Lautaro Andrés Russo Bertolez

Entrenamiento de modelos de deep learning para la clasificación de texto en base al sentimiento.

TREBALL DE FI DE GRAU

Dr. Domènec Puig

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2022

Tabla de contenido

Índice de figuras

1. Introducción

Vivimos en un mundo de progreso e innovación constante donde aprender siempre es un hábito fundamental para no quedarse desactualizado con respecto a los nuevos descubrimientos y tecnologías.

Debido a la revolución de la información que estamos viviendo cada vez más las personas optan por realizar este proceso de aprendizaje mediante herramientas online como cursos, audiolibros, herramientas interactivas, aplicaciones y todo tipo de recursos digitales.

Normalmente estas plataformas de cursos online tienen apartados donde los usuarios pueden dar feedback sobre su experiencia con este recurso de aprendizaje ya sea mediante puntuación o reseñas de texto, pero no para todas las plataformas es el caso, muchas veces aplicaciones o portales web nuevos que aún están en desarrollo no disponen de este tipo de funcionalidades dificultando la capacidad de valorar la calidad de su material didáctico o la calidad de la experiencia de sus usuarios.

Este proyecto apunta resolver ese problema utilizando tecnologías de nueva generación como son las del campo del aprendizaje automatizado [2] y las muy aclamadas hoy en día redes neuronales.

La idea general es desarrollar un modelo predictivo que consiga determinar con éxito, de forma consistente y en tiempo real si comentarios de usuarios al respecto de un tema son positivos o negativos.

Este modelo podría ser utilizado no solo como herramienta para medir la experiencia del usuario en base a sus comentarios en la plataforma sinó para buscar temas de interés en otras plataformas analizando qué temas gustan o no al público.

Las capacidades de comercialización están limitadas por la creatividad e imaginación de los desarrolladores debido a que al ser herramientas de nueva generación aún se están encontrando nuevas formas de implementarlas a diferentes ámbitos y campos.

1.1. Estado del campo del aprendizaje automático en la actualidad

Antes que nada tenemos que tener en mente que es el aprendizaje automático o como sus siglas en inglés indican ML (Machine Learning). El aprendizaje automático es un subcampo de las ciencias de la computación y una rama de la inteligencia artificial [1] cuyo objetivo es desarrollar técnicas que permitan a las máquinas aprender.

¿Que significa que una máquina aprenda?

Se dice que un ordenador aprende si en base a una experiencia E con respecto a una tarea T y medida de desempeño P si este este desempeño con respecto a la tarea T, medido mediante P, mejora con la experiencia E.

Por decirlo de alguna forma, es hacer que un programa de ordenador aprenda por sí solo a resolver un problema en base a ejemplos.

Este subcampo de las ciencias de la computación no es nuevo, sus orígenes se remontan a 1943 donde Warren McCulloch publicó un paper sobre matemáticas mapeando la toma de decisiones mediante redes neuronales del cerebro humano y desde entonces se han ido diseñando y desarrollando algoritmos que apuntan a imitar este sistema de aprendizaje.

Tradicionalmente el campo del aprendizaje automático se caracteriza por utilizar modelos matemáticos relativamente simples que ayudan a la interpretación de los datos, evitando cálculos excesivos. Esta eficiencia computacional viene con ciertas ataduras, teniendo que diseñar un gran abanico de modelos para la resolución de tareas de diferente índole y tratamiento de diferentes tipos de datos.

Una vertiente en auge y que ha progresado mucho estos últimos años son la utilización de modelos de aprendizaje profundo [3] (Deep Learning) basados en redes neuronales. Este tipo de modelos por contraparte a los modelos tradicionales son sumamente costosos a nivel computacional y requieren de un mayor volumen de datos, pero debido a la las nuevas técnicas de aceleración de cómputo mediante GPU y la recolección masiva de datos online están permitiendo que estos modelos superen a sus predecesores en ciertos aspectos.

A pesar de ser computacionalmente costosos, no requieren de un gran conocimiento o análisis sobre el problema para su resolución y aceptan una gran variedad de tipos de datos. Lo que permite su generalización y uso a diferentes tareas.

Hoy en día las mayores potencias tecnológicas del mundo como Google, Amazon, Meta, Netflix o HBO cuentan con una gran variedad de algoritmos basados en modelos de deep learning para resolver tareas de predicción, clasificación, segregación, recomendación o agentes inteligentes para la conducción autónoma.

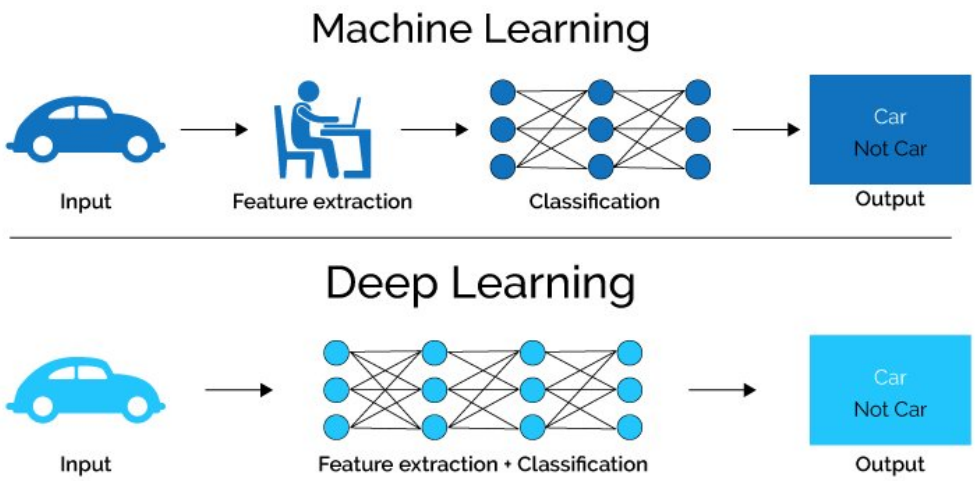

Figura 1 - Ejemplo diferencia entre ML y DL.

1.2. Motivación y objetivos

Debido al interés por parte de la industria tecnológica de desarrollar sus propias herramientas y modelos basados en inteligencia artificial, aprendizaje automático y aprendizaje profundo la demanda de desarrolladores en el campo está en alza.

Más allá de desarrollar una herramienta comercializable este proyecto apunta a ser un muestrario de todas las etapas, criterios y técnicas que conforman el desarrollo de un modelo de inteligencia artificial basado en redes neuronales para la clasificación de texto en base al sentimiento, todos los conocimientos y metodologías utilizadas en dicho desarrollo.

No solamente desarrollar un modelo funcional sinó abarcar también varias técnicas de procesamiento y codificación de datos para el posterior entrenamiento de modelos.

Los objetivos tanto didácticos tanto finales del proyecto se listan a continuación :

A. Diseño y definición del problema a resolver.
B. Diseño y definición del modelo a entrenar.
C. Análisis y definición del tipo de muestras a recolectar.
D. Recolección y selección de muestras.
E. Selección y depuración para garantizar y aumentar la calidad de los datos.
F. Procesamiento de muestras para extraer y condensar parámetros relevantes.
G. Codificación de muestras a un dominio vectorial.
H. Entender, definir, entrenar y retocar diferentes tipos de modelos de aprendizaje profundo.
I. Validación y pruebas independientes comparando diferentes métricas para el entendimiento del comportamiento del modelo y así garantizar la ausencia de sesgos.

1.3. Planificación del proyecto

La realización del proyecto se llevará a cabo en las siguientes fases de forma secuencial salvo por la tarea T1 que es paralela a todas las demás:

T1. Realización de la memoria : Consiste en confeccionar este documento durante la realización del proyecto.

T2. Análisis del estado de la tecnología : Analizar el abanico de técnicas existentes en el campo de la inteligencia artificial.

T3. Análisis del proyecto : Definir y acotar el trabajo a realizar con un enfoque concreto sobre las necesidades de la herramienta a desarrollar.

T4. Recolección de datos para entrenamiento supervisado : Una vez definido el enfoque, reunir todas las muestras necesarias para el entrenamiento del modelo.

T5. Análisis de la validez y calidad de los datos : Determinar si la validez y calidad de las muestras es la correcta para el correcto desarrollo del modelo.

T6. Selección y formateado de los datos : Creación de los correspondientes corpus de datos para el entrenamiento, validación y test independientes.

T7. Procesamiento de texto : Aplicar las modificaciones correspondientes a las muestras de texto para acondicionarse para el entrenamiento.

T8. Codificación de texto : Conversión de las secuencias de texto a un dominio numérico mediante el uso de modelos de embeddings [11] pre entrenados.

T9. Entrenamiento preliminar de modelos : Selección arquitecturas, diseño y entrenamiento de los diferentes modelos a comparar.

T10. Validación y selección de modelos : Mediante el conjunto de muestras de validación comparar las diferentes características de los modelos.

T11. Depuración de datos en base a modelos entrenados : Una vez entrenado un modelo preliminar podremos inferir un error en cada muestra y hacer un análisis de confianza del modelo sobre las muestras para corregir muestras mal clasificadas.

T12. Evaluación de eficacia y precisión de la herramienta : Fase de pruebas para evaluar cómo se comportaría el modelo en un entorno real.

1.4. Estructura de la memoria

A continuación en la sección 2 está descrito el diseño de la solución al problema previamente planteado, un análisis detallado de las metodologías utilizadas por la industria hoy en día, por qué se ha elegido un tipo de modelos con respecto a otros y ventajas e inconvenientes de unos modelos frente a otros.

En la sección 3 de esta memoria se detallan todas las librerías y dependencias utilizadas tanto el uso que se les ha dado a lo largo de este proyecto. También se detalla el criterio utilizado para la decisión del uso de dicha herramienta, librería o dependencia.

En la sección 4 vienen detallados de forma secuencial y en orden todos los apartados que componen el desarrollo técnico del modelo, resultados obtenidos y posibles mejoras antes de utilizarse en un entorno de producción.

