Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Trabajo Práctico Final

Fine-Tune de Distil-BERT para clasificación de textos

Introducción a la Inteligencia Artificial

LCC

Integrantes:

- Agustín Samper — Legajo: S-5552/2
- Zoe Granitto — Legajo: G-5749/5

Rosario, 12 de agosto de 2025

Introducción

En el siguiente trabajo discutimos sobre varios intentos para acercarnos a un modelo de `distil-bert` que pueda responder a la pregunta *"A qué área de estudio pertenece este texto?"*. Para ello, realizamos fine-tuning agregando al modelo ya entrenado una capa de clasificación.
Para realizar esta técnica de transfer learning de manera adecuada, fue necesario frizar las capas del modelo pre-entrenado y únicamente agregar una cabeza de clasificación y entrenarla. Si hubiésemos entrenado todo el modelo, al utilizar pocos datos, probablemente hubiéramos llegado a un sobreajuste. Por ello, esta capa de clasificación (junto con una capa de pre-clasificación) serán las encargadas de aprender este pequeño problema particular, con el conocimiento que ya tiene distil-bert.
Durante el trabajo, comparamos cómo funciona el modelo utilizando textos que se asemejen más o bien que difieran más en su contenido. Además, probamos ejemplos particulares para lograr entender qué fue lo que aprendió la red y cuáles fueron los casos que más le dificultaron. Por otro lado, fuimos variando distintos parámetros del entrenamiento, al igual que los tamaños de los conjuntos de datos para entrenamiento, validación y test. También, para poder entender mejor algunos resultados, realizamos distintos gráficos sobre ellos.
A lo largo del informe discutiremos los resultados obtenidos y las conclusiones que pudimos sacar con ellos.

Desarrollo

Modelos obtenidos

Al considerar la propuesta de clasificar textos en distintas áreas académicas, surge la pregunta de qué áreas considerar. Esta discusión es fundamental, pues la diferencia entre las categorías a utilizar influirá significativamente en los resultados de nuestro modelo.
Mientras que una opción pensada fue utilizar textos de distintas áreas dentro del área de computación, otra opción fue elegir textos de computación, física y matemática. Observamos claramente que el problema en este segundo caso parecería ser más sencillo, pues los textos en principio diferirán más entre sí.
Además, tuvimos en cuenta que en una problemática real donde se plantea la idea de clasificar textos académicos, es frecuente que haya más de dos categorías distintas, por ello, optamos por una categorización multi-clase.
Para realizar un análisis más completo, tomamos la decisión de evaluar el problema en ambos casos, y comparar las diferencias entre ellos.

Datasets

Para el armado de datasets, obtuvimos varias publicaciones de arxiv. Utilizando la API a través de su librería en python, armamos datasets limpios con los textos que necesitamos para el fine-tuning. El código sobre la creación de estos archivos se puede ver en la Sección `Datasets` del notebook.
Por un lado, armamos un dataset que consiste en 1620 ejemplos de a lo sumo, 300 caracteres cada uno. El mismo tiene una tercera parte de los textos comprendiendo el área de inteligencia artificial, etiquetados como `AI`, otra tercera parte sobre lenguajes formales y la teoría de autómatas, etiquetados como `FL` y los restantes sobre algoritmos y complejidad, etiquetados como `DS`. Para conseguir suficientes datos, combinamos varias categorías y así formamos las clases finales con el tamaño suficiente. Nuevamente, las subcategorías de arxiv utilizadas para poder armar estas clases se pueden ver en el notebook.
Por otro lado, armamos un segundo dataset que contiene tres clases, ciencias de la computación, física y matemática, etiquetadas como `Computer Science`, `Physics` y `Mathematics`, respectivamente. Cada clase cuenta con 16 ejemplos de cada subcategoría (para ver más información sobre las clases usadas y sus subcategorías puede consultar este sitio).
Finalmente, obtuvimos un conjunto que tiene 1504 ejemplos, donde 640 son de la clase 'Computer Science', 352 de 'Physics' y 512 de 'Mathematics'.
Para ambos conjuntos de datos, tomamos la decisión de no balancear las clases. La motivación detrás de ello fue no perder datos que creímos útiles, pues al balancear nos quedaríamos con una cantidad fija de datos para cada clase, determinada por el cardinal de la clase más chica. Por ello, en el análisis de los resultados se utilizaron varias medidas para entender correctamente lo que ocurre. Sin embargo, en la creación del dataset de áreas de computación, había suficientes datos como para que igualmente el conjunto quede balanceado. Las proporciones obtenidas se detallan en los Cuadros 1 y 2.

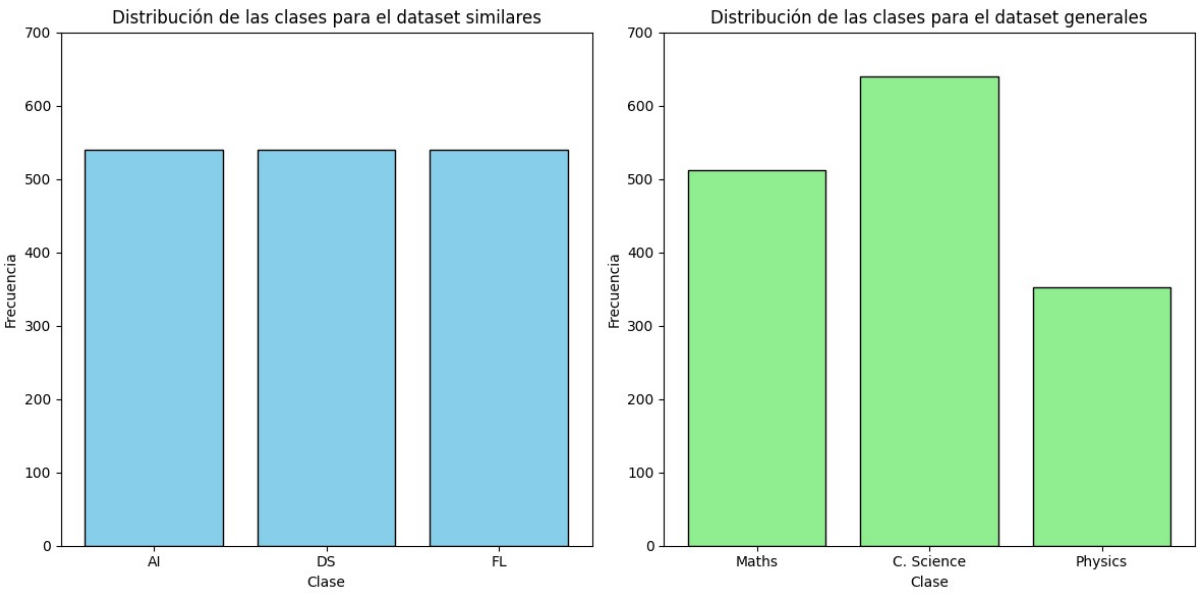

Figura 1: Gráfico de barras sobre la proporción de datos por clase para cada dataset.

