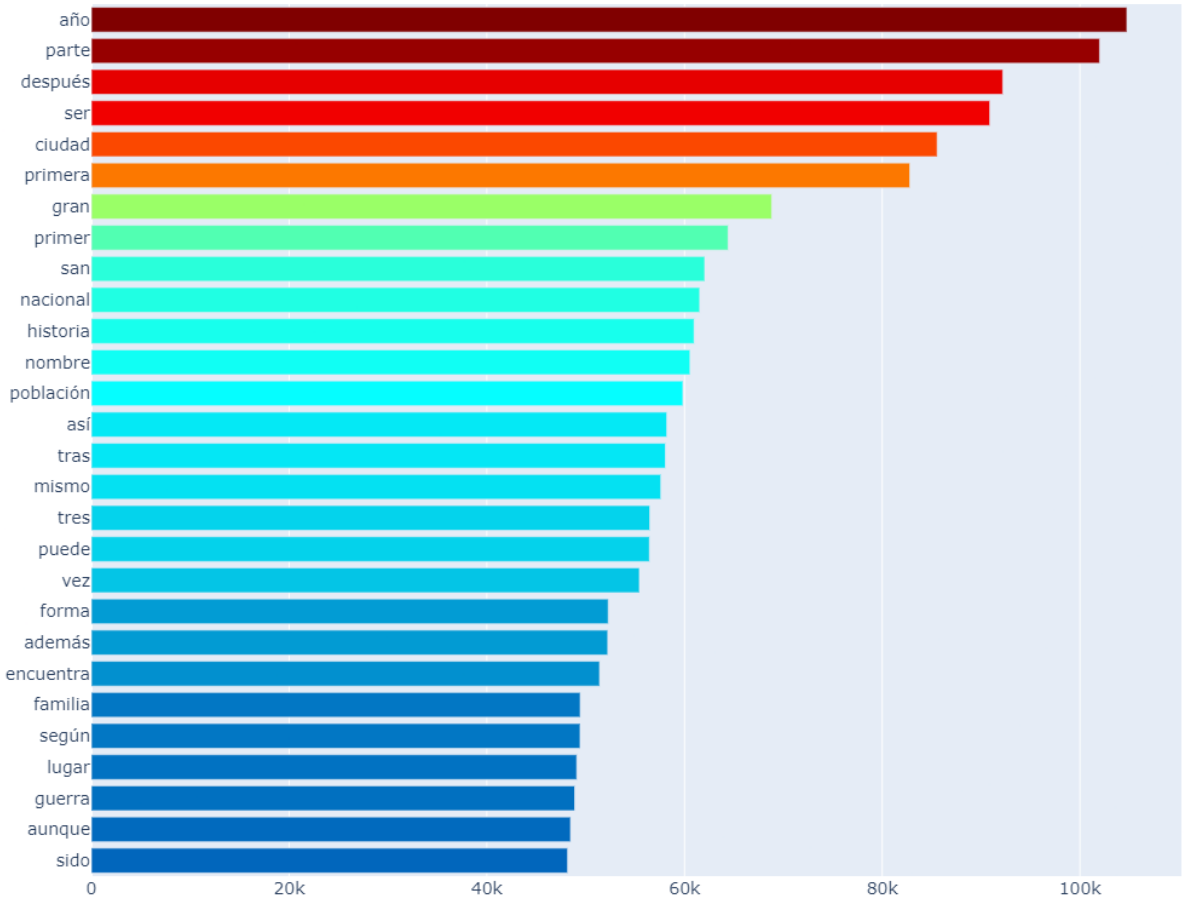


Figura 5.1.1: Diagrama de barras con frecuencias de las palabras del conjunto de datos de Wikipedia.

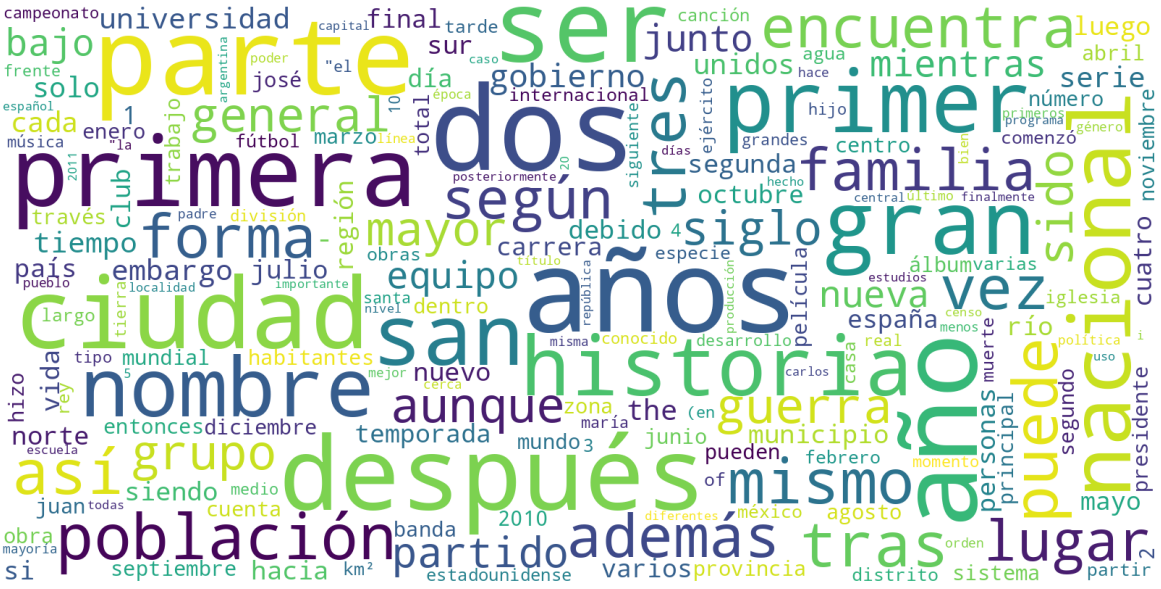


Figura 5.1.2: Nube de palabras con frecuencias de las palabras del conjunto de datos de Wikipedia.

5.1.2. Libros de economía

El segundo conjunto de datos es específico para resolver el problema planteado. Los siguientes libros contienen un vocabulario que se acerca razonablemente al objetivo final del trabajo: las biografías de los consejeros.

- Carlos Enrique Rodríguez. Diccionario de economía: etimológico, conceptual y procedimental: edición especial para estudiantes [52].
- Jonathan Berk, Peter DeMarzo. Finanzas corporativas [53].
- Marcela Astudillo Moya. Fundamentos de economía [54].
- Francisco Alburquerque. Conceptos básicos de economía: En busca de un enfoque ético, social y ambiental [55].
- Saving Trust S.A. El dinero y el ahorro: Un buen mañana se planifica hoy [56].
- Michael C. Ehrhardt, Eugene F. Brigham. Finanzas corporativas [57].
- Brealey Myers Allen. Finanzas corporativas [58].
- Ross Westerfield Jaffe. Principios de finanzas corporativas [59].

Todos los libros se encuentran en formato PDF. Se utiliza `Tika` [60], un parser de PDF para obtener el texto fuente. Al ser documentos cortos en comparación con Wikipedia, el procesa-

miento es bastante rápido. De la misma manera que en la sección 5.1.1 se separan los ficheros en sub-ficheros y se limpia el texto.

El conjunto final cuenta con un total de 1690924 palabras, de las cuales 72458 son palabras distintas. De la misma forma que con el conjunto anterior, visualizamos sus frecuencias en las Figuras 5.1.3 y 5.1.4. La mayoría de los términos se relacionan con la economía y las finanzas, como era de esperar.

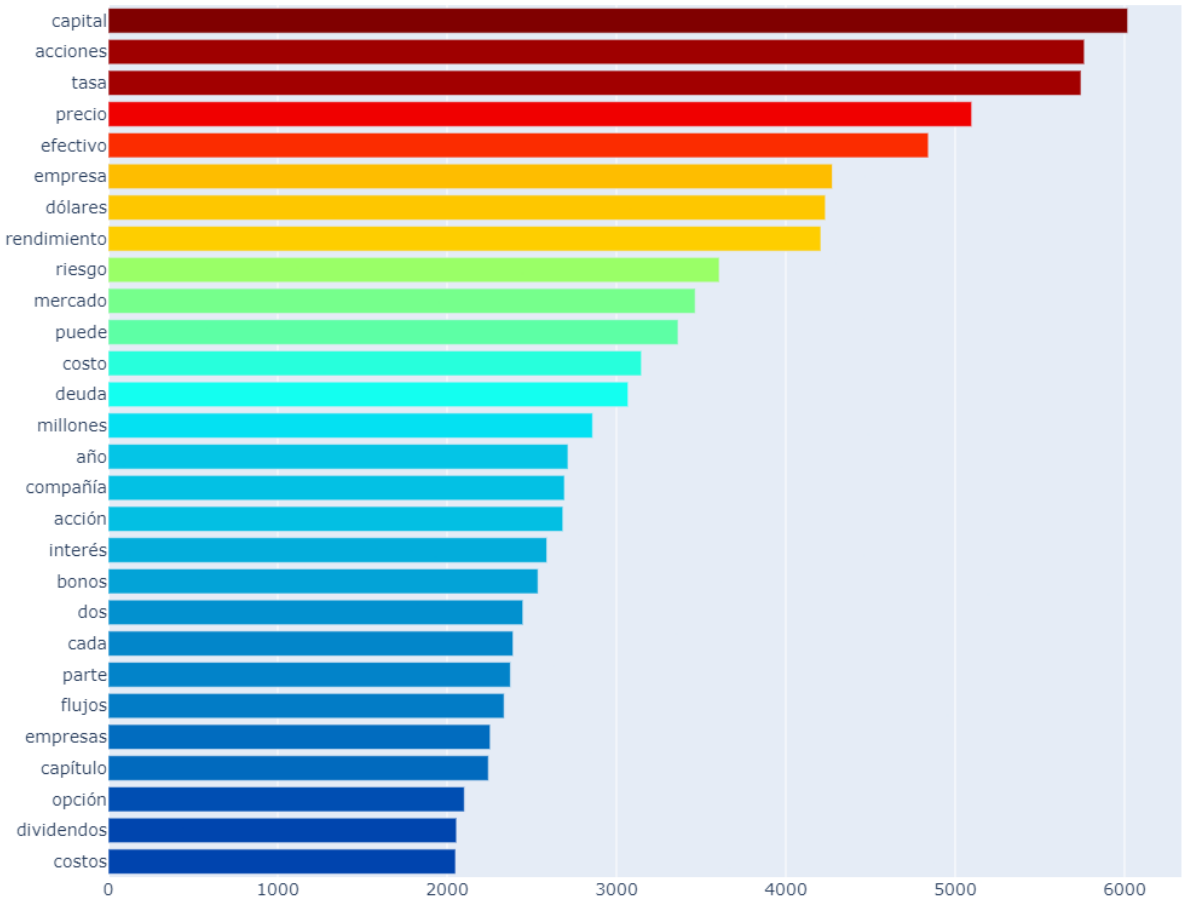


Figura 5.1.3: Diagrama de barras con frecuencias de las palabras del conjunto de datos de libros de finanzas.

Figura 5.1.4: Nube de palabras con frecuencias de las palabras del conjunto de datos de libros de finanzas.

5.2. Modelo LSTM

Para construir el primer modelo de lenguaje, utilizamos Fastai.

5.2.1. Arquitectura

Primero, hay que crear los *dataloaders* que alimentan al modelo. Para ello se utilizan las herramientas explicadas en la sección 3.2.3. A la hora de tokenizar el texto, se incluyen una serie de tokens especiales cuyo significado es el siguiente:

- `xxunk` representa la palabra desconocida, aquella que no está presente en el vocabulario.
- `xxpad` es el token de relleno, se incluye cuando las secuencias no tienen la longitud especificada.
- `xxbos` representa el inicio de una secuencia (*Beginning Of Sequence*).
- `xxeos` representa el final de una secuencia (*End Of Sequence*).
- `xxfld` indica la columna del conjunto de datos de donde se obtuvo el texto. En nuestro caso no se usa, puesto que todo se extrae de la misma columna.
- `xxrep` se utiliza para indicar que el siguiente caracter está repetido n veces en el texto original (`xxrep n {caracter}`).

- `xxwrep` se utiliza para indicar que la siguiente palabra está repetida n veces en el texto original (`xxrep n {palabra}`).
- `xxup` indica que la siguiente palabra está escrita en mayúsculas.
- `xxmaj` se usa para indicar que la siguiente palabra comienza por mayúscula.
- `xxfake` es el token «falso». Sirve para hacer que la longitud del vocabulario sea un múltiplo de 8 y facilitar el procesamiento para un ordenador.

Una vez disponemos de los datos ya procesados, se configura el modelo en lo que Fastai denomina `Learner`. En él, se implementa una red basada en la arquitectura AWD-LSTM de la sección 2.7.1. Para ello, se definen tres tipos de *dropout*:

1. *Dropout* sobre la entrada y capas ocultas. Un tensor de una capa recurrente tiene tres dimensiones: el tamaño del lote, la longitud de la secuencia y el número de sub-capas ocultas. Al tratarse de capas recurrentes, queremos reemplazar siempre por 0 las mismas posiciones dentro de las secuencias en vez de hacerlo aleatoriamente en cada sub-capa. Esto permite no perturbar la capacidad de la capa LSTM de recordar relaciones a «largo plazo».
2. *Dropout* sobre los pesos. El *dropout* convencional que fija a 0 los pesos de las matrices entre capas ocultas con una probabilidad prefijada.
3. *Dropout* sobre los *embeddings*. Se fijan a 0 filas de la matriz de *embeddings*, de forma que desaparecen las ocurrencias de una palabra en concreto.

A continuación se plantea la arquitectura del modelo de lenguaje.

- *Encoder*. Capa de *embeddings* encargada de crear la matriz. El tamaño de vocabulario será 60008.
- *Encoder dropout* (tipo 3). Capa de *dropout* sobre los elementos de la capa de *embeddings* como primera medida contra el sobreajuste. Inicialmente se toma una probabilidad de 0.02 de eliminar cada entrada.
- LSTM. Las capas LSTM. Se concatenan 3 capas de tamaño 1152.
- *Dropout* (tipo 2) sobre los pesos en cada una de las capas LSTM.
- *RNNDropout* (tipo 1). Se aplica otro *dropout* sobre las capas LSTM.
- *Decoder*. El decodificador lineal que predice la palabra siguiente de la secuencia.

- *Dropout* (tipo 2) sobre los pesos del decodificador.

Un resumen de la arquitectura anterior aparece en el Cuadro 5.2.1.

Capa	**Dimensión**	**Núm. Parámetros**
Embedding (encoder)	60008 x 400	24003200
Embedding Dropout (tipo 3)	60008 x 400	0
LSTM Weight Dropout (tipo 2)	4608 x 1152 + 4608 x 400	7160832
RNN Dropout (tipo 1)	4608 x 1152 + 4608 x 400	0
LSTM Weight Dropout (tipo 2)	4608 x 1152 + 4608 x 1152	10626048
RNN Dropout (tipo 1)	4608 x 1152 + 4608 x 1152	0
LSTM Weight Dropout (tipo 2)	1600 x 400 + 1600 x 1152	22486400
RNN Dropout (tipo 1)	1600 x 400 + 1600 x 1152	0
Lineal (decoder)	60008 x 400	24003200
	Total	57082856

Cuadro 5.2.1: Arquitectura y parámetros de la AWD-LSTM usada en el trabajo.

5.2.2. Entrenamiento

El modelo de lenguaje se ha entrenado en dos grandes etapas utilizando los dos conjuntos de datos de la Sección 5.1:

1. Wikipedia en español. Usando el optimizador Adam, función de pérdida de entropía cruzada binaria y una tasa de aprendizaje de $1 \cdot 10^{-4}$ durante 10 épocas, un total de 30 horas. Lógicamente no se ha utilizado el conjunto de datos completo debido a su gran tamaño. Se ha tomado una porción de unos 600 MB de texto seleccionados aleatoriamente.