La sección 5 presenta las conclusiones del trabajo. En primera instancia, se evalúa el cumplimiento de los objetivos. A continuación, se enumeran las diferentes conclusiones y opiniones personales obtenidas con la realización de todo el proyecto y, por último, el trabajo pendiente y posibilidades de futuro.

En el anexo se encuentran listados todos los recursos utilizados para la realización del proyecto, pruebas y memoria.

Finalmente, en la bibliografía se listan los enlaces raíz a las plataformas oficiales de todos los recursos utilizados.

2. Diseño

De todos los tipos de problemas en los que el campo del aprendizaje automático nos puede ayudar a resolver, regresión, clasificación, pronóstico de secuencias, detección de anomalías, recomendación, generación de datos u optimización, en este trabajo se busca resolver un problema de clasificación.

Concretamente el problema que plantea este trabajo es el de clasificar de forma binaria, en positivos y negativos, textos escritos en base a un sentimiento. Este tipo de problemas son típicamente abordados con métodos de aprendizaje supervisado, que consisten en que a nuestro modelo le vamos a ir enseñando muestras y corrigiendo en base a predicciones de este, como si de un ejercicio de palo y zanahoria se tratara.

2.1. Análisis del problema

En primer lugar necesitaremos definir las características de los tres elementos fundamentales para este sistema, datos, modelo y función de coste [5].

Estamos queriendo clasificar el texto en categorías binarias por lo tanto los datos que necesitamos estarán compuestos por muestras de texto acompañadas de una etiqueta que nos indique si el sentimiento de este texto es positivo o negativo.

Por otro lado y no menos importante para clasificación binaria disponemos de una gran variedad de funciones de coste diferentes, en este caso utilizaremos la función de coste más utilizada en la actualidad por la industria, conocida en inglés por “binary cross entropy log loss”.

Para los modelos tenemos gran variedad de opciones tanto de algoritmos con un enfoque más tradicional como son las regresiones logísticas, árboles de clasificación o bosques de clasificación; Como otras vertientes más modernas previamente mencionadas del sub campo del aprendizaje profundo como son las redes neuronales. En este caso optamos por modelos más novedosos del campo de las redes neuronales.

2.2. Planteamiento de modelo

Con tal de comparar el desempeño de diferentes tipos de redes neuronales en este trabajo se plantean tres modelos de redes neuronales que siguen tres arquetipos diferentes: Una red neuronal [4] densa, una red neuronal convolucional y una red neuronal recurrente mediante celdas lstm.

El funcionamiento de las redes neuronales es relativamente sencillo, las neuronas tienen una serie de variables en las que almacenan conocimiento de forma codificada, mediante la

experiencia de cada iteración los valores de cada neurona van cambiando ajustándose para mapear las características de los ejemplos y del texto.

Las redes más profundas y con mayor cantidad de neuronas tienen más parámetros para mapear estas características pudiendo describir problemas más complejos, pero por otro lado ello requiere más capacidad de cómputo y lidiar con problemas de optimización numérica que aparecen cuando el número de capas y/o neuronas es muy grande.

Para este problema utilizaremos redes relativamente pequeñas en comparación a los modelos estado del arte que utilizan plataformas como Google o Amazon. No obstante, para los recursos de los que dispone el entorno donde se ha llevado a cabo el proyecto, son exigentes a nivel computacional significando este un coste temporal a la hora de entrenarlas.

3. Dependencias

3.1. Entorno

Como entorno de desarrollo de la aplicación se utilizará la herramienta de desarrollo Jupyter notebook que consiste en un entorno de desarrollo interactivo basado en web. Tiene una interfaz flexible que permite a los desarrolladores configurar de forma modular secuencias de código y estructuras de datos. Se utiliza ampliamente en los campos de la ciencia de datos, ciencia computacional, periodismo computacional y aprendizaje automatizado.

Mediante el subsistema para linux de windows levantaremos un servidor que nos permitirá mediante localhost acceder a la herramienta, esta herramienta dispone también de gestión de ficheros mediante una interfaz con varias funcionalidades.

Todo el proceso de principio a fin ha sido ejecutado en un entorno local utilizando los recursos de los que dispone mi ordenador, los recursos que cabe mencionar son los siguientes: Procesador Intel(R) Core(™) i7-970H CPU @ 2.60GHz, Memoria RAM 16GB ddr4 2600MHz, GPU Nvidia GeForce RTX 2070, SSD Hard drive 256GB Kingston.

3.2. Librerías

A continuación se enlistan en orden alfabético todas las herramientas y librerías de desarrollo para el lenguaje de programación python que nos permitirán llevar a cabo todas las transformaciones sobre las series de datos y entrenar modelos de Deep Learning:

- ast : Abstract syntax trees - Este módulo nos ayuda a procesar árboles de sintaxis y gramática abstracta de Python.
- langdetect : Este módulo nos ayuda a detectar en qué lenguaje está escrito un texto.
- matplotlib : Este módulo es una herramienta para crear visualizaciones de datos estáticas, dinámicas e interactivas en Python.
- nltk : Natural language toolkit - Es la librería líder para el desarrollo de herramientas basadas en el lenguaje natural.
- numpy : Es una librería fundamental para las ciencias de la computación en Python. Proporciona tipos de representaciones matemáticas para Python y herramientas para el manejo y transformación de estas
- pandas : És una biblioteca de software como extensión de numpy para la manipulación y el análisis de datos para Python.
- pickle : Es un módulo que nos proporciona la capacidad de serializar y deserializar objetos en Python.
- re : Regular expression - Este módulo nos permite buscar patrones de texto mediante el uso de expresiones regulares.

- seaborn : Es un módulo construido encima de Matplotlib para la visualización de datos, proporciona herramientas de visualización menos flexibles pero más sencillas de entender.
- string : Este módulo incorpora diferentes métodos para la manipulación de strings o arrays de caracteres.

El uso de todas estas herramientas estará detallado tanto en la sección de desarrollo como en el anexo.

4. Desarrollo

Los siguientes cuatro sub apartados de desarrollo se corresponden a los cuatro cuadernos Jupyter en los que se han llevado a cabo las tareas. En el anexo se encuentran las secciones de código.

4.1. Recolección y selección

4.1.1. *Búsqueda y recolección preliminar de datos*

Una vez definido el tipo de datos a buscar, el siguiente paso es encontrar dichos datos. Se plantearon dos opciones: La primera consiste en hacer web scraping [6] para la recolección online, pero como los portales web más importantes de cursos online no disponían de APIs [7] para dicha recolección la tarea se complicaba bastante; Por otro lado la opción de utilizar un set de datos ya creado por un tercero, por lo que acudí a Kaggle [8] y empecé a investigar si alguien había hecho un trabajo de búsqueda similar.

Se consiguieron reunir dos conjuntos de datos con una considerable cantidad de muestras de reviews extraídas de las plataformas Coursera y Udemy, sumando un total aproximado de dos millones de ejemplos.

Las herramientas utilizadas tanto para la carga de los datos en memoria, las transformaciones y el guardado en ficheros de dichos datos fueron métodos de las librerías Pandas y Numpy.

	review_content	rating
0	Pretty dry, but I was able to pass with just t...	4
1	would be a better experience if the video and ...	4
2	Information was perfect! The program itself wa...	4

Figura 2 - Ejemplo muestras extraídas del portal Coursera.

	review_content	rating
0	Delivers more than expected. Thank you!	4.5
1	This course would be a good choice if 1) you a...	2.5
2	I am new to Ruby but have programmed in Python...	5.0

Figura 3 - Ejemplo muestras extraídas del portal Udemy.

4.1.2. Formato de los datos

Un problema de estas series de datos es que las etiquetas o ratings no pertenecían a un dominio binario siendo estas valoraciones del 1 al 5. Por lo que después de una primera limpieza de muestras sin etiqueta se procedió a transformar las etiquetas a un dominio binario.

Se trazó una línea y todas las valoraciones superiores a 2 se contabilizaron como 1 y todas las valoraciones iguales o inferiores a dos se contabilizaron como 0.

4.1.3. Filtrado por lenguaje

A continuación, debido a que nuestro objetivo consistía en clasificar textos únicamente en inglés se procedió a hacer un filtrado por lenguaje utilizando una herramienta de la librería langdetect y descartando todas las muestras que no se correspondiera al inglés.

Una vez filtrado por lenguaje el número total de muestras se redujo a aproximadamente 850.000 muestras lo que indica que menos de la mitad de las muestras estaban en inglés.

	course_name	review_content	rating	lang
0	google-cbrs-cpi-training	Pretty dry, but I was able to pass with just t...	1	en
1	google-cbrs-cpi-training	would be a better experience if the video and ...	1	en
2	google-cbrs-cpi-training	Information was perfect! The program itself wa...	1	en

Figura 4 - Ejemplo muestras post-binarización Coursera.

	course_name	review_content	rating	lang
0	Ruby Programming for Beginners	Delivers more than expected. Thank you!	1	en
1	Ruby Programming for Beginners	This course would be a good choice if 1) you a...	1	en
2	Ruby Programming for Beginners	I am new to Ruby but have programmed in Python...	1	en

Figura 5 - Ejemplo muestras post-binarización Udemy.

4.1.4. Selección de muestras de forma balanceada

Debido a que típicamente la distribución de reseñas no está balanceada con respecto a negativas y positivas, siendo las reseñas positivas alrededor del 95% de las reseñas, se ha hecho una selección balanceada. Esto quiere decir que se han seleccionado el mismo número

de reseñas negativas que positivas reduciendo así el número total de muestras a aproximadamente 70.000.

4.2. Procesamiento

Una vez realizada la selección, el filtrado y formateado de los datos debemos hacer que nuestro algoritmo sea capaz de trabajar con estas muestras. Hay muchas formas de llevar a cabo este proceso, las oraciones son secuencias de palabras y necesitamos una representación numérica del significado que tiene cada palabra.

Se han decidido utilizar representaciones vectoriales llamadas "embeddings" [11] para la representación semántica de cada palabra, entonces una vez hecha la conversión al algoritmo le llegarán secuencias de vectores para el entrenamiento.