Problema de las clases similares

En este problema, utilizamos el primer dataset discutido en la sección anterior, donde las clases fueron IA, Lenguajes y Complejidad algorítmica. A partir de este conjunto seleccionamos un 70 % para entrenamiento, un 15 % evaluación y el resto para test, manteniendo la proporción original de clases en cada subconjunto. En su análisis, se entrenaron modelos variando el parámetro de learning rate, y fijando los parámetros de regularización (weight decay), batch size, y número de épocas de entrenamiento. Luego, se eligió el mejor de ellos teniendo en cuenta su error en el conjunto de validación, y fue utilizado para probar diferentes ejemplos e intentar entender lo aprendido.

Learning Rate

Consideramos entrenar modelos variando la taza de aprendizaje para ver en qué medida nos afectaba la red, y qué valor nos daría un resultado más eficiente. Al estar realizando fine tuning, entendemos que el proceso que queremos realizarle a nuestro modelo tiene que ser sutil para no perder información que aprendió anteriormente. Por eso, para este tipo de procedimientos se suelen utilizar valores pequeños de learning rate.
Planteamos los distintos modelos cambiando la potencia de diez de dos en dos, desde 5×10^{-2} hasta 5×10^{-6}, esperando encontrar que un valor grande como lo es el primero no funcione tan bien como los restantes.
Además, exploramos dos estrategias de programación de la tasa de aprendizaje (Learning Rate Scheduler). Por un lado, el lineal, en el cual se realiza un decaimiento lineal hasta casi 0 al finalizar el entrenamiento de la tasa de aprendizaje, comenzando con el valor definido. Por otro lado, el constante (utilizado en clase) donde simplemente se mantiene fijo el parámetro.

- **Learning rate** 5×10^{-2}

  Aquí, al tener un *learning rate* alto, la red realiza una búsqueda más agresiva que

en los casos siguientes, haciendo actualizaciones significativas. Sin embargo, desde la base hay una gran diferencia entre las dos estrategias utilizadas.
En ambos modelos, se observa una disminución marcada del *training loss* entre la primera y la segunda época. Tras la primer época el modelo aún se encuentra lejos de un mínimo, en una región del espacio de parámetros donde los gradientes tienen una magnitud elevada. Combinado con el *learning rate* utilizado, esto provoca que para la segunda época se produzca una reducción pronunciada en el *training loss*.
Observamos por otro lado, que en el modelo lineal, en las primeras épocas, varía bastante el accuracy. Creemos que simplemente en la primer época el modelo acertó ejemplos por casualidad, y al estar realizando estos cambios abruptos en los pesos para mejorar la pérdida, la red puede fluctuar y predecir de manera aleatoria. A partir de la época 6, observamos que comienza a estabilizarse y mejorar.
Al tener una escala tan grande en los gráficos, no se llega a apreciar la diferencia entre las dos figuras. Por un lado, en el caso del LR Lineal, el modelo comienza con un 1,17 de *validation loss*, y logra bajarlo a un 0,69 durante el entrenamiento. En cambio, el modelo constante de la Figura 2 comienza con un 3,1 de *validation loss*, y solo logra disminuirlo a 0,89 en las primeras épocas, terminando antes por el *early stopping*.
Lo que creemos que está ocurriendo en ese segundo caso es que, al tener una taza de aprendizaje más significativa durante todo el entrenamiento, los ajustes en los pesos de la red son demasiado bruscos y no le permiten acercarse a una respuesta mejor para los datos. Incluso pensamos que si entrenamos la primer red durante más tiempo va a lograr acercarse a un mejor resultado (mientras que el *learning rate* no llegue a 0), mientras que la segunda va a seguir comportándose erráticamente.
Sin embargo, aunque el modelo de la Figura 1 es mejor que el de la Figura 2, igualmente puede mejorarse, como veremos en los siguientes casos.

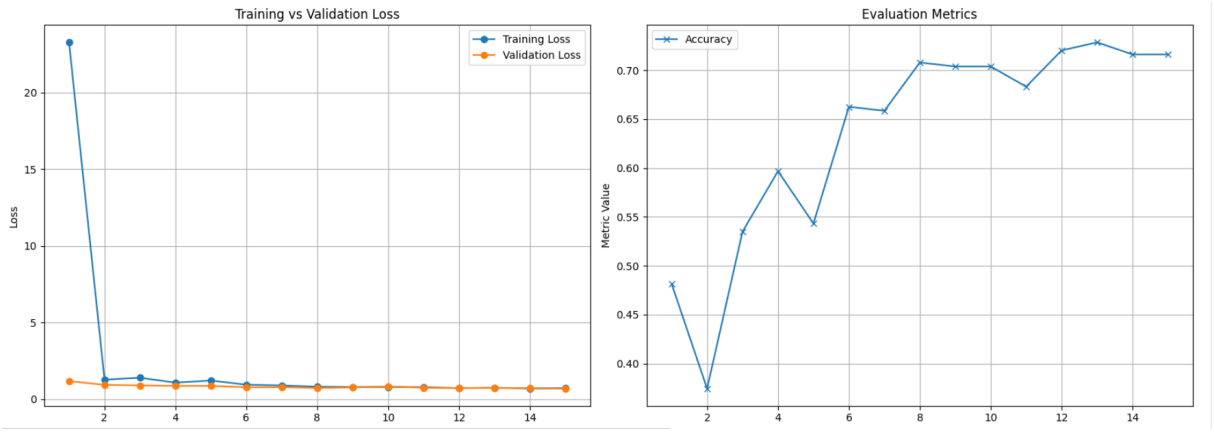

Figura 2: Evolución de las métricas para 5×10^{-2}; LR lineal.