2. Libros de finanzas. De la misma forma, utilizando el optimizador Adam, función de pérdida de entropía cruzada binaria y una tasa de aprendizaje de $1 \cdot 10^{-3}$ durante 10 épocas, un total de 7 horas. Utilizamos una tasa de aprendizaje mayor para obtener un mejor modelado de los datos relacionados con economía.

Las gráficas de las funciones de pérdida para los conjuntos de entrenamiento y validación se presentan en las Figuras 5.2.1 y 5.2.2.

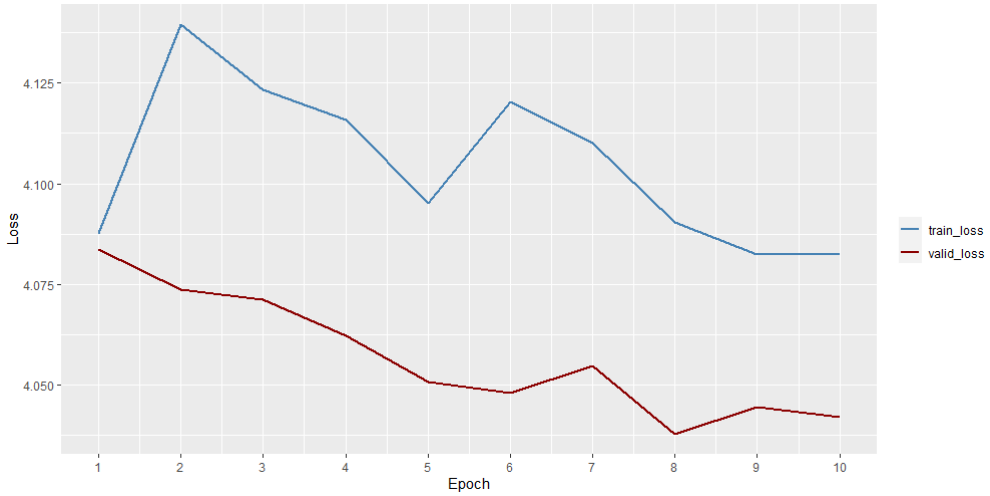


Figura 5.2.1: Evolución de la función de pérdida para el entrenamiento con Wikipedia (LSTM).

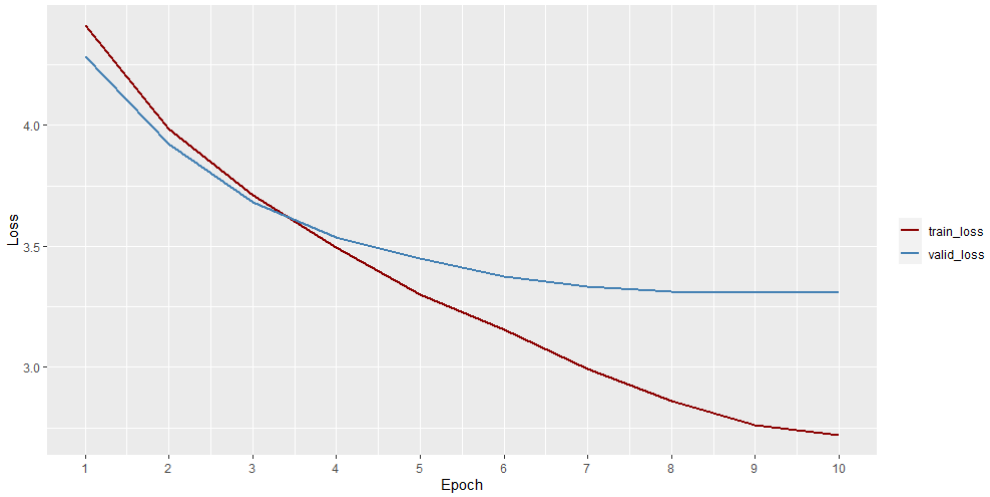


Figura 5.2.2: Evolución de la función de pérdida para el entrenamiento con libros de finanzas (LSTM).

5.2.3. Resultados

A continuación, probamos el modelo de lenguaje que acabamos de entrenar con el objetivo de comprobar que generan resultados coherentes y así tener un modelo válido para pasar a etapa de construcción del modelo de regresión. Proporcionando una entrada corta a la red y pidiendo una salida de 10 tokens obtenemos los resultados. A medida que se aumenta el número de tokens de salida, se pierde coherencia. Sin embargo, para la tarea final no nos interesa tanto que el modelo genere salidas coherentes, sino que «entienda» correctamente el idioma y los términos

que se le presentan.

La medida de calidad se ha establecido de forma parcialmente subjetiva. En ella se valora que la gramática sea correcta y que la oración tenga sentido. Los resultados para el modelo LSTM se encuentran en el Cuadro 5.2.2.

Datos	**Entrada**	**Predicción**	**Calidad**
Wikipedia	Mi amigo estudia	un desfile del hombre de Dios, luego de	Mala
	La oferta	realizó el posición de juez en su carrera en los	Mala
	Es imposible negociar	el papel de la vida de la vida del mundo	Normal
	Para solucionar el problema hay	que trata de la policía y la cierto que se	Normal
	La economía del país	de México se sitúa en el Museo	Buena
	Un índice bursátil	ostentan bien en el sistema de plantaciones de la	Mala
Wikipedia + Libros	Mi amigo estudia	los cuatro primeros mejores hombres de su país	Normal
	La oferta	fiscal de capital incluye el efectivo que se encuentra en	Buena
	Es imposible negociar el	préstamo real a una tasa fija del 1 de marzo	Buena
	Para solucionar el problema hay	dos errores fundamentales para aprender a hacer las preguntas	Normal
	La economía del país	es el único proceso de producción de capital, sino	Buena
	Un índice bursátil	se refiere a los tipos de cambio de una moneda	Buena

Cuadro 5.2.2: Resultados del modelo de lenguaje con AWD-LSTM.

Comprobamos que el modelo inicial con Wikipedia da predicciones bastante malas ya sea porque son demasiado incoherentes o gramaticalmente incorrectas. Cuando se añaden los libros de finanzas se adopta un lenguaje más depurado con bastantes términos relacionados con la economía.

5.3. Modelo con *transformers* y *transfer learning*

Para construir el modelo con transformadores, utilizamos PyTorch, HuggingFace y un modelo BERT (*Bidirectional Encoder Representations from Transformers*) [61] en español [62]. A partir de aquí nos referiremos al modelo con transformadores de forma abreviada como BERT.

5.3.1. Arquitectura

Como en la sección anterior, es necesario crear los *dataloaders* para el modelo. En este caso, los tokens especiales son distintos ya que hay que incluir características nuevas para entrenar los transformadores:

- `[MASK]` se utiliza para enmascarar una palabra. Recordamos que los transformadores reciben toda la secuencia de tokens simultáneamente en vez de secuencialmente de izquierda a derecha. Por tanto, se utiliza este token especial para ocultar el token original y realizar la predicción.

- `[PAD]` es equivalente a `xxpad` en Fastai: rellena las secuencias para que todas tengan la misma longitud.
- `[CLS]` indica el comienzo de una secuencia, de forma similar a `xxbos` en Fastai.
- `[SEP]` análogamente a `xxeos` en Fastai, señala el final de una secuencia.
- `[UNK]` representa el token desconocido o no presente en el vocabulario (`xxunk` en Fastai).

La arquitectura definida en BERT consta de varias capas, algunas de ellas diferentes a nivel estructural. A continuación se explican las capas diferentes con respecto a otras arquitecturas, donde la función de cada tipo de capa es el siguiente:

- *Embeddings* de palabras. De forma análoga al modelo LSTM, representa las palabras en una matriz.
- *Embeddings* de posiciones. Al no registrar la naturaleza secuencial de las entradas, se registra la posición de las palabras con ésta capa. Se trata de una matriz donde la primera fila representa la primera palabra, la segunda fila representa la segunda... Y así sucesivamente.
- *Embeddings* de segmentos. Es posible alimentar al modelo con dos oraciones o segmentos. Esta capa simplemente etiqueta cada oración por separado. Es de utilidad durante el entrenamiento como se verá en la Sección 5.3.2.
- Capas que representan las consultas y las parejas clave: valor, como se explica en la Sección 2.8.

En la Figura 5.3.1 se observa un ejemplo con las oraciones «*My dog is cute. He likes playing*». Los tokens representan las palabras, los segmentos separan las oraciones y las posiciones indican la secuencia.

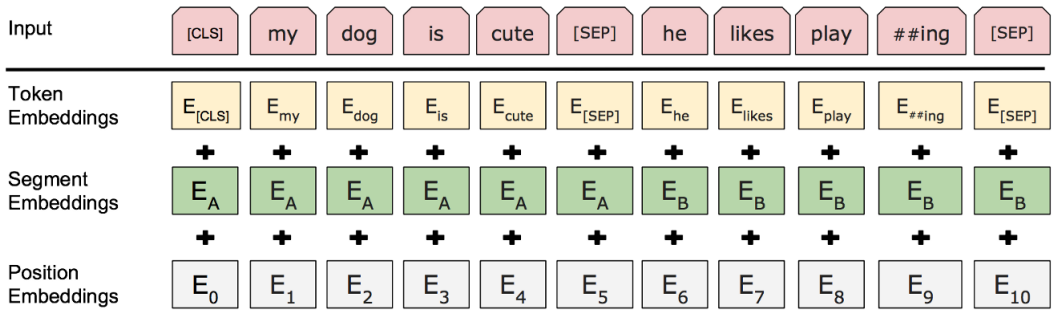


Figura 5.3.1: *Embeddings* de BERT. Fuente: [61].

La idea general de la arquitectura es la siguiente:

- *Embedding* que combina los tres tipos mencionados anteriormente. En este caso el modelo tiene un vocabulario de 31002 palabras distintas ya que ha sido entrenado sin mayúsculas. Se incluye un *dropout* de 0.1 para evitar el sobreajuste.
- *Encoder*. Está formado por capas que representan las consultas, los pares clave: valor, una capa *fully connected* con normalización y un *dropout* de 0.1. Se concatenan 12 repeticiones del codificador.
- *Decoder*. Se compone de una capa lineal, una normalización y otra lineal que devuelve las probabilidades o la predicción de cada palabra del vocabulario.

La arquitectura del modelo empleado se observa al completo en el Cuadro 5.3.1, donde se incluyen las repeticiones de cada capa para reducir el tamaño de la tabla.

Capa	**Dimensión**	**Repeticiones**	**Núm. Parámetros**
ID de posición	1 x 512	1	0
Embedding de palabras	31002 x 768	1	23809536
Embedding de posiciones	512 x 768	1	393216
Embedding de segmentos	2 x 768	1	1536
Consulta	768 x 768	12	589824 x 12
Clave	768 x 768	12	589824 x 12
Valor	768 x 768	12	589824 x 12
Dropout	768 x 768	12	0
Atención (lineal)	768 x 768	12	589824 x 12
Normalización	768 x 768	12	0
Dropout	768 x 768	12	0
Intermedia (lineal)	768 x 3072	12	2359296 x 12
Salida (lineal)	3072 x 768	12	2359296 x 12
Normalización	768 x 768	12	0
Dropout	768 x 768	12	0
Transformador (lineal)	768 x 768	1	589824
Normalización	768 x 768	1	0
Decoder	768 x 31002	1	23809536
	Total		133406304

Cuadro 5.3.1: Arquitectura y parámetros de la red BERT usada en el trabajo.

5.3.2. Entrenamiento

Si nos fijamos en las estructuras descritas en la arquitectura de BERT, no hay capas que actúen a modo de decodificador (referido a los transformadores, el decodificador del Cuadro 5.3.1 es un

decodificador común). El modelo de lenguaje que se plantea solo cuenta con un codificador. Esto se debe a que se basa en un modelo de lenguaje enmascarado. Como se ha dicho anteriormente, los transformadores permiten inferir el contexto de las palabras que tienen a su alrededor, por lo que se entrenan de forma paralela en vez de secuencial. De esta forma se pueden obtener los valores de atención de la Sección 2.8.

Es por eso que para realizar una predicción en una oración no basta con omitir la última palabra, sino que hay que enmascarar u ocultar una o varias palabras con el token especial `[MASK]`. Esto da más flexibilidad al modelo y permite realizar predicciones como las siguientes:

- «Nos vamos al cine mañana por la `[MASK]`» donde el resultado que se busca tiene la misma forma que en las redes LSTM: la siguiente palabra.
- «Nos vamos al `[MASK]` mañana por la mañana» donde se espera una predicción de mejor calidad que la anterior ya que se cuenta con palabras anteriores y posteriores a la enmascarada.

Como el procesamiento de los tokens enmascarados se realiza completamente en el codificador, no es necesario un decodificador. El proceso de entrenamiento del modelo BERT puede realizarse de dos formas:

- Enmascaramiento. Antes de alimentar la secuencia de palabras al modelo, un porcentaje (normalmente un 15 %) se reemplazan con el token `[MASK]`. El modelo intentará predecir el valor original de las palabras ocultas basándose en otras palabras de la secuencia que no lo estén. En este modo de entrenamiento, la función de pérdida solo ha de calcularse teniendo en cuenta los valores ocultos, ignorando la predicción de las palabras visibles.
- Predicción de la siguiente oración (*Next Sentence Prediction*, NSP). Para generar modelos de lenguaje más robustos, el modelo recibe parejas de oraciones como entrada e intenta predecir si la segunda oración de la pareja es la continuación de la primera en el documento original. Para ello, durante el entrenamiento la mitad de las entradas son subsiguientes mientras que la otra mitad se aligen aleatoriamente asumiendo que no lo serán. Es en este tipo de entrenamiento donde se utilizan los *embeddings* de segmentos.

Los datos empleados para entrenar este modelo en dos etapas han sido los siguientes:

1. (Preentrenado) Corpus español de gran tamaño disponible en [63]. Se utilizó el optimizador Adam y entropía cruzada binaria como función de coste. La tasa de aprendizaje fue $1 \cdot 10^{-4}$ y se entrenó durante 2 millones de iteraciones.
2. Libros de finanzas. Al igual que en el modelo LSTM entrenado desde cero, se realiza un

afinado con Adam como optimizador, entropía cruzada binaria como función de pérdida, tasa de aprendizaje de $1 \cdot 10^{-3}$, y 10 épocas durante un total de 10 horas.

En la Figura 5.3.2 se muestran los valores de la función de pérdida durante el afinado del modelo.

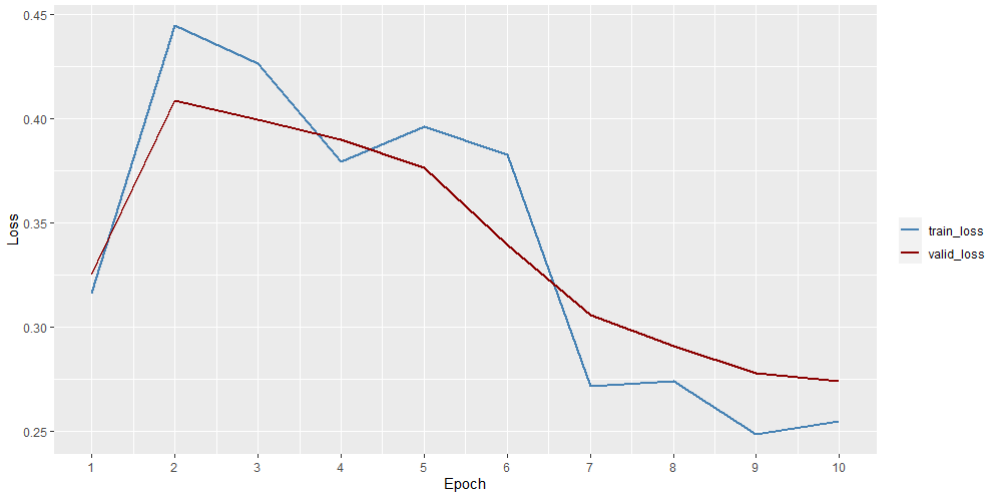


Figura 5.3.2: Evolución de la función de pérdida para el entrenamiento con libros de finanzas (Transformadores).