Los embeddings son una forma de representación de la semántica de las palabras de forma vectorial y tiene propiedades como la de que palabras con significados similares dan como resultado del embedding vectores similares.

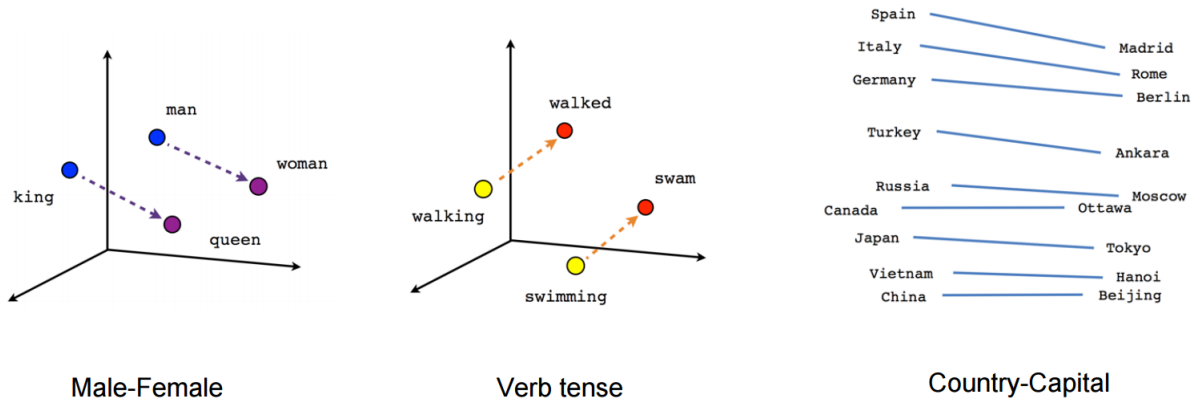

Figura 6 - Similitud entre palabras representadas mediante word embeddings.

Antes de la codificación mediante el uso de embeddings debemos convertir los textos a secuencias de palabras teniendo en cuenta que estas palabras han de tener un contenido semántico relevante, como son: nombres, verbos, adjetivos, adverbios, etc.

Por otro lado debemos asegurarnos que estos textos están libres de secuencias de caracteres que no aportan ningún significado o que son muy difíciles de tratar, como son: emojis, hipervínculos, carácteres sueltos, palabras compuestas de carácteres que no pertenecen al alfabeto, etc.

Una cosa más a tener en cuenta son las diferentes formas de palabras con la misma raíz que tienen un significado muy similar. Para esto se suelen utilizar dos técnicas muy populares en la industria, el “stemming” [9] y la “lematización” [10].

Stemming consiste en aislar únicamente la raíz de la palabra, esta contiene la mayor parte del significado, es un proceso muy rápido de ejecutar y da unos resultados relativamente buenos. Por contraparte la lematización consiste en convertir o transformar cada palabra a su forma canónica o lema, es un proceso computacionalmente más costoso, pero tiende a dar resultados más consistentes que el stemming.

Se diseñó una línea de procesamiento, utilizando herramientas de las librerías re y nltk, en la que se aplicaban las siguientes transformaciones:

- Convertir todos los carácteres de la oración a minúsculas.
- Eliminar hipervínculos.
- Separar la oración en palabras.
- Eliminar signos de puntuación.
- Eliminar lo que se conoce como “stop words”.
- Eliminar palabras que contienen carácteres que no pertenecen al alfabeto.
- Eliminar caracteres sueltos o palabras de longitud 1.
- Lematizar cada palabra de la oración.

"I am new to Ruby but have programmed in Python, Java, C/C++ for years. This course is truly aimed at non-programmers. Would recommend one of Huw's more advanced courses if you have any programming experience."

Figura 7 - Muestra de texto previo al procesamiento.

['new', 'ruby', 'programmed', 'python', 'java', 'cc', 'year', 'course', 'truly', 'aimed', 'nonprogrammers', 'would', 'recommend', 'one', 'huws', 'advanced', 'course', 'programming', 'experience']

Figura 8 - Secuencia de palabras extraída de texto procesado.

4.3. Codificación

Para la codificación de las muestras y basándonos en lo que se conoce como transfer learning [12] o transferencia de aprendizaje, utilizaremos un modelo preentrenado de embeddings.

Estos embeddings han sido generados a partir de corpus masivos de palabras o tokens, hablamos del orden de millones de palabras, lo que nos permite una representación de las palabras más precisa.

Para ello haremos uso del sistema GloVe [13], actualmente en conjunto con google, uno de los sistemas para representación de palabras mediante embeddings más utilizados.

4.3.1. *Tokenización*

Ahora que ya tenemos un conjunto de muestras procesadas, para poder codificarlas mediante embeddings, debemos generar un diccionario donde contendremos todas las palabras de nuestro corpus de entrenamiento para posteriormente asignarles un índice.

Para ello, utilizando la herramienta tokenizer de la librería keras, generamos dicho diccionario conteniendo todas las palabras de forma indexada y con la característica de que las palabras que este diccionario no contenga se les asignará un índice predeterminado.

'course': 2, 'good': 3, 'great': 4, 'really': 5, 'well': 6, 'lot': 7, 'learning': 8, 'learn': 9, 'instructor': 10, 'like': 11, 'would': 12, 'understand': 13, 'easy': 14, 'way': 15, 'much': 16, 'thank': 17, 'basic': 18, 'video': 19, 'time': 20, 'one': 21, 'concept': 22, 'excellent': 23, 'thing': 24, 'clear': 25, 'get': 26, 'also': 27, 'content': 28, 'example': 29

Figura 9 - Ejemplo palabras indexadas en el diccionario.

4.3.2. Embeddings

Utilizaremos el diccionario que previamente hemos generado en conjunto con estos embeddings pre entrenados del sistema GloVe para generar una lista de conversión de las palabras de nuestro diccionario a vectores numéricos.

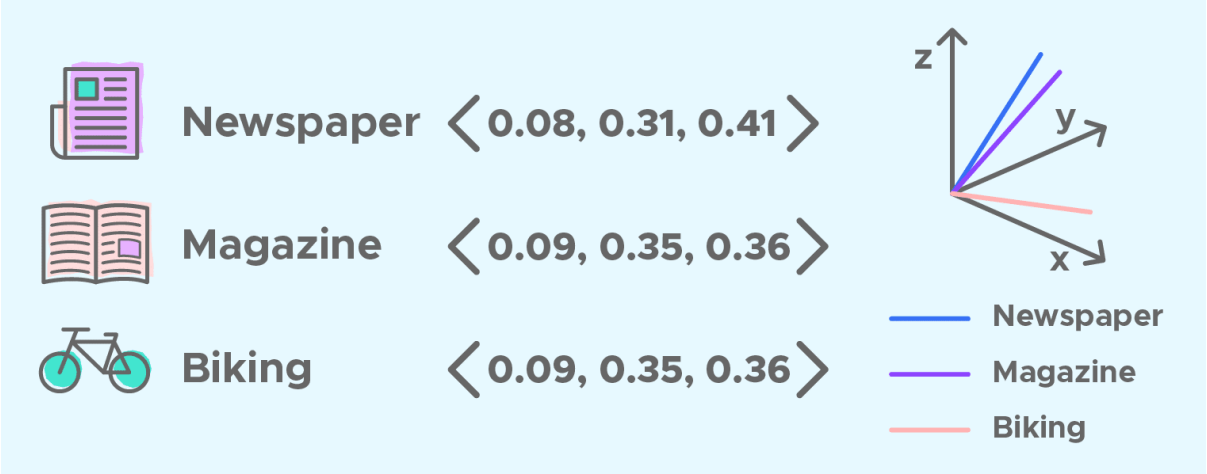

Figura 10 - Ejemplo de concepto word embeddings.

4.4. Entrenamiento y validación

4.4.1. Muestras de entrenamiento, validación y pruebas

Previo a la generación y entrenamiento de modelos debemos separar nuestras muestras en diferentes conjuntos de datos, esto nos permitirá realizar generalizaciones independientes y sin sesgos con respecto a los modelos.

Así pues se separan las muestras en tres conjuntos: conjunto de entrenamiento, conjunto de validación y conjunto de pruebas.

Utilizaremos el conjunto de entrenamiento para aproximar y optimizar los modelos que se estrenarán procesando y prediciendo estas muestras.

Utilizaremos el conjunto de validación para hacer un seguimiento del entrenamiento analizando el desempeño de este en ambos conjuntos, para evitar que haya lo que se conoce como overfitting.

Finalmente utilizaremos el conjunto de pruebas para medir qué modelos muestran un desempeño mejor con respecto a los otros.

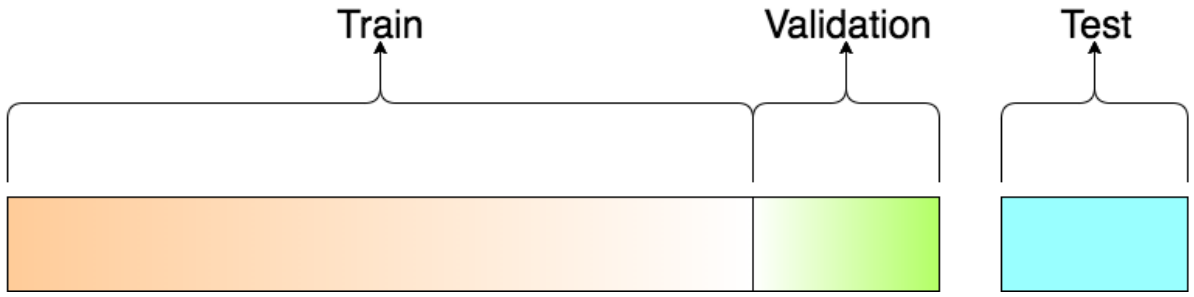

Figura 11 - Ejemplo de concepto conjuntos de entrenamiento, validación y pruebas.