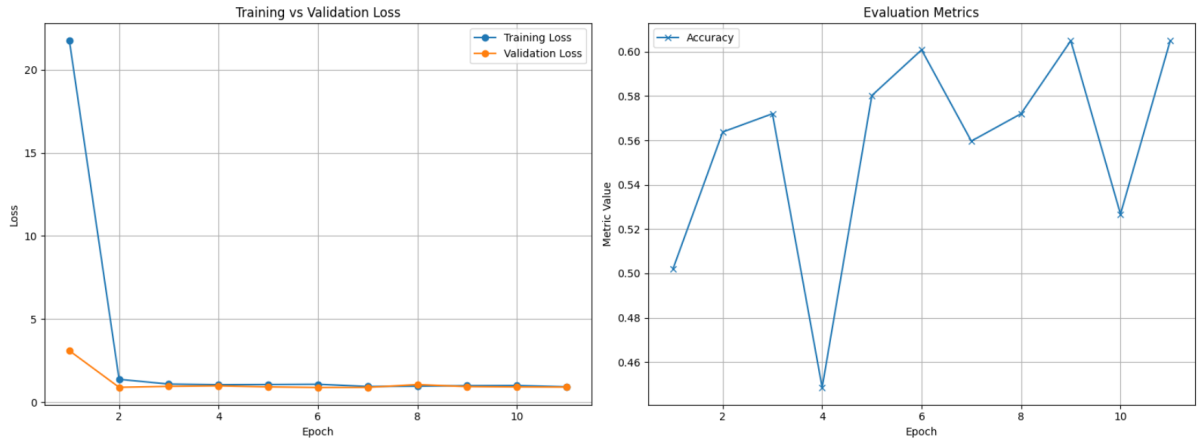

Figura 3: Evolución de las métricas para 5×10^{-2}; LR constante.

- **Learning rate** 5×10^{-4}

  Este es el valor con el cual vimos un mejor rendimiento en las métricas. Si comparamos ambas curvas, tanto en la Figura 3 como en la 4, vemos que, a medida que los *validation loss* comienzan a aumentar, los *training loss* siguen disminuyendo, lo cual nos da un indicio de que las redes comienzan a sobreajustar (*overfit*). Esto es, llega un momento a partir del cual lo que aprenden solamente les permite ajustarse a los datos de entrenamiento pero no mejora la generalización.
  De cualquier manera, en este contexto, la preocupación por el overfitting en el conjunto de entrenamiento es secundaria, pues estamos midiendo también con el conjunto de validación, por lo que podemos evitarlo. Lo que realmente nos importa es el rendimiento del modelo en este segundo conjunto, priorizamos elegir el modelo que mejor se desempeña con datos nuevos y no solo con los que ya vio. Por esta razón, entendemos que los entrenamientos que continúan más allá del punto donde el *validation loss* empieza a subir no son útiles para nuestros modelos. Es precisamente esta intuición la que impulsa el mecanismo de *early stopping*, deteniendo las ejecuciones antes de completar todas las épocas configuradas, asegurando que seleccionamos el modelo con la mejor capacidad de generalización.
  Si analizamos en particular las métricas obtenidas, vemos que el modelo entrenado con LR lineal logró un 0,60 de *validation loss* y aproximadamente 74 % de *accuracy*. El segundo modelo, consiguió un 0,59 de *validation loss* y aproximadamente un 75 % de *accuracy*.
  Si quisiéramos comparar los resultados de ambas estrategias con esta taza de aprendizaje, vemos que son prácticamente idénticos, a diferencia de lo ocurrido con un *learning rate* más alto. En este caso, lo que creemos que ocurre es que el *early stopping* está deteniendo el entrenamiento antes de que las diferencias entre ambos optimizadores se manifiesten de forma significativa. Es decir, el LR constante ya es lo suficientemente chico como para ser un valor más seguro o estable, y el LR adaptativo no tiene el tiempo necesario para diferenciarse con su estrategia de decaimiento.

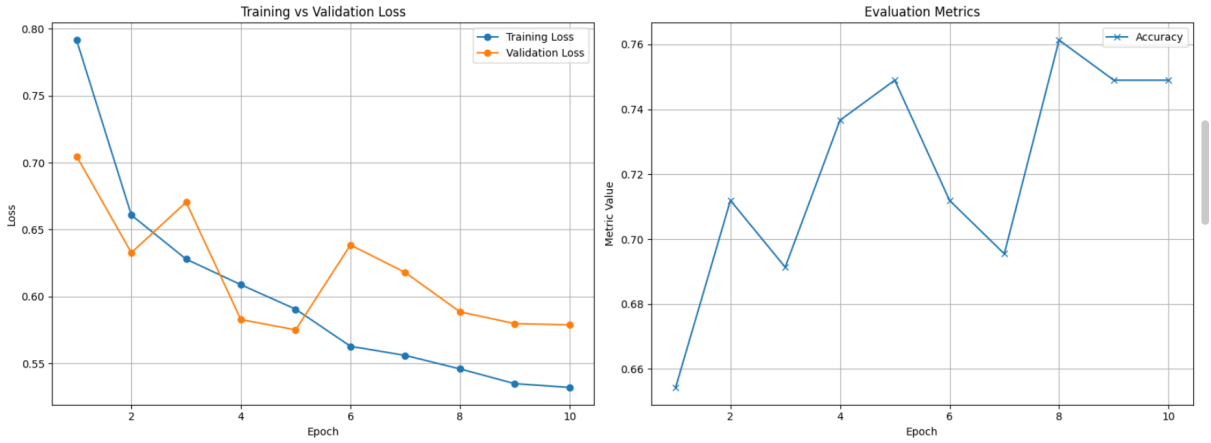

Figura 4: Evolución de las métricas para 5×10^{-4}; LR lineal.

