ANÁLISIS DE CONTENIDO EN CONVERSACIONES USANDO MACHINE LEARNING

Autor: Jorge Corral Losada

(enero, 2021)

1. Introducción

El proyecto planteado en estas prácticas era el de un modelo funcional capaz de predecir el tema de conversación de un audio usando Machine Learning. Para llevar a cabo el objetivo de dicho proyecto, hemos creado primeramente un modelo basado en TF-IDF y usando scikit-learn, y con el cual hemos alcanzado un porcentaje de éxito del 92,6%. Posteriormente hemos realizado un segundo modelo basado en redes neuronales y utilizando TensorFlow y Keras. Con este modelo se ha obtenido un porcentaje de acierto del 91.2%. La memoria consta principalmente de: una introducción, donde se contextualiza el trabajo; los objetivos que han de llevarse a cabo a lo largo del proyecto; la planificación que se ha seguido; el núcleo del trabajo, donde se explica en detalle cada una de las partes que componen el proyecto; los resultados del proyecto; las conclusiones que saco de haber realizado este trabajo; así como un anexo con el código empleado y la lista de referencias bibliográficas. El resultado de este proyecto ha sido muy satisfactorio. Creo que se ha logrado un modelo que funciona bastante bien, al menos con una tasa de acierto superior a la que al principio del proyecto me podía esperar lograr. He podido profundizar en el campo de la Inteligencia Artificial y aprender conceptos, ideas y herramientas muy útiles para el presente y para mi futuro, además de servirme como una primera toma de contacto con el mundo laboral.

Para comenzar nuestro trabajo, debemos definir nuestro proyecto bajo el marco de la inteligencia artificial y el procesamiento del lenguaje natural, aplicando diferentes técnicas de procesado de texto y clasificación de este texto en diferentes categorías predefinidas. Posteriormente, aplicamos el reconocimiento de voz con las técnicas aplicadas anteriormente para conseguir una conexión entre el reconocimiento de voz y el procesamiento del lenguaje natural.

En primer lugar, hablaremos de la inteligencia artificial. Esto es el intento de imitar la inteligencia humana mediante el uso de un software. Es la capacidad de un sistema para interpretar correctamente datos externos, para aprender de dichos datos y emplear esos conocimientos para lograr tareas y metas concretas.

Otro concepto importante es el de procesamiento de lenguaje natural, que es el campo dentro de la inteligencia artificial y la lingüística aplicada que estudia las interacciones mediante uso del lenguaje natural entre los seres humanos y las máquinas. No trata de la comunicación por medio de lenguas naturales de una forma abstracta, sino de diseñar

mecanismos para comunicarse que sean eficaces computacionalmente, es decir, que se puedan realizar por medio de programas que ejecuten o simulen la comunicación. La inteligencia artificial debería de ser capaz de interpretar lo que le escriben y ofrecer una respuesta adecuada. Hace posible que los ordenadores lean texto, escuchen la voz hablada, la interpreten, midan el sentimiento y determinen qué partes son importantes. Actualmente el NLP se utiliza en diferentes áreas entre las que podemos destacar la comprensión del lenguaje natural, la detección de sentimientos, la traducción automática o el resumen y clasificación de textos.

Aplicaremos ambos conceptos para obtener una inteligencia capaz de entender el lenguaje natural y de clasificarlo, mediante el uso de reconocimiento de voz. El reconocimiento de voz es la capacidad de una máquina o programa para identificar palabras y frases en lenguaje hablado y convertirlas a un formato legible por máquina. Actualmente las aplicaciones más comunes de esta tecnología son el dictado automático, el control por comandos y la telefonía, así como sistemas portátiles y sistemas que hagan más accesible su uso para discapacitados.

2. Objetivos

Los objetivos para aprender que se nos plantearon a principio de este curso eran estos:

- Familiarizarme con el término aprendizaje automático, así como con los métodos existentes y su funcionamiento.
- Aprender lo que es la clasificación de los datos dentro del aprendizaje automático.
- Conocer la forma correcta de clasificar datos con Python y llevarlo a cabo para lograr la épica del proyecto.
- Aprender e investigar acerca de la idea sobre la que gira el proyecto, el procesamiento de lenguajes naturales.
- Profundizar en el concepto TF-IDF y crear un modelo basado en este.
- Crear un modelo basado en redes neuronales y conocer el funcionamiento de estas.
- Conocer las métricas que podemos aplicar a los modelos de nuestro proyecto y ser capaces de seleccionar aquellas que mejores resultados nos proporcionan.
- Profundizar en la programación orientada a objetos a través del lenguaje de programación Python.
- Desarrollar nuestro código en un entorno nuevo para nosotros como es Google Colaboratory.
- Conseguir un buen modelo de análisis de contenido funcional, que posea un alto porcentaje de éxito.

Después de varias iteraciones y con el seguimiento de nuestra tutora para la consecución de nuestro objetivo principal, que era la creación de un modelo funcional, podemos asegurar que todos estos conceptos han sido desarrollados e implementados en nuestro modelo final, adquiriendo al fin una cantidad de conocimientos aplicable para otro tipo de proyectos, como puede ser la programación orientada a objetos de python y el uso del Google Colab.

3. Planificación

Nuestra planificación ha sido bastante libre, siguiendo desde el principio lo que hemos puesto en la plataforma de administración de proyectos Trello, como podemos observar en la figura 3.1. Gracias a esta herramienta, y el seguimiento por Whatsapp, hemos conseguido ir coordinando nuestros esfuerzos para ir sacando un modelo funcional. Por lo tanto, podemos ir definiendo nuestra aplicación en diferentes estados de desarrollo y diseño, que iremos describiendo en cada uno de los siguientes párrafos.

- **Lectura de los papers**: En este apartado, hemos tardado en completarlo las dos primeras semanas, lo que ha resultado en una lectura concienzuda de los papers, debido a su complejidad y al desconocimiento sobre el tema sobre el que trataban. Una vez que la lectura y comprensión de estos se terminó, lo siguiente que hicimos fue decidir cuáles de los papers realizaremos, debido a que había uno, el segundo, que no nos llamaba la atención en absoluto. Este consiste en una técnica de tratamiento de textos como imágenes, que no nos interesaban
- **Selección del dataset**: Para esta parte, decidimos aplicar nuestros esfuerzos en conseguir un dataset que nos sirviese para la tarea de clasificación que queríamos realizar.
- **Conocer las tecnologías y técnicas** que íbamos a utilizar, ya que nuestro conocimiento sobre los textos y su clasificación y limpieza era muy limitado, dándonos nuestra tutora unas pautas a seguir y qué tecnologías era recomendable utilizar.
    - **Aprender módulo Pandas:** Librería para el análisis de datos que cuenta con las estructuras de datos que necesitamos para limpiar los datos en bruto y que sean aptos para el análisis. Con ayuda del mismo crear el dataset que se usará en la práctica.
    - **Aprender módulo speech recognition:** Módulo que se usó para transcribir los audios a texto y poder trabajar con ellos.
    - **Aprender Word2vec:** Word2vec es un grupo de modelos relacionados que se utilizan para generar vectores de palabras.
    - **Aprender Keras y TensorFlow:** TensorFlow es una librería de código libre para computación numérica (usando grafos de flujo de datos) creada por Google Brain y que permite principalmente procesar fácilmente

matrices o tensores (de 2 dimensiones). Keras, es otro framework para trabajar con redes neuronales pero a más alto nivel, mediante APIs. Aprendimos a usar esta tecnología para poder crear los modelos.