4.4.2. Definición de modelos

Como anteriormente hemos comentado los modelos a definir son tres redes neuronales basadas en tres arquetipos diferentes : Un modelo basado en redes neuronales densas (FCNN) al que llamaremos “FCNN” , Un modelo basado en redes neuronales convolucionales (CNN) [14] al que llamaremos “CNN” y Un modelo basado en redes neuronales recurrentes (RNN) al que llamaremos “LSTM”.

Las redes neuronales densas FCNN, se usan como modelos de propósito general, estas aprenden a mapear las características de los datos y de forma gradual a medida que avanzan en las capas de la red aprenden a entender conceptos cada vez más complejos.

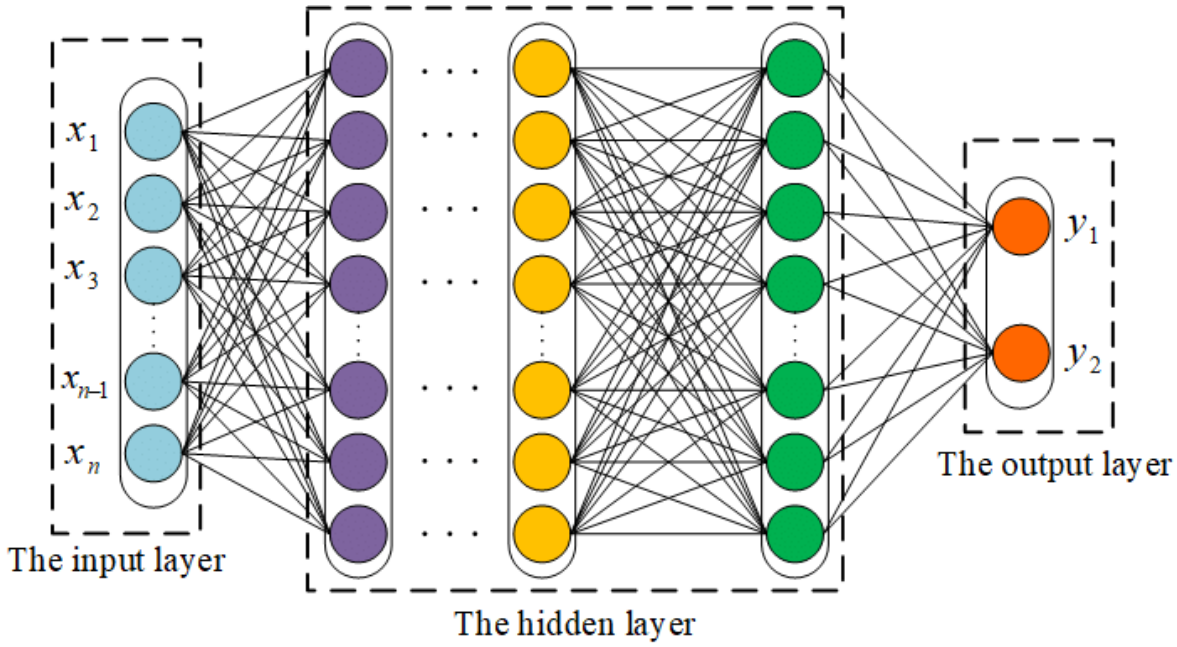

Figura 12 - Ejemplo de concepto red neuronal artificial.

Una red neuronal convolucional CNN , similares a las redes neuronales densas pero con el añadido de que contienen capas internas que funcionan aplicando convoluciones, son muy efectivas en el tratamiento de imágenes.

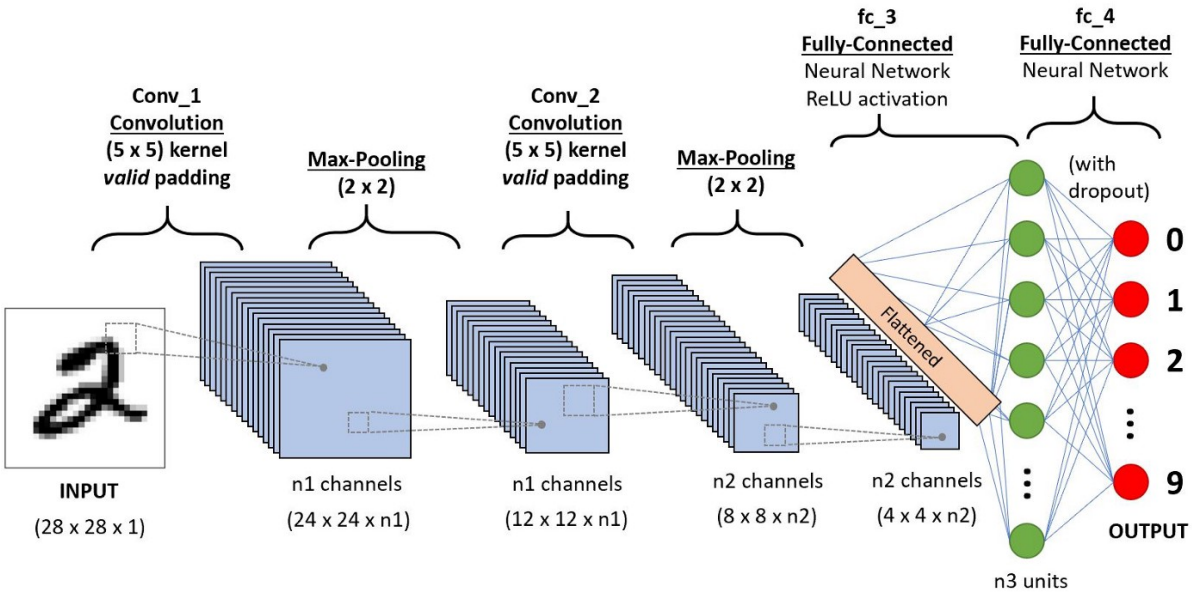

Figura 13 - Ejemplo de concepto red neuronal convolucional.

Una red neuronal con celdas de tipo LSTM [15] , las celdas LSTM son un tipo de neuronas que almacenan contexto entre muestras, pudiendo aprender patrones y relaciones entre

secuencias de datos. Se suelen utilizar para tratamiento de videos, voz y texto, concretamente datos que tengan un formato secuencial.

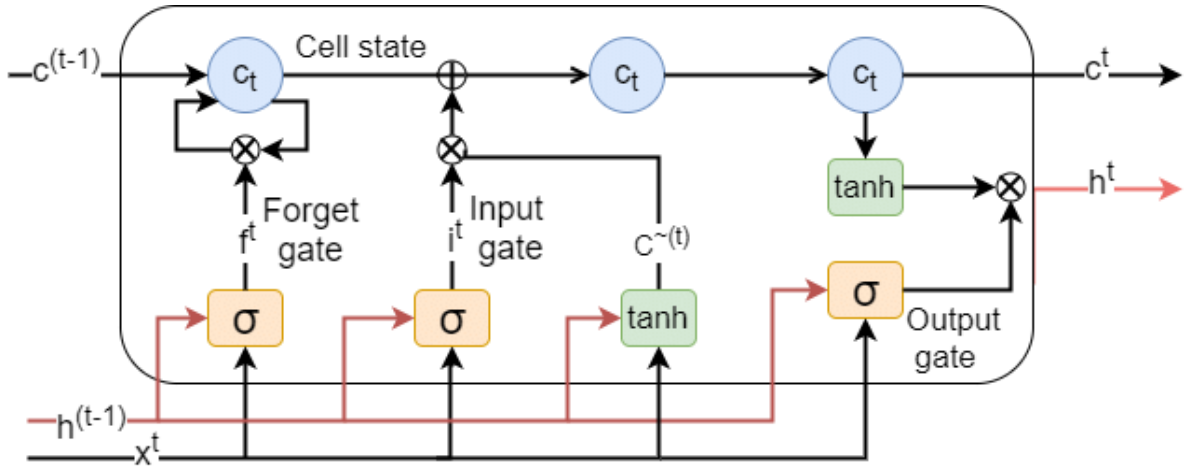

Figura 14 - Ejemplo estructura celda LSTM.

4.4.3. Entrenamiento de modelos

Una vez definidos los tres arquetipos de modelo procederemos al entrenamiento. Este consiste en que de forma iterativa al modelo se le irán dando muestras, el modelo hará una predicción, se evaluará esta predicción midiendo mediante la función de coste y utilizando el algoritmo conocido como algoritmo de backpropagation [16] se modificarán los parámetros a aprender de la red.

El entrenamiento se detendrá cuando el desempeño del modelo en el conjunto de validación deje de mejorar, previniendo así el overfitting [17].

