ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

DESARROLLO Y APLICACIÓN DE UN MODELO COMPUTACIONAL PARA EL PROCESAMIENTO DE LENGUAJE NATURAL

DETECCIÓN DE ERRORES ORTOGRÁFICOS EN TEXTOS OBTENIDOS POR RECONOCIMIENTO DE CARACTERES ÓPTICOS (OCR) MEDIANTE REPRESENTACIONES DE CODIFICADOR BIDIRECCIONAL DE TRANSFORMERS (BERT)

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN CIENCIAS DE LA COMPUTACIÓN

MICHAEL LEONARDO CHILÁN RIVERA

michal.chilan@epn.edu.ec

DIRECTOR: PhD. JOSAFÁ DE JESÚS AGUIAR PONTES

josafa.aguiar@epn.edu.ec

DMQ, febrero 2024

CERTIFICACIONES

Yo, MICHAEL LEONARDO CHILÁN RIVERA declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

MICHAEL LEONARDO CHILÁN RIVERA

Certifico que el presente trabajo de integración curricular fue desarrollado por MICHAEL LEONARDO CHILÁN RIVERA, bajo mi supervisión.

PhD. JOSAFÁ DE JESÚS AGUIAR PONTES

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

MICHAEL LEONARDO CHILÁN RIVERA

PhD. JOSAFÁ DE JESÚS AGUIAR PONTES

DEDICATORIA

Este trabajo se lo dedico a mi familia, en especial a mis padres Marlene Rivera y Vito Chilán que han sido parte fundamental para poder obtener mi título profesional, que pese a las dificultades transcurridas a lo largo de mi existencia me han comprendido y me han dado su apoyo incondicional.

AGRADECIMIENTO

Agradezco a la Escuela Politécnica Nacional ya que, a pesar de las dificultades, he logrado obtener mi título profesional y he llevado un crecimiento académico con la suficiente fortaleza para poder afrontar un futuro profesional de la mejor manera posible.

Extiendo un gran agradecimiento a mi tutor de carrera y director de tesis, Josafá Aguiar, ya que es quién me dirigió durante el proceso del desarrollo del trabajo de integración curricular y me dio las indicaciones para poder llevar a cabo el presente trabajo.

Agradezco a compañeros que he conocido a lo largo de la carrera con quienes ha sido muy grato compartir vivencias durante el transcurso de esta época de estudio.

Finalmente, me agradezco a mí mismo por presentar la fortaleza y las ganas de superarme y poder conseguir este logro.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente trabajo está enfocado en el procesamiento de lenguaje natural (NLP), abordando las tareas de procesamiento y alineación de un corpus paralelo identificando las etiquetas que son requeridas para aplicarlas en el desarrollo del modelo supervisado para la Detección de errores ortográficos obtenidos mediante una herramienta OCR.

Para cumplir con este enfoque, se aplica la técnica del Ajuste Fino para aprovechar el codificador de un modelo previamente entrenado de arquitectura Transformer, conocido como Codificador Bidireccional de Transformers BERT.

De esta manera se utiliza el modelo para procesar la información y contextualizar los datos que recibe, formando una arquitectura específica para el decodificador que está conformado con las capas requeridas para la estructuración de la capa clasificadora, la cual tiene como objetivo asignar las clases según corresponda a las palabras, clases que están catalogadas como correctas e incorrectas.

PALABRAS CLAVE: Corpus, OCR, REF, Fine Tuning, Ajuste Fino, BERT, Transformers, Detección, Clasificación, Alineación

ABSTRACT

This work focuses on natural language processing (NLP) tasks, specifically addressing the processing and alignment of a parallel corpus by identifying the necessary tags for applications in the development of a supervised model for detecting spelling errors obtained through an OCT tool.

To achieve this goal, the Fine-Tuning technique is applied to leverage the encoder of a pre-trained Transformer architecture model, known as Bidirectional Encoder of Transformers (BERT).

In this manner, the model is utilized to process information and contextualize the received data, forming a specific architecture for the decoder. The decoder is constructed with the required layers to structure the classification layer, which aims to assign classes accordingly to words, categorizing then as either correct or incorrect.

KEYWORDS: Corpus, OCR, REF, Fine Tuning, BERT, Transformers, Detection, Classification, Alignment.

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

El componente consiste en el manejo de un corpus paralelo, que consiste en dos textos con el número de oraciones y número de palabras similares, conformado por oraciones que tiene espacios entre palabras, espacios que son utilizados para delimitar el entendimiento de la escritura y lectura. El corpus paralelo es la data contextual a cuál se aplican técnicas de tokenización (conversión de texto a números) que permiten el desarrollo de un modelo para la detección de errores gramaticales a partir de un modelo BERT preentrenado. Está conformado por un texto referencial y un texto derivado, el cual es obtenido mediante un procesamiento de reconocimiento de caracteres ópticos (OCR). El texto OCR tiene fallos ortográficos inherentes al proceso, mientras que el texto referencial se supone no debe tener errores ortográficos.

Para la aplicación de técnicas de tokenización que implica tomar grupos de caracteres en secuencia que tengan una estructura contextualizada, el corpus pasa por un proceso de limpieza que consta de la aplicación de expresiones regulares, las cuales consisten en eliminar caracteres especiales (comas, puntos, etc.), espacios innecesarios y, debido a la aplicación de la herramienta OCR la cual genera oraciones donde cada palabra es un solo carácter y que cada oración contiene un solo carácter, se eliminen estas oraciones al no presentar el contexto suficiente para ingresarlas al modelo a entrenarse.

Posterior a la realización del preprocesamiento con respecto a la limpieza de la data textual del corpus paralelo, se aplica el algoritmo de alineación de palabras llamado Smith-Waterman el cual mediante un proceso de comparación de caracteres de manera bidireccional toma cada oración del corpus paralelo (dos a la vez) para encontrar diferencias entre las palabras de las oraciones, en el cual se detecta errores gramaticales. Estos errores se toman con respecto al texto OCR dando como resultado eliminación y agregación de palabras, sustitución y transposición de letras o números, además de un caso especial donde se eliminan o agregan espacios entre palabras.

Finalizado el proceso de alineación, se diseñan las entradas mediante la aplicación de la técnica de tokenización provista por el modelo preentrenado llamado Wordpiece la cual forma sufijos colocando los caracteres ## para indicar que forman parte de una palabra compuesta, tomando las oraciones del corpus paralelo para generar las etiquetas a predecir comparando el texto OCR con el texto referencial, siendo las etiquetas correcto e incorrecto.

Tras obtener las etiquetas se procede a formar la arquitectura para el modelo de detección de errores gramaticales. Mediante el proceso conocido como Ajuste Fino (Fine Tuning) que aprovecha un modelo entrenado para adaptarlo a una tarea distinta, se toma el modelo preentrenado y se agrega el clasificador por cada palabra además de la implementación de hiperparámetro como valores la tasa de aprendizaje, función de pérdida utilizada y formación de grupos de grupos o lotes de muestras.

Durante el entrenamiento, el modelo se evalúa con respecto a la pérdida que se obtiene en cada época al predecir las clases definidas. Al finalizar el entrenamiento, para la evaluación del modelo entrenado se debe considerar la precisión y la capacidad que tiene de clasificación, calculando las métricas f1 score, precisión, exactitud y exhaustividad (recall), además de la curva ROC para definir cuan bien el modelo aprende en el transcurso del entrenamiento.

Luego del cálculo de estas métricas con respecto al entrenamiento del modelo, este es comparado con el enfoque del modelo Baseline que consiste en aplicar un algoritmo que obtiene predicciones al azar tomando como parámetros la distribución porcentual entre las clases correctas e incorrectas.

1.1 Objetivo general

Desarrollar un modelo supervisado en base a un modelo BERT preentrenado. Empleándose dos textos procesados y alineados para la generación de etiquetas correspondientes, facilitando la detección de errores gramaticales en oraciones extraídas por una herramienta OCR. La comparación se la realizará con un texto Referencial que contiene oraciones gramaticalmente correctas, permitiendo la precisa identificación de los errores.

1.2 Objetivos específicos

1. Procesar el corpus paralelo mediante la aplicación de expresiones regulares para la limpieza y mejora de la calidad de las oraciones del texto OCR.
2. Alinear el corpus paralelo para formar las etiquetas que serán aplicadas en la comparación entre las etiquetas verdaderas con las predicciones del modelo.

3. Diseñar los parámetros de entrada que el modelo tomará para su entrenamiento y posterior evaluación.
4. Aplicar Ajuste Fino para diseñar la arquitectura del modelo a partir de un modelo preentrenado.
5. Evaluar métricas en base a los resultados del entrenamiento.
6. Comparar los resultados del entrenamiento con el enfoque del modelo Baseline y evaluar si el modelo entrenado es mejor en la detección de errores gramaticales que el azar.