- **Realización del primer modelo**: Para la realización de este modelo, lo que hicimos fue seguir varios tutoriales de scikit-learn para poder adaptarlos a lo que nosotros necesitábamos en ese momento.
- **Realización de las variantes del tercer modelo:** Para esto, dividimos nuestros esfuerzos en crear una variedad de modelos para escoger de ellos qué tipos de capas de redes neuronales eran las más adecuadas para el desarrollo de nuestro proyecto.

4. Núcleo del trabajo

El proyecto desarrollado consiste en la implementación de 3 modelos de predicción de texto, Tf-idf y redes neuronales. Por otra parte, el proyecto cuenta con la capacidad de convertir audio en texto a partir de un archivo con extensión .wav, para posteriormente clasificar el texto obtenido.

El Tf-idf es un método que verifica la relación entre el documento y la palabra, para concretar si su uso es importante en el contexto en el que se aplica.

Las redes neuronales son un concepto utilizado en la inteligencia artificial para hacer software que sea capaz de imitar el razonamiento y pensamiento humano para aplicarlo a diferentes situaciones o problemas que nos surjan.

4.1. TECNOLOGÍA UTILIZADA

La implementación de los modelos de predicción se ha realizado en el lenguaje de programación Python y desarrollado y ejecutado a través de la plataforma Google Colaboratory. A su vez, se ha hecho uso de diversos módulos del lenguaje Python para facilitar la implementación de los modelos:

- Google.colab: Habilita la posibilidad de conectar la plataforma Google Colaboratory con Google Drive para acceder a archivos. Colab es un servicio cloud, basado en los Notebooks de Jupyter, que permite el uso gratuito de las GPUs y TPUs de Google, con librerías como: Scikit-learn, PyTorch, TensorFlow, Keras y OpenCV. Todo ello con bajo Python 2.7 y 3.6.
- Scikit-learn: Módulo que ofrece diversas funciones para implementar aprendizaje automático.
- TensorFlow: Nos permite generar redes neuronales con alta exactitud y se centra en la simplicidad y facilidad de uso.
- Keras: Biblioteca de redes neuronales capaz de ejecutarse en TensorFlow. Está especialmente diseñada para posibilitar la experimentación en mas o menos poco tiempo con redes de Deep Learning.
- Joblib: Módulo que ofrece diversas funciones para implementar pipelining ligero.
- Gensim.parsing.preprocessing: Módulo que ofrece diversas funciones para la limpieza de cadenas de texto, cómo eliminar una lista de stopwords.

- Speech Recognition: Módulo que ofrece diversas funciones para el reconocimiento de voz y su transcripción a texto.
- Soundfile: Módulo que ofrece diversas funciones para la lectura y escritura de archivo de audio.
- Pandas: Módulo que ofrece diversas funciones para el análisis y manipulación de datos. Se utiliza para extraer los datos y almacenarlos en el formato necesario.
- RegEx: Las expresiones regulares se pueden utilizar para describir cadenas y números de cadenas en una forma lógica general con el fin de buscarlas, sustituirlas, manipularlas o procesarlas en documentos, código fuente o una base de datos.
- Numpy: Módulo que ofrece diversas funciones para fines científicos computacionales, como crear y manipular matrices multidimensionales.
- Nltk: Módulo que ofrece diversas funciones para programas que requieran trabajar con lenguaje natural.
- Matplotlib.pyplot: Módulo que ofrece diversas funciones para la creación y representación de gráficos.
- Seaborn: Librería de visualización de datos de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos.

Por otra parte, otros módulos fueron probados sin éxito y por ello hubo que descartarlos:

- String: Módulo que ofrece diversas funciones referentes a cadenas de texto.
  Se probó su implementación para la limpieza de caracteres especiales (signos, números, letras, símbolos, figuras de las diferentes monedas del mundo…), pero no ofreció resultados satisfactorios.
- TensorFlow puro: Módulo que ofrece diferentes herramientas de trabajo en inteligencia artificial. Descartado para mezclarlo con Keras y facilitar nuestra comprensión sobre las capas utilizadas.

4.2. MODELO DE DATOS

Los algoritmos empleados son de carácter supervisado. Esto implica que cada dato del conjunto facilitado para el entrenamiento de los modelos de predicción debe estar asignado a una etiqueta o categoría.

Teniendo en cuenta el aspecto anterior, el conjunto de datos de entrada, dataset, utilizado inicialmente en la implementación de este modelo consiste en un conjunto de noticias, escritas en inglés, de carácter público recogidas por la cadena de televisión BBC entre los años 2004 y 2005. El total de noticias asciende a 2.225, cada una de ellas asignada a una categoría según su tema de discusión: "business" (negocios), "entertainment" (entretenimiento), "politics" (política), "sport" (deporte) y "tech" (tecnología).

En cuanto a su tratamiento, para este modelo se comenzó utilizando archivos dataset con extensión de archivo .csv, así como un archivo txt que contenía los mismos datos y que en ocasiones era más fácil de tratar. El problema radica en que al hacer uso del módulo Pandas, el contenido del archivo con extensión .csv era convertido en una estructura de datos tipo lista, esto es, ["Texto 1", "Texto 2", Texto 3"], lo cual ocasiona conflictos y dificulta el procesado de la información. Haciendo uso del archivo con extensión .txt, el módulo Pandas convierte la información introducida en una variable de tipo de dato string (cadena de caracteres), con formato "Texto 1 Texto 2 Texto 3", más fácil de trabajar. El dataset en formato csv fue creado gracias a NumPy y Pandas, pero debido a la forma en la que se encontraban los datos, al ser noticias cada una de ellas venia en un txt, dentro del cual se encontraba el titulo de la noticia y separado por líneas en blanco cada oración del cuerpo de la misma, lo que nos llevó a tener que limpiarla de caracteres creados por esos saltos de línea entre otros después de haber importado a todas ellas al .csv.

En la siguiente figura 4.1, mostramos la distribución por categoría del dataset de las noticias de la BBC:

Categoría	Número de noticias
business	534
entertainment	488
politics	417
sport	402
tech	380

Tabla 4.1. Total de noticias de cada categoría

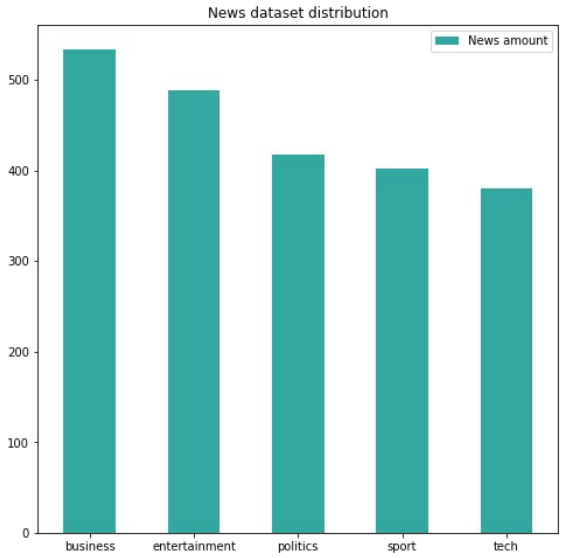

Figura 4.1. Gráfica de noticias por categoría

Por último, destacar la organización del dataset una vez que insertado en una estructura de datos funcional aplicando librerías de Python, tales como Pandas o Gensim. En la figura 4.2 podemos ver como han quedado las noticias con sus respectivas categorías.

```
                                                  news  category
0     ad sales boost time warner profit quarterly pr...  business
1     dollar gains greenspan speech the dollar hit h...  business
2     yukos unit buyer faces loan claim the owners e...  business
3     high fuel prices hit ba's profits british airw...  business
4     pernod takeover talk lifts domecq shares uk dr...  business
...                                                 ...       ...
2216  new consoles promise big problems making games...      tech
2217  bt program beat dialler scams bt introducing t...      tech
2218  spam e-mails tempt net shoppers computer users...      tech
2219  be careful code a new european directive could...      tech
2220  us cyber security chief resigns the man making...      tech

[2221 rows x 2 columns]
```