5.3.3. Resultados

De la misma forma que se ha hecho con el modelo LSTM, evaluamos la calidad del afinado de BERT. El procedimiento que se ha seguido ha sido equivalente al funcionamiento del primer modelo, obteniendo secuencialmente las nuevas palabras a medida que se genera el texto. Los resultados se presentan en el Cuadro 5.3.2.

Datos	**Entrada**	**Predicción**	**Calidad**
Corpus español	Mi amigo estudia	el libro.	Buena
	La oferta	oficial de venta incluye la siguiente versión.	Buena
	Es imposible negociar	.	Normal
	Para solucionar el problema hay	que saber como hacer esto.	Buena
	La economía del país	es que tiene un nombre.	Normal
	Un índice bursátil	de 1 en enero de enero de enero de enero	Mala
Corpus español + Libros	Mi amigo estudia	el tema de la empresa.	Buena
	La oferta	de acciones comunes de empresas grandes.	Buena
	Es imposible negociar	con el dinero en el tiempo.	Buena
	Para solucionar el problema hay	que hacer una serie de preguntas: 1.	Buena
	La economía del país	es de gran envergadura.	Buena
	Un índice bursátil	de rentabilidad de mercado en el futuro.	Buena

Cuadro 5.3.2: Resultados del modelo de lenguaje con BERT.

Es importante destacar que el propósito inicial de BERT no es la generación de texto secuencial. Se especializa en rellenar palabras faltantes deduciendo el contexto y clasificación de oraciones. Sin embargo, se ha realizado una prueba en «igualdad de condiciones» contra el modelo LSTM, pidiendo al modelo 10 tokens de forma secuencial.

El modelo con transformadores destaca por generar predicciones más cortas y conservadoras, normalmente con menos de 10 palabras y terminando con un punto y seguido. Las predicciones, a pesar de ser sencillas, son bastante buenas. El modelo podría continuar generando texto después del punto y seguido, pero infiriendo un contexto diferente al que se le dio en la oración de entrada. De nuevo, para mantener la igualdad de condiciones con el modelo anterior se decide parar la predicción una vez se genere el punto.

Resulta interesante visualizar el mecanismo de atención [64] de los transformadores una vez se ha realizado el entrenamiento con los dos conjuntos de datos. Podemos ver qué palabras tienen valores de atención altos dada una oración: «El consejero redacta un informe para el presidente». Recordamos que BERT tiene 12 capas de transformadores y en cada capa se calculan 12 valores de atención en paralelo. En la Figura 5.3.3 observamos algunos valores de atención: pesos representados por rectas entre palabras, donde una recta más gruesa indica un peso o atención más alto desde la palabra de la izquierda hacia la palabra de la derecha.

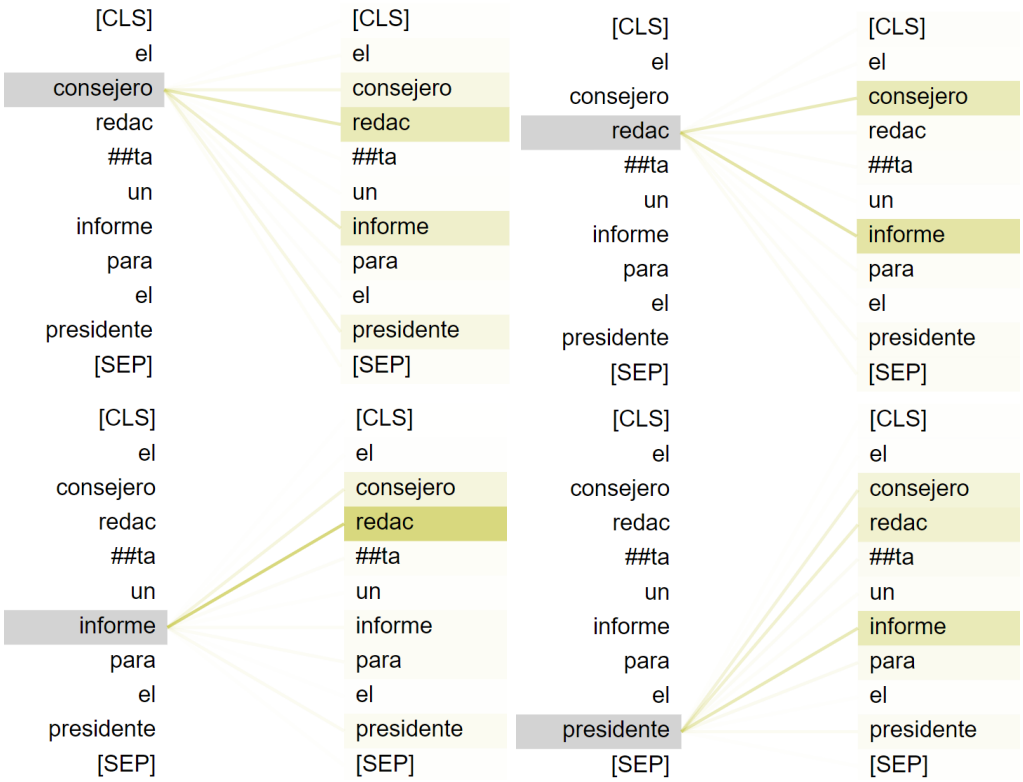


Figura 5.3.3: Atención para el transformador 0 y valor de atención 8.

Comprobamos que la palabra *consejero* se relaciona con *redacta* e *informe*, lo mismo ocurre con *redacta*, *informe* y *presidente*. Se podría decir que la capa 0 se ha encargado de extraer el significado relacionando las palabras más importantes de la oración.

En la Figura 5.3.4 se representa una matriz donde cada celda contiene los pesos (líneas entre palabras) de las atenciones para cada capa de atención en paralelo y capa del transformador. Se trata de una vista general de lo que se ha mostrado en la imagen anterior. Podemos observar que las primeras capas tienen valores de atención amplios, se fijan en varios elementos de la oración. Los tokens de las capas intermedias prestan atención a los siguientes dentro de la oración. Las últimas capas y en algunos casos las intermedias tienen valores de atención altos para los tokens separadores `[CLS]` y `[SEP]`. Este comportamiento es habitual en modelos de este tipo [65].

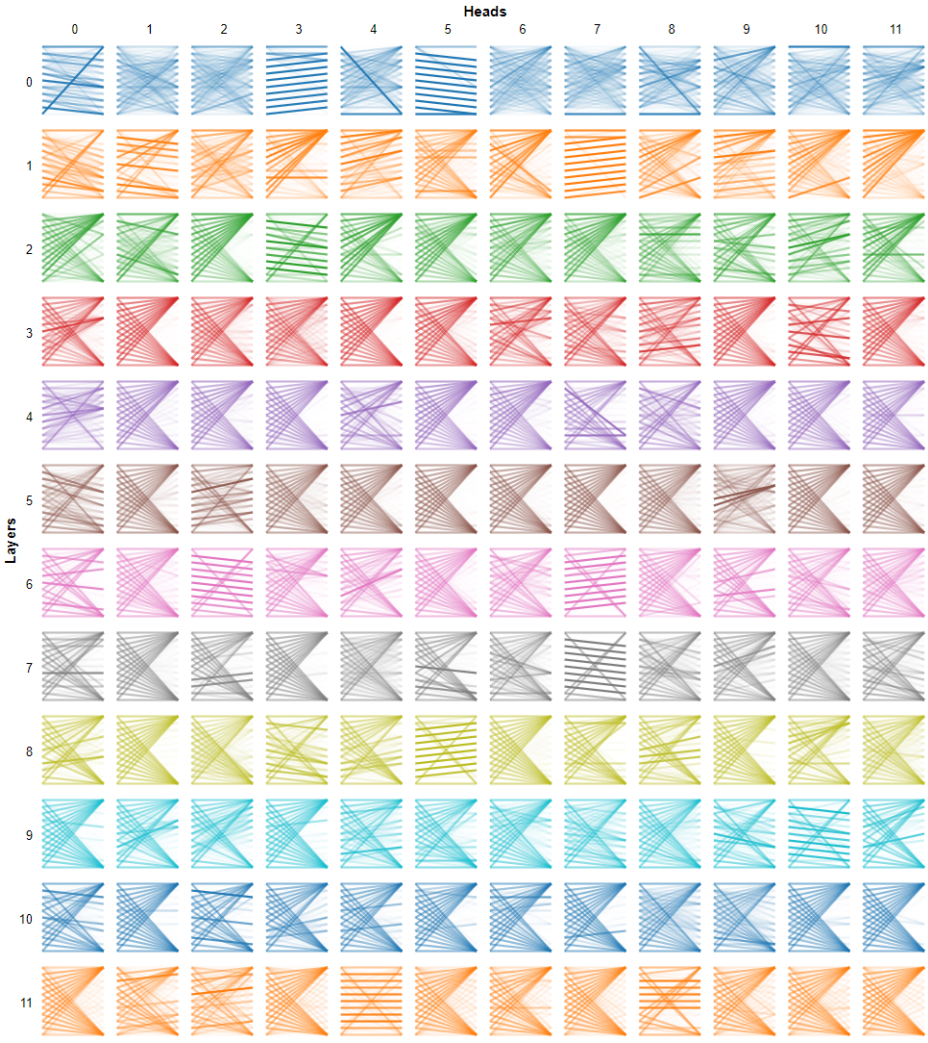


Figura 5.3.4: Matriz de atenciones para todo el modelo.

Capítulo 6

Modelo de regresión

En esta sección se detalla el procedimiento de construcción del modelo de regresión para las biografías de los consejeros a partir de los modelos de lenguaje obtenidos anteriormente. Se sigue la técnica ya explicada del *transfer learning*. En este capítulo se detallan el procedimiento del último paso de la Figura 5.0.1. Esta última versión del modelo busca, dada una biografía, asignar un valor a cada atributo asociado con el perfil profesional de un consejero.

6.1. Conjunto de datos

La obtención de las biografías es una parte fundamental del trabajo, ya que son los datos sobre los que se va a entrenar el modelo final. Sin embargo, conseguirlas no es tan sencillo como lo es obtener el texto de Wikipedia o el texto de libros de finanzas. La información está disponible en la página web de la Comisión Nacional del Mercado de Valores (CNMV) y se encuentra en formato PDF. Por tanto, será necesario automatizar el trabajo de extracción y procesamiento de los datos mediante un procedimiento ETL (*Extract, Transform, Load* o extraer, transformar, cargar).

1. Obtener los PDF fuente. Cada año se genera un PDF con el informe anual de gobierno corporativo para cada empresa del Mercado Continuo de la Bolsa de Madrid.
2. Extraer la información de los PDF. Cada PDF contiene en una sección determinada las biografías de los consejeros externos independientes de la empresa.
3. Organizar la información en un conjunto de datos etiquetable, es decir, un conjunto de datos organizado por filas y columnas de forma que cada fila contenga una biografía y sus correspondientes atributos.

La obtención de los archivos se ha automatizado mediante *web scraping*, ya que la web del CNMV permite hacerlo y está organizada en directorios.

El principal problema aparece con el formato PDF de los archivos. Como su nombre indica «*Portable Document Format*» o formato de documento portátil, su objetivo es la portabilidad. Esto significa que el documento debe ser legible por un humano en el mayor número de situaciones posible. Es por eso que se asume que no va a ser necesario editar el documento y que va a ser impreso en un futuro. Por tanto, los programas que generan PDFs lo hacen de forma que sus datos son imágenes o no están estructurados.

6.1.1. Web scraping

El directorio principal a extraer es la web www.cnmv.es. En él, está disponible una tabla de empresas con diferentes columnas como se ve en la Figura 6.1.1.

Informe anual de gobierno corporativo

Nombre del emisor	Mercado continuo	Nº registro oficial	Fecha registro oficial	Ejercicio	Fecha modificación	Apartados modificados	Ampliación (1)	Documento
ABANCA CORPORACION BANCARIA, S.A.		2021038607	29/03/2021	2020				
ABANCA CORPORACION BANCARIA, S.A.		2020031371	17/03/2020	2019				
ABANCA CORPORACION BANCARIA, S.A.		2019034999	26/03/2019	2018				
ABANCA CORPORACION BANCARIA, S.A.		2018034443	20/03/2018	2017				

Figura 6.1.1: Extracto de la tabla de compañías del CNMV. Fuente: [66].

En ella aparece el nombre de la empresa (Nombre del emisor), el registro del informe (Nº registro oficial) y la fecha en la cual se realizó (Ejercicio) junto con un enlace al documento PDF (Documento). Para filtrar la lista y realizar una descarga más uniforme es posible acceder a enlaces más concretos con la forma siguiente:

`https://www.cnmv.es/.../InformacionGobCorp.aspx?nif={NIF}&pageIAGC={PAGE}`

Donde `{NIF}` se sustituye por el NIF correspondiente a la empresa y `{PAGE}` por el número de página si es que el listado de informes no cabe en una sola. El proceso de *web scraping* se programa en Python y se utilizan las librerías `requests` y `bs4` para realizar peticiones HTTP

y extracción de HTML. A continuación vemos un ejemplo sencillo de cómo usarlo para extraer el contenido de una web:

```
URL = f'https://www.cnmv.es/.../InformacionGobCorp.aspx?nif={nif}&pageIAGC={page}'
response = requests.get(URL)
soup = BeautifulSoup(response.text, 'html.parser')
tabla = str(soup.find_all('tbody')[0])
```

Obtenemos el contenido HTTP con `requests.get()` y una estructura XML con `BeautifulSoup()`. Finalmente la tabla se encuentra bajo la primera etiqueta `tbody` de la web. Se extraen los datos mediante expresiones regulares sobre la tabla y se descargan los PDF con el enlace al documento bajo la etiqueta `a`, dándole un nombre con formato `{NIF}_{CODE}_{DATE}` (NIF, código y fecha respectivamente). Finalmente se obtienen 1835 documentos que ocupan en torno a 900 MB.

6.1.2. Parsing

El siguiente paso es obtener las biografías de los consejeros disponibles en los PDF. Hay una gran heterogeneidad en los documentos, que varían cada cierto número de años y existen empresas que usan su propio formato (Figura 6.1.2).

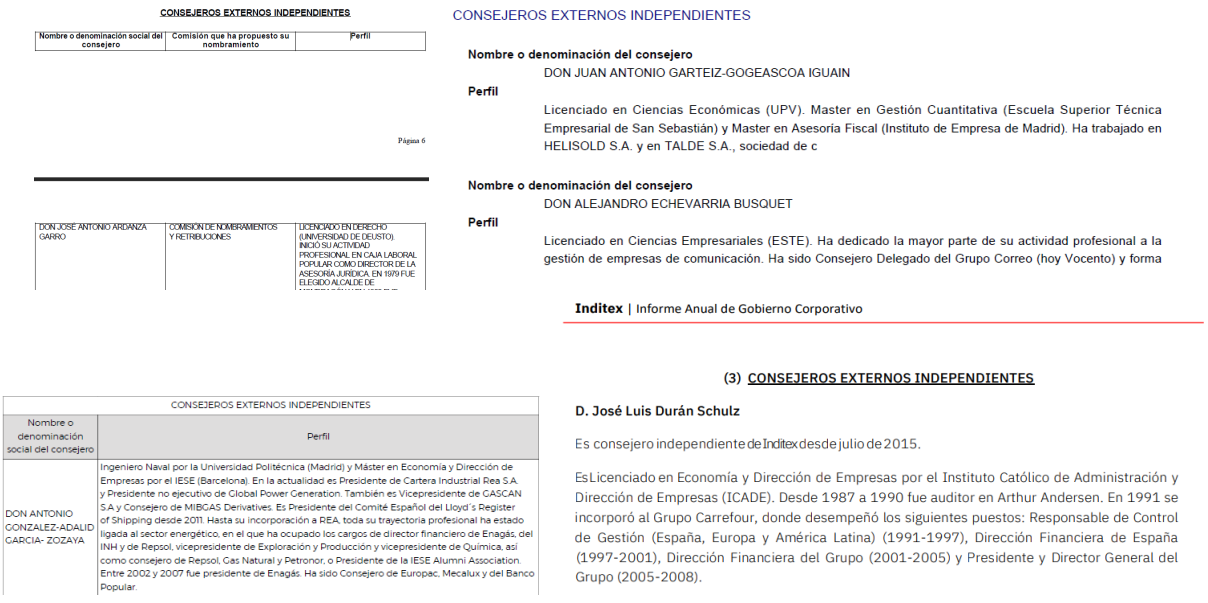


Figura 6.1.2: Heterogeneidad entre años y empresas.