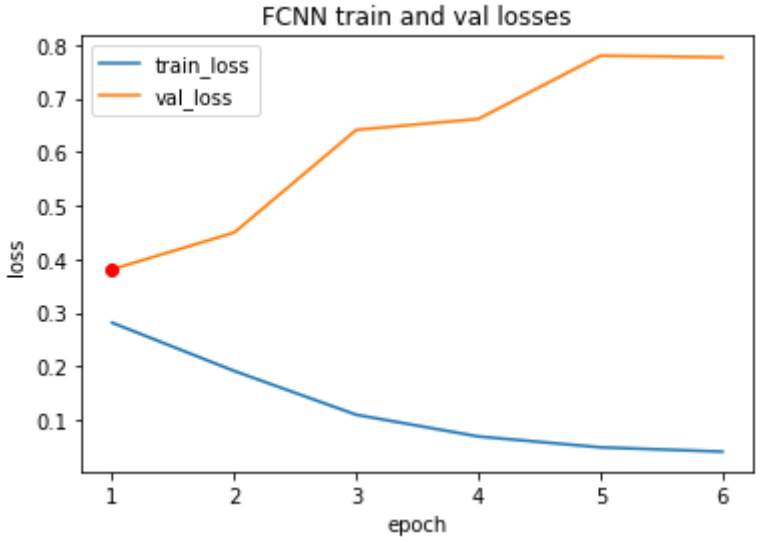

Figura 15 - Progresión de entrenamiento modelo FCNN

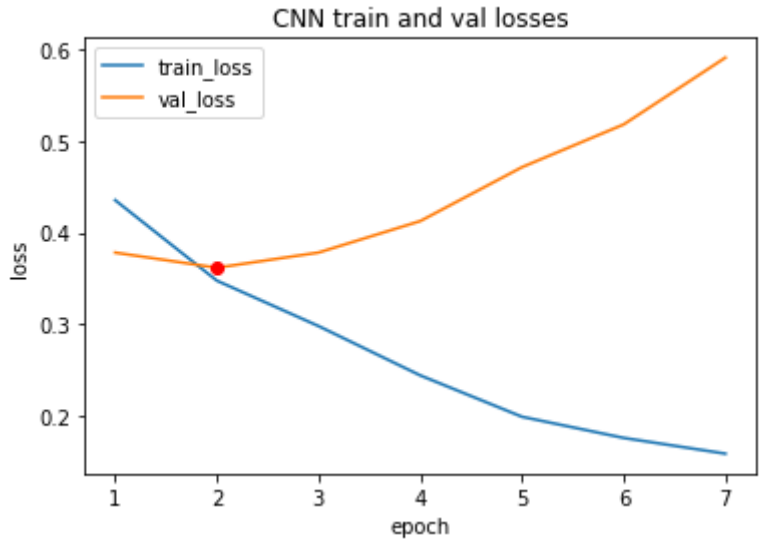

Figura 16 - Progresión de entrenamiento CNN

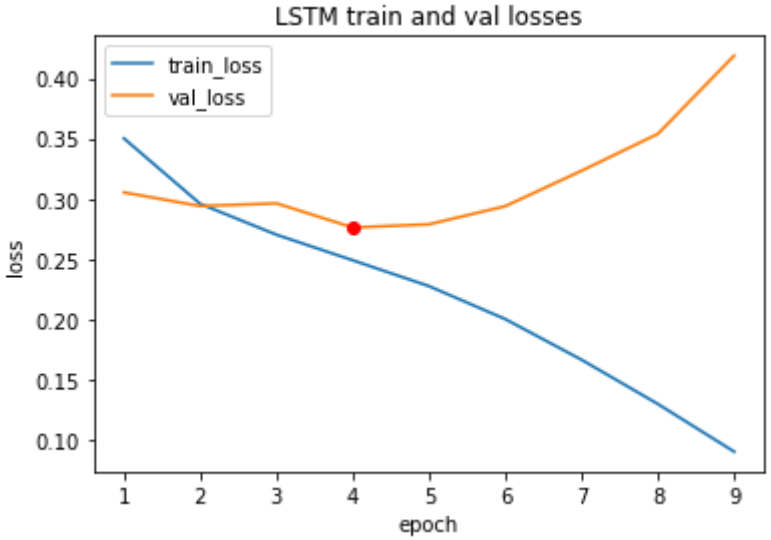

Figura 17 - Progresión de entrenamiento modelo LSTM

4.4.4. Reclasificación de muestras

Una cosa que impide que los modelos aprendan de forma correcta y efectiva es que haya en nuestro conjunto de datos muestras mal clasificadas o muestras cuya categoría es dudosa.

Este problema de muestras anómalas se puede solucionar haciendo que un modelo previamente entrenado haga una estimación de las muestras y analizar las muestras que tengan un error significativamente más grande que las otras. De esta forma podemos encontrar muestras que claramente pertenecen a la categoría contraria y clasificarlas de nuevo para un posterior reentrenamiento, lo que mejora el rendimiento del modelo.

Para identificar estos modelos añadiremos el sufijo "_flip_outliers" al nombre original del arquetipo.

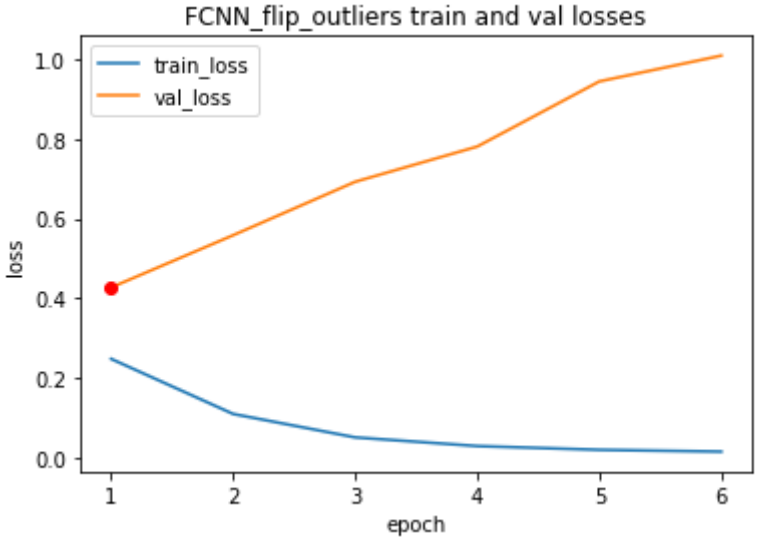

Figura 18 - Progresión de entrenamiento modelo FCNN_flip_outliers

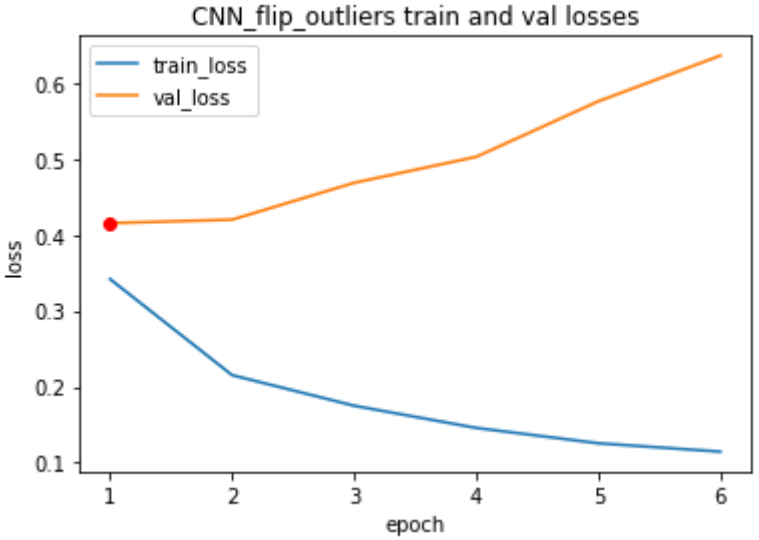

Figura 19 - Progresión de entrenamiento modelo CNN_flip_outliers

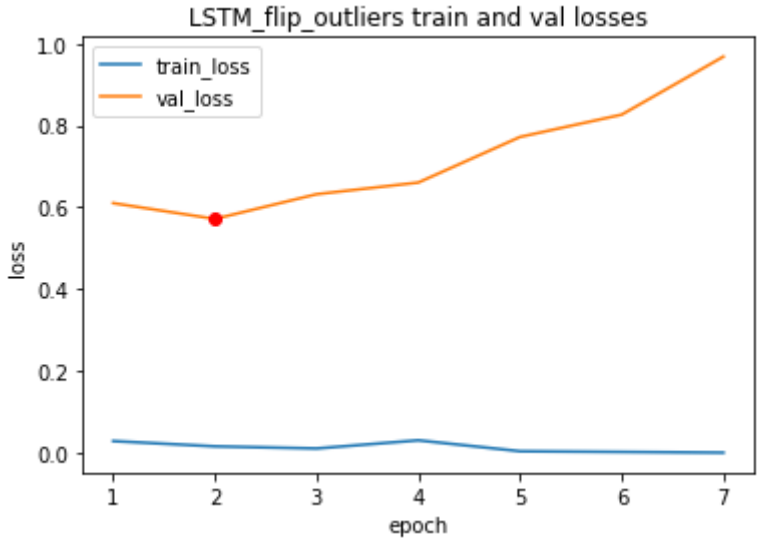

Figura 20 - Progresión de entrenamiento modelo LSTM_flip_outliers

	model	train_loss	val_loss	epoch
0	FCNN	0.281279	0.380947	1
7	CNN	0.347350	0.361450	2
16	LSTM	0.248947	0.276153	4
22	FCNN_flip_outliers	0.247194	0.426680	1
28	CNN_flip_outliers	0.341818	0.415553	1
35	LSTM_flip_outliers	0.016199	0.571763	2

Figura 21 - Coste de entrenamiento, validación y número de epochs de convergencia de cada modelo.

	FCNN	CNN	LSTM
default	180	280	1800
flip_outliers	120	200	800

Figura 22 - Tiempo total de entrenamiento de cada modelo en segundos.

4.5. Pruebas finales

Llegados a este punto, disponemos de nueve modelos efectivamente entrenados en total. Cada arquetipo de modelo entrenado de dos formas diferentes teniendo así dos versiones de cada uno: Una versión que llamaremos por defecto y una segunda versión en la que habremos entrenado a los modelos en base a los datos con las muestras que habremos reclasificado en el proceso de detección de anomalías.
Las métricas que utilizaremos para medir el desempeño de los modelos son las siguientes:

- Exactitud (Accuracy) : La exactitud mide el porcentaje de casos que el modelo ha acertado. Esta es una de las métricas más usadas y favoritas pero puede llegar a ser engañosa. Dado nuestro caso, donde aproximadamente el 94% de las muestras son positivas, si el modelo predijese que todas las muestras son positivas, nos daría que tiene una exactitud del 94% lo que a priori parece un muy buen resultado pero resulta ser un modelo efectivo a la hora de detectar muestras negativas.

- Precisión (Precision) : Con la métrica de precisión podemos medir la calidad del modelo en tareas de clasificación. En este caso la precisión sería la respuesta a la pregunta ¿qué porcentaje de los comentarios están bien clasificados?

- Exhaustividad (Recall) : La métrica de exhaustividad nos va a informar sobre la cantidad que el modelo es capaz de identificar. En este caso la exhaustividad (recall) respondería a la pregunta ¿qué porcentaje de los comentarios positivos somos capaces de identificar?

- Valor F1 (F1-score) : El valor F1 se utiliza para combinar las medidas de precisión y exhaustividad en un sólo valor. Esto es práctico porque hace más fácil el poder comparar el rendimiento combinado de la precisión y la exhaustividad entre varias soluciones.