Figura 4.2. Muestra de las noticias del dataset en un dataframe

4.3. PREPROCESAMIENTO DE DATOS

En este apartado vamos a explicar los métodos de limpieza y uso de los datos que vienen de los dataset que hemos escogido. Para ello, lo primero que hemos hecho ha sido el dataset en formato txt, con la noticia y su categoría. Pero las noticias tienen algún que otro inconveniente para su uso en la clasificación de textos

Lo primero que hemos tenido que solucionar ha sido la eliminación de los signos de puntuación, para que no fuesen vectorizados por el programa y, por lo tanto, tratados como una de las palabras importantes del texto. Para ello, hemos utilizado una función que

permite eliminar de los textos todos los puntos, comas y signos de puntuación correspondientes, para así, mejorar la capacidad de acierto del modelo.

El otro problema de limpieza de texto que nos hemos encontrado ha sido la repetición de stopwords en todas las noticias. Estas stopwords son aquellas palabras que no aportan ningún significado a la oración o texto y que, por lo tanto, solo son útiles para la comunicación oral, no para una clasificación del contenido de un texto. Para ello, hemos utilizado una función auxiliar que las detecta y elimina del texto.

El último problema que hemos encontrado ha sido en la categorización de las noticias, que, al ser palabras, los diferentes modelos fallaban al no poder relacionar sus cálculos con palabras. Este problema lo hemos solucionado indexando cada una de las categorías, esto es, atribuyendo un número a cada categoría, para facilitar la implementación de nuestros modelos.

4.4. MODELO TF-IDF

4.4.1. MODELOS UTILIZADOS

Para llevar a cabo la tarea de clasificación de texto haciendo uso de la medida Tf-idf, hemos empleado 3 algoritmos de aprendizaje supervisado de manera independiente y comparado sus resultados. De forma general, los algoritmos han dado buenos resultados con los ajustes adecuados, rondando entre los 3 algoritmos una media de acierto de entre el 90% y 95% en la mayoría de las pruebas.

TF-IDF (term frequency-inverse document frequency) es una medida estadística que evalúa que tan relevante es una palabra para un documento dentro de un conjunto de documentos. Esto se calcula a través de dos parámetros, el TF(Term Frequency), que es la frecuencia con la que una palabra aparece en el texto, es decir el número de repeticiones de esa palabra dividido por el número total de palabras del texto. El otro parámetro es el IDF(Inverse Document Frequency), que viene a ser como de rara es una palabra en el conjunto de textos. Cuanto más cercano a 0 sea este valor, más común será la palabra. Este valor puede ser calculado tomando el número total de textos y dividiéndolo por el número de textos que contienen la palabra y calculando el logaritmo de eso. Multiplicando ambos valores obtendremos la puntuación de una palabra. Cuanto mayor sea este valor, más relevante será la palabra en ese documento en particular.

Necesitaremos vectorizar el texto, es decir, transformar el texto en números, pues los algoritmos usados en Machine Learning trabajan con números, y lo que nosotros tenemos

es texto. Al vectorizar estamos creando vectores de palabras, que representan un texto como una lista de números, con un número para cada palabra. Tf-idf nos permite asociar cada palabra del texto con un número que representa cuán relevante dicha palabra es en ese documento. Por lo tanto, documentos que posean palabras relevantes iguales poseerán vectores similares. Una vez vectorizado podemos usar la puntuación del Tf-idf en algoritmos como los que explicaremos a continuación para mejorar los resultados.

El primer clasificador que se ha probado en esta clase hace uso del algoritmo Multinomial Naive Bayes. El algoritmo simple de Naive Bayes consiste en una ecuación que describe la relación de probabilidades condicionales.

$$P(h|D) = P(D|h)P(h)/P(D) \quad \text{(Ecuación 4.1)}$$

El algoritmo busca la probabilidad de que el texto bajo estudio pertenezca a una clase concreta a partir del conjunto de datos previamente estudiados. Los clasificadores Naive Bayes han funcionado bastante bien en muchas situaciones del mundo real, como la clasificación de documentos y el filtrado de spam. Requieren una pequeña cantidad de datos de entrenamiento para estimar los parámetros necesarios. Los Naive Bayes son algoritmos probabilísticos para predecir la clasificación de un texto. La forma en que obtienen estas probabilidades es utilizando el Teorema de Bayes, que describe la probabilidad de una característica, basándose en el conocimiento previo de las condiciones que podrían estar relacionadas con esa característica.

El algoritmo se amplía haciendo uso de la distribución multinomial (MultinomialNB), que es una generalización de la distribución binomial, un tipo de distribución de la probabilidad. Entre los diversos parámetros que el algoritmo presenta, el que mayor interés ocupa es el parámetro de suavizado aditivo (Additive Smoothing), alpha. Este parámetro está involucrado en la técnica de suavizado de datos categóricos y admite valores entre 0 y 1.

El siguiente modelo que se probó fue el SVM, que son las máquinas de vectores de soporte, las cuales permiten encontrar la forma óptima de clasificar entre varias clases. Conocidos los datos(los puntos en la figura 4.3), el clasificador genera el hiperplano (la linea roja de la figura 4.3), que separa las etiquetas. Dependiendo de donde caiga el dato lo clasificará de una manera o de otra. La clasificación óptima se realiza maximizando el margen de separación entre las clases. Los vectores que definen el borde de esta separación son los vectores de soporte.

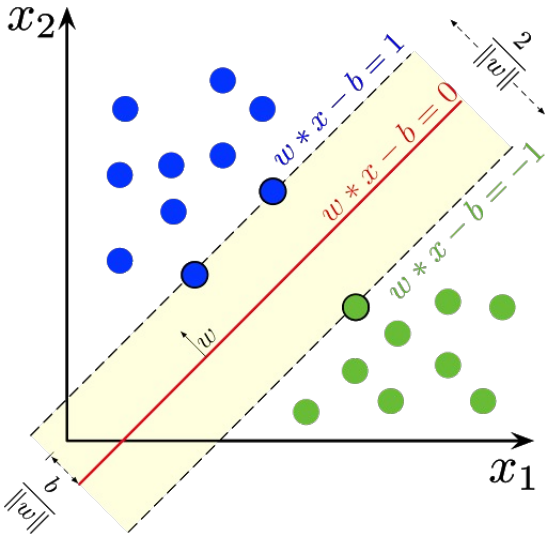

Figura 4.3. Ejemplo del SVM

El último algoritmo empleado en el modelo es el conocido KNN (k vecinos más cercanos). Este modelo sirve esencialmente para clasificar valores buscando los puntos de datos "más similares" (por cercanía) aprendidos en la etapa de entrenamiento y haciendo conjeturas de nuevos puntos basado en esa clasificación. Es un algoritmo supervisado, es decir, que tenemos etiquetado nuestro conjunto de datos, y basado en instancia, esto quiere decir que no aprende explícitamente de un modelo, sino que memoriza las instancias de entrenamiento que son usadas como base para la fase de predicción. En la figura 4.4. podemos ver un ejemplo de KNN.