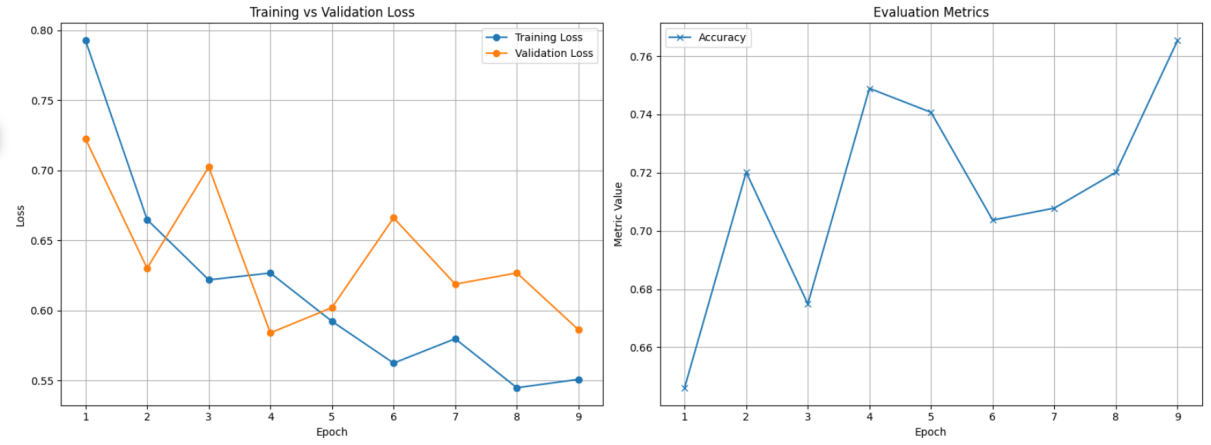

Figura 5: Evolución de las métricas para 5×10^{-4}; LR constante.

- **Learning rate** 5×10^{-6}

  En este último caso, al ser el valor muy bajo, consideramos que el entrenamiento podía necesitar ser más extenso para que el modelo pudiera aprender, por ello, lo entrenamos con 30 épocas. Luego, lo que observamos en los resultados fue que, aunque ambos completaron las 30 épocas (es decir, no se utilizó el procedimiento de early stopping), aprendieron más en las primeras. Esto se aprecia en la forma en la que tanto el *training loss* como el *validation loss* disminuyen con una pendiente cada vez menor.
  Claramente, al ver los resultados obtenidos con otro learning rate, entendemos que este no es el entrenamiento adecuado, ya que las actualizaciones son demasiado pequeñas como para salir de regiones no óptimas del espacio de parámetros.
  Si comparamos el entrenamiento con y sin scheduler lineal, vemos que al contrario de lo que ocurrió en el primer caso, funcionó mejor con un valor constante. Obtuvimos *accuracys* similares pero un *loss* menor en el caso constante. En particular, nuestra intuición era que un learning rate adaptativo siempre sería mejor. Sin embargo, en la interpretación de estos resultados entendemos que la principal ventaja de los LR

adaptativos era permitir pasos iniciales más grandes para acelerar la convergencia, una ventaja que se pierde cuando el punto de partida (LR inicial) ya es demasiado conservador. En cambio, un LR constante pero bajo, garantiza un progreso incremental y constante.

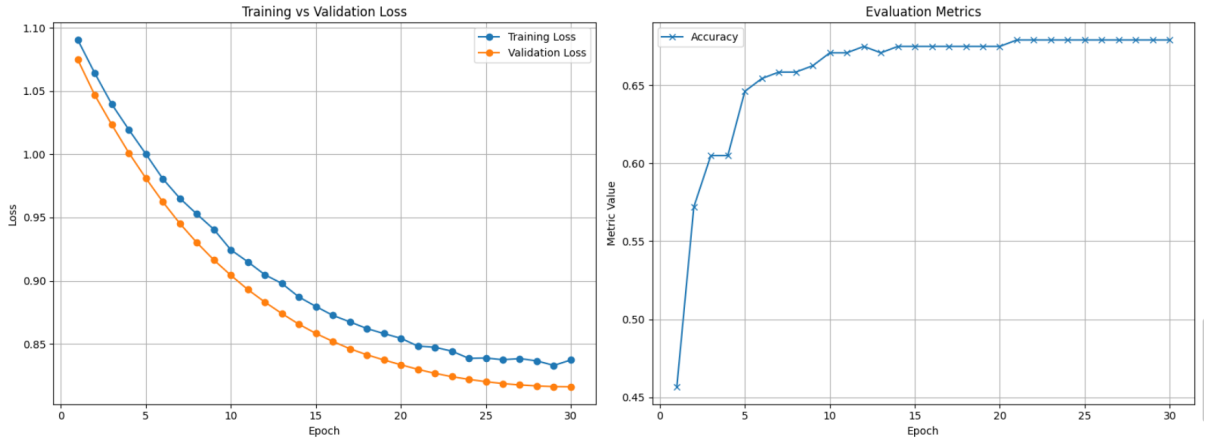

Figura 6: Evolución de las métricas para 5×10^{-6}; LR lineal.

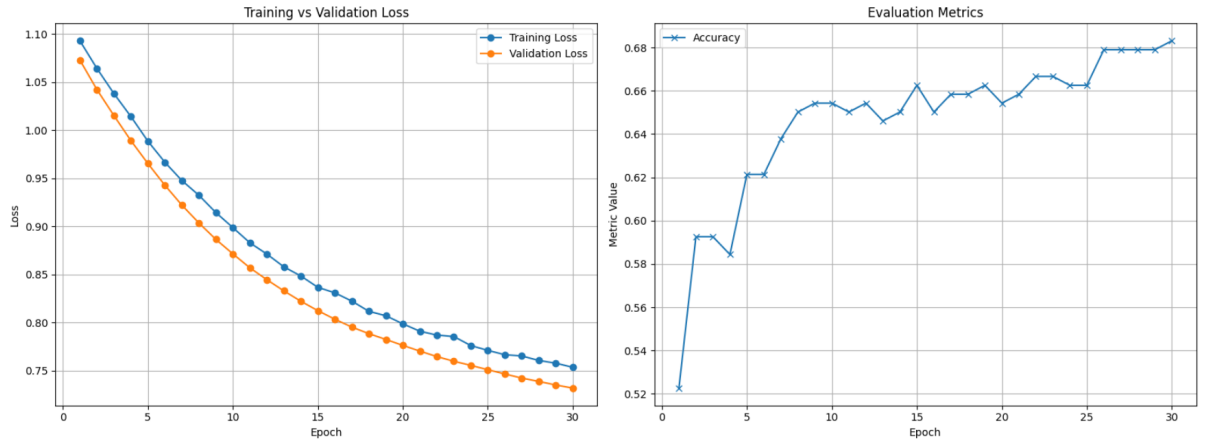

Figura 7: Evolución de las métricas para 5×10^{-6}; LR constante.