	FCNN	CNN	LSTM
default	11	10	106
flip_outliers	11	10	106

Figura 23 - Tiempo de inferencia por muestra de cada modelo en milisegundos.

	model	accuracy	precision	recall	f1_score
0	FCNN	0.848664	0.877270	0.799134	0.836381
1	FCNN_flip_outliers	0.844997	0.835880	0.845833	0.840827
2	CNN	0.821795	0.791109	0.858512	0.823433
3	CNN_flip_outliers	0.813113	0.880123	0.710685	0.786380
4	LSTM	0.888930	0.893043	0.875367	0.884117
5	LSTM_flip_outliers	0.885488	0.882654	0.880470	0.881561

Figura 24 - Desempeño en las métricas de exactitud (accuracy), precisión (precision), exhaustividad (recall) y valor F1 (F1-score) de todos los modelos sobre el conjunto de pruebas finales.

5. Conclusiones

Finalmente, después de todo un largo proceso de colección, selección, procesamiento, codificación, entrenamiento, validación y pruebas, podemos sentarnos a analizar los resultados y las diferencias en el desempeño de los diferentes modelos.

Este apartado de conclusiones intentará arrojar un poco de luz sobre todos esos valores en tablas que pueden parecer confusos y estará dividido en cuatro subsecciones donde se detallarán pros y contras de los modelos.

5.1. Métricas

Como se puede observar en la *Figura 24* los modelos LSTM, pese a que son mas costosos a nivel de inferencia y entrenamiento, tienden a dar mejores resultados:

- Exactitud :
    - LSTM con un valor de 0.888930
- Precisión:
    - LSTM con un valor de 0.893043
- Exhaustividad:
    - LSTM_flip_outliers con un valor de 0.880470
- Valor F1:
    - LSTM con un valor de 0.884117

5.2. Tiempo de inferencia

En cuanto al tiempo de inferencia si hay una diferencia notoria entre los modelos FCNN y CNN con respecto a los modelos LSTM, siendo estos últimos más lentos y costosos de computar.
De ser esta métrica importante a la hora de la implementación deberá considerarse el descartar los modelos LSTM. Si por otro lado lo que importa es puramente el desempeño del modelo, habría que valorar mediante las métricas de desempeño anteriormente mencionadas.

5.3. Mejoras

Es importante destacar que en este proyecto el margen de mejoras es considerablemente amplio, el campo del aprendizaje automático está viviendo un crecimiento sin precedentes lo que hace que cada pocos meses se desarrollen nuevas técnicas y mejoras en las metodologías de desarrollo de este tipo de sistemas.

A continuación se enlistan posibles mejoras que en base al criterio personal, se consideran oportunas:

- Aumento del volumen de datos: La mayor parte de los recursos en esta industria se están destinando a la recolección de datos, se ha demostrado que un mayor volumen de datos siempre mejora el desempeño de los modelos.

- Modelos más grandes y complejos : Se ha observado a lo largo de los años que modelos más complejos y con mayor cantidad de parámetros desempeñan mejor.
- Un entorno de desarrollo más potente : Aumentar la capacidad de cómputo podría reducir considerablemente los tiempos de entrenamiento, permitiendo así un entorno de optimización más dinámico, lo que aceleraría el proceso de reajuste de los modelos.
- Una mejor utilización y entendimiento de las métricas : Un mejor análisis estadístico con respecto a la predicción de modelos podría señalar puntos críticos a optimizar.

5.4. Que sucede a continuación

El siguiente paso de este proyecto es darle uso en un entorno real a los modelos entrenados y evaluar si en un entorno real se comportan de la forma deseada.

Se ha preparado un cuaderno de tipo Jupyter donde se pueden hacer pruebas en tiempo real de los modelos y probar así su efectividad.

6. Anexo

6.1. Código

6.1.1. *data_selection.ipynb*

```
#Import data frame dependencies
import pandas as pd
import numpy as np
import time
from langdetect import detect

#Load udemy and coursera raw data
coursera_reviews = pd.read_csv("coursera reviers
1.4M/Coursera_reviews.csv")

udemy_reviews = pd.read_csv("udemy courses/udemy review.csv",sep=';')
udemy_courses = pd.read_csv("udemy courses/udemy course.csv",sep=';')

#Drop non useful features
coursera_reviews = coursera_reviews.drop(['date_reviews','reviewers'],
axis=1)
#Rename useful features
coursera_reviews = coursera_reviews.rename(columns={'course_id' :
'course_name', 'reviews' : 'review_content', 'rating' : 'rating'})
#Reorder the features
coursera_reviews = coursera_reviews[['review_content','rating']]

#Here we need to merge both udemy datasets due that info is split in
different tables
udemy_reviews =
udemy_reviews.merge(udemy_courses[['course_id','course_name']],on="course_i
d")
#Drop non useful features
udemy_reviews = udemy_reviews.drop(['date','course_id'], axis=1)
#Reorder the features
udemy_reviews = udemy_reviews[['review_content','rating']]

#drop empty reviews and reviews without rating
udemy_reviews = udemy_reviews.dropna(axis=0)
coursera_reviews = coursera_reviews.dropna(axis=0)

#first we fillter reviews longer than N characters
N = 1500
mask = (udemy_reviews.review_content.str.len() <= N)
udemy_reviews = udemy_reviews.loc[mask]
mask = (coursera_reviews.review_content.str.len() <= N)
coursera_reviews = coursera_reviews.loc[mask]

#Language detection and filtering for english only reviews
def lang_detection(x):
    try:
        return detect(x)
    except:
        return np.nan
```

```
udemy_reviews['lang']= udemy_reviews.review_content.apply(lambda x :
lang_detection(x))
udemy_reviews = udemy_reviews[udemy_reviews['lang'] == 'en']

coursera_reviews['lang']= coursera_reviews.review_content.apply(lambda x :
lang_detection(x))
coursera_reviews = coursera_reviews[coursera_reviews['lang'] == 'en']

coursera_reviews = coursera_reviews[['review_content','rating']]
udemy_reviews = udemy_reviews[['review_content','rating']]

#Convert values to the same domain
udemy_reviews.rating = udemy_reviews.rating.replace(0.5,1)
udemy_reviews.rating = udemy_reviews.rating.replace(1.5,2)
udemy_reviews.rating = udemy_reviews.rating.replace(2.5,3)
udemy_reviews.rating = udemy_reviews.rating.replace(3.5,4)
udemy_reviews.rating = udemy_reviews.rating.replace(4.5,5)

udemy_reviews = udemy_reviews[udemy_reviews.rating != 3.9]
udemy_reviews.rating = udemy_reviews.rating.astype(int)

#Binarize the datasets
udemy_reviews.rating = udemy_reviews.rating.replace([1,2],0)
coursera_reviews.rating = coursera_reviews.rating.replace([1,2],0)

udemy_reviews.rating = udemy_reviews.rating.replace([3,4,5],1)
coursera_reviews.rating = coursera_reviews.rating.replace([3,4,5],1)

#save cleaned datasets
udemy_reviews.to_csv('data_selection/cleaned_udemy_reviews.csv',index=False
)
coursera_reviews.to_csv('data_selection/cleaned_coursera_reviews.csv',index
=False)

coursera_reviews =
pd.read_csv("data_selection/cleaned_coursera_reviews.csv")
udemy_reviews = pd.read_csv("data_selection/cleaned_udemy_reviews.csv")

data_set = pd.concat([udemy_reviews, coursera_reviews], axis=0)
data_set = data_set.reset_index(drop=True)
negatives = data_set[data_set.rating == 0]
positives = data_set[data_set.rating == 1].sample(data_set[data_set.rating
== 0].rating.count(),ignore_index=True)
data_set = pd.concat([negatives, positives], axis=0)
data_set = data_set.reset_index(drop=True)
data_set.to_csv('data_selection/data_set.csv',index=False)
```

6.1.2. *data_preprocessing.ipynb*

```
import pandas as pd
import numpy as np
import time
import pickle
```

```
import string
import nltk
import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from ast import literal_eval

nltk.download('stopwords')
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()

def tokenize(doc):
    #delete hyperlinks
    doc = re.sub(r'\w+:\/{2}[\d\w-]+(\.[\d\w-]+)*(?:(?:\/[^\s/]*))*', '',
doc)
    #Normalize the string (improvement 1)
    doc = doc.lower()
    # split into tokens by white space
    tokens = doc.split()
    # prepare regex for char filtering
    re_punc = re.compile('[%s]' % re.escape(string.punctuation))
    # remove punctuation from each word
    tokens = [re_punc.sub('', w) for w in tokens]
    # filter out stop words
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]
    # remove remaining tokens that are not alphabetic
    tokens = [word for word in tokens if word.isalpha()]
    # filter out short tokens
    tokens = [word for word in tokens if len(word) > 1]
    # lemmatize (improvement 2)
    tokens = [(lambda word: lemmatizer.lemmatize(word))(word) for word in
tokens]

    return tokens

def encode(tokens, tokenizer):
    #tokens = tokenize(text)
    encoded = tokenizer.texts_to_sequences([tokens])
    encoded = pad_sequences(encoded, maxlen=200)
    return encoded[0]

#load datasets
data_set = pd.read_csv('data_selection/data_set.csv')

train_data = pd.DataFrame(columns=['text','X','y'])
test_data = pd.DataFrame(columns=['text','X','y'])
```

```
train_data.text, test_data.text, train_data.y, test_data.y =
train_test_split(data_set.review_content, data_set.rating, test_size=0.2,
random_state=1)

train_data.X = train_data.text.apply(str).apply(tokenize)
test_data.X = test_data.text.apply(str).apply(tokenize)

train_data = train_data.astype(str).drop_duplicates(subset=['X'])
train_data = train_data.reset_index(drop=True)

test_data = test_data.astype(str).drop_duplicates(subset=['X'])
test_data = test_data.reset_index(drop=True)

train_data.X = train_data.X.apply(lambda x: literal_eval(str(x)))
train_data.y = train_data.y.apply(lambda x: literal_eval(str(x)))

test_data.X = test_data.X.apply(lambda x: literal_eval(str(x)))
test_data.y = test_data.y.apply(lambda x: literal_eval(str(x)))

tokenizer = Tokenizer(oov_token=1)
tokenizer.fit_on_texts(train_data.X)

train_data.X = train_data.X.apply(lambda x: encode(x,tokenizer))
test_data.X = test_data.X.apply(lambda x: encode(x,tokenizer))

with open('auxiliar_resources/tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)

data = train_data
train_data = data.sample(frac=0.8,random_state=1)
val_data = data[~data.index.isin(train_data.index)]

train_data.to_parquet('data_preprocessing/train_data.gzip',compression='gzi
p')
val_data.to_parquet('data_preprocessing/val_data.gzip',compression='gzip')
test_data.to_parquet('data_preprocessing/test_data.gzip',compression='gzip'
)

path_to_glove_file = "glove.42B.300d/glove.42B.300d.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

embedding_matrix = np.zeros([len(tokenizer.word_index)+1, 300])
for word, i in tokenizer.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
np.save('auxiliar_resources/embedding_matrix.npy',embedding_matrix)
```