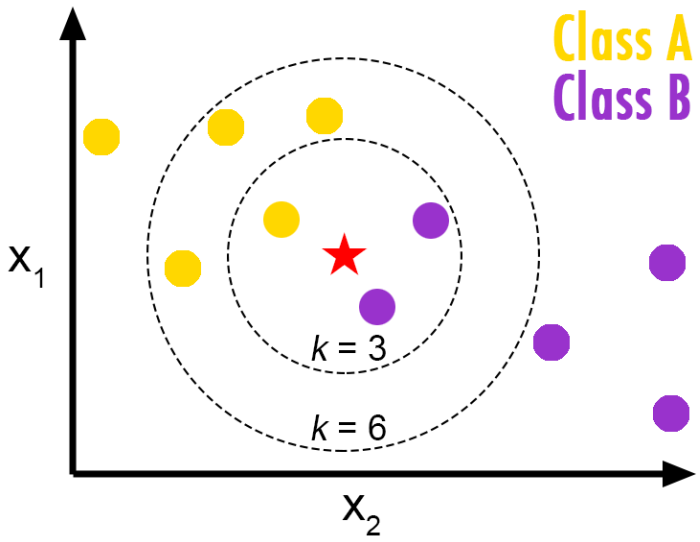

Figura 4.4. Ejemplo del KNN

4.4.2. ESTRUCTURA Y FUNCIONAMIENTO

El modelo está construido en torno a la clase "ContentAnalyzer", compuesta de los siguientes métodos:

- **Método: __init__ (self, path_model, txt_path, categories):**

  Argumentos:

  - self: *ContentAnalyzer*. Instancia del objeto de la clase ContentAnalyzer.
  - path_model: *String*. Ubicación del donde se va a guardar el modelo entrenado.
  - txt_path: *String*: Ubicación en la que se encuentra el fichero del dataset.
  - categories *list(String)*: Lista con cada una de las categorías de noticias.

  Desarrollo:

  Método predefinido por el modelo de clases de python que se utiliza a modo de constructor. Este es el que inicia cada una de las partes que componen el modelo y sus argumentos, que estarán en un alcance de clase y podrán ser utilizados en cada uno de los métodos que tengan como argumento el self.

- **Método: fit(self, dataset_part)**

  Argumentos:

  - self: *ContentAnalyzer*. Instancia del objeto de la clase ContentAnalyzer.
  - dataset_part: *pandas.dataframe:* Argumento que sirve para definir qué parte del dataset general mandamos. Es decir, podemos enviar al método X_train-Y_train, X_val-Y_val, etc.

  Desarrollo:

  Método de la clase ContentAnalyzer que sirve para entrenar un modelo y guardarlo en el atributo de la clase **text_clf** para que podamos usarlo en cada una de las siguientes partes de la clase.

  Lo primero que ocurre en este método es la transformación de la parte del dataset en una array de conteo de frecuencia de palabras gracias a la clase de CountVectorizer(). Posteriormente, lo que hace la clase es transformar ese array de conteo en un TF-IDF array, que lo que hace es convertirlo todo un conteo de frecuencia, pero basado en la relevancia que tiene la palabra que vamos a

vectorizar, para ver qué impacto tiene en la clase. Todo ello gracias a la clase TfidfTransformer.

A continuación, crea el clasificador elegido. Todo ello, se agrupa en un pipeline, que es la que recogerá tanto el contador de frecuencias, como el inversor de frecuencias en base a la relevancia y el clasificador en un array que podrá ser accedido con métodos especiales basados en cada una de las partes de esta clase.

Para terminar, lo que hace es entrenar el modelo con las partes del dataset_part que se le pasan como parámetro y lo guarda todo en el atributo del modelo text_clf.

- **Método: predict(self, list_news)**

  Argumentos:

  - self: Instancia del objeto de la clase ContentAnalyzer.
  - list_news: *list(String)* : Atributo dedicado a almacenar todas las noticias en formato texto tiene que predecir el clasificador.

  Desarrollo:

  Este método se encarga de recoger el modelo ya entrenado o en su defecto, entrenarlo con un dataset de entrenamiento y, después, hacer el mismo procedimiento del **método fit,** es decir, vectorizar y ordenar por relevancia en la oración, cada una de las noticias para poder hacer una predicción con el modelo ya entrenado.

- **Método: print_predict(self,news)**

  Argumentos:

  - self: ContentAnalyzer. Instancia del objeto de la clase ContentAnalyzer.
  - news: *list(String)*: Conjunto de noticias que se muestran con su predicción siguiendo el formato de la expresión regular definida.

  Desarrollo:

  Método auxiliar que sirve para imprimir cada una de las noticias que estén dentro del array de los argumentos news, siguiendo la siguiente expresión regular. ***Noticia -> Predicción de Noticia***

- **Método: predict_audio(self,sound)**

Argumentos:

- self: ContentAnalyzer. Instancia del objeto de la clase ContentAnalyzer.
- sound: String: Ruta del archivo de audio a identificar.

Desarrollo:

Método que predice la noticia contenida en el audio, mediante el uso de la biblioteca speech recognition, que convierte el audio en un String, que luego se le pasará al método predict para sacar por pantalla el resultado que da el modelo sobre la noticia.

- **Método: store(self, path_model)**

Argumentos:

- self: ContentAnalyzer. Instancia del objeto de la clase ContentAnalyzer.
- path_model: *String.* Ruta en la que se almacenará el modelo entrenado.

Desarrollo:

Método que se basa en el dump de scikit-learn para guardar los modelos en una ruta seleccionada. El modelo guardado luego puede ser cargado debido a que se guarda en formato joblib.

- **Método: eval(self, x, y)**

Argumentos:

- self: ContentAnalyzer. Instancia del objeto de la clase ContentAnalyzer.
- x: ***
- y ***

Desarrollo:

Método que evalúa el modelo entrenado para poder evaluar diferentes combinaciones de modelos y clasificadores y sacar conclusiones sobre la mejor combinación para la predicción de noticias.

La métrica utilizada en el caso de esta clase es la precisión en la predicción, contando las falladas y acertadas y dando el resultado en tanto por ciento.

- **Método: describe(self)**

Argumentos:

- self: Instancia del objeto de la clase ContentAnalyzer.

Desarrollo:

Método que describe los resultados obtenidos después de todo el proceso de evaluación, entrenamiento y predicción. Todo ello acompañado de unos datos sobre la capacidad de error y el tamaño de los datos.

- **Método: getNumericalType(self,speech_dataset)**

Argumentos:

- self: ContentAnalyzer. Instancia del objeto de la clase ContentAnalyzer.
- speech_dataset: *pandas.dataframe:* Conjunto del dataset con todas las partes unidas antes del split.

Desarrollo:

Método auxiliar que crea una nueva columna en el dataframe con los índices integer de cada una de las categorías de las noticias, para facilitar su uso.

- **Método: quitarStopWords(self, text)**

Argumentos:

- self: ContentAnalyzer. Instancia del objeto de la clase ContentAnalyzer.
- text: *pandas.dataframe:* Texto al que se le han de quitar las stopwords.