Modelo Elegido

Ahora, utilizando como modelo final el entrenado con una tasa de aprendizaje lineal de 5×10^{-4}, realizaremos distintas pruebas. Para ello, planteamos ejemplos que nos parecen interesantes para entender el problema, y evaluamos el funcionamiento de la red en el conjunto de test (no visto por el modelo hasta este momento).
En primer instancia, vimos que el *accuracy* en el conjunto de test fue de un 0,74. Este número nos indica la exactitud que debemos esperar sobre nuestro modelo cuando lo usemos en la realidad. Observamos además que este valor se acerca mucho al obtenido sobre el conjunto de evaluación, que fue de 0,75, por lo que está generalizando de manera efectiva a datos nuevos y no vistos. Era esperable también el hecho de que en el de evaluación funcione un poco mejor, pues hay un *bias inductivo* al haberlo utilizado para

mejorar el entrenamiento.
Por otro lado, presentamos algunos ejemplos y los resultados obtenidos sobre ellos.

Cuadro 1: Ejemplos de predicciones del modelo fine-tuneado.

Ejemplo de texto	**Predicción del Modelo**		**Etiqueta Verdadera**
"The perceptron training rule converges after a finite number of iterations to a hypothesis that perfectly classifies the training data."	AI: DS: FL:	0.04 0.79 0.17	AI
"In automata theory, the result of a nondeterministic automaton is defined to be the the union of the results of all possible executions."	AI: DS: FL:	0.00 0.31 0.69	FL
"A master recurrence describes the running time of a divide-and-conquer algorithm that divides a problem of size n into a subproblems."	AI: DS: FL:	0.00 0.99 0.01	DS
"The probabilistic grammar of the input string defines a series of state transitions within our learning model. We trained a recognizer to classify sequences of symbols, optimizing the production rules for an accurate parse of the linguistic data."	AI: DS: FL:	0.08 0.02 0.90	AI - FL
"The systematic approach that is used in software engineering is sometimes called a software process. A software process is a sequence of activities that leads to the production of a software product."	AI: DS: FL:	0.14 0.13 0.73	None
"The Imitation Game. I propose to consider the question, 'Can machines think?'. This should begin with definitions of the meaning of the terms 'machine' and 'think'."	AI: DS: FL:	0.04 0.11 0.84	AI

Para comenzar el testeo, evaluamos el modelo en 3 ejemplos, esperando que al ser ejemplos simples el modelo funcione correctamente. Esto de hecho ocurrió con los ejemplos para las clases DS y FL. Sin embargo, el ejemplo 1 (de IA) lo clasifica mal, y con una probabilidad casi completa para DS. Al revisar el dato, sacado de un libro de Inteligencia Artificial, observamos que en realidad está comentando sobre un algoritmo, su convergencia e iteraciones, por lo que tiene sentido que lo clasifique como DS. Creemos que es un buen ejemplo de la sobreposición de los datos, lo cual detallamos más adelante.
Además, evaluamos el funcionamiento de la red con casos mas ruidosos. Lo que se esperaría de un modelo ideal es que, ante un caso ambiguo, asignara una probabilidad similar a las clases relevantes, reflejando su incertidumbre.
Por un lado, vimos que al pasarle un texto que unía conceptos de AI y FL, fue completamente por el lado de FL con una gran probabilidad a favor. Luego, al intentar con un

texto sobre ingeniería del software (ejemplo 6 del cuadro), es decir que no pertenecía a ninguna de las clases, nuevamente decidió FL. Creemos por estos resultados que la red puede llegar a tener complicaciones con dicha clase. Esto quedaría respaldado por las métricas conseguidas sobre el conjunto de test. Con ellas, vimos que el modelo tiene un buen rendimiento general, pero su eficacia no es uniforme en todas las clases, y en particular, en FL obtiene el peor desempeño. En la clase DS obtiene mejor recall y precision, lo que nos muestra que la red logra diferenciar los ejemplos de DS de mejor manera, tanto para identificar correctamente sus instancias (alto recall) como para no confundirla con otras clases (alta precision).
Finalmente, probamos con textos más largos y más cortos, encontrando que al darle más palabras, obtiene un mayor contexto y clasifica mejor. Estos ejemplos pueden observarse en el notebook.

Problema de las clases generales

En este problema, utilizamos el segundo dataset planteado, donde las clases fueron Matemática, Física y Computación. Para poder hacer un análisis desde otra perspectiva que el problema anterior, decidimos evaluar cómo variaba el modelo al ser entrenado con más o menos datos. Para ello, entrenamos 3 modelos. Antes de entrenarlos dividimos el dataset en 70 % para datos de entrenamiento, 15 % para validación y el 15 % restante para test. Notar que los conjuntos de evaluación y test son los mismos para los tres modelos, lo cual permite que la comparación de rendimiento entre ellos sea justa.
Los tres modelos fueron entrenados usando distintas cantidades del conjunto de entrenamiento: todo el conjunto para el primer caso, un 50 % para el segundo y para el tercero un 25 %. A continuación, planteamos ciertos gráficos que son de ayuda para visualizar los resultados obtenidos.

- **Training dataset size:** 100 %

  En la gráfica de la figura 8 se observa que, al comienzo del entrenamiento, el accuracy es de aproximadamente un 65 %. Sin embargo, esto no implica un rendimiento "aceptable" del modelo en ese momento. Al analizar las métricas por clase, se puede concluir que predice correctamente varios ejemplos de las clases 'Computer science' y 'Mathematics', pero obtiene un peor desempeño sobre la clase restante.
  Se nota además, para las primeras épocas, un recall muy bajo y una precisión muy alta en la clase 'Physics'. Esto nos indica que al predecir esta clase generalmente acierta, a excepción de algunos pocos falsos positivos, pero hay muchos falsos negativos, no suele predecir 'Physics' en muchos casos que sí debería.
  Esta combinación, sugiere que el modelo predice la clase solo cuando está muy seguro, por lo tanto, sigue sin reconocer una gran cantidad de ejemplos reales pertenecientes a la misma. Puede deberse a una necesidad del modelo de ajustar más los pesos para aprender sobre la clase por ser la minoritaria. En otras palabras, le resulta más difícil en un comienzo generalizar sobre esta categoría a comparación de las restantes debido a la escasez de datos.
  Observamos además, que al finalizar el entrenamiento en la época 16, el F1 score de la clase 'Physics' (la media armónica entre las métricas recall y precision) queda por debajo de los demás. Este hecho pone en evidencia que al modelo le está costando más aprender sobre la clase 'Physics', lo esperado por su cardinal. Sin embargo,

en esta época conseguimos muy buenos resultados a comparación del principio del entrenamiento, concluyendo que la red pudo aprender correctamente con los datos disponibles.