6.1.3. *model_selection.ipybn*

```
import pandas as pd
import numpy as np
import pickle
import keras
from keras import metrics
from tensorflow.keras.models import Sequential
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from keras import callbacks
from tensorflow.keras.layers import Conv1D, MaxPooling1D
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Embedding

with open('auxiliar_resources/tokenizer.pickle', 'rb') as handle:
    tokenizer = pickle.load(handle)

word_index = tokenizer.word_index
embedding_matrix = np.load('auxiliar_resources/embedding_matrix.npy')
train_data = pd.read_parquet('data_preprocessing/train_data.gzip')
val_data = pd.read_parquet('data_preprocessing/val_data.gzip')

embedding_layer = Embedding(input_dim=len(word_index) + 1,
                            output_dim=300,
                            weights=[embedding_matrix],
                            input_length=200,
                            trainable=False)

model = Sequential()
model.add(embedding_layer)
model.add(Flatten())
model.add(Dense(300, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam')
model.summary()
model.save('models/fcnn.h5')

model2 = Sequential()
model2.add(embedding_layer)
model2.add(Conv1D(filters=64, kernel_size=8, activation='relu'))
model2.add(MaxPooling1D(pool_size=2))
model2.add(Conv1D(filters=32, kernel_size=8, activation='relu'))
model2.add(MaxPooling1D(pool_size=2))
model2.add(Flatten())
model2.add(Dense(10, activation='relu'))
model2.add(Dense(1, activation='sigmoid'))
model2.compile(loss='binary_crossentropy', optimizer='adam')
```

```
model2.summary()
model2.save('models/cnn.h5')

model3 = Sequential()
model3.add(embedding_layer)
model3.add(LSTM(200))
model3.add(Dense(10, activation='relu'))
model3.add(Dense(1, activation='sigmoid'))
model3.compile(loss='binary_crossentropy', optimizer='adam')
model3.summary()
model3.save('models/lstm.h5')

es1 = callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=5)
sm1 =
callbacks.ModelCheckpoint(filepath='models/checkpoints/FCNN_flip_outliers/c
heckpoint', save_weights_only=True, monitor='val_loss', mode='min',
save_best_only=True)

history = model.fit(np.expand_dims(train_data.X.to_list(), -1),
                    np.asarray(train_data.y),
                    batch_size=128,
                    epochs=100,
                    verbose=2,
                    callbacks=[es1,sm1],
                    validation_data=[np.expand_dims(val_data.X.to_list(),
-1),np.asarray(val_data.y)])

pd.DataFrame(history.history).to_csv('train_history/fcnn_flip_outliers.csv'
,index=False)

es2 = callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=5)
sm2 =
callbacks.ModelCheckpoint(filepath='models/checkpoints/CNN_flip_outliers/ch
eckpoint', save_weights_only=True, monitor='val_loss', mode='min',
save_best_only=True)

history = model2.fit(np.expand_dims(train_data.X.to_list(), -1),
                    np.asarray(train_data.y),
                    batch_size=128,
                    epochs=100,
                    verbose=2,
                    callbacks=[es2,sm2],
                    validation_data=[np.expand_dims(val_data.X.to_list(),
-1),np.asarray(val_data.y)])

pd.DataFrame(history.history).to_csv('train_history/cnn_flip_outliers.csv',
index=False)

es3 = callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=5)
```

```
sm3 =
callbacks.ModelCheckpoint(filepath='models/checkpoints/LSTM_flip_outliers/c
heckpoint', save_weights_only=True, monitor='val_loss', mode='min',
save_best_only=True)

history = model3.fit(np.expand_dims(train_data.X.to_list(), -1),
                     np.asarray(train_data.y),
                     batch_size=128,
                     epochs=100,
                     verbose=2,
                     callbacks=[es3,sm3],
                     validation_data=[np.expand_dims(val_data.X.to_list(),
-1),np.asarray(val_data.y)])

pd.DataFrame(history.history).to_csv('train_history/lstm_flip_outliers.csv'
,index=False)

train_data['temp'] = np.zeros(len(train_data.X))
model3.load_weights('models/checkpoints/LSTM/checkpoint')
train_data['temp'] = model3.predict(np.expand_dims(train_data.X.to_list(),
-1))
train_data['error'] = abs(train_data.y-train_data.temp)

sorted_by_error = train_data.sort_values('error',axis=0,ascending=True)
outliers_to_flip = sorted_by_error[sorted_by_error.error> 0.7].index
outliers_to_delete = sorted_by_error[(sorted_by_error.error < 0.7) &
(sorted_by_error.error > 0.4)].index

train_data.at[outliers_to_flip,'y'] =
abs(train_data.loc[outliers_to_flip].y-1)
train_data.drop(outliers_to_delete,inplace=True,axis=0)
```

6.1.4. *model_testing.ipybn*

```
import keras
import pandas as pd
import numpy as np
```

```
train_data = pd.read_parquet('data_preprocessing/train_data.gzip')
val_data = pd.read_parquet('data_preprocessing/test_data.gzip')
test_data = pd.read_parquet('data_preprocessing/test_data.gzip')
train_predictions = pd.DataFrame(index=train_data.index)
val_predictions = pd.DataFrame(index=val_data.index)
test_predictions = pd.DataFrame(index=test_data.index)

fcnn = keras.models.load_model('models/fcnn.h5')
cnn = keras.models.load_model('models/cnn.h5')
lstm = keras.models.load_model('models/lstm.h5')

fcnn.load_weights('models/checkpoints/FCNN/checkpoint')
train_predictions['FCNN'] =
fcnn.predict(np.expand_dims(train_data.X.to_list(), -1))
val_predictions['FCNN'] = fcnn.predict(np.expand_dims(val_data.X.to_list(),
-1))
test_predictions['FCNN'] =
fcnn.predict(np.expand_dims(test_data.X.to_list(), -1))
train_predictions.to_csv('results/train_predictions.csv',index=False)
val_predictions.to_csv('results/val_predictions.csv',index=False)
test_predictions.to_csv('results/test_predictions.csv',index=False)

fcnn.load_weights('models/checkpoints/FCNN_flip_outliers/checkpoint')
train_predictions['FCNN_flip_outliers'] =
fcnn.predict(np.expand_dims(train_data.X.to_list(), -1))
val_predictions['FCNN_flip_outliers'] =
fcnn.predict(np.expand_dims(val_data.X.to_list(), -1))
test_predictions['FCNN_flip_outliers'] =
fcnn.predict(np.expand_dims(test_data.X.to_list(), -1))
train_predictions.to_csv('results/train_predictions.csv',index=False)
val_predictions.to_csv('results/val_predictions.csv',index=False)
test_predictions.to_csv('results/test_predictions.csv',index=False)

cnn.load_weights('models/checkpoints/CNN/checkpoint')
train_predictions['CNN'] =
cnn.predict(np.expand_dims(train_data.X.to_list(), -1))
val_predictions['CNN'] = cnn.predict(np.expand_dims(val_data.X.to_list(),
-1))
test_predictions['CNN'] = cnn.predict(np.expand_dims(test_data.X.to_list(),
-1))
train_predictions.to_csv('results/train_predictions.csv',index=False)
val_predictions.to_csv('results/val_predictions.csv',index=False)
test_predictions.to_csv('results/test_predictions.csv',index=False)

cnn.load_weights('models/checkpoints/CNN_flip_outliers/checkpoint')
train_predictions['CNN_flip_outliers'] =
cnn.predict(np.expand_dims(train_data.X.to_list(), -1))
val_predictions['CNN_flip_outliers'] =
cnn.predict(np.expand_dims(val_data.X.to_list(), -1))
test_predictions['CNN_flip_outliers'] =
cnn.predict(np.expand_dims(test_data.X.to_list(), -1))
train_predictions.to_csv('results/train_predictions.csv',index=False)
val_predictions.to_csv('results/val_predictions.csv',index=False)
```

```
test_predictions.to_csv('results/test_predictions.csv',index=False)

lstm.load_weights('models/checkpoints/LSTM/checkpoint')
train_predictions['LSTM'] =
lstm.predict(np.expand_dims(train_data.X.to_list(), -1))
val_predictions['LSTM'] = lstm.predict(np.expand_dims(val_data.X.to_list(),
-1))
test_predictions['LSTM'] =
lstm.predict(np.expand_dims(test_data.X.to_list(), -1))
train_predictions.to_csv('results/train_predictions.csv',index=False)
val_predictions.to_csv('results/val_predictions.csv',index=False)
test_predictions.to_csv('results/test_predictions.csv',index=False)

lstm.load_weights('models/checkpoints/LSTM_flip_outliers/checkpoint')
train_predictions['LSTM_flip_outliers'] =
lstm.predict(np.expand_dims(train_data.X.to_list(), -1))
val_predictions['LSTM_flip_outliers'] =
lstm.predict(np.expand_dims(val_data.X.to_list(), -1))
test_predictions['LSTM_flip_outliers'] =
lstm.predict(np.expand_dims(test_data.X.to_list(), -1))
train_predictions.to_csv('results/train_predictions.csv',index=False)
val_predictions.to_csv('results/val_predictions.csv',index=False)
test_predictions.to_csv('results/test_predictions.csv',index=False)

train_predictions = pd.read_csv('results/train_predictions.csv')
val_predictions = pd.read_csv('results/val_predictions.csv')
test_predictions = pd.read_csv('results/test_predictions.csv')

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

train_scores =
pd.DataFrame(columns=['model','accuracy','precision','recall','f1_score'])
val_scores =
pd.DataFrame(columns=['model','accuracy','precision','recall','f1_score'])
test_scores =
pd.DataFrame(columns=['model','accuracy','precision','recall','f1_score'])

threshold = 0.5
for model in train_predictions.columns:
    row = {'model' :model,
    'accuracy' : accuracy_score(train_data.y,
train_predictions[model]>=threshold),
    'precision' : precision_score(train_data.y,
train_predictions[model]>=threshold),
    'recall' : recall_score(train_data.y,
train_predictions[model]>=threshold),
    'f1_score' : f1_score(train_data.y,
train_predictions[model]>=threshold)}
    train_scores = train_scores.append(row,ignore_index=True)
```

```
threshold = 0.5
for model in val_predictions.columns:
    row = {'model' :model,
    'accuracy' : accuracy_score(val_data.y,
val_predictions[model]>=threshold),
    'precision' : precision_score(val_data.y,
val_predictions[model]>=threshold),
    'recall' : recall_score(val_data.y, val_predictions[model]>=threshold),
    'f1_score' : f1_score(val_data.y, val_predictions[model]>=threshold)}
    val_scores = val_scores.append(row,ignore_index=True)

threshold = 0.5
for model in test_predictions.columns:
    row = {'model' :model,
    'accuracy' : accuracy_score(test_data.y,
test_predictions[model]>=threshold),
    'precision' : precision_score(test_data.y,
test_predictions[model]>=threshold),
    'recall' : recall_score(test_data.y,
test_predictions[model]>=threshold),
    'f1_score' : f1_score(test_data.y, test_predictions[model]>=threshold)}
    test_scores = test_scores.append(row,ignore_index=True)
```