Por tanto, hay que separar el tratamiento de cada tipo de PDF individualmente:

- Los PDF anteriores a 2007 contienen tablas con cabecera formada por Nombre, Comisión y Perfil.
- Los PDF de 2007 a 2017 contienen un listado de consejeros con formato Nombre y Perfil.
- Los PDF posteriores a 2017 contienen tablas con cabecera formada por Nombre y Perfil.
- Los PDF que no se encuentren en las categorías anteriores se consideran independientes y se tratarán manualmente.

Para obtener el texto de los PDF que no contienen tablas se utiliza la librería `tika` en Python, que añade una estructura que permite obtener el contenido de cada página de forma organizada. Dada una ruta al archivo obtenemos el texto fuente de esta forma:

```
1 texto = parser.from_file(pdf, xmlContent=True)
2 texto = texto['content']
```

A partir del texto mediante expresiones regulares, se obtienen los nombres y las biografías de los consejeros, guardándolos en un DataFrame junto a su empresa, NIF y fecha.

En el caso de las tablas, la obtención de los datos es más enrevesada. Debido a que las tablas no son tablas reales, es decir, la información no está estructurada como una tabla sino que simplemente son líneas que dividen el texto, no se puede extraer la información directamente. Para ello se utiliza la librería `camelot`, que permite extraer las tablas mediante un algoritmo de detección de bordes (Figura 6.1.3).

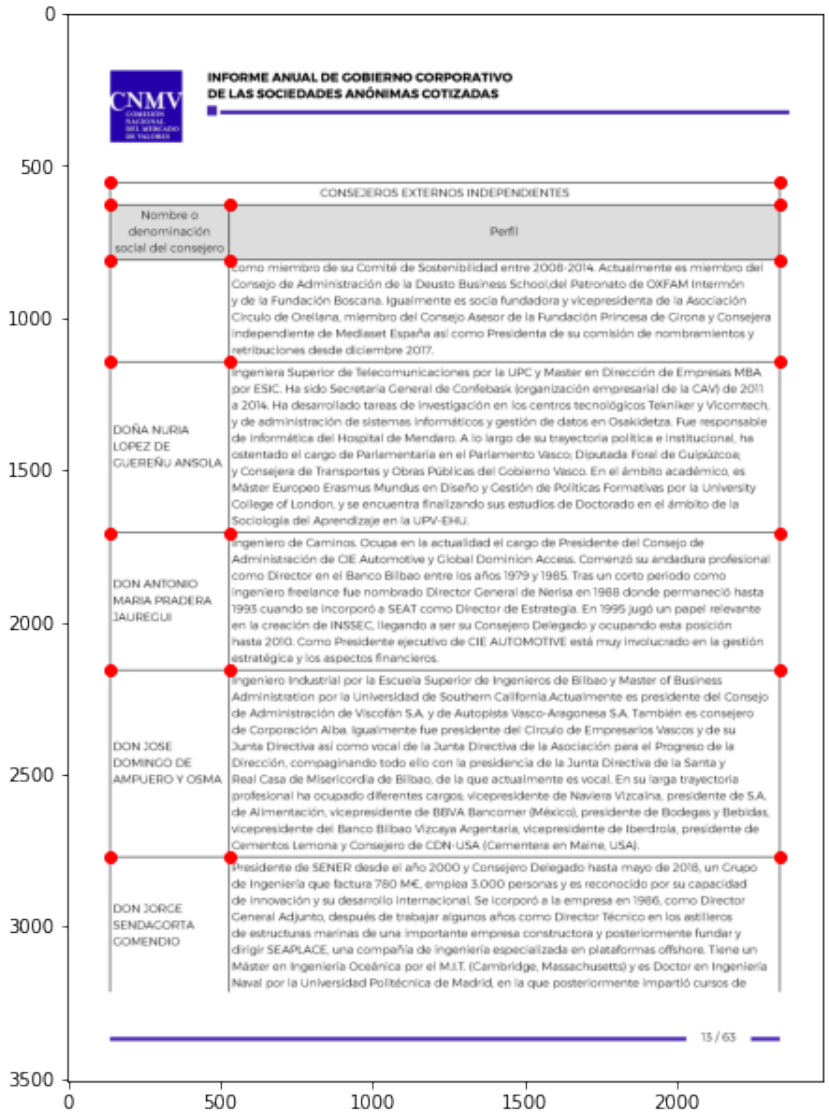


Figura 6.1.3: Extracción automática de tablas con detección de bordes (puntos rojos).

Para usarlo, se buscan los rangos de páginas en los que se encuentra la tabla de los consejeros independientes (que contiene las biografías que buscamos) con expresiones regulares. La librería proporciona un listado de DataFrames que tendremos que depurar y unir si la tabla se extiende en varias páginas. A continuación se muestra un ejemplo sencillo:

```
tablas = camelot.read_pdf(pdf, pages='13-17')
df_list = [t.df for t in tables]
```

El resultado es una lista de DataFrames que abarca las páginas 13 a 17 del documento. Para encontrar la que queremos, basta con comprobar la cabecera de cada uno y elegir el que nos

interesa.

6.1.3. Conjunto de datos final

El conjunto de datos final consiste en una tabla con las siguientes columnas:

- Variables independientes:
  - `bio`. El texto con la biografía del consejero.
  - `tipo`. Tipo de consejero. Inicialmente todos son independientes.
  - `empresa`. Nombre de la empresa a la que pertenece el consejero.
  - `nif`. Número de identificación fiscal de la empresa.
  - `fecha`. Año en el que se realizó el informe
- Variables dependientes:
  - `F`. Nivel de experiencia financiero.
  - `D/C`. Nivel de experiencia como directivo o consultor.
  - `A/C/F`. Nivel de experiencia como auditor, contable o fiscal.
  - `L`. Nivel de experiencia legal.
  - `P`. Nivel de experiencia como político.
  - `Ac`. Nivel de experiencia como académico.

Donde las variables dependientes que el modelo va a predecir han sido etiquetadas por el profesor doctor en economía financiera D. Fernando Tejerina. El etiquetado se realiza sobre un subconjunto de las biografías y se utilizará para entrenar la red neuronal con el objetivo de predecir el resto de valores si los resultados sobre el subconjunto son buenos. Son puntuaciones que varían en el rango $[0, 1]$ dependiendo del nivel de experiencia del consultor en cada campo, donde un nivel más cercano a 1 se considera un nivel mayor.

Las biografías cuentan con 105221 palabras, de las cuales 10105 son distintas. Para explorar el conjunto de datos, se realizan visualizaciones para las biografías y las variables dependientes relacionadas con los perfiles profesionales de los directivos. Comenzamos con un diagrama de frecuencias y una nube de palabras como se ha hecho anteriormente. Comprobamos que las palabras más comunes se relacionan con términos de juntas directivas, estudios y cargos impor-

tantes en una compañía. Además, los términos empleados complementan bien a los modelados con el conjunto de libros de finanzas (ver Figuras 6.1.4 y 6.1.5).

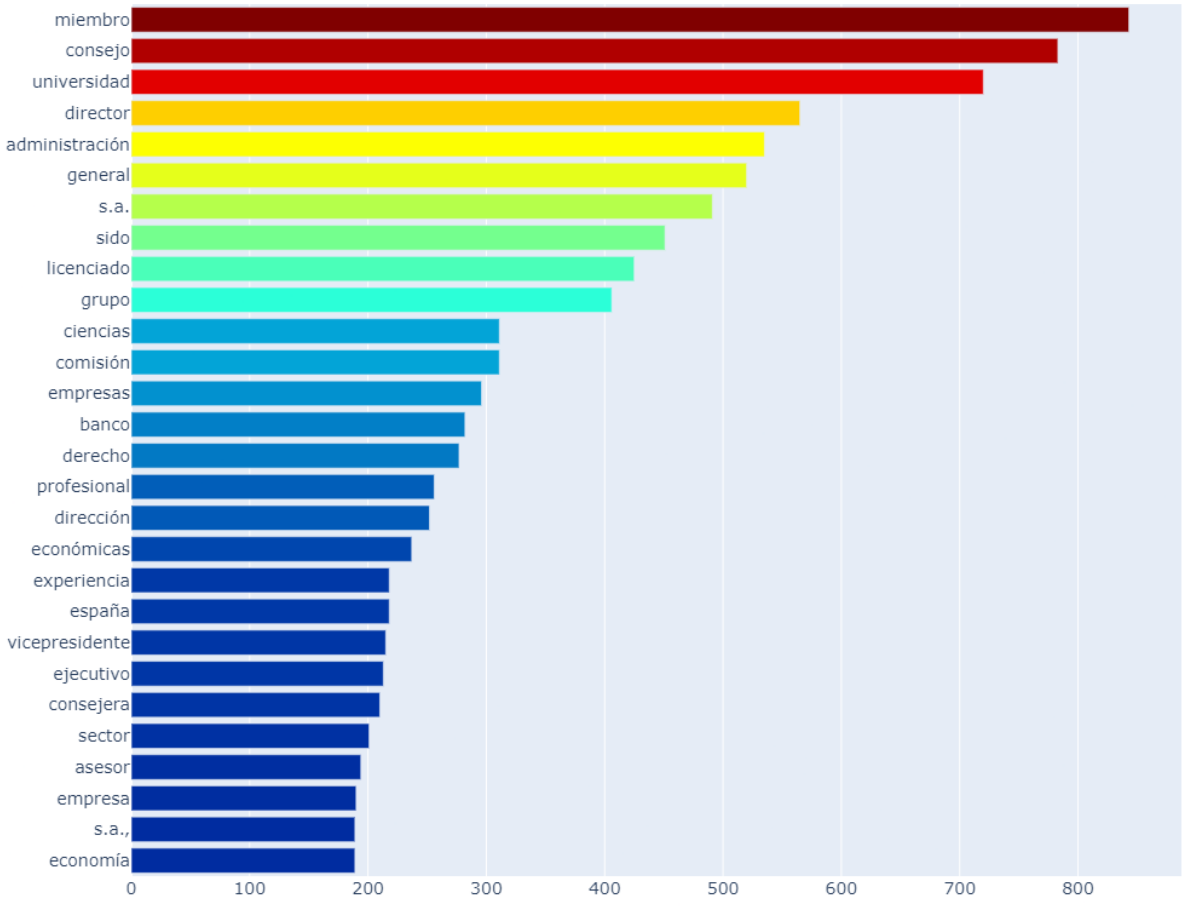


Figura 6.1.4: Diagrama de barras con frecuencias de las palabras del conjunto de datos de biografías.

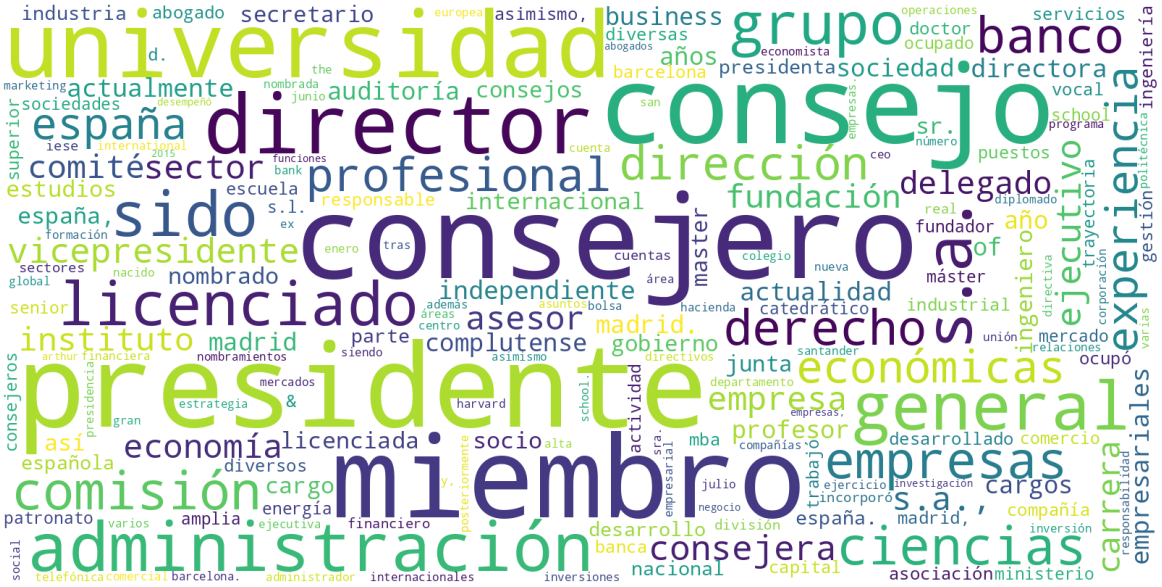


Figura 6.1.5: Nube de palabras con frecuencias de las palabras del conjunto de datos de biografías.

En cuanto a la distribución de las variables dependientes, observamos una breve descripción en el Cuadro 6.1.1. Comprobamos que la distribución de valores es bastante extrema. La mediana en todos los casos es 0 excepto en el atributo Directivo/Consultor.

	Media	Mín.	Q1	Mediana	Q3	Máx.
F	0.281	0	0	0	0.6	1
D/C	0.524	0	0	0.7	1	1
A/C/F	0.096	0	0	0	0	1
L	0.165	0	0	0	0	1
P	0.108	0	0	0	0	1
Ac	0.16	0	0	0	0	1

Cuadro 6.1.1: Estadísticos resumen para las variables dependientes de los consejeros.

Visualizamos la información anterior con un gráfico de cajas en la Figura 6.1.6, donde se comprueba que la gran mayoría de consejeros suelen tener puntuaciones mayores que 0 en las primeras variables (F y D/C) mientras que los otros atributos suelen ser complementarios, ya que hay muy pocas observaciones con valores superiores a 0.

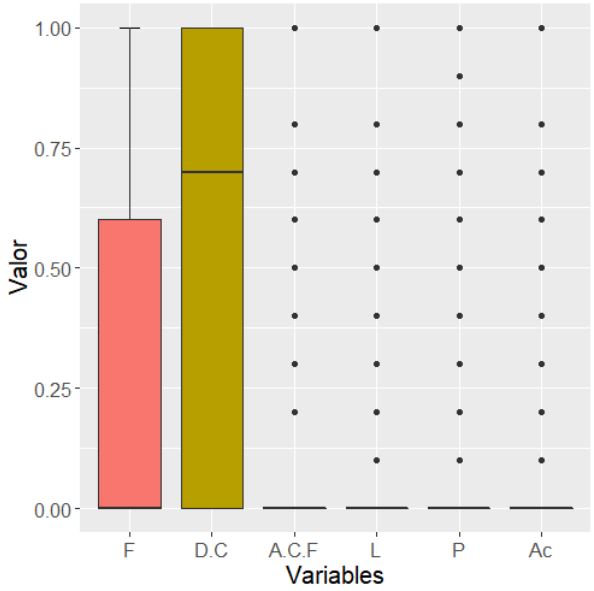


Figura 6.1.6: Diagrama de cajas para las variables dependientes de los consejeros.

6.2. Modelo LSTM

Al igual que en la sección 5.2, utilizamos Fastai para modificar el modelo de lenguaje y añadir las capas necesarias para la regresión. Debido a que el modelo de lenguaje predice la siguiente palabra dada una cadena, queremos que prediga las variables dependientes dada una biografía. Por tanto, hay que realizar modificaciones sobre las últimas capas y adaptarlas para que su salida sea la deseada.

6.2.1. Arquitectura

De nuevo, se crean los *dataloaders* de la misma forma que en la parte anterior, por lo que no se profundiza en ello. A la arquitectura del modelo de lenguaje se le elimina el decodificador y se añaden las siguientes capas para realizar la regresión:

- Normalización del lote. Tiene el objetivo de acelerar el entrenamiento, bajo la premisa de que la distribución de las salidas de la capa anterior será la misma.
- *Dropout* convencional fijado en 0.2.
- Capa lineal y activación ReLU.
- Normalización del lote.
- *Dropout* convencional fijado en 0.1.