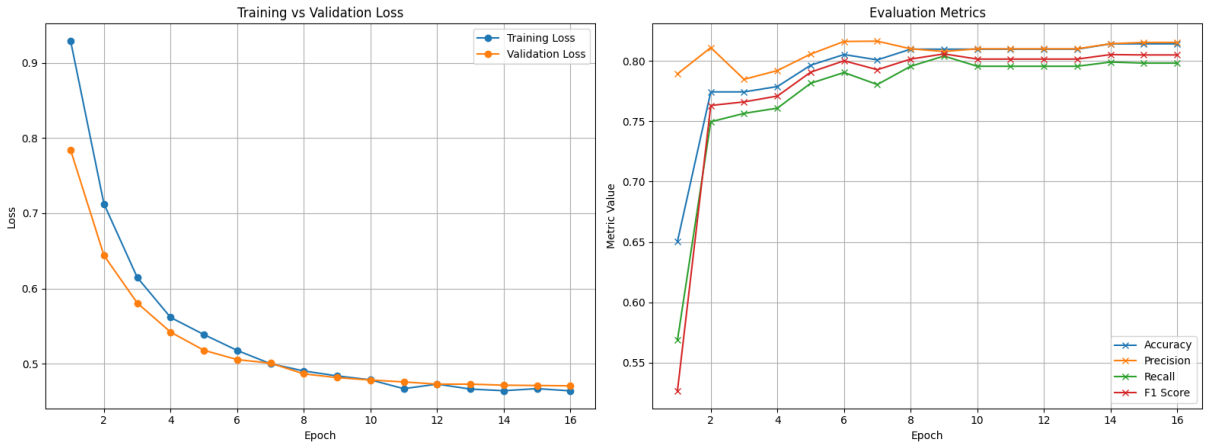

Figura 8: Evolución de las métricas, las métricas de la derecha, sin contar el accuracy, fueron calculadas con average='macro'.

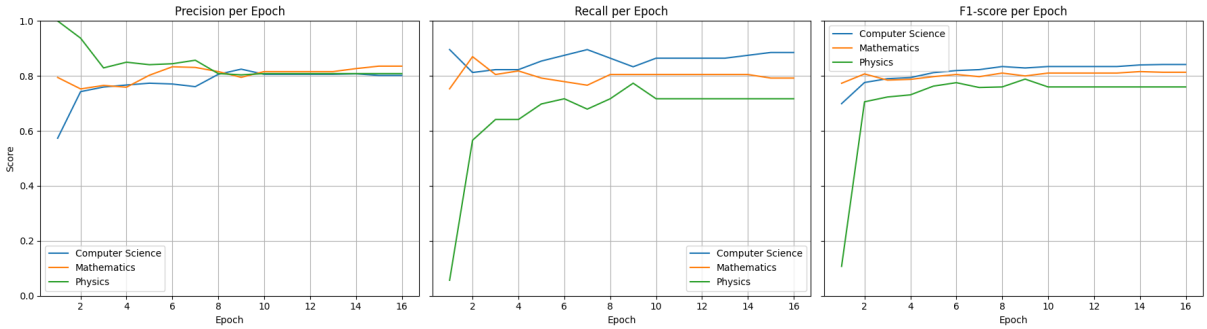

Figura 9: Evolución de las métricas calculadas por clase.

- **Training dataset size:** 50 %

  En este caso, al igual que con el 100 % de los datos, las curvas de pérdida descienden y se estabilizan. Sin embargo, en la época final, el modelo presenta un mayor error de evaluación y un menor accuracy, lo cual es razonable al usar menos datos.
  Según las métricas por clase, vemos que tenemos un rendimiento similar al modelo anterior para las clases 'Computer Science' y 'Mathematics'. Sin embargo, le cuesta incluso aún más reconocer los ejemplos de la clase 'Physics', pues tiene todavía menos ejemplos de la misma. Notar para este caso, que en la época 1 y 2 tanto su precision como recall son 0, esto quiere decir que el modelo hasta ese momento no había podido reconocer ningún ejemplo de la clase. Vemos que necesitó más épocas con respecto al modelo que usó el 100 % del conjunto de entrenamiento para únicamente poder empezar a ver el problema por completo.
  Finalmente, vemos en la gráfica que durante todo el entrenamiento (las 16 épocas que planteamos nosotros), la red mejoró sus resultados. Sería interesante entonces

observar qué ocurriría si lo entrenamos durante más tiempo, a qué resultados puede llegar.

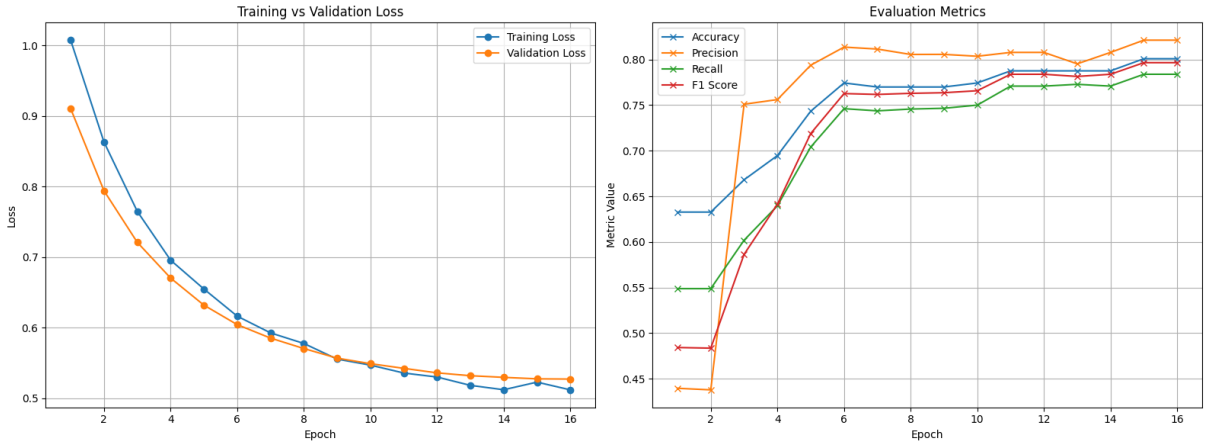

Figura 10: Evolución de las métricas, las métricas de la derecha menos el accuracy, fueron calculadas con average=‘macro’.