1.3 Alcance

Se empezará con el procesamiento del corpus paralelo mediante la aplicación de expresiones regulares; proceso con el cual se detectará caracteres que no sean alfanuméricos (número o letras) y espacios adicionales (doble espacio, espacios al inicio de la oración).

Luego de realizar la limpieza del corpus, se procede a la aplicación del algoritmo de alineación Smith-Waterman, algoritmo con el cual se procederá a la detección de los distintos errores gramaticales (errores de inserción, eliminación, sustitución y trasposición de caracteres, espacios insertados o eliminados entre palabras) que serán considerados, en general, como errores gramaticales.

Utilizando la función de tokenización y el vocabulario que el modelo BERT preentrenado proporciona en su contenido, se procede con la formación de los vectores de índices, establecer tamaños válidos para cada oración y etiquetas correspondientes a cada palabra que serán las entradas del modelo durante el proceso de entrenamiento.

Finalmente, se evaluarán los resultados, donde se aplicarán las métricas: medida F1, precisión, exhaustividad y exactitud del entrenamiento del modelo; además se interpretará la curva ROC con respecto a cómo el modelo aprende durante su entrenamiento. Adicionalmente, también se comparará con el enfoque base line para indicar si el modelo es superior en predecir palabras correctas e incorrectas que el azar.

1.4 Marco teórico

1.4.1 OCR

Como lo indica [1], el Reconocimiento óptimo de Caracteres (OCR) se refiere a la técnica de convertir texto a un formato codificado por máquina, es decir digitalizar textos escritos ya sea en libros, textos mecanografiados o incluso imágenes para que sean legibles y modificables en cualquier aparato computarizado. Es útil realizar este procedimiento para obtener información mediante una herramienta OCR con el fin de evitarse la transcripción y ahorrarse tiempo el transcribir el texto a dispositivo para tener el recurso digitalizado.

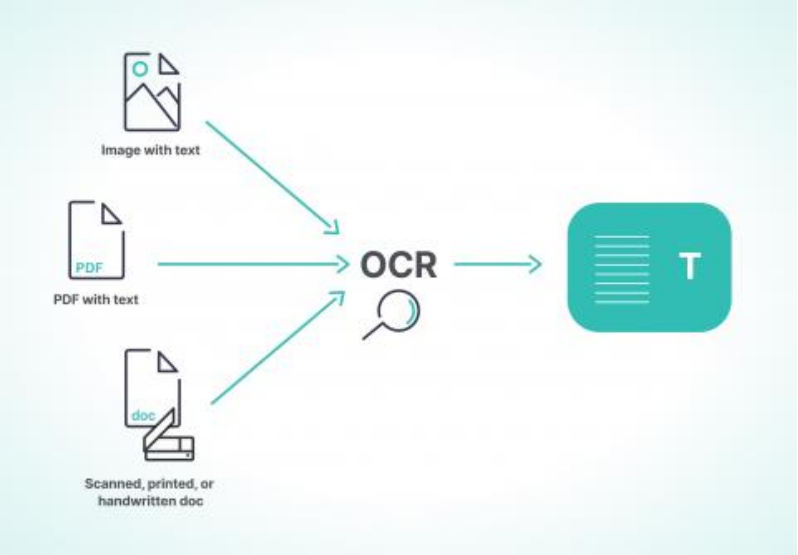


Figura 1. Ejemplo para usar una herramienta OCR [3]

Como lo muestra la Figura 1, el uso de una herramienta OCR abarca distintos tipos de textos contenidos, ayudando a una mejor digitalización y conversión del texto.

1.4.2 Problema

Expresado en [2] y [3], el proceso de aplicar la técnica OCR produce degradaciones con respecto a los caracteres que se detectan, ya sea porque la herramienta no es lo suficientemente sofisticada, el texto no es legible o simplemente la herramienta no pudo detectar el carácter correspondiente. Estas degradaciones producen que no se detecten los caracteres correctamente, exista transposición, sustitución entre caracteres e incluso se eliminen, ya sean caracteres individuales o incluso palabras completas.

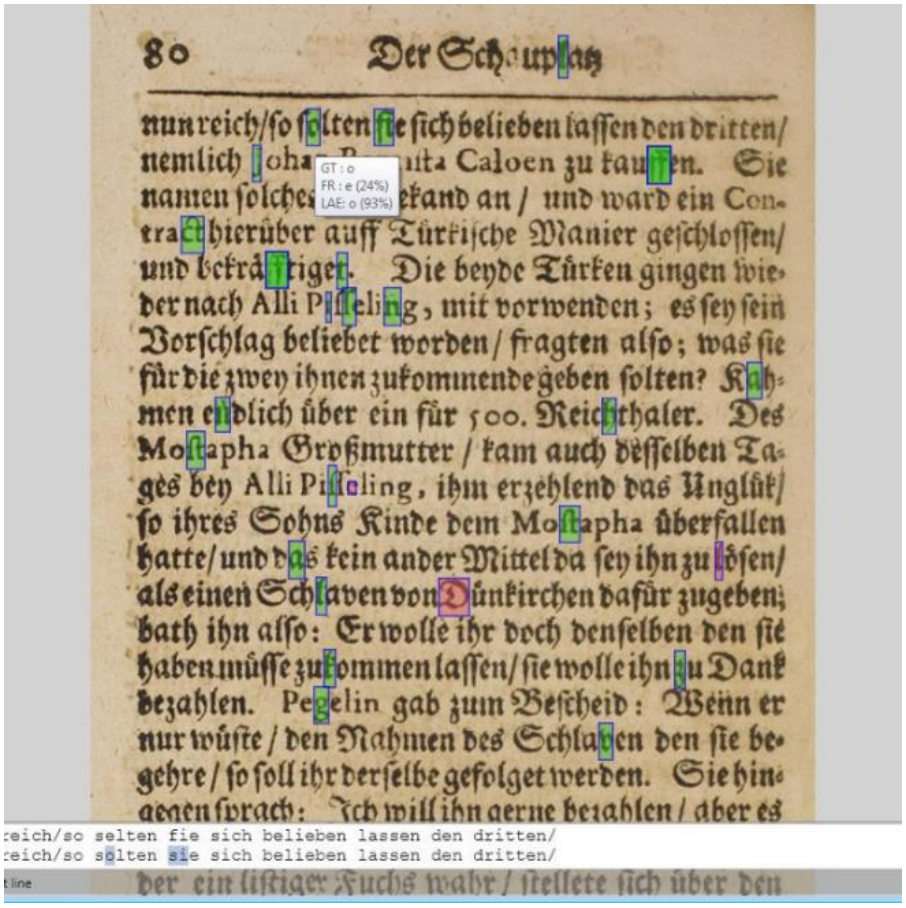


Figura 2. Ejemplo de degradación e ilegibilidad de un texto [4]

Como ilustra la Figura 2, la imagen presentada se muestra ilegible durante el proceso de digitalización por lo que pueden darse los problemas de caracteres perdidos, convertidos erróneamente o no detectados por la herramienta OCR.

1.4.3 Propuesta

La propuesta para el problema de caracteres malinterpretados por una herramienta OCR, es presentar un modelo que se encargará de detectar los errores ortográficos inherentes al proceso, donde se comprobará como actúa el modelo frente a los errores encontrados y cuál es la capacidad de detección de estos.

1.4.4 Modelo Transformer

Para aplicar la solución al problema de errores ortográficos que puede causar el proceso OCR, se requiere de la aplicación de modelos de aprendizaje. Aquí hace su entrada el modelo Transformer, el cual consiste en una red neuronal que aprende contexto mediante el seguimiento de la relación de datos secuenciales [5], que en este caso será aplicado a oraciones en las cuales se detectará aquellas palabras que contienen errores ortográficos.

La aplicación de un modelo Transformer presenta una mejor alternativa que las redes recurrentes [6], esto ya que el Transformer se basa en la atención y la sintaxis del texto a procesarse. La atención consiste en enfocarse en las partes importantes de información y mezclar las piezas sin importancia [7], indicando que se centra en las partes más relevantes de las entradas que recibe el modelo, en este caso serán las palabras que se requiere detectar si tienen errores ortográficos o no.

1.4.5 Representaciones de Codificador Bidireccional de Transformers (BERT)

Los modelos transformers son adecuados para este tipo de tareas ya que se centra en prestar atención en los aspectos más relevantes de la oración y coloca todo su enfoque en esta parte, a diferencia de word2vec que trabaja manera unidireccional únicamente, es decir que empieza desde la parte izquierda hacia la derecha, pero si la parte más importante se encuentra en el límite de la parte derecha, se pierde el contexto debido a que han pasado muchas características para llegar a la parte importante, perdiéndose en el camino el contexto o requiriéndose más contexto que puede encontrarse al principio de la oración.