6.1.5. data_exploration.ipybn

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

fcnn_hist = pd.read_csv('train_history/fcnn.csv')
cnn_hist = pd.read_csv('train_history/cnn.csv')
lstm_hist = pd.read_csv('train_history/lstm.csv')
fcnn_flip_outliers_history =
pd.read_csv('train_history/fcnn_flip_outliers.csv')
cnn_flip_outliers_history =
pd.read_csv('train_history/cnn_flip_outliers.csv')
lstm_flip_outliers_history =
pd.read_csv('train_history/lstm_flip_outliers.csv')

fcnn_hist['model'] = 'FCNN'
cnn_hist['model'] = 'CNN'
lstm_hist['model'] = 'LSTM'
fcnn_flip_outliers_history['model'] = 'FCNN_flip_outliers'
cnn_flip_outliers_history['model'] = 'CNN_flip_outliers'
lstm_flip_outliers_history['model'] = 'LSTM_flip_outliers'

models_hist = pd.concat([fcnn_hist,
                         cnn_hist,
                         lstm_hist,
                         fcnn_flip_outliers_history,
                         cnn_flip_outliers_history,
                         lstm_flip_outliers_history])

model_losses = models_hist[['model','loss','val_loss']]
```

```
model_losses['epoch'] = model_losses.index
model_losses.reset_index(drop=True, inplace=True)
model_losses.epoch = model_losses.epoch+1

model_name = 'FCNN'
model = model_losses[model_losses.model == model_name]
plt.plot(model.epoch, model.loss, label='train_loss')
plt.plot(model.epoch, model.val_loss, label='val_loss')
plt.plot(model[model.val_loss == model.val_loss.min()].epoch,
         model[model.val_loss == model.val_loss.min()].val_loss,
        'ro')
plt.title(model.model[0]+" train and val losses")
plt.xticks(model.epoch)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend()

model_name = 'FCNN_flip_outliers'
model = model_losses[model_losses.model == model_name]
plt.plot(model.epoch, model.loss, label='train_loss')
plt.plot(model.epoch, model.val_loss, label='val_loss')
plt.plot(model[model.val_loss == model.val_loss.min()].epoch,
         model[model.val_loss == model.val_loss.min()].val_loss,
        'ro')
plt.title(model_name+" train and val losses")
plt.xticks(model.epoch)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend()

model_name = 'CNN'
model = model_losses[model_losses.model == model_name]
plt.plot(model.epoch, model.loss, label='train_loss')
plt.plot(model.epoch, model.val_loss, label='val_loss')
plt.plot(model[model.val_loss == model.val_loss.min()].epoch,
         model[model.val_loss == model.val_loss.min()].val_loss,
        'ro')
plt.title(model_name+" train and val losses")
plt.xticks(model.epoch)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend()
model_name = 'CNN_flip_outliers'
model = model_losses[model_losses.model == model_name]
plt.plot(model.epoch, model.loss, label='train_loss')
plt.plot(model.epoch, model.val_loss, label='val_loss')
plt.plot(model[model.val_loss == model.val_loss.min()].epoch,
         model[model.val_loss == model.val_loss.min()].val_loss,
        'ro')
plt.title(model_name+" train and val losses")
plt.xticks(model.epoch)
plt.xlabel("epoch")
plt.ylabel("loss")
```

```
plt.legend()

model_name = 'LSTM'
model = model_losses[model_losses.model == model_name]
plt.plot(model.epoch, model.loss, label='train_loss')
plt.plot(model.epoch, model.val_loss, label='val_loss')
plt.plot(model[model.val_loss == model.val_loss.min()].epoch,
         model[model.val_loss == model.val_loss.min()].val_loss,
        'ro')
plt.title(model_name+" train and val losses")
plt.xticks(model.epoch)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend()

model_name = 'LSTM_flip_outliers'
model = model_losses[model_losses.model == model_name]
plt.plot(model.epoch, model.loss, label='train_loss')
plt.plot(model.epoch, model.val_loss, label='val_loss')
plt.plot(model[model.val_loss == model.val_loss.min()].epoch,
         model[model.val_loss == model.val_loss.min()].val_loss,
        'ro')
plt.title(model_name+" train and val losses")
plt.xticks(model.epoch)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend()
```

6.1.6. model_tryouts.ipybn

```
import time
import numpy as np
import pickle
import string
import nltk
import re
import keras
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from ast import literal_eval

FCNN = keras.models.load_model('models/FCNN.h5')
FCNN.load_weights('models/checkpoints/FCNN/checkpoint')

nltk.download('stopwords')
nltk.download('wordnet')
with open('auxiliar_resources/tokenizer.pickle', 'rb') as handle:
    tokenizer = pickle.load(handle)
lemmatizer = WordNetLemmatizer()

def tokenize(doc):
    #delete hyperlinks
```

```
    doc = re.sub(r'\w+:\/{2}[\d\w-]+(\.[\d\w-]+)*(?:(?:\/[^\s/]*))*', '',
doc)
    #Normalize the string (improvement 1)
    doc = doc.lower()
    # split into tokens by white space
    tokens = doc.split()
    # prepare regex for char filtering
    re_punc = re.compile('[%s]' % re.escape(string.punctuation))
    # remove punctuation from each word
    tokens = [re_punc.sub('', w) for w in tokens]
    # filter out stop words
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]
    # remove remaining tokens that are not alphabetic
    tokens = [word for word in tokens if word.isalpha()]
    # filter out short tokens
    tokens = [word for word in tokens if len(word) > 1]
    # lemmatize (improvement 2)
    tokens = [(lambda word: lemmatizer.lemmatize(word))(word) for word in
tokens]

    return tokens

def encode(tokens, tokenizer):
    #tokens = tokenize(text)
    encoded = tokenizer.texts_to_sequences([tokens])
    encoded = pad_sequences(encoded, maxlen=200)
    return encoded[0]

def predict(text, model, tokenizer):
    text = tokenize(text)
    print(text)
    text = encode(text ,tokenizer)
    return model.predict(np.expand_dims(text, 0))

text_to_try = "i am excited with this course"
predict(text_to_try, FCNN, tokenizer)
```

6.2. Reproducción y puesta en marcha

Para la reproducción, pruebas y puesta en marcha de los modelos se ha preparado un cuaderno de jupyter “model_tryouts.ipybn” donde se podrán probar los modelos de forma independiente.

7. Referencias

[1] Inteligencia Artificial - https://es.wikipedia.org/wiki/Inteligencia_artificial
[2] Aprendizaje automático - https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico
[3] Aprendizaje profundo - https://es.wikipedia.org/wiki/Aprendizaje_profundo
[4] Red neuronal - https://es.wikipedia.org/wiki/Red_neuronal_artificial
[5] Función de coste - https://en.wikipedia.org/wiki/Cross_entropy
[6] Web scraping - https://es.wikipedia.org/wiki/Web_scraping
[7] API - https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones
[8] Kaggle - https://es.wikipedia.org/wiki/Kaggle
[9] Stemming - https://es.wikipedia.org/wiki/Stemming
[10] Lematización - https://es.wikipedia.org/wiki/Lematizaci%C3%B3n
[11] Embedding - https://es.wikipedia.org/wiki/Word_embedding
[12] Transfer learning - https://en.wikipedia.org/wiki/Transfer_learning
[13] GloVe - https://en.wikipedia.org/wiki/GloVe
[14] Convolución - https://es.wikipedia.org/wiki/Convoluci%C3%B3n
[15] Celda LSTM - https://en.wikipedia.org/wiki/Long_short-term_memory
[16] Algoritmo de backpropagation - https://es.wikipedia.org/wiki/Propagaci%C3%B3n_hacia_atr%C3%A1s
[17] Overfitting - https://es.wikipedia.org/wiki/Sobreajuste