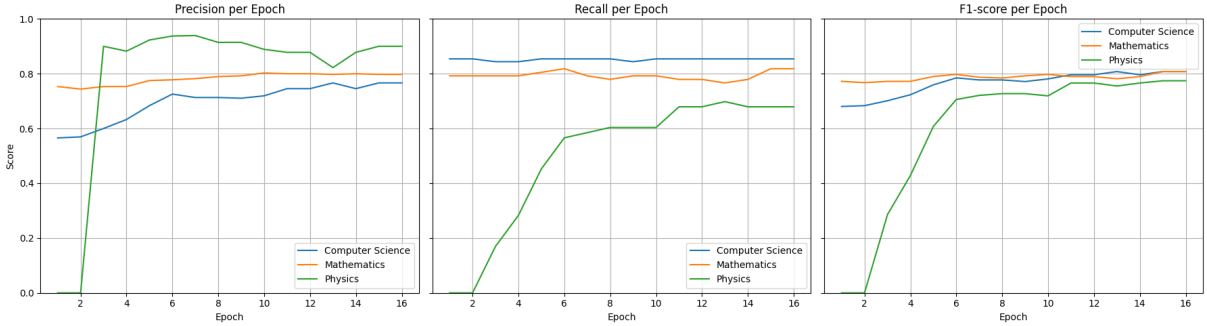

Figura 11: Evolución de las métricas calculadas por clase.

- **Training dataset size:** 25 %

  En este último caso, vemos que los valores de error fueron más altos, además el accuracy disminuyó, al igual que las métricas promediadas (precision, recall y f1), especialmente el recall.
  Si nos enfocamos en la clase ‘Physics’, se ve una dificultad del modelo para generalizar correctamente sobre ella. Esto se confirma al ver las métricas por clase: el recall para esta clase sólo alcanza el 43 %, lo que significa que menos de la mitad de las instancias reales de esta clase son en efecto detectadas por el modelo.
  Por lo tanto, aunque el rendimiento global pueda parecer aceptable, creemos este caso nos muestra por qué es importante entender también las métricas por clase.

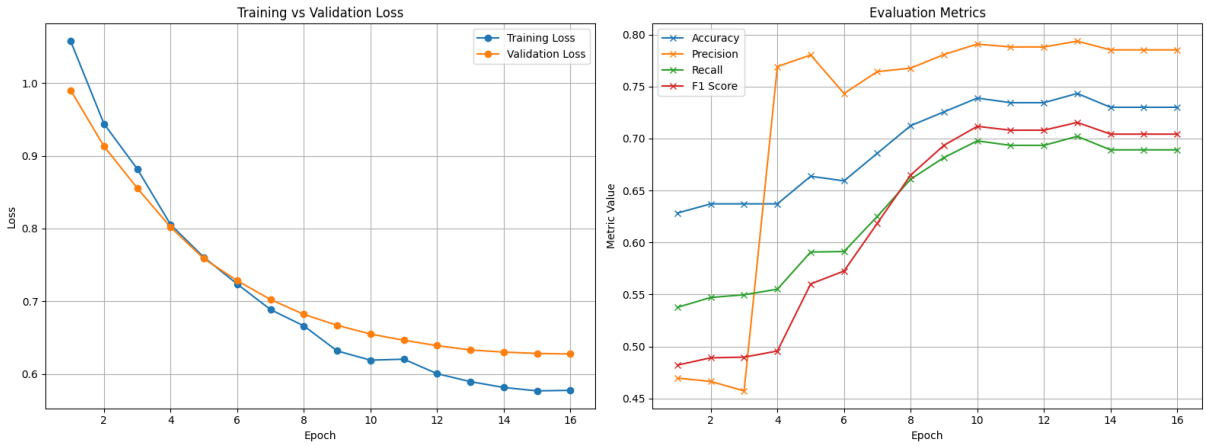

Figura 12: Evolución de las métricas, las métricas de la derecha menos el accuracy, fueron calculadas con average=‘macro’.

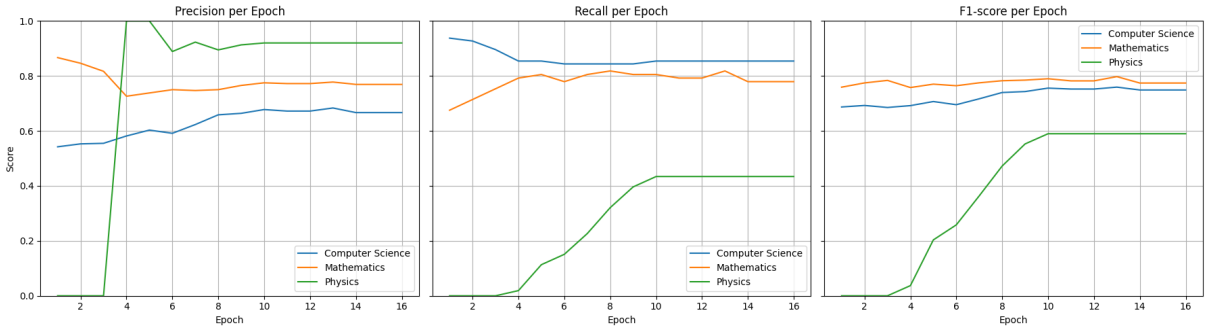

Figura 13: Evolución de las métricas calculadas por clase.

Conclusiones y observaciones

En la comparación de los 3 modelos, encontramos que mientras mayor es el tamaño del training set, menor son tanto el evaluation loss como training loss. Este hecho nos indica que todos los datos que le estamos agregando al aumentar los conjuntos, le aportan información nueva al modelo. Creemos que nuestra tarea es ‘grande’, y nunca llegaríamos a conseguir la cantidad suficiente de información como para entenderla por completo. Si lo pensamos, siempre va a haber términos nuevos o textos particulares para los cuales, sin más contexto, no se podrá decidir correctamente una categoría. Por ello, mientras más información agregamos, más estamos aprendiendo del problema, debido a su complejidad. Mirando las métricas de las redes sobre el conjunto de evaluación, consideramos que la mejor estrategia sería quedarnos con aquel entrenado con el 100 % de los datos, pues nos están aportando información relevante. Cabe destacar que el conjunto de evaluación utilizado es el mismo en todos los casos, lo que nos permite una comparación justa.