- Capa lineal y activación sigmoide para obtener las salidas en el rango deseado $[0, 1]$.

Si modificamos el Cuadro 5.2.1 la arquitectura del modelo final se puede observar en el Cuadro 6.2.1.

Capa	**Dimensión**	**Núm Parámetros**
Embedding (encoder)	60008 x 400	24003200
Embedding Dropout (tipo 3)	60008 x 400	0
LSTM Weight Dropout (tipo 2)	4608 x 1152 + 4608 x 400	7160832
RNN Dropout (tipo 1)	4608 x 1152 + 4608 x 400	0
LSTM Weight Dropout (tipo 2)	4608 x 1152 + 4608 x 1152	10626048
RNN Dropout (tipo 1)	4608 x 1152 + 4608 x 1152	0
LSTM Weight Dropout (tipo 2)	1600 x 400 + 1600 x 1152	22486400
RNN Dropout (tipo 1)	1600 x 400 + 1600 x 1152	0
Normalización	2 x 1200	2400
Dropout (tipo 2)	2 x 1200	0
Lineal	1200 x 50	60000
ReLU	1200 x 50	0
Normalización	2 x 50	100
Dropout (tipo 2)	2 x 50	0
Lineal	50 x 6	300
Sigmoide	50 x 6	0
	Total	**33142456**

Cuadro 6.2.1: Arquitectura y parámetros de la AWD-LSTM para el modelo de regresión.

Se separan las capas «antiguas» de las nuevas con una línea en el centro. Comprobamos también que el número de parámetros final se ha reducido debido a que hemos eliminado el decodificador, ya que no hay necesidad de generar texto.

6.2.2. Entrenamiento

Se divide el conjunto de biografías de consejros etiquetadas mediante *hold-out*, es decir, en *train* y *test* con proporciones 0.7 y 0.3 respectivamente. Del conjunto de *train* se toman un 20 % de observaciones como conjunto de validación. Como en casi todos los problemas de aprendizaje profundo, resulta inviable realizar validación cruzada debido a los altos tiempos de entrenamiento.

Se entrena el modelo durante 250 épocas con una tasa de aprendizaje de $1 \cdot 10^{-2}$ usando el optimizador Adam. En este caso la función de pérdida no puede ser la entropía cruzada debido a que se trata de un problema de regresión (se predicen atributos continuos en vez de discretos), por lo que se utiliza el error cuadrático medio.

En esta ocasión hacemos uso del descongelamiento o *unfreezing* gradual. Como ya se ha explicado anteriormente, evitamos la actualización de los pesos de las capas del modelo de lenguaje durante las primeras épocas de entrenamiento. A medida que se entrena durante más épocas, se procede a actualizar todos los pesos. Con esto se consigue una mejor generalización en los resultados ya que se conservan las propiedades del modelo de lenguaje inicial.

En la Figura 6.2.1 se muestran los valores de la función de pérdida (MSE) para los conjuntos de entrenamiento y validación. Se pueden apreciar los «saltos» en la gráfica a medida que se descongelan las capas del modelo.

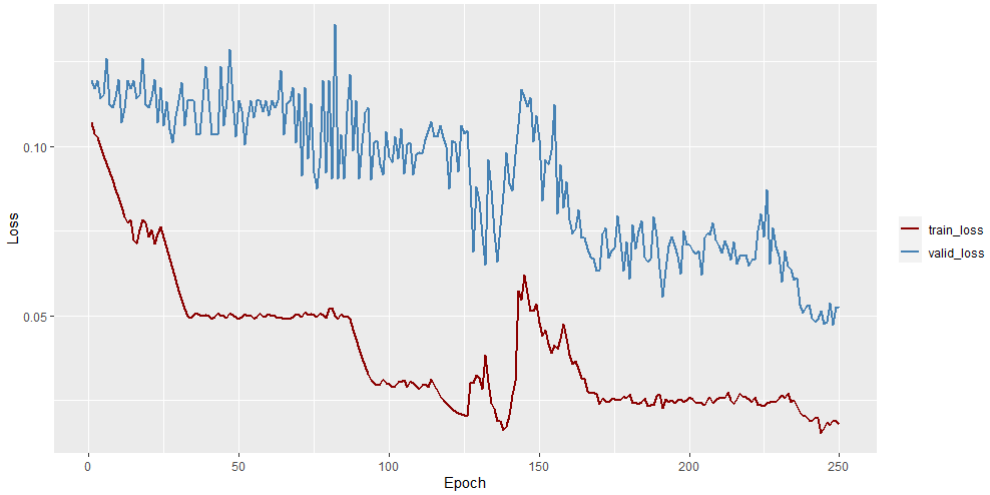


Figura 6.2.1: Evolución de la función de pérdida para el entrenamiento con biografías de consejeros (LSTM).

6.3. Modelo con *transformers*

Al igual que en la sección anterior, es necesario modificar las últimas capas e incluir la regresión sobre el texto.

6.3.1. Arquitectura

Se han añadido la misma estructura que en el modelo LSTM, por lo que modificamos el Cuadro 5.3.1 y observamos el resultado final en el Cuadro 6.3.1.

Con el objetivo de situar ambos modelos en el mismo nivel hemos hecho que sus capas de regresión sean iguales. De esta forma la comparación es más objetiva y permite evaluar lo bien que las redes entienden los textos.

Capa	Dimensión	Repeticiones	Núm. Parámetros
ID de posición	1 x 512	1	0
Embedding de palabras	31002 x 768	1	23809536
Embedding de posiciones	512 x 768	1	393216
Embedding de segmentos	2 x 768	1	1536
Consulta	768 x 768	12	589824 x 12
Clave	768 x 768	12	589824 x 12
Valor	768 x 768	12	589824 x 12
Dropout	768 x 768	12	0
Atención (lineal)	768 x 768	12	589824 x 12
Normalización	768 x 768	12	0
Dropout	768 x 768	12	0
Intermedia (lineal)	768 x 3072	12	2359296 x 12
Salida (lineal)	3072 x 768	12	2359296 x 12
Normalización	768 x 768	12	0
Dropout	768 x 768	12	0
Transformador (lineal)	768 x 768	1	589824
Normalización	2 x 768	1	1536
Dropout	2 x 768	1	0
Lineal	768 x 50	1	38400
ReLU	768 x 50	1	0
Normalización	2 x 50	1	100
Dropout	2 x 50	1	0
Lineal	50 x 6	1	300
Sigmoide	50 x 6	1	0
	Total		**109637104**

Cuadro 6.3.1: Arquitectura y parámetros de la red BERT usada en el modelo de regresión.

6.3.2. Entrenamiento

El conjunto de biografías etiquetadas se divide en *train* y *test* mediante *hold-out* con proporciones 0.7 y 0.3 respectivamente, al igual que en el modelo LSTM. Del conjunto de *train* se toman un 20 % de observaciones como conjunto de validación. Destacamos que se utiliza exactamente la misma división en ambas arquitecturas para realizar una comparación justa.

Se entrena el modelo durante 140 épocas. Las primeras 40 mantienen las capas de transformadores congeladas y las 100 siguientes descongelan el modelo entero. El optimizador es AdamW con una tasa de aprendizaje de $1 \cdot 10^{-2}$ para las primeras 40 épocas y $1 \cdot 10^{-3}$ para las 100 últimas. De nuevo, se emplea el error cuadrático medio como función de pérdida.

Los resultados del entrenamiento para las funciones de pérdida en el conjunto de entrenamiento

y validación se observan el la Figura 6.3.1. Podemos comprobar cómo durante las primeras 40 épocas, la función de pérdida es muy variable ya que está ajustando pesos inicializados aleatoriamente. Una vez se estabiliza, el entrenamiento es menos variable hasta el final.

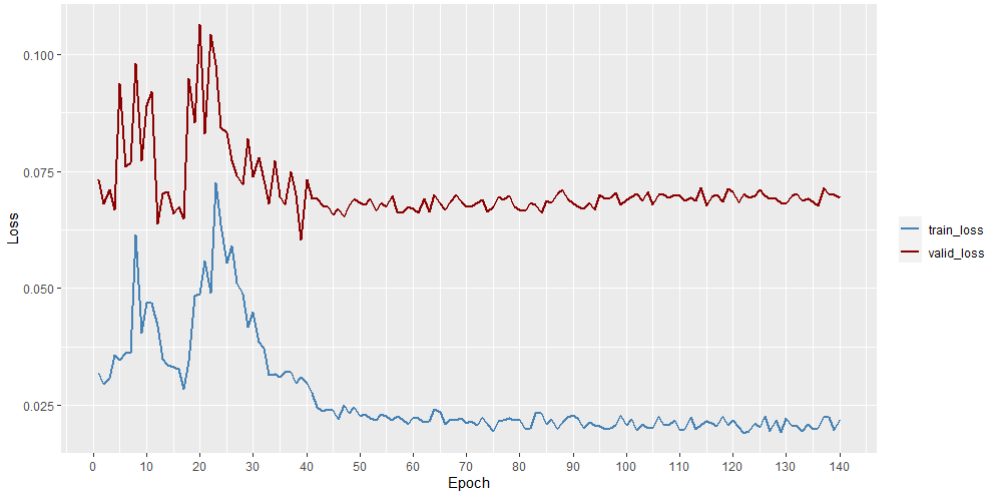


Figura 6.3.1: Evolución de la función de pérdida para el entrenamiento con biografías de consejeros (Transformadores).

Capítulo 7

Resultados finales

Recordamos que el conjunto de datos se ha dividido en entrenamiento y test mediante *hold-out* con proporciones 0.7 y 0.3 respectivamente. Después, del conjunto de entrenamiento se ha reservado un 20 % de las observaciones como conjunto de validación. En esta sección se presentan los resultados finales obtenidos sobre un conjunto de test con 309 biografías etiquetadas para cada modelo de regresión y se realiza una comparación entre ellos. Finalmente, se presenta un análisis de interpretabilidad de cada modelo para algunos ejemplos del conjunto.

En las siguientes subsecciones se utilizan tres métricas para evaluar los resultados sobre el conjunto de test. Las dos primeras se han explicado en la sección 2.3.1.

- $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$. Es la raíz del error cuadrático medio o MSE. Es importante debido a que ha sido la función de pérdida utilizada durante el entrenamiento. Obtenemos su raíz cuadrada para mejorar la interpretabilidad. Esta medida destaca por penalizar más los errores altos que los bajos.
- $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$. El error medio absoluto es la medida más interpretable ya que nos indica cuánto se desvían de media las predicciones del valor real. Penaliza todos los errores por igual, a diferencia del RMSE.
- $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}$. En cuanto al R^2, se trata de la proporción de variabilidad explicada de la variable dependiente por la variable independiente. Toma valores entre 0 y 1. Un modelo que proporciona una salida constante como la media de los datos tendrá un $R^2 = 0$, mientras que otro cuyas predicciones coincidan con los valores reales tendrá un $R^2 = 1$. En problemas de regresión sencillos, donde se cuenta con pocas variables suele emplearse el R^2 ajustado, que penaliza la inclusión de variables innecesarias al modelo

evitando el sobreajuste. En nuestro caso, las variables independientes son las palabras que forman las biografías y no tiene sentido usarlo.

7.1. Modelo LSTM

Los resultados sobre el conjunto de test se presentan en el Cuadro 7.1.1.

	Total	**F**	**D/C**	**A/C/F**	**L**	**P**	**Ac**
RMSE	0.2575	0.2578	0.4002	0.2499	0.2425	0.1542	0.2402
MAE	0.1335	0.1452	0.2559	0.1010	0.0898	0.0726	0.1367
$\mathbf{R^2}$	0.4742	0.5819	0.2982	0.3177	0.5686	0.6208	0.4582

Cuadro 7.1.1: Estadísticos resumen del modelo LSTM.

- RMSE. Comprobamos que su valor medio es 0.2575 y generalmente todos los atributos presentan valores similares, a excepción de D/C y P, que presentan errores inusualmente altos y bajos, respectivamente. Es la métrica más complicada de interpretar ya que a los errores más altos se les asigna un peso mayor.
- MAE. El error medio absoluto nos indica que la predicción total se desvía de la realidad un 13.13 % de media. De nuevo, destaca D/C al acumular gran parte del error.
- R^2. El modelo LSTM explica en torno a un 47 % de la variabilidad de los datos. Es relativamente poco y se ve afectado por el atributo D/C. Junto con las métricas anteriores podemos deducir que este atributo se modela con bastantes palabras distintas y extraer su valor es complicado. Ampliando el conjunto de entrenamiento con más observaciones que den información sobre D/C sería recomendable. Destacamos A/C/F que a pesar de tener un R cuadrado bajo su error también lo es. Esto indica que hay pocas palabras que lo modelen pero son suficientes para obtener un buen resultado.

7.2. Modelo con transformers

Los resultados sobre el conjunto de test se presentan en el Cuadro 7.2.1. En general, las métricas presentan comportamientos similares al modelo LSTM con valores más bajos en los errores y más altos en el R cuadrado.

	Total	F	D/C	A/C/F	L	P	Ac
RMSE	0.2273	0.2535	0.3901	0.1655	0.1978	0.1789	0.1778
MAE	0.1012	0.1335	0.2206	0.0426	0.0684	0.0709	0.0712
$\mathbf{R^2}$	0.5891	0.5957	0.3331	0.7005	0.7128	0.4896	0.7030

Cuadro 7.2.1: Estadísticos resumen del modelo con transformadores.

- RMSE. La raíz del error cuadrático medio mejora en el total y en cada atributo individualmente. Destaca de nuevo el atributo D/C ya que se cometen prácticamente los mismos errores que en el modelo anterior.
- MAE. La desviación con respecto a los valores reales es ahora de un 10 %, llegando a reducirse hasta un 4 % en los atributos complementarios. En el caso de Directivo o Consultor, el error aumenta hasta un 22 %.
- R^2. Es la métrica que más ha mejorado con respecto al modelo anterior, alcanzando casi un 60 % de variabilidad acumulada. Todos los atributos presentan un aumento individualmente excepto D/C que se mantiene igual. Junto con los resultados del apartado anterior, podemos confirmar que la cantidad de datos para modelar D/C ha sido insuficiente. Con respecto a A/C/F comprobamos que BERT modela mucho mejor los resultados y extrae el significado de las palabras de este atributo más eficientemente.

7.3. Interpretabilidad

Interpretar las predicciones de una red neuronal con muchos parámetros es muy complicado. Normalmente estos modelos se consideran una caja negra que, a pesar de generar buenas predicciones, tienen una interpretabilidad prácticamente nula. En nuestro problema puede resultar interesante comprender qué partes de la biografía de un consejero contribuyen a cada puntuación de los seis niveles de experiencia que se han predicho.

Para realizar este análisis, utilizamos SHAP [67] (*SHapley Additive exPlanations*) o las explicaciones aditivas de Shapley. Se trata de un algoritmo que permite obtener una explicación de las predicciones de cualquier modelo de *machine learning*. Está construido sobre las técnicas LIME y valores de Shapley, asignando a cada variable del modelo una importancia para una predicción en particular. En nuestro caso las variables son cada uno de los tokens que conforman la biografía, por lo que obtendremos una medida de la importancia de cada una de las palabras. Para obtener dichas medidas se construyen regresores con las permutaciones de las variables, de forma que se generan predicciones con cada variable presente y ausente, viendo cómo influye en las puntuaciones del modelo.

Tomaremos 6 bigorafías del conjunto de test que tengan puntuaciones altas en cada uno de los niveles de experiencia y obtendremos dos gráficos:

- Diagrama de barras. Muestra los tokens con más importancia en cada predicción del subconjunto. El último token de cada gráfico (que suele acumular la mayor importancia) es la suma del resto.
- *Force plot* o gráfico de fuerza. Muestran cómo las palabras de una biografía específica «empujan» la predicción de la variable en concreto hacia valores altos (rojo) o bajos (azul). Se observa una anotación llamada *base value* o valor base que es el valor que habría sido predicho si no hubiera suficientes palabras para realizar una predicción, que no es más que la media de las predicciones.