Para solucionar este problema en relación con el contexto, aparece el modelo de Representaciones de Codificador Bidireccional de Transformes (BERT) [8] el cual realiza el mismo proceso que un Transformer que es una arquitectura base base, pero en este caso el contexto es rescatado debido a que BERT actúa tanto de izquierda a derecha y viceversa obteniendo un mayor enfoque al contexto obtenido a lo largo de la oración.

Como lo indica el libro “Getting Started with Google BERT: Build and train stat models using BERT” [9], BERT es un modelo de última generación publicado por Google el cual ha tenido un gran impacto referente al campo del Procesamiento de Lenguaje Natural (NLP, por sus siglas en inglés), el cual proporciona mejores resultados con respecto a tareas de

responder preguntas, generación de texto y clasificación de textos ya sea a nivel de oraciones o a nivel de palabras.

La importancia del porque el contexto es destacado en este modelo radica en el entendimiento de las palabras con respecto a la oración en la que se encuentra, esto debido a que la misma palabra puede tener distintos contextos con relación a otras oraciones, por ejemplo, al tener estas dos oraciones:

Oración A: The Burmese python, native to Southeast Asia, is known for its impressive size and distinctive coloration.

Oración B: Python is a programming language suitable for data analysis.

En estas oraciones se habla sobre *python* pero en dos contextos distintos, en la oración A se refiera a una especia de serpiente existente, y en la oración B se refiere al lenguaje de programación.

Con esto indicado, al comparar BERT con otros modelos como word2vec, el cual no se basa en el contexto de las palabras por lo cual se perdería la información referente al significado de la palabra en una oración perdiendo el contexto y por qué esta palabra está en la oración. Para esto BERT, comprenderá y entenderá el significado de p*ython* en cada oración, logrando identificar si se trata del tipo de serpiente o del lenguaje de programación. Por este motivo el contexto es muy relevante y por qué la importancia que tiene con respecto al uso de un modelo BERT.

1.4.6 Configuraciones BERT

Se lanzaron distintos modelos BERT que contienen cierta complejidad, esto debido a que al ser un modelo muy grande y que ocupa demasiados recursos computacionales. Esto causó que se requieran de distintos tipos de modelos, esto para adaptarlos a los recursos computacionales que requiera el usuario que utilice este modelo. Los más importantes son:

BERT-Base

Consta de 12 capas de codificación apiladas uno sobre la otra. Cada capa codificadora utiliza 12 cabezas de atención (attention-head). La red de avance en cada codificador presenta 768 unidades ocultas. Con estas configuraciones, el modelo BERT-Base será de tamaño 768. Un ejemplo de un codificador en la Figura 3.

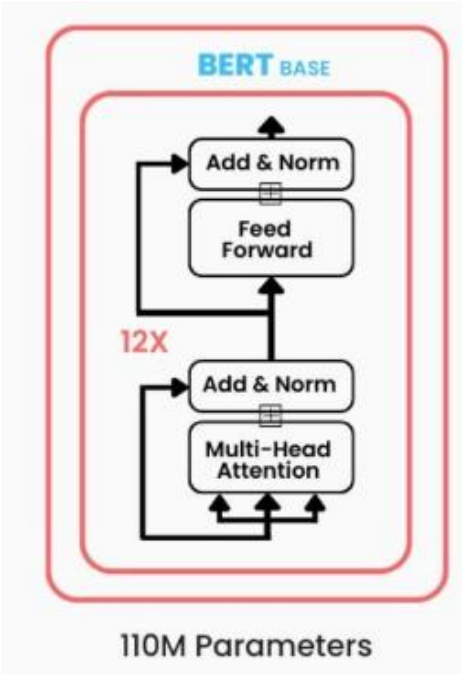


Figura 3. Representación de un codificador para BERT-Base [11]

Con los 12 cabezales de autoatención bidireccionales por cada del codificador permite que el modelo se centre en diferentes partes de la entrada de manera simultánea. Este modelo presenta 110 millones de parámetros, siendo más fácil de manejar computacionalmente que BERT-Grande [11].

Existen ciertos tipos de modelos BERT-Base, los cuales están orientados a distintos idiomas, para textos con o sin mayúsculas y la eliminación de marcadores de acento, esto para adaptarlo a idiomas como el inglés y el japonés [10].

BERT-Grande

Conformado por 24 capas de codificación (el doble que BERT-Base) apiladas entre sí. Los codificadores utilizan 16 mecanismos de atención en sus cabezales y la red de avance en el codificador contiene 1024 unidades ocultas, por lo que el tamaño de representación

BERT obtenida será de 1024. Un ejemplo de la conformación de un codificador Bert-Grande se muestra en la Figura 4.

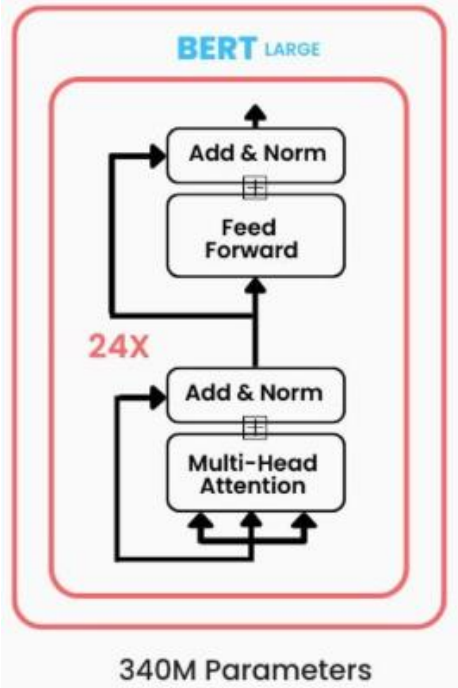


Figura 4. Representación de BERT-Grande [11]

Con un total de 24 capas de codificador, mejora sustancialmente la capacidad para la captura de relaciones más complejas dentro del texto. Este modelo presenta 340 millones de parámetros, lo que indica que el modelo es altamente expresivo en la manera de comprender estructuras de lenguaje [10].

1.4.7 Componentes fundamentales en la implementación de BERT

Para la utilización de un modelo BERT, ya sea en su versión Base o Grande, es un requisito el procesar la información de manera correcta para que el modelo identifique de manera correcta los datos que recibirá como entradas. Este proceso es fundamental para el modelo preentrenado, ya que espera recibir los datos en un formato especifico con respecto a la estructura usada durante su entrenamiento previo. Para ello, se deben aplicar técnicas de tokenización, con las cuales, los datos recibidos como entradas pasaran a un formato de identificadores (ids) que son reconocidos por el modelo.

1.4.8 Tokenización BERT

El modelo BERT al ser previamente entrenado, ya formó un vocabulario limitado, esto para evitar la sobrecarga de ids formados, con respecto a los datos que usó para su entrenamiento, por lo cual su formato servirá para tokenizar los nuevos datos que se requerirán para una tarea en específico.

El proceso de tokenización trata en dividir el texto en fracciones más pequeñas, las cuales se denominan tokens y tienen un valor asignado indicando su valor único con respecto al vocabulario que se requirió para su formación denominados ids [12].

Un ejemplo de cómo están ligados los tokens con su respectivo id, obtenido mediante la tokenización, es indicado en la Figura 5.

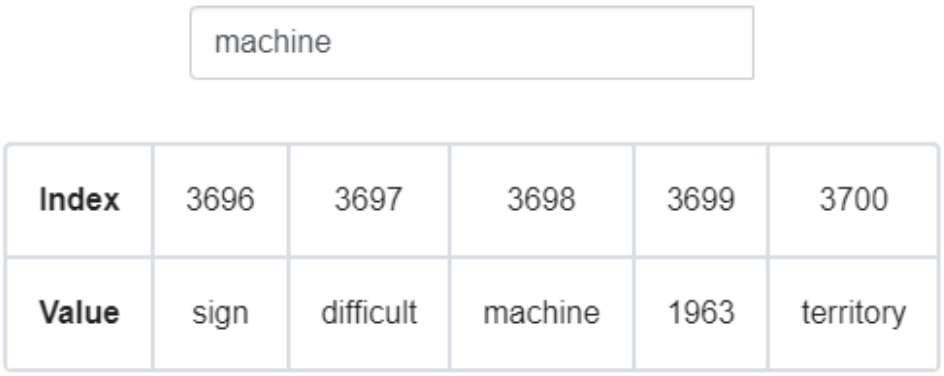


Figura 5. Ejemplo de tokenización [12]

Este ejemplo indica la relación de un token, en este caso machine, con su respectivo id. Se muestra además los ids (representaciones numéricas) cercanos y sus respectivos tokens.