Desarrollo:

Método auxiliar que elimina las stopwords de un texto.

- **Método: _preprocess_data(self, x)**

Argumentos:

- self: ContentAnalyzer. Instancia del objeto de la clase ContentAnalyzer.
- x: *String:* Datos que preprocesar.

Desarrollo:

Método que preprocesa los datos que le han de entrar al modelo para que la predicción sea la más correcta. Para ello, lo que hace es quitarle todos los símbolos de puntuación y todas aquellas palabras que no añaden ningún significado ni contexto a la noticia. Para ello hay dos alternativas, el uso de la biblioteca gensim, o la más lenta, que es con la función auxiliar stopwords.

4.5. MODELO BI-LSTM

Se ha realizado un modelo Bi-LSTM, para profundizar en el funcionamiento de las redes neuronales e indagando en este caso en el uso de Keras y TensorFlow.

En primer lugar, se ha llevado a cabo una limpieza del dataset, el cual estaba presentado en forma de .csv. Se hizo uso del módulo Pandas para crear dicho .csv, cuyo contenido se presentaba en una estructura de datos tipo lista, esto es, " ["Texto 1", "Texto 2", Texto 3"], tipoNoticia ". Se probó a usar un dataset en formato .txt que poseía todos los textos unidos sin encontrarse separados por comillas y con el tipo de la noticia al final separado por una coma, pero al encontrarse algunas noticias con comas en la misma, el programa no era capaz de diferenciarlas de la coma final que separaba el tipo de la noticia y daba por resultado que el tipo de la noticia era NaN, por ello se optó por utilizar el de formato .csv. Para llevar a cabo la limpieza del texto y eliminación del ruido del mismo, se ha utilizado RegEx. Es una tecnología que ha resultado fácil de usar y que no dominábamos, por lo que su elección ha sido ideal no solo para llevar a cabo la práctica sino también para ampliar nuestros conocimientos. También se ha limpiado el dataset eliminando URLs, emails, stopwords, signos de puntuación y poniendo todo el texto en minúsculas.

Llevamos a cabo una función de detokenización, ya que vamos a usar word embeddings. Para manipular toda esta información debemos convertirla a números, ya que no se puede trabajar con ella mediante texto, la vectorizaremos. Los word embeddings o word vectors son vectores densos (low-dimensional floating-point vectors). Se usan estos porque los modelos de Deep Learning funcionan mejor con ellos que con los sparse vectors. Así que pasaremos nuestro array de texto en una matriz numérica.

Usaremos Redes Neuronales Recurrentes (RNN) que son redes que incluyen conexiones que apuntan "hacia atrás", una especie de retroalimentaciones entre las neuronas dentro de las capas. Otras redes neuronales no tienen memoria, esto es que cada input se procesa independientemente, sin ninguna relación con los otros. Esto es lo contrario a lo que una persona haría cuando lee un texto, para darle sentido a lo que está leyendo tiene en cuenta lo que ha leído con anterioridad; pues esto es lo que vendrían a hacer las RNN. En la figura 4.5 podemos ver un ejemplo de cómo se podría implementar una capa de neuronas recurrentes de tal manera que, en cada instante de tiempo, cada

neurona recibe dos entradas, la entrada correspondiente de la capa anterior y a su vez la salida del instante anterior de la misma capa.

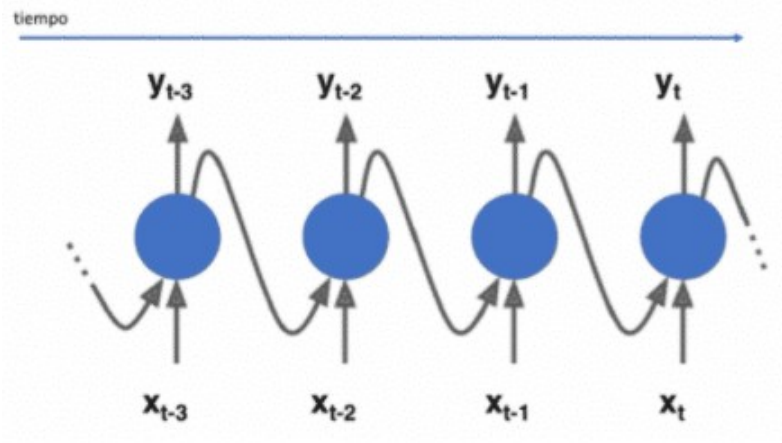

Figura 4.5. Ejemplo de RNN

En este modelo, se han realizado cuatro tipos de redes neuronales recursivas, entre las que tenemos un modelo SimpleRNN, un modelo Single LSTM, un modelo Bi-LSTM y un modelo de redes neuronales convolucionales 1D, para dirimir cual de todos ellos obtenía mejores resultados.

Las redes LSTM (Long-Short Term Memory) son una extensión de las redes neuronales recurrentes, que básicamente amplían su memoria para aprender de experiencias importantes que han pasado hace mucho tiempo. Las LSTM permiten a las RNN recordar sus entradas durante un largo período de tiempo. Esto se debe a que LSTM contiene su información en la memoria, que puede considerarse similar a la memoria de un ordenador, en el sentido que una neurona de una LSTM puede leer, escribir y borrar información de su memoria. Esta memoria se puede ver como una "celda" bloqueada, donde "bloqueada" significa que la célula decide si almacenar o eliminar información dentro (abriendo la puerta o no para almacenar), en función de la importancia que asigna a la información que está recibiendo. La asignación de importancia se decide a través de los pesos, que también se aprenden mediante el algoritmo. Esto lo podemos ver como que aprende con el tiempo qué información es importante y cuál no.

La red convolucional de una dimensión es una red neuronal que tiene un coste computacional menor. Es una red que tiende rápidamente al sobreajuste en datasets pequeños. Es la red que más rápido ha entrenado pero que en este caso ha dado resultados mucho peores que el resto.

Para entrenar el modelo SimpleRNN se ha usado una capa SimpleRNN, para el SingleLSTM se ha aplicado una capa LSTM y para el Bi-LSTM una capa Bidirectional con una LSTM dentro. Las redes bidireccionales consisten básicamente de dos RNN que procesan la información en direcciones diferentes. Una de las capas interpreta en orden cronológico y la otra en orden anti cronológico. Es por ello que estas redes se usan más y obtienen mejores resultados que las RNN, son capaces de captar patrones más complejos de lo que lo haría una RNN.

```
Model2 = Sequential()
model2.add(layers.Embedding(max_words, 40, input_length=max_len))
model2.add(layers.Bidirectional(layers.LSTM(20,dropout=0.6)))
model2.add(layers.Dense(5,activation='softmax'))
model2.compile(optimizer='rmsprop',loss='categorical_crossentropy'
, metrics=['accuracy'])
#Implementing model checkpoins to save the best metric and do not
lose it on training.
Checkpoint2 = ModelCheckpoint("best_model2.hdf5",
monitor='val_accuracy', verbose=1,save_best_only=True,
mode='auto', period=1,save_weights_only=False)
history = model2.fit(X_train, y_train,
epochs=70,validation_data=(X_test,
y_test),callbacks=[checkpoint2])
```

En estos tres modelos hemos usado la función softmax como función de activación que básicamente, lo que hace es convertir un vector z de k números reales en otro vector de k números en el rango [0, 1] de tal forma que estos k números sumen 1. Como optimizador se ha usado RMSprop. Un optimizador es un mecanismo que calcula constantemente el gradiente de la perdida y dice como moverse contra la función de perdida para obtener los mejores parámetros para la red. El optimizador RMSprop es similar al algoritmo de descenso de gradiente con momentum. Restringe las oscilaciones en la dirección vertical, por lo tanto, podemos aumentar nuestra tasa de aprendizaje y nuestro algoritmo podría dar pasos más grandes en la dirección horizontal convergiendo más rápido. Como función de perdida se usa la categorical_crossentropy, que es la que usa para clasificar cuando hay mas de dos clases. Por último, en los modelos se usan checkpoints para guardar el mejor resultado durante la fase de entrenamiento.