En el Cuadro 7.3.1 se muestran las biografías que se han escogido para analizar junto con las puntuaciones de cada variable, real y predicha, por los modelos. Se utiliza la notación de las variables dependientes de la sección 6.1.3.

Cuadro 7.3.1: Biografías y puntuaciones asignadas.

Biografía	F	DC	ACF	L	P	Ac	Pred.
Licenciado en Derecho y Ciencias Empresariales (E3) por ICADE, es actualmente socio de las firmas Akiba Partners y Meridia Capital Partners. Ha sido director general para España y Portugal de Dresdner Kleinwort y consejero delegado y responsable de Relaciones con los Inversores de la sociedad de valores BBVA Bolsa. Previamente desempeñó diversas responsabilidades en JP Morgan en México, Nueva York, Londres y Madrid.	**1**	**0**	**0**	**0**	**0**	**0**	**Real**
	0.715	0.5	0.05	0.005	0.028	0.079	LSTM
	0.89	0.97	0	0	0	0	BERT
La carrera profesional de la Sra. Recio se ha desarrollado principalmente en el ámbito de consultoría, telecomunicaciones y transformación digital en compañías como PWC, Grupo AON (Directora de Recursos Humanos para España y Portugal) y Grupo OrangeFrance Telecom, llegando al puesto de Directora General Corporativa y Secretaria General.	**0**	**1**	**0**	**0**	**0**	**0**	**Real**
	0.358	0.918	0.005	0.003	0.01	0.036	LSTM
	0.01	0.96	0	0	0	0.01	BERT
Licenciado en Ciencias Económicas por la Universidad Complutense de Madrid. Pertenece al Cuerpo de Inspectores de Hacienda del Estado. Master en Business Administration (MBA) por el Instituto de Empresa de Madrid. Inició su carrera profesional como Inspector de Hacienda del Estado en la Delegación de Hacienda de Madrid y en la Oficina Nacional de Inspección. En el año 2000 fue nombrado Director General de Tributos del Ministerio de Hacienda, asumiendo posteriormente la Secretaría de Estado de Hacienda.	**0**	**0**	**1**	**0**	**0.7**	**0**	**Real**
	0.028	0.006	1	0.01	0.87	0.021	LSTM
	0.01	0.02	0.97	0.06	0.01	0	BERT

Continúa en la siguiente página

Cuadro 7.3.1: Biografías y puntuaciones asignadas. (Continuado)

Biografía	**F**	**DC**	**ACF**	**L**	**P**	**Ac**	**Pred.**
Abogado del Estado en Excedencia. Fue Vicesecretario del Consejo de Administración de Repsol S.A., Secretario del Consejo de Terra Lycos S.A. y Secretario y Consejero de Altadis S.A.	**0**	**0**	**0**	**1**	**0**	**0**	**Real**
	0	0.002	0.065	1	0.02	0	LSTM
	0	0	0	0.94	0	0	BERT
Ex Director General de la Energía. Ex Secretario General de la Energía y Recursos Minerales. Ex Secretario General Técnico del Ministerio de Industria. Ex Profesor en la Universidad Autónoma de Madrid.	**0**	**0**	**0**	**0**	**1**	**0.3**	**Real**
	0.003	0.24	0.02	0.997	0.997	0.312	LSTM
	0	0	0	0	0.34	0.9	BERT
Nacido en el año 1956 (Roma). Licenciado en Derecho, Doctor en Derecho y Catedrático de Universidad. Presidente de Endesa y de la Fundación Endesa, Socio del Estudio Jurídico Sánchez Calero y Catedrático de Universidad de Derecho Mercantil de la Facultad de Derecho de la Universidad Complutense de Madrid.	**0**	**0**	**0**	**1**	**0**	**1**	**Real**
	0.002	0.008	0.006	0.88	0.02	0.995	LSTM
	0.01	0.02	0	0.11	0	0.96	BERT

7.3.1. Modelo LSTM

Comenzando con el modelo LSTM construido con Fastai, primero tenemos que realizar algunas modificaciones sobre las funciones básicas de la librería para poder calcular los valores SHAP. Se define una función de predicción para cada variable dependiente de la siguiente manera:

```
def predict(lista):
    test_dl = learn.dls.test_dl(pd.DataFrame(lista, columns=['text']))
    preds, _ = learn.get_preds(dl = test_dl)
    preds = torch.nan_to_num(preds)
    return [t[0] for t in preds]
```

La función recibe una lista de cadenas de texto (biografías) y devuelve los valores de predicción para la variable `F` o nivel de experiencia financiero en este caso. Basta con modificar el índice de `t[0]` (de 0 a 5) para obtener el resto de variables dependientes.

También es necesario reescribir el tokenizador para poder construir el gráfico. En este caso tiene que tener un formato similar al un tokenizador de un transformador, por lo que lo definimos de la siguiente manera:

```
def custom_tokenizer(s, return_offsets_mapping=True):
    pos = 0
    offset_ranges = []
    input_ids = []
    for m in re.finditer(r"\W", s):
        start, end = m.span(0)
        offset_ranges.append((pos, start))
        input_ids.append(s[pos:start])
        pos = end
    if pos != len(s):
        offset_ranges.append((pos, len(s)))
        input_ids.append(s[pos:])
    out = {}
    out["input_ids"] = input_ids
    if return_offsets_mapping:
        out["offset_mapping"] = offset_ranges
    return out
```

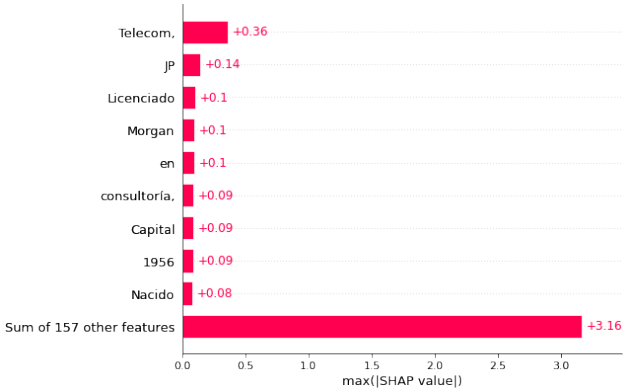
Dividimos la entrada por palabras y obtenemos su numericalización en `input_ids` además del offset de cada palabra dentro de la cadena de texto en `offset_mapping`. Ahora basta con llamar a las funciones básicas de SHAP con las siguientes líneas:

```
1 tokenizer = train_dls.tokenizer
2 explainer = shap.Explainer(predict_F, masker=shap.maskers.Text(custom_tokenizer),
3     max_evals=1500)
4 shap_values = explainer(lista)
```

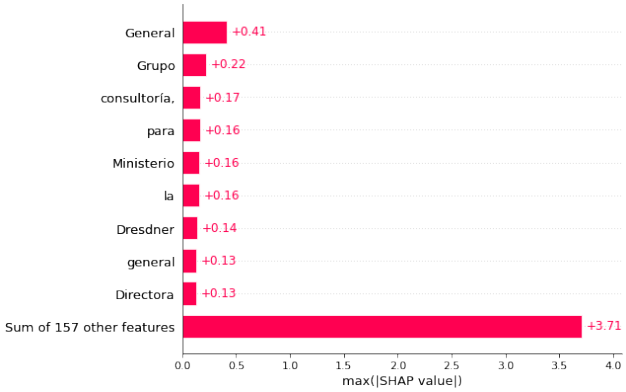
En ellas se proporcionan las funciones que hemos creado anteriormente al constructor del objeto `Explainer` (línea 2) y la lista que contiene las biografías en la siguiente línea. Destacamos que se limitan las permutaciones a 1500 para mejorar el tiempo de ejecución, ya que las funciones hechas a mano no son tan eficientes como las que hay por defecto. No obstante, las puntuaciones seguirán siendo bastante precisas.

Con esto ya tenemos lo necesario, por lo que creamos una lista con las seis biografías elegidas y analizamos las predicciones. Los siguientes gráficos están condicionados a las biografías del Cuadro 7.3.1, debido a que evaluar el conjunto de datos entero llevaría cientos de horas al ser necesario obtener las permutaciones de todas las palabras presentes.

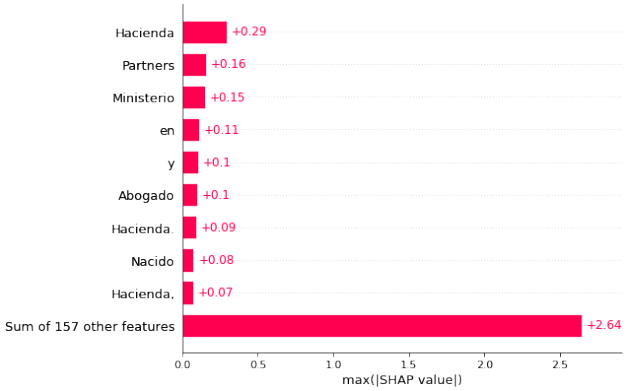
Se presentan en la Figura 7.3.1 los diagramas de barras que muestran los tokens con más influencia en cada una de las variables. Observamos que las variables que presentaban errores más bajos muestran palabras muy relacionadas con la cualidad que representan: *profesor* para Académico, *jurídico* para Legal o *estado* para Político. Sin embargo, en las tres primeras se muestra un modelado más pobre con palabras específicas que no acaban de describir la cualidad, a excepción de *hacienda* en Auditor/Contable/Fiscal.



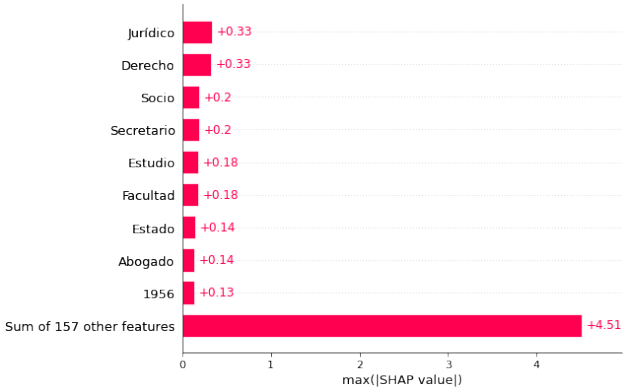
Financiero



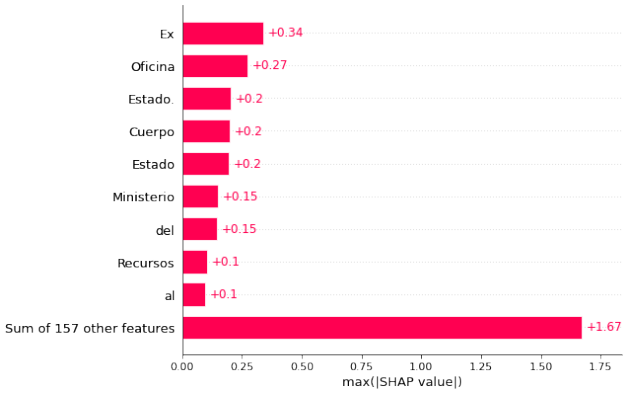
Directivo/Consultor



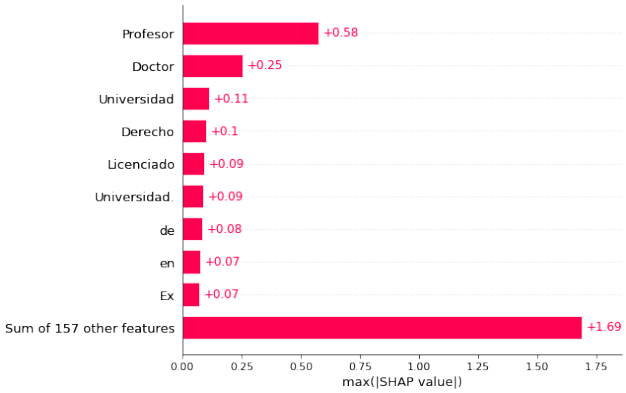
Auditor/Contable/Fiscal



Legal



Político



Académico

Figura 7.3.1: Gráficos de barras para los tokens más importantes en cada variable en LSTM.

Experiencia financiera

En la Figura 7.3.2 comprobamos que los tokens *Capital, JP Morgan, Nueva Tork* empujan la predicción hacia valores cercanos a 1, mientras que *Licenciado en Derecho* contribuye a reducir la predicción a 0.77.

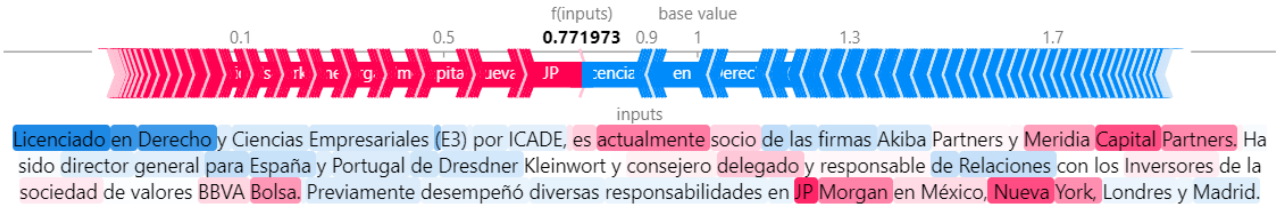


Figura 7.3.2: *Force plot* para una biografía con experiencia financiera.

Experiencia como directivo o consultor

Directora General, consultoría tienen importancia a la vez que *telecomunicaciones y transformación* reducen ligeramente la puntuación, como se observa en la Figura 7.3.3.

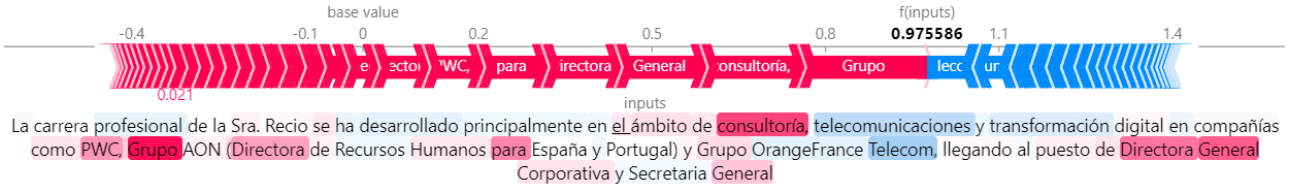


Figura 7.3.3: *Force plot* para una biografía con experiencia como director o consultor.

Experiencia como auditor, contable o fiscal

La predicción es completamente acertada en este caso, donde *Hacienda* tiene casi toda la importancia (ver Figura 7.3.4).

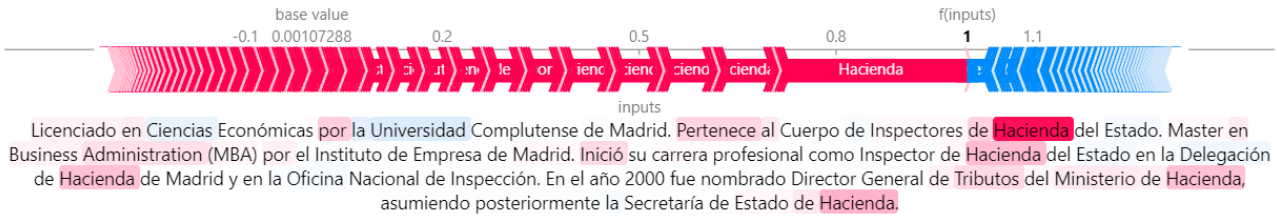


Figura 7.3.4: *Force plot* para una biografía con experiencia como auditor, contable o fiscal.

Experiencia legal

De nuevo, la predicción es acertada y los tokens con importancia son razonables en términos legales, como aparece en la Figura 7.3.5.

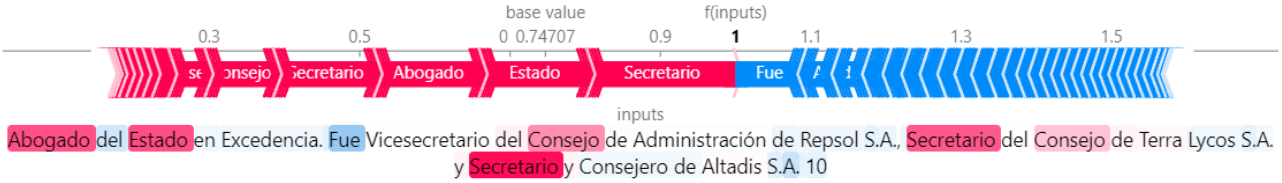


Figura 7.3.5: *Force plot* para una biografía con experiencia legal.