Este podría ser un buen método con respecto a cómo transforma palabras en ids, pero existe un problema, como se muestra en la Figura 6, al ingresar una palabra compuesta ocurre un error con respecto a la asignación de un id para la palabra indicada,

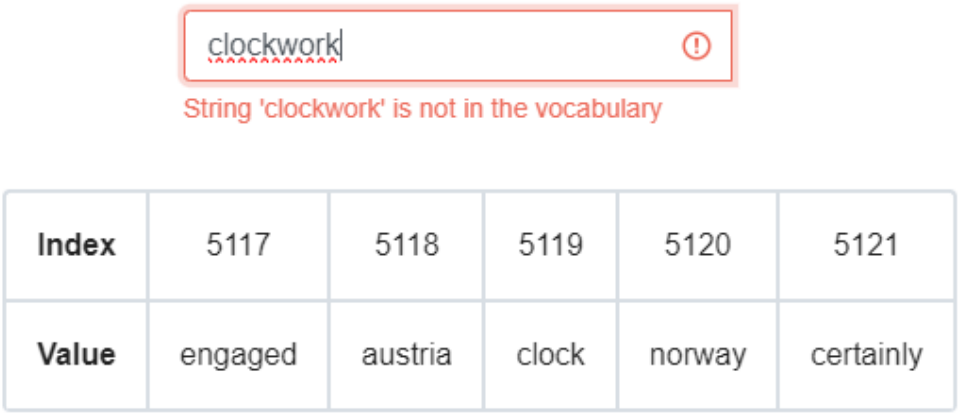


Figura 6. Error de tokenización palabra clockwork [12]

Este error indica que el vocabulario no contiene esta palabra, esto se debe a que el vocabulario es limitado y durante el proceso de su formación esta palabra no estuvo presente o no tuvo la suficiente frecuencia para ser considerada para pertenecer al vocabulario.

La opción de limitar el vocabulario es debido a que no se requiere usar todas las palabras habidas y por haber en un idioma o en varios, por esta razón se debe tomar solo aquellas palabras o partes de palabras que se repiten. En este ejemplo, *clockwor*k está compuesta por dos palabras, *clock* (Figura 7) y *work* (Figura 8), por lo que estas palabras por separado deberían existir en el vocabulario.

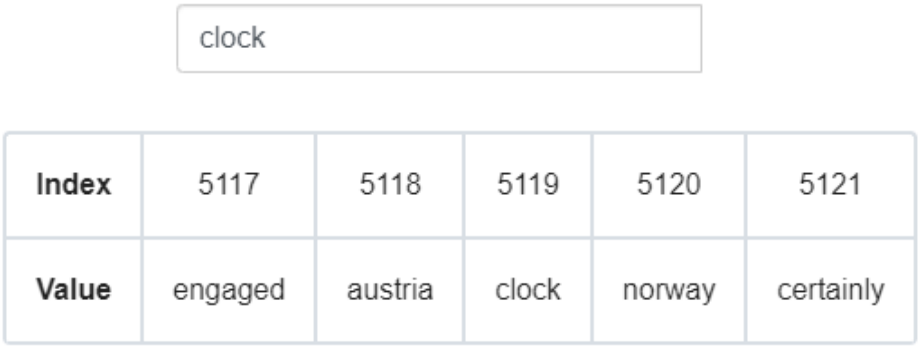
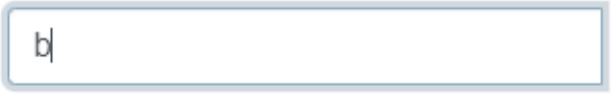


Figura 7. Existencia de la palabra clock en el vocabulario [12]

Index	2145	2146	2147	2148	2149
Value	still	long	work	south	us

Figura 8. Existencia de la palabra work en el vocabulario [12]

Para solventar este tipo de errores, el tokenizador de BERT tiene implementada la técnica de tokenización Wordpiece [13]. Esta técnica consiste en dividir cada palabra en conjuntos de caracteres dependiendo de la frecuencia de aparición en el corpus, tomando como conjuntos principales a los caracteres y números de manera individual tomando los primeros lugares en el vocabulario dependiendo de la implementación que use el tokenizador (Figura 9).



Index	1036	1037	1038	1039	1040
Value	`	a	b	c	d

Figura 9. Ids por nivel de caracteres [12]

A partir de esta tokenización a nivel de caracteres es donde se irán formando, por nivel de frecuencia, los grupos de caracteres que más presencia tuvieron a lo largo del corpus, colocándolos uno tras otro.

Ahora, a diferencia de otra técnica de tokenización como lo es BPE que consiste en realizar esta separación de conjuntos de caracteres frecuentes donde al terminar une las palabras

compuestas para formar una sola, Wordpiece no realiza esta juntara de los conjuntos de caracteres que forman la palabra compuesta. En este caso Wordpiece asigna dos caracteres numerales (##) indicando que ese conjunto de caracteres es la continuación de otras palabras que son denominados sufijos como lo indica la Figura 10.

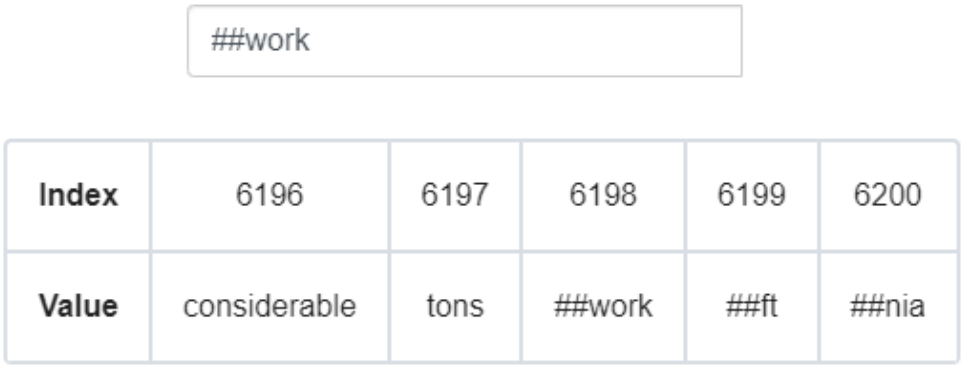


Figura 10. Representación de palabras sufijos [12]

Como se observa no es el único sufijo existente, sino que existen algunas palabras compuestas o que no han sido lo suficientemente frecuentes que tuvieron que ser divididas.

En la Figura 11 se indica como quedaría la tokenización final al aplicar la técnica Wordpiece.

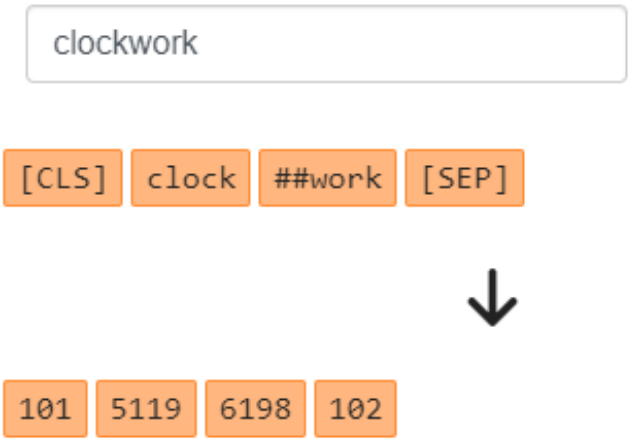


Figura 11. Tokenización Wordpiece [12]

De esta manera al usar un modelo preentrenado BERT que contiene su propio vocabulario, si en un nuevo corpus hay palabras nuevas, compuestas o que por inherencia de algún proceso OCR se cambien, agreguen o eliminen caracteres, siempre se tendrá una representación para realizar la tokenización. La Figura 12 ejemplifica este caso.

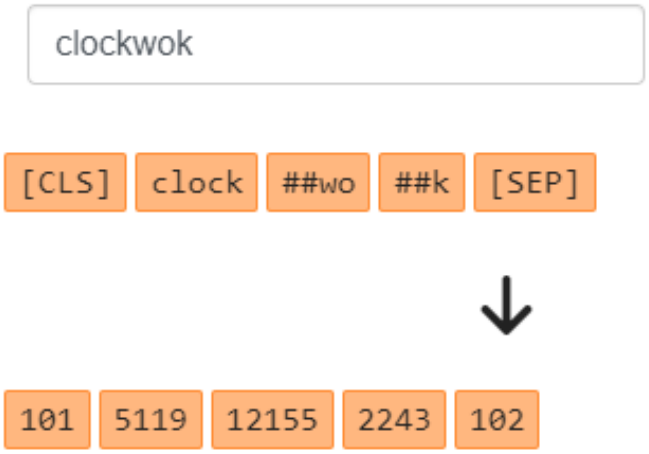


Figura 12. Tokenización palabras inconsistentes [12]

1.4.9 Incrustaciones (Embeddings) BERT

Como se indicó anteriormente, BERT es popular debido a que considera el contexto de las palabras y aplica un enfoque bidireccional a diferencia de word2vec que lo hace unidireccional. El contexto obtenido es gracias a los embeddings, que consisten en un vector que presenta números reales indicando la información semántica y contextual de cada token [14] y [15]. Un ejemplo está en la Figura 13. La cual indica en una representación vectorial de tamaño 8 como se puede representar el género de películas asignando 8 valores de punto flotante, valores que serían aprendidos durante el entrenamiento.