El modelo que ha mostrado un mayor porcentaje de acierto es el modelo bidireccional LSTM, que es con el que se ha trabajado en el resto del código.

En este modelo hemos presentado los resultados en una matriz de confusión de manera que nos permite no solo saber el porcentaje de noticias que han sido bien clasificadas sino, además, poder conocer en qué otra categoría se clasificaron aquellas que lo hicieron mal.

El segundo modelo que hemos llevado a cabo es un modelo de capas densas, con la misma estructura de métodos que se han explicado con anterioridad en el modelo Tf-idf.

Para ello, lo primero que hemos hecho es tokenizar el texto esta vez usando la clase Tokenizer, y aplicando los símbolos de filtro que deseamos, además de pasarlo a minúsculas. Posteriormente, después de la ejecución del Tokenizer, pasamos este texto a una matriz, que será codificada y pasada a enteros gracias a la clase labelBinarizer, para que sea posible ordenarlas en base a este índice numérico.

El modelo es secuencial, por lo tanto, decidimos que nuestra primera capa fuese una capa densa de entrada, que es donde va a entrar la información al modelo. Luego añadimos una capa de activación ReLU, para normalizar los datos para que sean todos positivos y en los rangos adecuados, mejorando la eficacia del modelo. La función devuelve 0 si recibe una entrada negativa, pero para cualquier valor positivo x, devuelve ese valor. Por lo tanto, se puede escribir como f (x) = max (0, x). Podemos ver como es esta función en la figura 4.6.

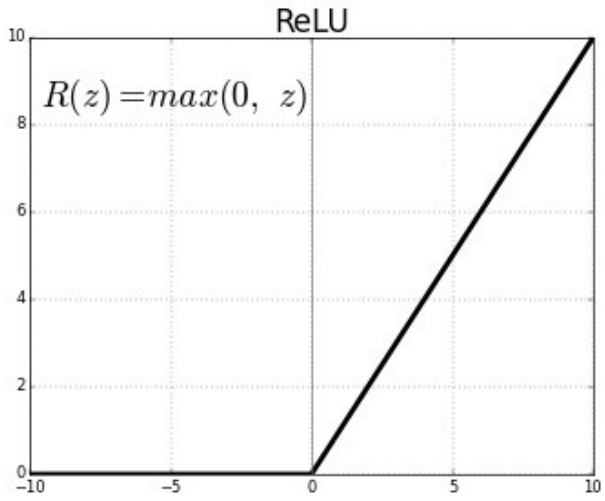

Figura 4.6. Función ReLU

La siguiente capa es una capa de dropout, que, por cada nueva entrada a la red en fase de entrenamiento, desactivará aleatoriamente un porcentaje de las neuronas en cada capa oculta, acorde a una probabilidad de descarte previamente definida. Lo que se consigue

con esto es que ninguna neurona memorice parte de la entrada; que es precisamente lo que sucede cuando tenemos sobreajuste. Podemos ver un ejemplo en la figura 4.7.

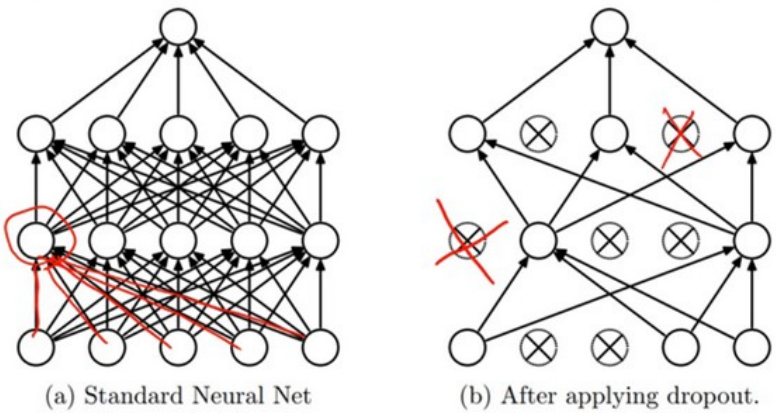

Figura 4.7. Función Dropout

Volvemos a añadir una capa de densa, otra relu y otra de dropout para terminar con una capa de softmax, que lo que hace es calcular la categoría en base a la probabilidad que decida el modelo a parecerse a una categoría. Softmax asigna probabilidades decimales a cada clase en un caso de clases múltiples. Esas probabilidades decimales deben sumar 1.0. Esta restricción adicional permite que el entrenamiento converja más rápido. Podemos ver un ejemplo en la figura 4.8.

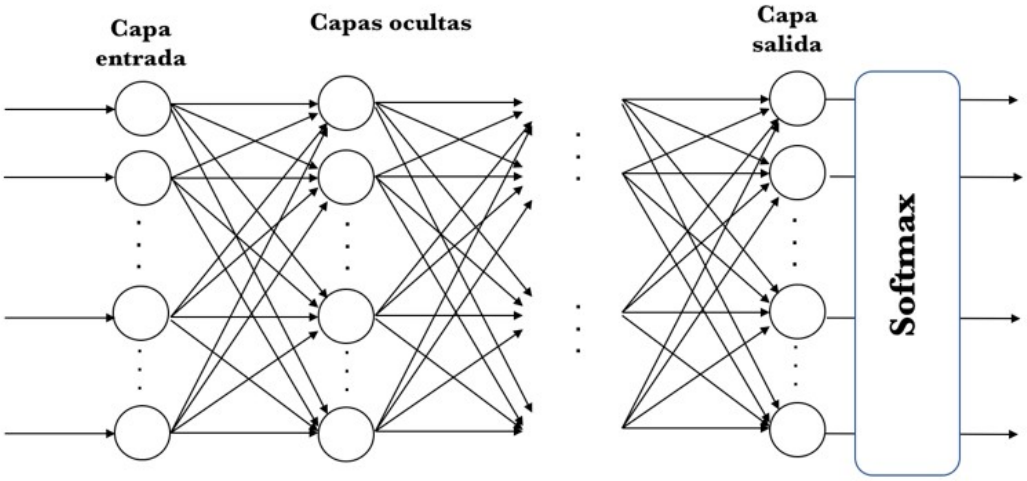

Figura 4.8. Función Softmax

5. Resultados

5.1. MODELO TF-IDF

Los resultados obtenidos por el modelo de predicción de texto tf-idf (term frequency – inverse document frequency) se ven reflejados en la figura 5.1., generadas por el propio modelo durante su ejecución:

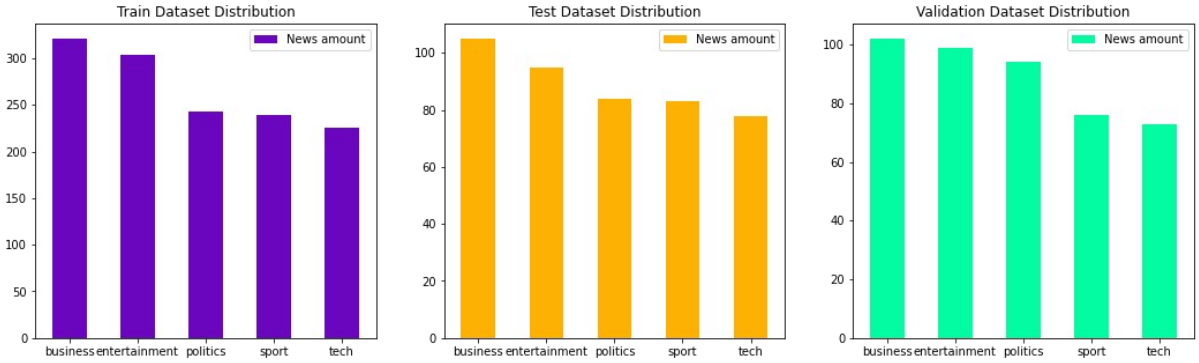

Figura 5.1. Distribución del conjunto de datos de entrada

Como vemos en la figura 5.2., es un dataset con un equilibrio bastante adecuado, debido a que la diferencia entre los diferentes tipos de noticias no es muy pronunciada, sin llegar a alcanzar una descompensación que estropee el trabajo de nuestro modelo clasificador.

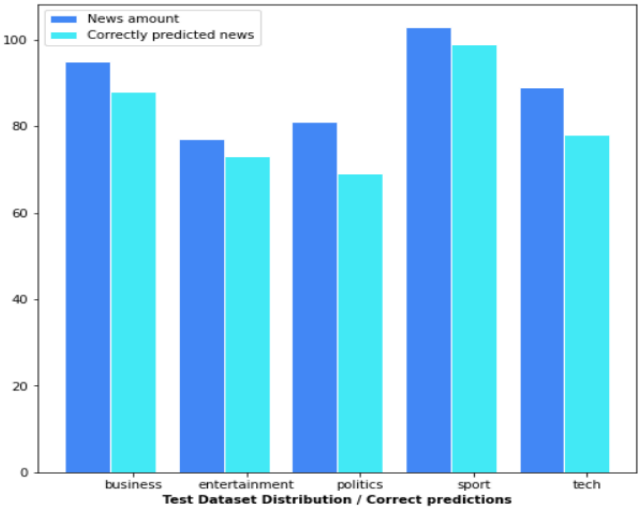

Figura 5.2. Relación del conjunto de datos de test respecto a su correcta clasificación

En la ejecución de esta instancia del modelo, vemos como acierta casi todas las noticias que le pasamos, llegando a alcanzar el 92.6% de accuracy en este caso.

5.2.MODELO BI-LSTM

5.2.1. PRIMER MODELO

Estos son resultados correspondientes a NewsTypePredictions.

Gracias a la matriz de confusión de la figura 5.3 podemos ver que clases son las que mejor se clasifican, que son negocios y deporte, y cuales son las que peor lo hacen, como son las de tecnología y política. Además podemos observar en que categoría coloca las noticias de cada tipo mal clasificadas.

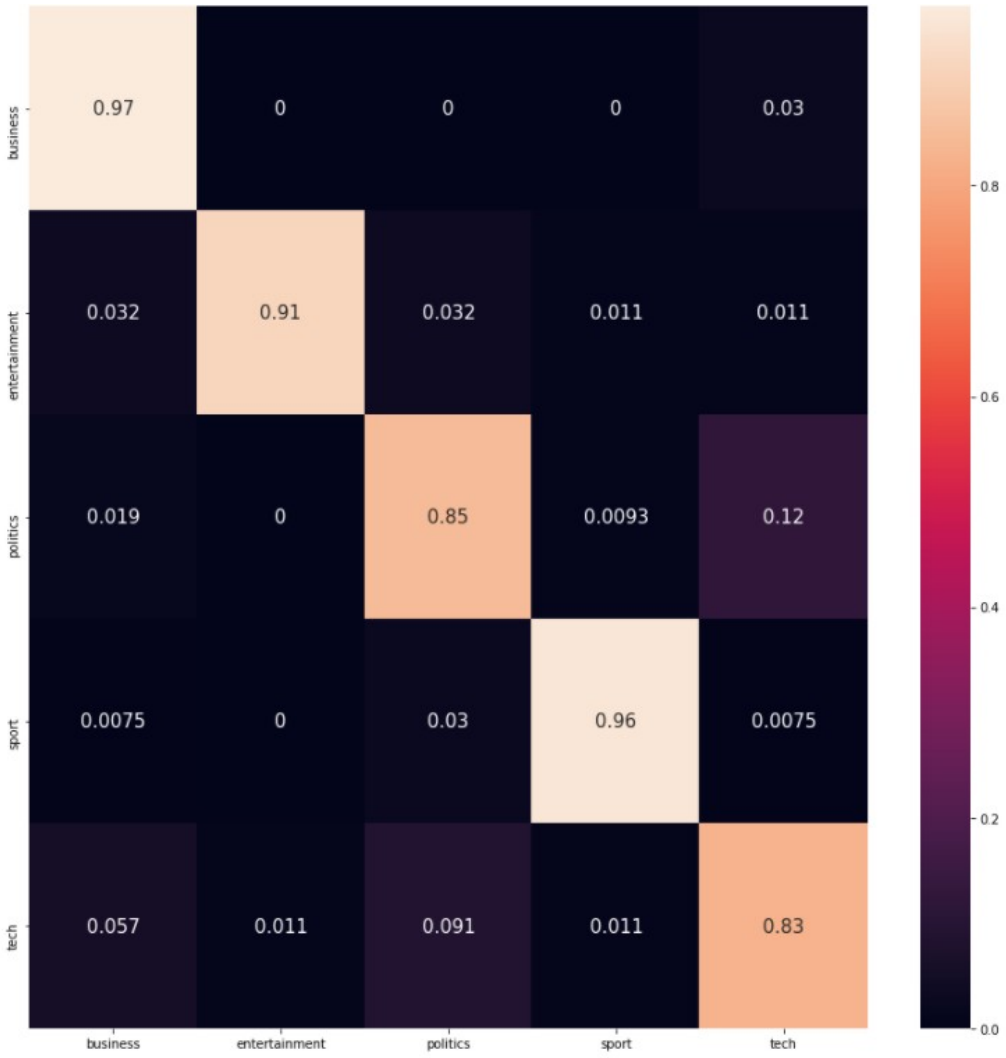

Figura 5.3. Matriz de confusión del modelo Bi-LSTM

En la ejecución de esta instancia del modelo, vemos como acierta casi todas las noticias que le pasamos, llegando a alcanzar el 91.2% de accuracy en este caso.