Experiencia como político

En la Figura 7.3.6 observamos que tienen importancia los tokens *Ministerio, Ex* donde el último es razonable ya que la mayoría de biografías con un valor alto en el atributo `P` lo contienen.

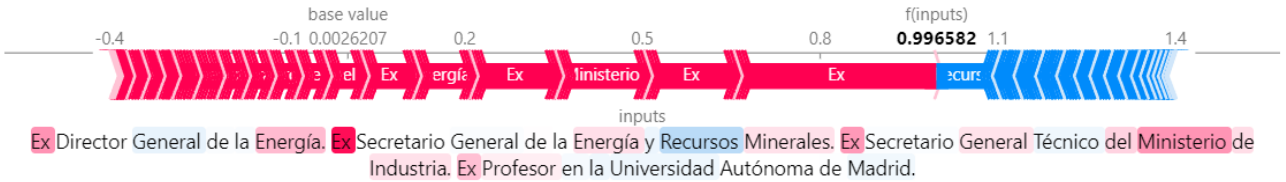


Figura 7.3.6: *Force plot* para una biografía con experiencia política.

Experiencia como académico

Finalmente, en la Figura 7.3.7 las palabras destacadas en rojo son razonables. Reduce ligeramente la puntuación *Presidente*, ya que no suele tener un contexto académico.

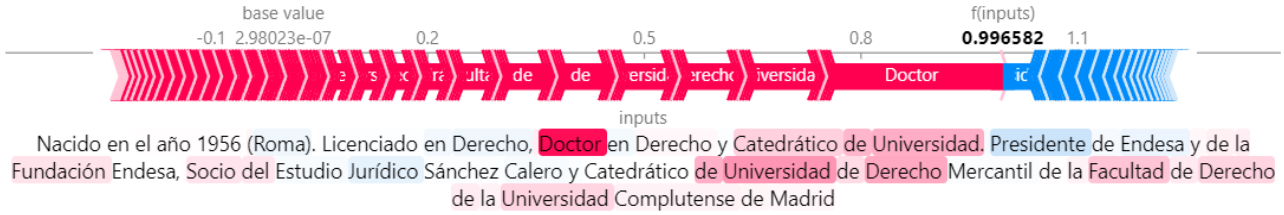
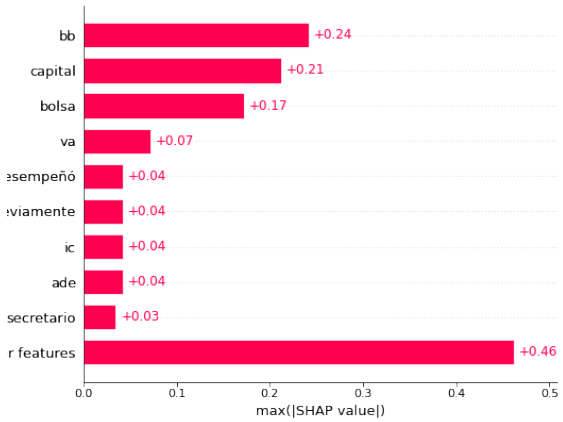


Figura 7.3.7: *Force plot* para una biografía con experiencia académica.

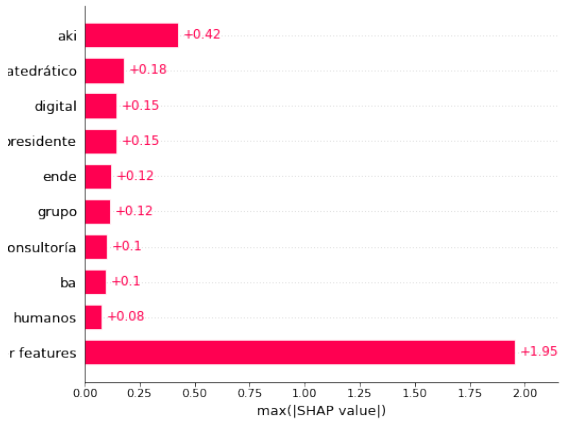
7.3.2. Modelo con transformers

Realizar una interpretación con un modelo de transformadores es más sencillo ya que las funciones por defecto para calcular los valores SHAP son más rápidas y eficientes. Tan solo hay que iniciar el objeto `Explainer` con la función de predicción y el tokenizador utilizado en la red y calcular los valores directamente.

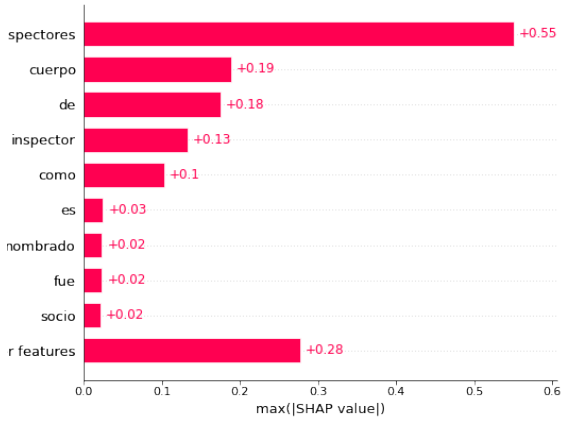
En la Figura 7.3.8 aparecen los diagramas de barras que muestran los tokens con la máxima influencia en cada una de las variables. Comprobamos que generalmente, todas las palabras son coherentes y están relacionadas con la variable. Destaca el gráfico de Directivo/Consultor por no cumplir la anterior premisa, ya que palabras como *catedrático* o digital no deberían relacionarse con el atributo. Esto justifica el elevado error en esta variable que se observó anteriormente.



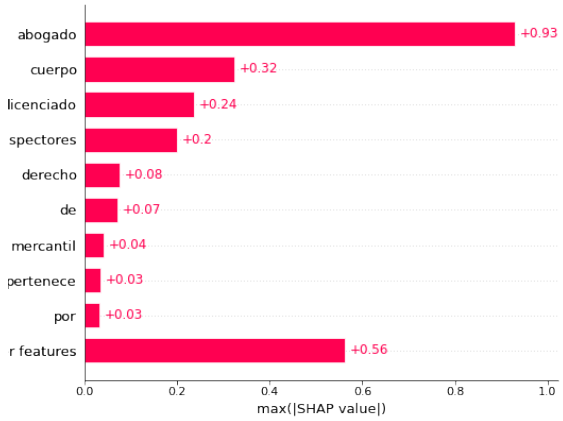
Financiero



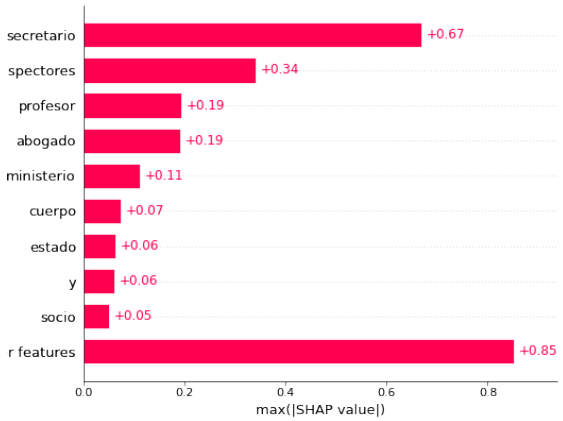
Directivo/Consultor



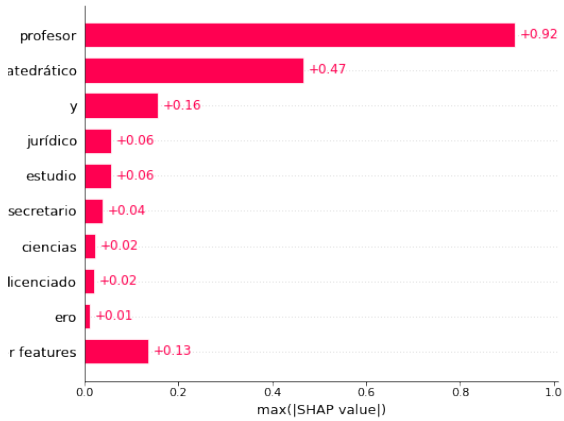
Auditor/Contable/Fiscal



Legal



Político



Académico

Figura 7.3.8: Gráficos de barras para los tokens más importantes en cada variable en BERT.

Experiencia financiera

Destacan *BBVA, bolsa, capital* como se observa en la Figura 7.3.9. Reduce la puntuación *Previamente desempeñó* al estar en pasado, como ocurre con algunas otras biografías. La red parece interpretar que una experiencia en pasado contribuye negativamente a la puntuación.

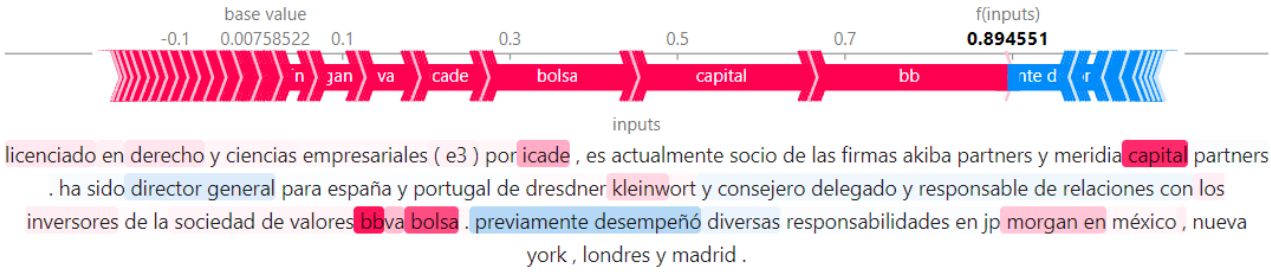


Figura 7.3.9: *Force plot* para una biografía con experiencia financiera.

Experiencia como directivo o consultor

Según la Figura 7.3.10, se reparte la puntuación entre varios tokens. Todos ellos parecen razonables.

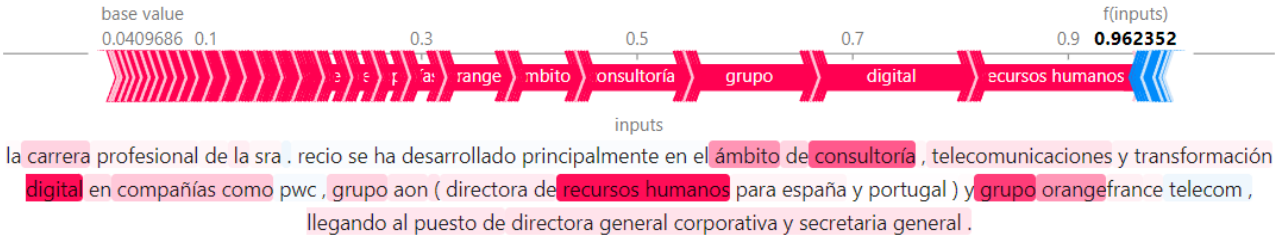


Figura 7.3.10: *Force plot* para una biografía con experiencia como director o consultor.

Experiencia como auditor, contable o fiscal

Las palabras *inspector, cuerpo de inspectores* tienen casi toda la influencia en la predicción (ver Figura 7.3.11).

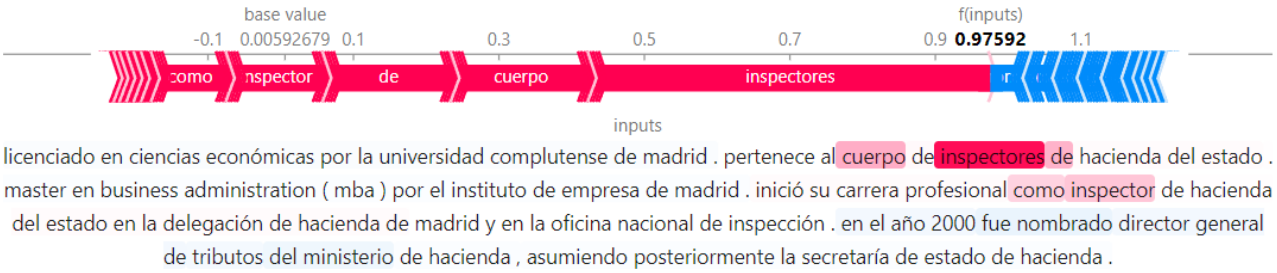


Figura 7.3.11: *Force plot* para una biografía con experiencia como auditor, contable o fiscal.

Experiencia legal

Se relaciona a la perfección *abogado* con la experiencia legal, como aparece en la Figura 7.3.12.

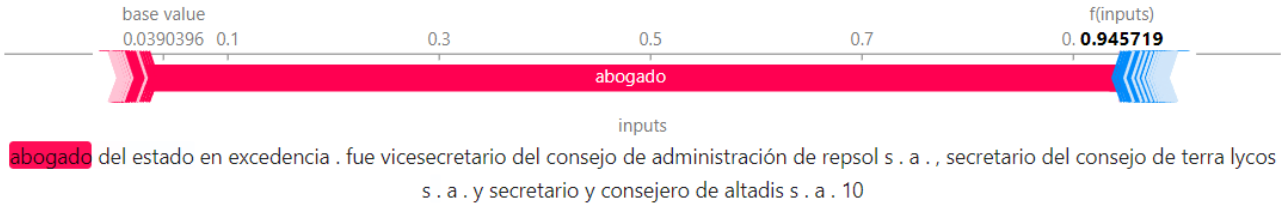


Figura 7.3.12: *Force plot* para una biografía con experiencia legal.

Experiencia como político

La predicción de la Figura 7.3.13 no es buena (siendo el valor real 1). La justificación yace en que *profesor* reduce el valor del atributo. No obstante, las palabras influyentes en el atributo político son correctas: *secretario, ministerio*.

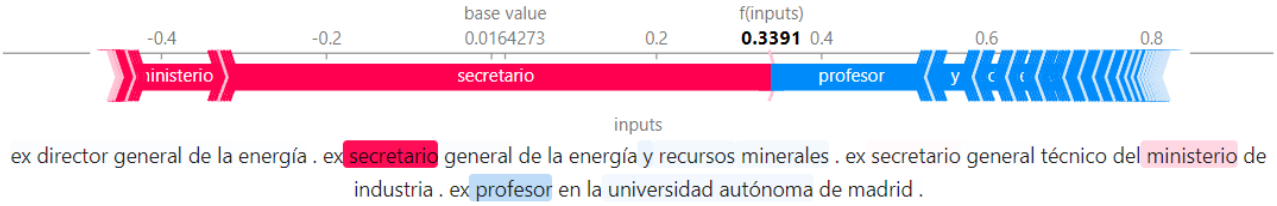


Figura 7.3.13: *Force plot* para una biografía con experiencia política.

Experiencia como académico

Finalmente, en la Figura 7.3.14 comprobamos que *catedrático* acumula casi toda la influencia en la predicción.

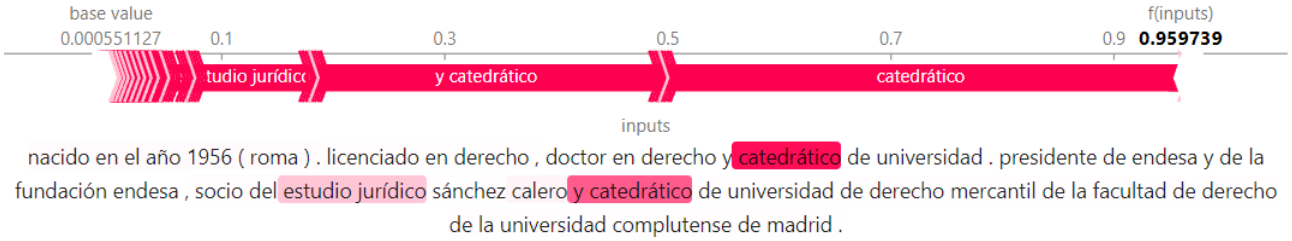


Figura 7.3.14: *Force plot* para una biografía con experiencia académica.