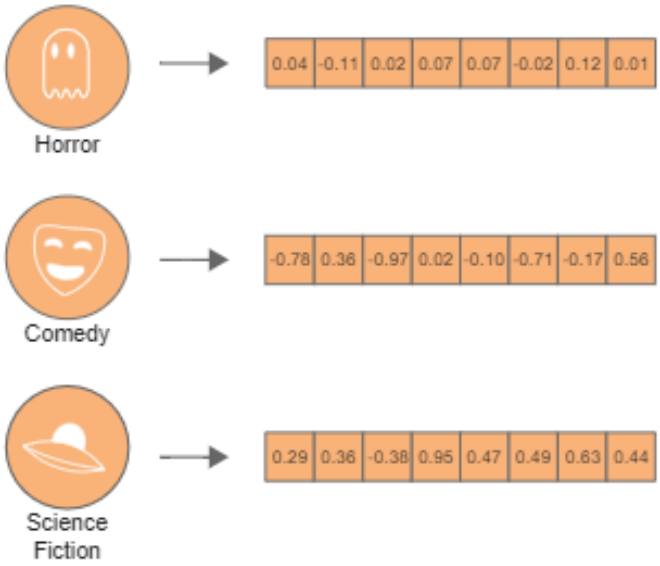


Figura 13. Ejemplo Embeddings [14]

Para poder obtener estos embeddings, es necesaria la parte anterior sobre el proceso de tokenización para poder realizar la representación vectorial en base a los ids. La Figura 14 muestra el proceso para la formación de los embeddings.

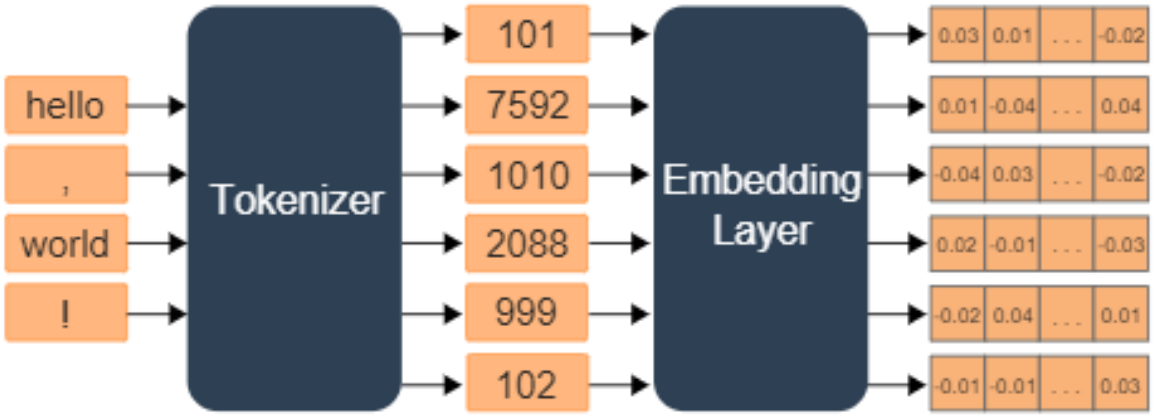
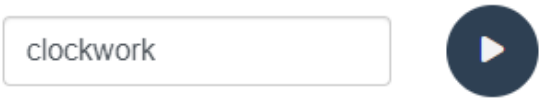


Figura 14. Proceso para la formación de los embeddings [14]

Para el modelo BERT Base, los vectores de los embeddings que se formarán aquí tendrán las dimensiones del vocabulario indicado y las capas ocultas de este modelo, en este caso son 768.

Estas representaciones vectoriales o embeddings, se podría tomar como valores aleatorios ya que no son legibles a simple vista debido a las dimensiones anterior mencionadas. Estos valores significan la comprensión durante el entrenamiento del modelo de un token en específico por lo que es posible tener embeddings que consideren palabras similares como indica la Figura 15.



Word	clock	bells	bell	tower	phone	alarm
Cosine Similarity	1.00	0.48	0.47	0.47	0.45	0.43

Figura 15. Representación y similitud de embeddings [14]

1.4.10 Capa de Codificador BERT

La capa de Codificador es la más importante para el entrenamiento de BERT, aquí es donde toma como entrada los datos procesados y tokenizados, por lo cual la salida son los embeddings contextualizados y listos para usarlos en la parte del decodificador [16].

Debido a que es basado en la arquitectura Transformer propuesta en [17], cada codificar está conformado por una capa de autoatención de múltiples cabezales (attention-head) y una red neuronal de alimentación directa.

1.4.11 Capa de Decodificador BERT

La capa de Decodificador BERT es la capa final del modelo, aquí se toman los resultados codificados y se puede aplicar distintas tareas, como las de clasificación, generación de texto, etc. Por consecuencia y como indica en BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [8] al preentrenamiento del modelo, BERT contiene dos tareas en específico, MASKED LM [18] que abarca la tarea de enmascarar

aleatoriamente un porcentaje de tokens en una oración para que el modelo trate de predecir el valor del token enmascarado, tomando en cuenta el contexto de las palabras alrededor del enmascaramiento. También cuenta con la tarea de Predecir la siguiente oración (NSP, por sus siglas en inglés) [18] la cual, como su nombre lo indica, es detectar si la siguiente oración es la que sigue o no, realizando una clasificación para las oraciones.

Estas son tareas con las cuales el modelo fue preentrenado, pero para usarlo para otras tareas, como es el caso de la Detección de Errores Ortográficos en un texto obtenido mediante herramientas OCR, el modelo en sí no abarcaría este tema ya que es una tarea muy distinta con relación a las tareas del entrenamiento del modelo BERT.

1.4.12 Ajuste Fino (Fine Tuning)

Aquí entra en juego el proceso del Ajuste Fino (Fine Tuning) el cual, a partir del uso del modelo preentrenado que ya contiene el contexto de los datos con los que fue entrenado, sería ajustado en la parte del decodificador para ser adaptado a la tarea requerida. Una representación de cómo se aplica el Ajuste Fine se indica en la Figura 16.

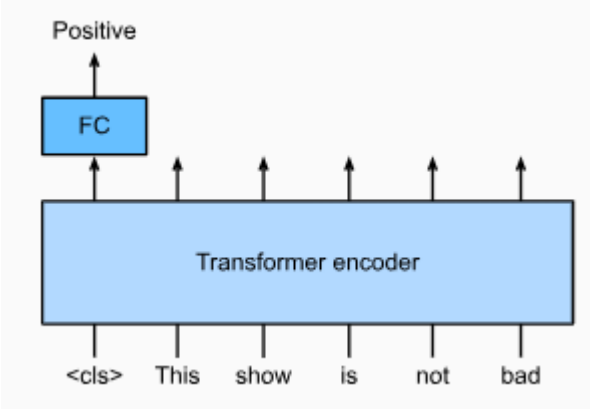


Figura 16. Ejemplo Ajuste Fino [23]

Aplica el Ajuste Fino al recibir las entradas al codificador del modelo y aplicar la tarea de análisis de sentimientos donde la clasificación es binaria, indicando si la oración es positiva o negativa.

2 METODOLOGÍA

El componente sigue el enfoque cuantitativo predominantemente, por lo que se basa en la aplicación de métodos cuantitativos para medir y evaluar la eficiencia del modelo de detección de errores gramaticales basado en un modelo preentrenado BERT utilizando oraciones de prueba para entender cómo actúa el modelo con oraciones nuevas.

El trabajo es netamente experimental, ya que implica la implementación y evaluación de un modelo para la detección de errores gramaticales en un corpus paralelo. Aquí se destacan mediciones cuantitativas para proporcionar una evaluación rigurosa como el cálculo de la precisión, exhaustividad y exactitud.

En el presente trabajo, para el desarrollo del componente indicado en la sección 1, se empleó la metodología “Proceso Estándar Interindustrial para la Minería de Datos” (Cross-Industry Standard Process for Data Minin, CRISP-DM) [19]

El proceso de ciclo de vida que ofrece esta metodología tiene los siguientes puntos: (1) Entendimiento del negocio, (2) Entendimiento de los datos, (3) Preparación de los datos, (3) Modelado, (4) Evaluación e implementación. De estos puntos, los usados para este trabajo son la Preparación de los datos (3), Modelado (4) y Evaluación e implementación (5).