5.2.2. SEGUNDO MODELO

Estos son resultados correspondientes a ContentAnalyzer_Neuronal_Neutwork

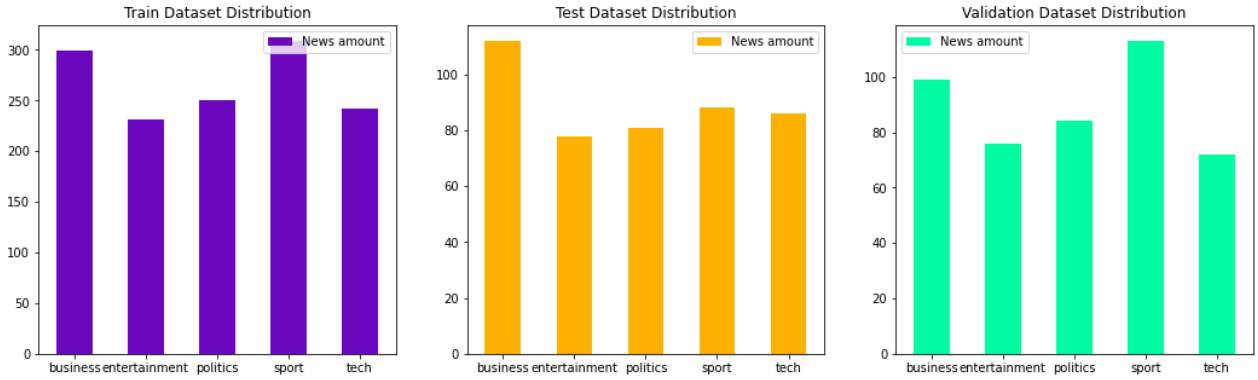

Figura 5.4. Distribución del conjunto de datos de entrada

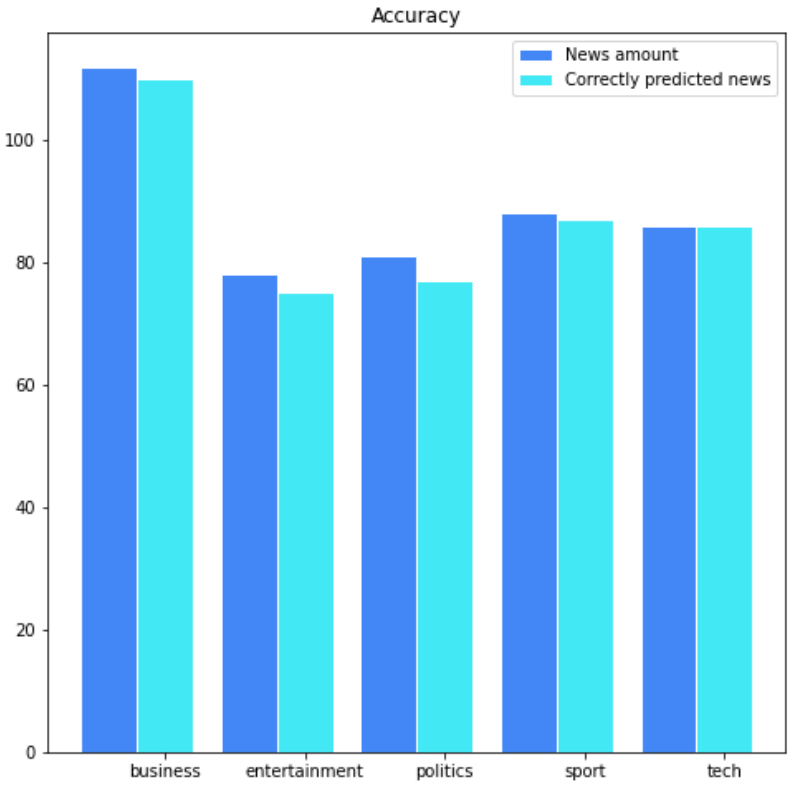

Figura 5.5. Relación del conjunto de datos de test respecto a su correcta clasificación

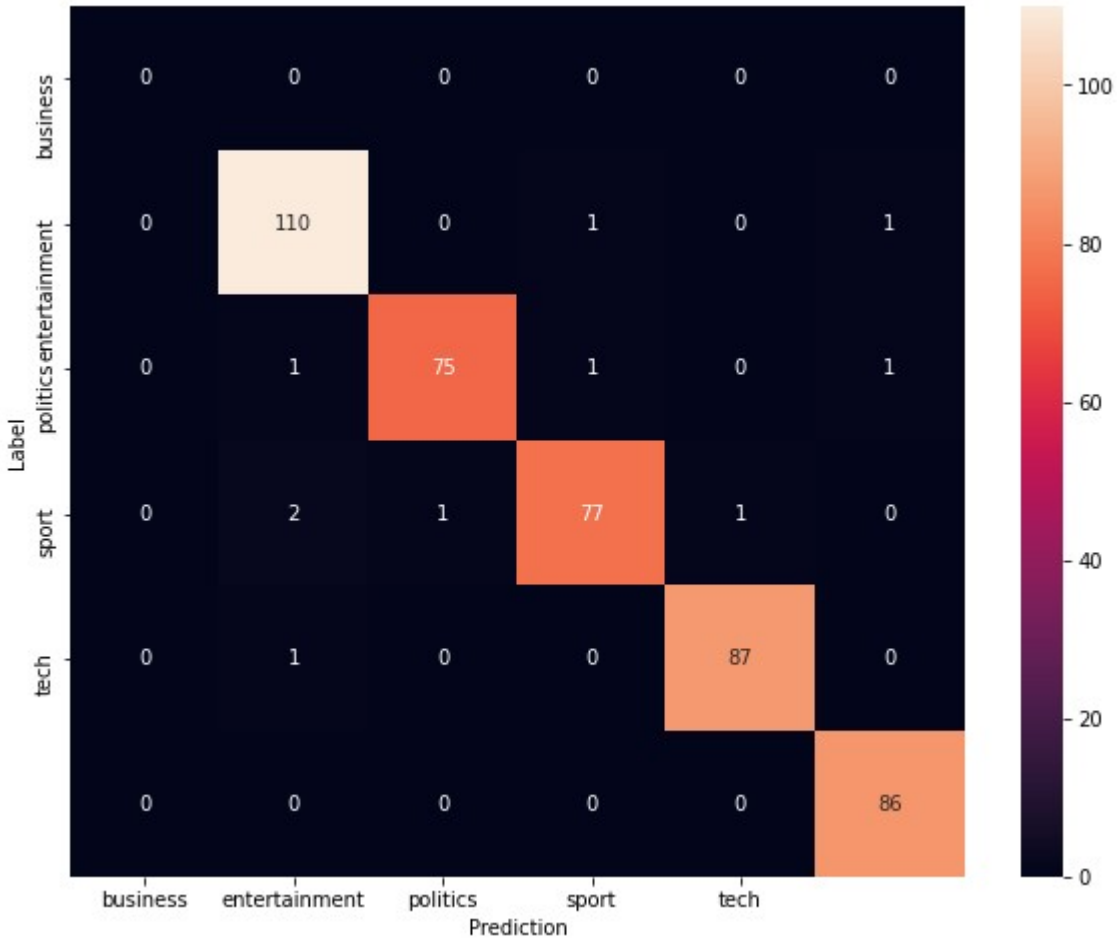

Figura 5.6. Matriz de confusión del modelo Bi-LSTM

6. Conclusiones

Podemos concluir que se han logrado satisfactoriamente los objetivos utilizando las tecnologías y pautas recomendadas. El trabajo se realizó dentro de los plazos establecidos y con amplio margen. Con una mejor organización y comunicación dentro del grupo estoy seguro de que el ya buen trabajo que se ha realizado podría a ver sido excelente. Se han logrado varios modelos que resuelven el problema propuesto con un grado de acierto satisfactorio (superior al 91%). Se ha trabajado con muchas tecnologías que eran nuevas para nosotros y con las cuales hemos podido aprender acerca de la Inteligencia Artificial y Machine Learning.