Modelo Elegido

Utilizaremos como modelo final el entrenado con el 100 % de los datos, y con él, probaremos ejemplos diversos para entender el problema. Al igual que en el caso anterior, evaluamos el funcionamiento de la red en el conjunto de test, independiente del entrenamiento.
Primero presentamos los resultados de las predicciones sobre el conjunto de test:

Metric	**Value**
Accuracy	0,810
Precision	0,813
Recall	0,810
F1-Score	0,809

Cuadro 2: Metricas promedio (Macro).

Class	**Precision**	**Recall**	**F1-Score**
Computer Science	0,781	0,854	0,816
Mathematics	0,818	0,818	0,818
Physics	0,864	0,717	0,784

Cuadro 3: Metricas por clase.

Las métricas obtenidas en el conjunto de test son las esperadas si consideramos que usamos siempre conjuntos representativos y el de test independiente a los demás. Nuevamente, las métricas son similares a las obtenidas en el conjunto de evaluación, siendo por ejemplo apenas menor el accuracy, pues el de evaluación tiene una ligera estima optimista al haberse visto durante el aprendizaje.

Presentamos algunos ejemplos concretos en el cuadro 6. Allí se observa que el modelo clasifica con seguridad los cuatro ejemplos presentados, incluso cuando los textos son reducidos, como en el último caso, donde la clase predicha es la minoritaria en el conjunto de entrenamiento. Esto refleja los buenos resultados obtenidos en el conjunto de test.

Cuadro 4: Ejemplos de predicciones del modelo.

Ejemplo de texto	**Predicción del Modelo**	**Etiqueta Verdadera**
"KuaiLive: A Real-time Interactive Dataset for Live Streaming Recommendation Live streaming platforms have become a dominant form of online content consumption"	CS: 0.91 MATH: 0.02 PHY: 0.05	CS
"Coefficient Identification Problem with Integral Overdetermination Condition for Diffusion Equations In this paper, we investigate a nonlinear inverse problem aimed at recovering a coefficient $a(t, x)$, dependent on both time and a subset of spatial variables, in a diffusion equation \(u_t - "	CS: 0.09 MATH: 0.82 PHY: 0.08	MATH
"On operation and control of CW magnetrons for superconducting accelerators CW magnetrons, developed for industrial RF heaters, were suggested to feed RF cavities of superconducting accelerators due to higher efficiency and lower cost of RF power than provide traditionally used klystrons, IOTs or"	CS: 0.05 MATH: 0.01 PHY: 0.92	PHY
"On operation and control of CW magnetrons"	CS: 0.04 MATH: 0.06 PHY: 0.89	PHY

Comparación de ambos problemas

Como era de esperar, fue más sencillo obtener mejores rendimientos para el problema de las clases generales. Lo que nos llamó la atención, es que esto se logró incluso con menos datos de entrenamiento y con clases desbalanceadas.
Nos parece interesante plantear la diferencia de la dificultad que encuentra el modelo de las clases similares con la de otros problemas vistos en clase. En nuestra opinión, no estamos ante un problema 'fácil' que por algún motivo se le dificulte al modelo, como fue el caso de diagonales para árboles de decisión. Por lo contrario, creemos que el modelo llega a resultados probablemente muy cerca de lo mejor posible.
Nuestra hipótesis se basa en que la dificultad de la tarea radica en el conjunto de datos y etiquetas, pues el dataset contiene ruido, porque la clasificación en clases tan parecidas siempre tendrá ambigüedades. Vimos en el primer dataset ocurrencias de ejemplos duplicados en el conjunto de entrenamiento pero etiquetados con clases distintas, e incluso encontramos muchos textos para los que es realmente difícil o no existe la posibilidad de entender a qué clase deberían pertenecer, o bien que están clasificados en áreas en las cuales nosotros no los hubiéramos clasificado.
En este punto, consideramos importante observar que la diferencia entre los dos proble-

mas radica únicamente en los datos que se le proporcionaron al modelo, pues la tarea subyacente es la misma: la clasificación de texto en áreas académicas. Teniendo esto en mente, entendemos que la dificultad en el primer caso estuvo sobre todo en la similitud de las clases, es decir el solapamiento (*overlapping*) de las mismas.
En cambio, en el problema de las clases distintas, generalmente habrá menos ruido en los datos, en efecto, es menos común encontrar textos que tengan etiquetas tanto de subcategorías de física como computación, antes que textos que tengan etiquetas de dos o más categorías de la misma área científica.
El hecho de que los datos para el problema de las clases similares presenten mayor ruido, dificulta el aprendizaje y produce peores resultados. Sin embargo, consideramos que obtuvo buen desempeño, justamente porque el mejor que podríamos encontrar en este problema no es un 100 % de exactitud. Creemos que si tuviéramos un clasificador de Bayes, nos daría un error mínimo considerable, y sobre todo mayor al del problema de las clases generales.

Conclusiones

Nuestra hipótesis principal es que, como el modelo ya está altamente optimizado, un entrenamiento prolongado no fue necesario. Debemos tener en cuenta que distil-Bert tiene 66 millones de parámetros y fue pre-entrenado con cientos de miles de millones de palabras para aprender patrones del lenguaje muy complejos, por lo que adquirió buen conocimiento general.
Dado este contexto, realizar una técnica de transfer learning, como lo es fine-tuning, con menos de 2000 datos, una taza de aprendizaje relativamente baja y solamente 10-15 épocas de entrenamiento, logró resultados que creemos muy buenos. Pensamos que el modelo aprendió correctamente como adaptar su conocimiento sobre el mundo en general al problema específico que le planteamos, permitiendo que un entrenamiento corto y sencillo sea suficiente y eficiente para obtener buenos resultados.

Bibliografía

1. *https://arxiv.org/category_taxonomy*
2. *https://www.evidentlyai.com/classification-metrics/multi-class-metrics*
3. *https://medium.com/@heyamit10/fine-tuning-neural-network-a-practical-guide-8f59eec3ef75*