Los recursos utilizados para el desarrollo del componente se utilizó el entorno de Jupyter disponible en los servidores de la Escuela Politécnica Nacional, el cual proporciona los dos lenguajes utilizados (Perl y Python). Con la ayuda de este entorno se generan los notebooks con los cuales se realiza el procesamiento y alineación de las oraciones aplicando el lenguaje de programación Perl. Para la construcción del modelo y la evaluación se usa el lenguaje de programación Python.

Tanto para Perl como para Python, se trabajó con las librerías MXNet y Gluon, ya que cuentan con soporte para los dos lenguajes y esto facilita la comprensión del código empleado. Adicional, la librería Gluon cuenta con el modelo preentrenado, por lo que su aplicación resulta cómoda de usar.

Para empezar, se requiere de la obtención del corpus paralelo para iniciar el proceso de preparación de las oraciones, este corpus está indicado en los Anexos.

2.1 Preparación de las oraciones

Para diferenciar cada uno de los componentes del corpus paralelo, al corpus que contiene las oraciones obtenidas por una herramienta OCR se lo denominará corpusOCR y para el corpus que contiene las oraciones correctas con las que se comparan los resultados se lo denominará como corpusREF, tomando en cuenta que cada oración de corpusOCR corresponde a una oración en el corpusREF. Al obtener el corpus paralelo se observa que tiene el contenido indicado en la Figura 17.

```
1 I N . U s e r . a u t h o r i z e # f u n c t i o n # # #
2 I N . U s e r . l o g o u t # f u n c t i o n # # #
3 u s e r . e m a i l _ # r e s . e m a i l A d d r e s s #
4 u s e r . f i r s t N a m e _ # _ r e s . l a s t N a m e #
5 # d e # _ # _ D e u t s c h e _ #
6 # n o c o m e n t s #
```

Figura 17. Contenido del corpus OCR

2.1.1 Identificación y normalización del contenido del corpus paralelo

Se observa que el contenido contiene espacios y caracteres que no son alfanuméricos por lo que esto dificultará el proceso de alineación y tokenización para preparar las oraciones para las entradas del modelo. Para solventar esto, en la Figura 18 se muestra el proceso de limpieza de las oraciones.

```perl
sub leer_archivo {
    my $archivo = shift;

    open my $fh, '<:encoding(UTF-8)', $archivoor die "ERROR: no se pudo abrir el archivo '$archivo'\n";
    my @lineas;

    while (my $linea = <$fh>) {
        chomp $linea;
        $linea =~ s/\s//g;
        $linea =~ s/^\W+|\W+$|\W+/ /g;
        $linea =~ s/^\s+|\s+$//g;
        push @lineas, lc($linea) unless $linea =~ /^\s+$/;
    }
    close $fh;

    return \@lineas;
}
```

Figura 18. Carga y limpieza de caracteres no alfanuméricos.

Al cargar las oraciones de los corpus se implementan expresiones regulares para localizar espacios y caracteres no necesarios y eliminarlos.

2.1.2 Eliminación de oraciones que no aportan contexto al entrenamiento del modelo

Se requiere conocer cuál es el tamaño de las oraciones contenidas en el corpus paralelo por lo cual se realiza un gráfico que indica el número de palabras por oración indicado en la Figura 19 y Figura 20.

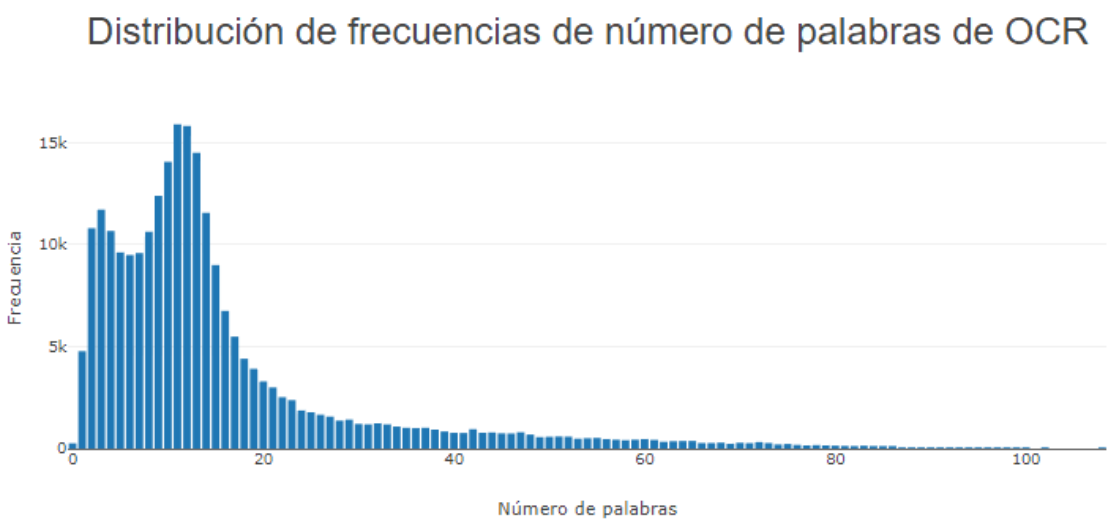


Figura 19. Frecuencia de número de palabras en cada oración en el corpusOCR

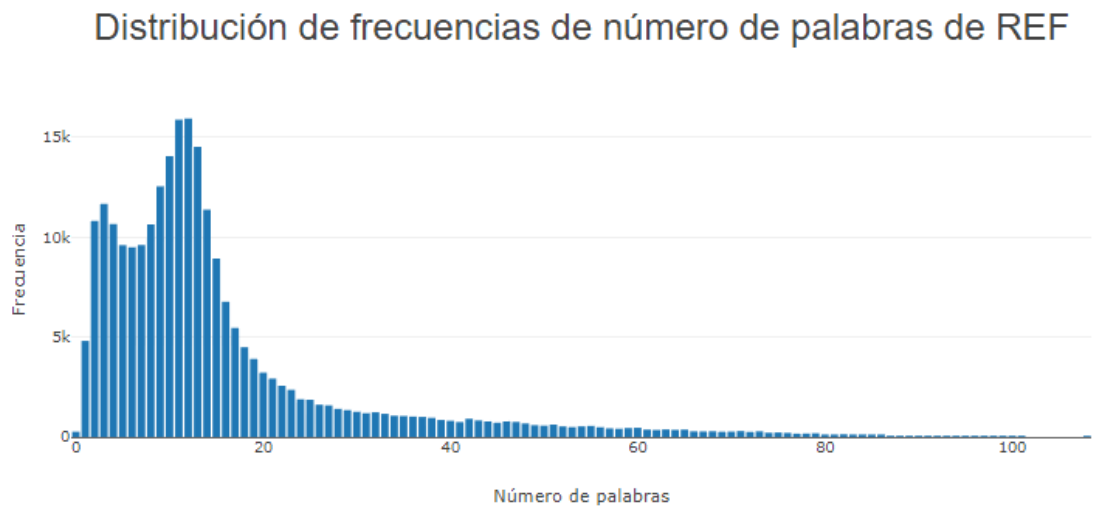


Figura 20. Frecuencia de número de palabras en cada oración en el corpusREF

Las gráficas indican que existen oraciones con más de 100 palabras y por el otro lado hay oraciones vacías o que contienen una o dos palabras. Las oraciones vacías se dan en el caso de que en el corpusOCR o en el corpusREF contienen un solo carácter o una sola

palabra, pero en su oración equivalente en el otro corpus este vacío. Por ahora el foco está sobre las oraciones vacías o que contienen pocas palabras, esto no es lo indicado ya que si se tiene pocas palabras no habrá el contexto suficiente para poder tener las suficientes características en los embeddings que obtendrá el modelo. Por lo cual se identifican las oraciones que tienen pocas palabras y se las elimina. La Figura 21 indica las expresiones regulares que se aplican a este proceso de eliminación de oraciones con pocas palabras y la Figura 22 muestra la eliminación de oraciones con una sola palabra.

```
for my $i (0 .. $#corpus_OCR) {
    my $linea_OCR = $corpus_OCR[$i];
    my $linea_REF = $corpus_REF[$i];

    chomp $linea_OCR;
    chomp $linea_REF;

    next if $linea_OCR =~ /^(\w\s\w\s?\w\s?)+\w$/ || $linea_REF =~ /^(\w\s\w\s?\w\s?)+\w$/;
    next if $linea_OCR =~ /^\s*$/ && $linea_REF =~ /^\s*$/;
    next if $linea_OCR =~ /^\w+$/ && $linea_REF =~ /^\w+$/;

    if ($linea_OCR =~ /^\s*$/ && $linea_REF =~ /^\w+$/) {
        next;
    }

    if ($linea_OCR =~ /^\w+$/ && $linea_REF =~ /^\s*$/) {
        next;
    }
```