7.4. Comparativa

Para realizar una comparación de arquitecturas nos basaremos en diferentes aspectos que se han ido observando durante su construcción, rendimiento sobre el conjunto de test e interpretabilidad.

- Construcción del modelo de lenguaje. El modelo LSTM utilizó una porción de Wikipedia con aproximadamente 79 millones de tokens y se afinó con los libros de finanzas durante unas 37 horas de entrenamiento que tuvieron que repartirse en 4 sesiones debido a la limitación de 12 horas de Kaggle. El modelo con transformadores fue entrenado con un conjunto de datos mucho más grande (en torno a 300 millones de tokens), por lo que se espera que tenga un mejor ajuste. Como se ha visto en la sección 5.2.3, la calidad de las predicciones es generalmente mejor con el modelo preentrenado. Debido a esto y que el tiempo de entrenamiento por nuestra parte (37 horas contra 10 horas en LSTM y BERT respectivamente) concluimos que el modelo BERT es mejor en este aspecto.
- Construcción del modelo de regresión. Inicialmente fue más sencillo modificar el modelo LSTM para adaptarlo al problema de regresión ya que se utilizó Fastai. Con unas pocas líneas de código se realiza todo el trabajo de entrenamiento, además de encontrar la tasa de aprendizaje óptima con más facilidad. El modelo con transformadores BERT requiere aprender PyTorch y trabajar a un nivel más bajo, lo que llevó más tiempo. El tiempo de entrenamiento fue mucho menor en la red LSTM aunque la diferencia no llegó a ser tan importante como en la etapa anterior. Preparar un modelo con Fastai es más sencillo y rápido, pero ofrece menos flexibilidad a largo plazo. No obstante, para el propósito de este trabajo consideraremos el modelo LSTM como el «ganador» en esta categoría.
- Rendimiento e interpretabilidad. En todas las métricas de calidad de ajuste empleadas (RMSE, MAE y R^2) el modelo con transformadores ha dado mejores resultados. En cuanto a la interpretabilidad, ambos ofrecen resultados razonables en los gráficos que se han planteado. No obstante, comprobamos que el modelo BERT tiende a dar mucho más peso a tokens en concreto (ver Figura 7.3.8), lo que nos lleva a pensar que identifica mejor las palabras que dan peso al atributo en concreto, ya que la mayoría de las veces los resultados eran coherentes. Debido a que la API de SHAP está diseñada en torno a PyTorch, es más sencillo obtener las medidas de interpretabilidad con el modelo BERT.

Con los criterios anteriores, podemos concluir que el modelo BERT ha dado mejores resultados y es más adecuado para realizar el trabajo.

Capítulo 8

Conclusiones y trabajo futuro

En este último apartado se exponen las conclusiones obtenidas a partir del análisis realizado en el proyecto. Además, se presentan posibles mejoras del trabajo y estudios futuros. El código del proyecto se encuentra en el siguiente enlace.

En este trabajo se han propuesto dos modelos de inteligencia artificial con dos arquitecturas (LSTM y *transformers*) que son capaces de analizar el perfil profesional de un consejero a partir de su biografía. Además, se ha obtenido una medida de interpretabilidad que permite observar el razonamiento del análisis que realiza el modelo, lo que tiene un gran valor en el ámbito de las finanzas. Además, se ha profundizado en el funcionamiento de las dos arquitecturas utilizadas, sus ventajas, desventajas e importancia del *transfer learning*.

Durante los últimos años, el procesamiento del lenguaje natural con redes neuronales profundas ha ido ganando importancia gracias a la mejora del *hardware* que permite construir y entrenar modelos de gran tamaño. Gracias a esto, es posible aplicar el PLN a múltiples disciplinas que anteriormente se consideraban demasiado complejas, como lo son el análisis de sentimiento, la predicción de precios o el análisis de consejos de administración.

Conclusiones

- El procesamiento del texto es una etapa esencial para la la construcción de un modelo de lenguaje. Obtener un texto limpio requiere muchas horas de trabajo y conforme se avanza en el proceso de depuración, aumenta la dificultad.
- Utilizar librerías de más alto nivel como Fastai es muy útil para obtener resultados de forma rápida. No obstante, si queremos un mayor control sobre el modelo y una mejor

integración con otras librerías de inteligencia artificial lo más adecuado es dedicar tiempo a aprender PyTorch. A largo plazo, se obtendrán resultados de mejor calidad y en plazos de tiempo reducidos.

- Como se ha mostrado en la Sección 5.2.3, el modelo de lenguaje BERT, obtenido a partir de transfer learning ha dado mejores resultados que AWD-LSTM habiendo requerido menos horas de cómputo. El factor limitante de la calidad de los modelos es el coste computacional de entrenar con grandes cantidades de texto. La transferencia de aprendizaje ha demostrado ser de gran utilidad al proporcionar un modelo más complejo y adecuado.
- Según se muestra en la Sección 7, el resultado final del modelo de regresión a grandes rasgos ha sido similar para ambas arquitecturas. No obstante, el modelo BERT preentrenado y afinado muestra mejores comportamientos en todos los aspectos, reduciendo las métricas del error y explicando una variabilidad de los datos mayor.
- En el ámbito de las finanzas, el razonamiento para la toma de decisiones es muy importante. La interpretabilidad de modelos complejos con SHAP permite en cierta medida obtener la importancia de las palabras y cómo contribuyen a realizar las predicciones. Los resultados de este análisis se muestran en la Sección 7.3.

Trabajo futuro y posibles mejoras

- Utilizar una proporción de datos mayor de Wikipedia para entrenar un modelo de lenguaje más completo. Esto requerirá un mayor tiempo de cómputo.
- Aumentar el tamaño del conjunto de datos etiquetado para regresión, especialmente para el atributo Directivo/Consultor que es el que peores resultados ha dado. Es posible hacerlo manualmente, lo cual generará los mejores resultados o utilizar *data augmentation*.
- Realizar una búsqueda más exhaustiva de hiperparámetros y obtener mejores estimaciones del error mediante validación cruzada.

Valoración personal del proyecto

Este trabajo ha sido de mis primeras experiencias trabajando con proyectos de inteligencia artificial a gran escala. He podido realizar al completo todas las etapas de elaboración de un modelo, desde la recogida de datos hasta la evaluación de los resultados finales. Gracias a ello,

he adquirido una visión global de la importancia y dificultades de cada etapa de desarrollo del modelo.

La obtención y procesamiento de los datos es una parte esencial que requiere una gran cantidad de trabajo, lo que explica el aumento en la demanda de ingenieros de datos que sean de realizar esta tarea de forma eficiente. La tecnología que rodea a las redes neuronales, tanto teórica como práctica, está constantemente actualizándose. Cada pocos meses se presentan nuevas arquitecturas y nuevo *software* que mejora el rendimiento de los modelos, por lo que es importante mantenerse al día y estar al tanto de las nuevas publicaciones.

En general, este proyecto ha sido muy enriquecedor ya que me ha permitido profundizar al máximo en la formación de la mención de Computación y complementar mis conocimientos en otra disciplina como las finanzas. Poder investigar y ampliar los conocimientos que he adquirido durante la carrera ha sido realmente interesante.

Bibliografía

[1] S. Hwang y J. Kim, «Toward a chatbot for financial sustainability», *Sustainability*, vol. 13, n.º 6, pág. 3173, 2021.

[2] F. Rosenblatt, «The perceptron: A probabilistic model for information storage and organization in the brain», *Psychological Review*, vol. 6, n.º 65, págs. 386-408, 1958.

[3] W. McCulloch y W. Pitts, «A logical calculus of the ideas immanent in nervous activity», *The bulletin of mathematical biophysics*, vol. 5, págs. 115-133, 1943.

[4] Wikipedia. «Perceptrón». (2022), dirección: https://es.wikipedia.org/wiki/Perceptr%C3%B3n.

[5] ——, «Activation function». (2022), dirección: https://en.wikipedia.org/wiki/Activation_function.

[6] F. Chollet, *Deep Learning with Python*. Manning, 2017.

[7] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016.

[8] S. Ruder, «An overview of gradient descent optimization algorithms», *ArXiv*, vol. abs/1609.04747, 2016.

[9] H. E. Robbins, «A Stochastic Approximation Method», *Annals of Mathematical Statistics*, vol. 22, págs. 400-407, 2007.

[10] N. Qian, «On the momentum term in gradient descent learning algorithms», *Neural Networks*, vol. 12, n.º 1, págs. 145-151, 1999.

[11] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning representations by back-propagating errors», *Nature*, vol. 323, págs. 533-536, 1986.

[12] PyTorch. «SGD». (2022), dirección: https://pytorch.org/docs/stable/generated/torch.optim.SGD.html#torch.optim.SGD.

[13] I. Sutskever, J. Martens, G. Dahl y G. Hinton, «On the importance of initialization and momentum in deep learning», Proceedings of Machine Learning Research, vol. 28, n.º 3, S. Dasgupta y D. McAllester, eds., págs. 1139-1147, 2013.

[14] G. Hinton, *Overview of mini-batch gradient descent*.

[15] D. P. Kingma y J. Ba, «Adam: A Method for Stochastic Optimization», 2017.

[16] J. Zhang e I. Mitliagkas, «YellowFin and the Art of Momentum Tuning», 2018.

[17] I. Loshchilov y F. Hutter, «Decoupled Weight Decay Regularization», 2019.

[18] Educative. «Overfit». (2022), dirección: https://www.educative.io/edpresso/overfitting-and-underfitting.

[19] Research Gate. «Dropout». (2022), dirección: https://www.researchgate.net/figure/Dropout-neural-network-model-a-is-a-standard-neural-network-b-is-the-same-network_fig3_309206911.

[20] M. A. Nielsen, *Neural Networks and Deep Learning*, 2018.

[21] S. Y. Feng, V. Gangal, J. Wei y col., «A Survey of Data Augmentation Approaches for NLP», 2021.

[22] J. Wei y K. Zou, «EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks», págs. 6382-6388, nov. de 2019. dirección: https://aclanthology.org/D19-1670.

[23] R. Sennrich, B. Haddow y A. Birch, «Improving Neural Machine Translation Models with Monolingual Data», págs. 86-96, ago. de 2016. DOI: 10.18653/v1/P16-1009. dirección: https://aclanthology.org/P16-1009.

[24] A. M. Turing, «Computing Machinery and Intelligence», *Mind*, New Series, vol. 59, n.º 236, págs. 433-460, 1950.

[25] T. Mikolov, I. Sutskever, K. Chen, G. Corrado y J. Dean, «Distributed Representations of Words and Phrases and their Compositionality», *CoRR*, vol. abs/1310.4546, 2013.

[26] D. Masters y C. Luschi, «Revisiting Small Batch Training for Deep Neural Networks», *CoRR*, 2018.

[27] Wikipedia. «Recurrent neural network». (2022), dirección: https://en.wikipedia.org/wiki/Recurrent_neural_network.

[28] Towards Data Science. «Redes neuronales recurrentes». (2022), dirección: https://towardsdatascience.com/understanding-rnns-lstms-and-grus-ed62eb584d90.

[29] Wikipedia. «Redes neuronales recurrentes». (2022), dirección: https://es.wikipedia.org/wiki/Red_neuronal_recurrente.

[30] I. Sutskever, «Training recurrent neural networks», *University of Toronto, Toronto, Ont., Canada*, 2013. dirección: http://www.cs.utoronto.ca/~ilya/pubs/ilya_sutskever_phd_thesis.pdf.

[31] S. Hochreiter y J. Schmidhuber, «Long Short-term Memory», *Neural computation*, vol. 9, págs. 1735-80, dic. de 1997.

[32] Wikipedia. «Long short-term memory». (2022), dirección: https://en.wikipedia.org/wiki/Long_short-term_memory.

[33] S. Merity, N. S. Keskar y R. Socher, «Regularizing and Optimizing LSTM Language Models», *CoRR*, 2017. dirección: http://arxiv.org/abs/1708.02182.

[34] A. Vaswani, N. Shazeer, N. Parmar y col., «Attention Is All You Need», 2017.

[35] ". N. del Mercado de Valores", «El accionista de una compañía cotizada», *Guía de la CNMV*,

[36] ——, «Informes de gobierno corporativo de las entidades emisoras de valores admitidos a negociación en mercados regulados», 2020.

[37] S. Gray y J. Nowland, «The diversity of expertise on corporate boards in Australia», *Accounting & Finance*, vol. 57, n.º 2, págs. 429-463, 2017.

[38] J. Howard y S. Ruder, «Fine-tuned Language Models for Text Classification», *CoRR*, vol. abs/1801.06146, 2018. dirección: http://arxiv.org/abs/1801.06146.

[39] Google Colab. «Transfer learning». (2022), dirección: https://colab.research.google.com/github/juansensio/blog/blob/master/044_cnn_transfer_learning/cnn_transfer_learning.ipynb.

[40] Wikipedia. «PyTorch». (2022), dirección: https://en.wikipedia.org/wiki/PyTorch.

[41] ——, «CUDA». (2022), dirección: https://en.wikipedia.org/wiki/CUDA.

[42] ——, «Automatic differentiation». (2022), dirección: https://en.wikipedia.org/wiki/Automatic_differentiation.

[43] J. Howard y S. Gugger, «fastai: A Layered API for Deep Learning», *CoRR*, vol. abs/2002.04688, 2020. dirección: https://arxiv.org/abs/2002.04688.

[44] Hugging Face. «Hugging Face». (2022), dirección: https://huggingface.co/.

[45] Google LLC. «Kaggle». (2022), dirección: https://www.kaggle.com/.

[46] J. C. Arias y R. Stern, «Review of risk management methods», *Business Intelligence Journal*, vol. 4, n.º 1, págs. 51-78, 2011.

[47] Instituto Nacional de Estadística. «Salario medio anual por sector». (2022), dirección: https://www.ine.es/jaxiT3/Tabla.htm?t=10911.

[48] NVIDIA. «Tesla P100 PCIe 16GB». (2022), dirección: https://www.nvidia.com/en-us/data-center/tesla-p100/.

[49] Wordfrequency. «Considerations when obtaining a word list». (), dirección: https://www.wordfrequency.info/comparison.asp.

[50] G. Attardi. «WikiExtractor». (2015).

[51] Microsoft. «BlingFire». (2021).

[52] C. E. Rodríguez, *Diccionario de economía : etimológico, conceptual y procedimental : edición especial para estudiantes*. 2013.

[53] P. D. Jonathan Berk, *Finanzas corporativas*. 2008.

[54] M. A. Moya, *Fundamentos de economía*. 2012.

[55] F. Alburquerque, *Conceptos básicos de economía: En busca de un enfoque ético, social y ambiental*. 2018.

[56] S. T. S.A., *El dinero y el ahorro: Un buen mañana se planifica hoy*. 2006.

[57] E. F. B. Michael C. Ehrhardt, *Finanzas corporativas*. 2007.

[58] R. W. Jaffe, *Finanzas corporativas*. 2012.

[59] B. M. Allen, *Principios de finanzas corporativas*. 2010.

[60] The Apache Software Foundation. «Tika». (2022).

[61] J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding», 2018.

[62] J. Cañete, G. Chaperon, R. Fuentes, J.-H. Ho, H. Kang y J. Pérez, «Spanish Pre-Trained BERT Model and Evaluation Data», en *PML4DC at ICLR 2020*, 2020.

[63] J. Cañete, *Compilation of Large Spanish Unannotated Corpora*, Zenodo, mayo de 2019. dirección: https://doi.org/10.5281/zenodo.3247731.

[64] J. Vig, «A Multiscale Visualization of Attention in the Transformer Model», en *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Florence, Italy: Association for Computational Linguistics, 2019, págs. 37-42.

[65] K. Clark, U. Khandelwal, O. Levy y C. D. Manning, «What Does BERT Look At? An Analysis of BERT's Attention», 2019.

[66] CNMV. «Tabla de informe de gobierno corporativo». (2022), dirección: https://www.cnmv.es/portal/home.aspx.

[67] S. Lundberg y S.-I. Lee, «A Unified Approach to Interpreting Model Predictions», 2017.