Figura 21. Expresiones regulares para eliminar oraciones con pocas palabras

```
my @palabras_OCR = split /\s+/, $linea_OCR;
my @palabras_REF = split /\s+/, $linea_REF;

if ((scalar @palabras_OCR<2 || scalar @palabras_REF<2)) {
    next;
}
```

Figura 22. Expresión para eliminar oraciones con una sola palabra

Al eliminar las oraciones con pocas palabras se considera que se eliminan las oraciones a la par, por lo que el proceso actúa tanto en el corpusOCR como en el corpusREF. Si encuentra una oración corta en el corpusOCR se eliminará su equivalente en el corpusREF y viceversa. También se agrega la lógica para eliminar oraciones cuyas palabras están definidas solo como caracteres o solo por números, Figura 23, esto también resultaría en que no se obtendrá el contexto suficiente.

```
Se encontró una fila con caracteres individuales separados por espacios en la fila 22577: 3 3 go web
Se encontró una fila con caracteres individuales separados por espacios en la fila 22584: 3 3 go web
Se encontró una fila con caracteres individuales separados por espacios en la fila 31275: 0 5 0 0 0 75
Se encontró una fila con caracteres individuales separados por espacios en la fila 32562: 2 11 7
Se encontró una fila con caracteres individuales separados por espacios en la fila 32567: 2 11 7
Se encontró una fila con caracteres individuales separados por espacios en la fila 71591: 4 5 8 9
Se encontró una fila con caracteres individuales separados por espacios en la fila 73206: o i a i j x j
Se encontró una fila con caracteres individuales separados por espacios en la fila 73208: o i a i j x j
Se encontró una fila con caracteres individuales separados por espacios en la fila 73927: m ppc shr srw
Se encontró una fila con caracteres individuales separados por espacios en la fila 101105: a c c f
Se encontró una fila con caracteres individuales separados por espacios en la fila 102102: x max 0 x 0 max
Se encontró una fila con caracteres individuales separados por espacios en la fila 105199: b nan
Se encontró una fila con caracteres individuales separados por espacios en la fila 116189: b nan
Se encontró una fila con caracteres individuales separados por espacios en la fila 116936: 2 10 6
Se encontró una fila con caracteres individuales separados por espacios en la fila 124024: 1 0 11
Se encontró una fila con caracteres individuales separados por espacios en la fila 129980: s id s
Se encontró una fila con caracteres individuales separados por espacios en la fila 135112: 0 0 81
Se encontró una fila con caracteres individuales separados por espacios en la fila 167251: x y z y
Se encontró una fila con caracteres individuales separados por espacios en la fila 175310: 1 2 3 4
Se encontró una fila con caracteres individuales separados por espacios en la fila 175897: t x t y 100 100
```

Figura 23. Oraciones que no aportan contexto

2.2 Alineación del corpus paralelo

2.2.1 Técnicas de alineación de caracteres

Después del proceso de limpieza o procesamiento de oraciones, se aplican técnicas de alineación para encontrar diferencias entre las oraciones del corpusOCR y del corpusREF. Esto para obtener las palabras correctas e incorrectas.

El algoritmo utilizado para realizar la alineación de caracteres y obtener las diferencias que se tienen entre las oraciones del corpus paralelo es la de Smith-Waterman [20], un ejemplo en la Figura 24. Este algoritmo trabaja de manera local, es decir toma el contexto carácter por carácter y va analizando las diferencias encontradas esto de manera secuencial de izquierda a derecha y luego viceversa para encontrar con mayor exhaustividad los errores que se pueden encontrar.

```
string by building it up taking into account whether the supplied options and whether it is
string by building it up taking into account --------the supplied options and whether it is
```

Figura 24. Ejemplo de implementación del algoritmo Smith-Waterman

Otro algoritmo para comparar es el algoritmo Needleman-Wunsch [21] que trabaja de manera similar, pero a nivel global que no toma el contexto con tanta exhaustividad, mirar Figura 25.

```
string by building it up taking into account whether the supplied options and whether it is
string by building it up taking into accoun-----t--- the supplied options and whether it is
```

Figura 25. Ejemplo de implementación del algoritmo Needleman-Wunsch

También con el algoritmo Ukkonen [22] que trabaja a nivel de árbol de sufijos que además de obtener las diferencias en la alineación muestra la cantidad de diferencias que existen en cada oración, Figura 26.

```
8
string by building it up taking into account whether the supplied options and whether it is
string by building it up taking into accoun-----t--- the supplied options and whether it is
```

Figura 26. Ejemplo de implementación del algoritmo Ukkonen

En la aplicación de los tres algoritmos se observa resultados similares, pero con la diferencia de que el algoritmo Smith-Waterman al trabajar de manera local, encuentra con más exhaustividad los errores sin afectar a las demás palabras. Como se observa en los ejemplos, Needleman-Wunsch y Ukkonen al realizar el alineamiento pueden llegar a cambiar los caracteres de las palabras generando errores que no debieran existir ya que al compararr las oraciones se observa que existe un error en la palabra *account*¸ error que no debe ser detectado por que en las dos oraciones existen. Por esta razón el mejor algoritmo de alineación es Smith-Waterman para esta tarea. Aquí una muestra de ejemplos que indican distintos tipos de errores en la Figura 27 y Figura 28.

```
user firstname res -lastname
user firstname res firstname

de deutsche
de deutsch-

you can disable comments for the current post using the nocomment-
you can disable comments for the current post using the nocomments

if build-- support app
if builder support app

with their excepr-ts
with their exce-rpts

compile cat-log http babel edgewall org wiki documentation setup html id4
compile catalog http babel edgewall org wiki documentation setup html id4

oc kernel patcher skips a a patch at u due to version u u u--
oc kernel patcher skips a a patch at u due to version u u u n

string by building it up taking into account whether the supplied options and whether it is
string by building it up taking into account --------the supplied options and whether it is

checks if the provided intiger is --prime number
checks if the provided integer is a prime number

on intel cpus and gpus deep learning practictitioners should use one of the
on intel cpus and gpus deep learning pract---itioners should use one of the

be able to set conditions to write custom access log
---able to set conditions to write custom access log

returns an array with n elements removed from the beggin-ing
returns an array with n elements removed from the beg-inning

number to array to- digits
number to array -of digits
```

Figura 27. Ejemplos de la alineación Smith-Waterman

```
creates a promis---e- version of the given callback style function in node 8 you
creates a promisified version of the given callback style function in node 8 you

onlyoneshortbreaktooltip activ--ing this option stops counter of the short break
onlyoneshortbreaktooltip activating this option stops counter of the short break

restartrequired chan-ing language takes effect in next start
restartrequired changing language takes effect in next start

param a address usually represention- from
param a address usually representi-ng from

only the latest released version of each package in this framework-are in scope
only the latest released version of each package in this framework are in scope

supporterd blockchain
supporte-d blockchain

description a uri poni-t- to the evidence json with data needed to certify this asset
description a uri po-ints to the evidence json with data needed to certify this asset
```

Figura 28. Ejemplos alineación Smith-Waterman

2.2.1 Implementación del algoritmo Smith-Waterman

El algoritmo Smith-Waterman está disponible en la librería d2l disponible en el entorno Jupyter mencionado al principio. Si se requiere obtener la librería, se debe instalar e importar la librería, como indica la Figura 29.

```
!pip d2l

from d2l import mxnet as d2l
```

Figura 29. Disponibilidad de la librería para usar el algoritmo Smith-Waterman

El contenido de la librería, además de contener el algoritmo Smith-Waterman, también contiene los algoritmos Needleman-Wunsch y Ukkonen por lo que la implementación es la misma para realizar la comparación entre los tres algoritmos. En la Figura 30, Figura 31 y Figura 32 se muestra cómo se aplica cada uno de estos algoritmos.

1.3.3 Smith-Waterman Algorithm

```
my $tiempo = new d2l::Timer();

open(my $archivo_SW, '>:encoding(UTF-8)', '../../Tesis/0_Corpus/3_Corpus_Alineamiento/Smith-Waterman/corpus_SW.txt')
                      or die "No se puede abrir el archivo corpus_SW.txt: $!";

for (my $i=0; $i<scalar(@$corpus_OCR); $i++) {
    my $linea1 = $corpus_OCR->[$i];
    my $linea2 = $corpus_REF->[$i];

    my ($alineamiento_SW1, $alineamiento_SW2) = d2l->smith_waterman($linea1, $linea2);

    print $archivo_SW "$alineamiento_SW1\n";
    print $archivo_SW "$alineamiento_SW2\n";
    print $archivo_SW "\n";
}

close($archivo_SW);

print "Duracion: ", $tiempo->stop(), "s.\n";
```

Figura 30. Implementación algoritmo Smith-Waterman

```
my ($alineamiento_NW1, $alineamiento_NW2) = d2l->needleman_wunsch($linea1, $linea2);
```

Figura 31. Implementación algoritmo Needleman-Wunsch

```
my ($distancia, $alineamiento_Uk1, $alineamiento_Uk2) = d2l->ukkonen($linea1, $linea2);
```

Figura 32. Implementación algoritmo Ukkonen

La implementación es la misma para los 3 algoritmos, donde reciben como entrada una oración del corpusOCR y una oración del corpusREF, motivo por el cual el corpus paralelo debe contener el mismo número de oraciones en cada corpus (ver Figura 33).

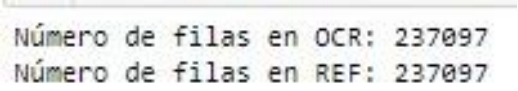


Figura 33. Número de filas (oraciones) en el corpus paralelo

2.2.2 Procesamiento de los resultados de la alineación

Una vez terminado el proceso de alineación aplicando el algoritmo Smith-Waterman, donde se obtuvieron resultados indicados en las Figura 27 y Figura 28, se observa que se han colocado guiones donde corresponden los errores que se encontraron en cada palabra, implicando un desbalance con respecto al número de palabras en cada oración, por ejemplo, en la Figura 34 se muestra como actúo el algoritmo de alineación con respecto a la desigualdad de palabras en cada oración.

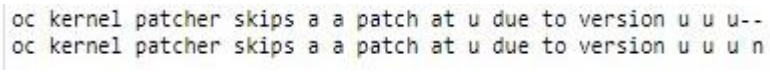


Figura 34. Desigualdad en el proceso de alineación

El orden de las oraciones es el siguiente: en la parte superior la oración del corpusOCR y debajo la oración del corpusREF. Al observar la Figura 34 muestra que en la oración del corpusOCR encontró 15 palabras y en la oración del corpusREF encontró 16. Por esta razón se colocó dos guiones, una indicando la palabra faltante, o en este caso el carácter faltante, y uno por el espacio que está antes de la última palabra ya que en la primera oración no existen. Esto provoca un desbalance en el número de palabras y provocará errores al momento de etiquetar las oraciones que serán usadas para el entrenamiento.

Para solucionar este problema, en la Figura 35 se muestra la implementación usada para eliminar los guiones que afectan al número de palabras.

```perl
sub comparar_caracteres {
    my ($corpus_OCR, $corpus_REF) = @_;

    my @corpus_OCR_modificado;
    my @corpus_REF_modificado;

    for (my $i = 0; $i < @$corpus_OCR; $i++) {
        my @ocr_chars = split //, $corpus_OCR->[$i];
        my @ref_chars = split //, $corpus_REF->[$i];

        my @palabras_OCR = split /\s+/, $corpus_OCR->[$i];
        my @palabras_REF = split /\s+/, $corpus_REF->[$i];

        if (@palabras_OCR != @palabras_REF) {
            for (my $j = 0; $j < @ocr_chars; $j++) {
                if ($ocr_chars[$j] eq '-' && $ref_chars[$j] eq ' ') {
                    $ocr_chars[$j] = ' ';
                } elsif ($ocr_chars[$j] eq ' ' && $ref_chars[$j] eq '-') {
                    $ref_chars[$j] = ' ';
                }
            }
        }

        push @corpus_OCR_modificado, join('', @ocr_chars);
        push @corpus_REF_modificado, join('', @ref_chars);
    }

    return (\@corpus_OCR_modificado, \@corpus_REF_modificado);
}
```

Figura 35. Eliminación de guiones entre palabras

Cada oración es dividida en base al espacio que las separa, formando un vector con cada palabra como elemento, luego se comparan los caracteres de cada elemento (palabra) y si existe un guión donde debe corresponder un espacio, este es removido, para que se iguale el número de palabras en cada oración paralela.

También ocurre otro tipo de errores con respecto a la desigualdad del número de palabras y esto corresponde a que se no solo se diferencian por caracteres, sino que toda la palabra ha sido sustituida o cambiada por otra, como indica la Figura 36.

```
Fila 1330:
OCR: along the way we re going to explore software that will help you understand if your customers are excited sad or disgus
ted by your products we ll see how simple api calls can identify what objects are in an image we ll show how to build your o
wn iphone app that cal tell you whether what you re pointing your camera at is a business card
REF: along the way we re -di--scove----r- software that will help you understand if your customers are excited sad or disgus
ted by your products we ll see how simple api calls can identify what objects are in an image we ll show how to build your o
wn iphone app that can tell you whether ---- you re pointing your camera at -- a business card
```

Figura 36. Ejemplo desigualdad de palabras en las oraciones paralelas

Aquí se muestra la fila donde se encontró las oraciones que no tienen el mismo número de palabras, el error se encuentra en que en la oración OCR se encontró las palabras *going to explore* pero en la oración REF encontró *discover*, por lo que el algoritmo trató de coordinar y empatar este grupo de caracteres con *-di—scove----r-*. La solución implementada se muestra en la Figura 37.

```
sub ajustar_numero_palabras {
    my ($corpus_OCR, $corpus_REF) = @_;

    my @corpus_OCR_modificado;
    my @corpus_REF_modificado;

    for (my $i = 0; $i < @$corpus_OCR; $i++) {
        my @ocr_chars = split //, $corpus_OCR->[$i];
        my @ref_chars = split //, $corpus_REF->[$i];

        my @palabras_OCR = split /\s+/, $corpus_OCR->[$i];
        my @palabras_REF = split /\s+/, $corpus_REF->[$i];

        if (@palabras_OCR != @palabras_REF) {
            for (my $j = 0; $j < @ocr_chars; $j++) {
                if ($ocr_chars[$j] eq ' ' && $ref_chars[$j] =~ /[a-zA-Z0-9]/) {
                    $ocr_chars[$j] = '-';
                } elsif ($ocr_chars[$j] =~ /[a-zA-Z0-9]/ && $ref_chars[$j] eq ' ') {
                    $ref_chars[$j] = '-';
                }
            }
        }

        push @corpus_OCR_modificado, join('', @ocr_chars);
        push @corpus_REF_modificado, join('', @ref_chars);
    }

    return (\@corpus_OCR_modificado, \@corpus_REF_modificado);
}
```

Figura 37. Función para el ajuste de palabras.

Se usa expresiones regulares para comparar si existen espacios o algún carácter en la oración OCR y si en esa misma posición se encuentra lo contrario (un carácter o un espacio), de ser así se coloca un guión. Esto para unir las palabras y formar una sola. Esto se implementó de esta manera ya que se tomará como un solo error en relación con el corpusREF. El resultado se muestra en la Figura 38.

```
Oración OCR: along the way we re going-to-explore software that will help you understand if your customers are excited sad or d
isgusted by your products we ll see how simple api calls can identify what objects are in an image we ll show how to build your
own iphone app that cal tell you whether what you re pointing your camera at is a business card

Oración REF: along the way we re -di--scove----r- software that will help you understand if your customers are excited sad or d
isgusted by your products we ll see how simple api calls can identify what objects are in an image we ll show how to build your
own iphone app that can tell you whether ---- you re pointing your camera at -- a business card
```

Figura 38. Resultado de la función para el ajuste de palabras.

El resultado indica que mantiene la forma en cómo se alineó la palabra *discover* en la oración REF, pero en la oración OCR se unió las 3 palabras para formar una sola, quedando como *going-to-explore*. Esta solución se aplica de esta manera ya que los errores se los debe tomar en base al corpusREF, de este modo solo indicaría un error con respecto al texto Referencial, y no tres con respecto al texto OCR.

2.3 Desarrollo e implementación del modelo para Detección de errores ortográficos en un texto obtenido mediante OCR

2.3.1 Preparación de los parámetros de entradas del modelo

La preparación de las entradas del modelo consiste en formar tensores mxnet que requiere el modelo preentrenado. Para ello se requiere de 4 tensores que son:

- token_ids: consisten en las oraciones procesadas anteriormente luego del preprocesamiento y de la alineación, estas oraciones son tokenizadas usando la técnica Wordpiece, creando así vectores de los ids que son las representaciones que el modelo requiere como entradas.
- segments: este vector es usado en tareas como Predicción de la Siguiente Oración, donde al pasar dos oraciones se debe identificar cual es la primera oración y la segunda oración, esto es indicado por ceros para la primera y unos para la segunda. En este caso, como la clasificación es a nivel de palabras, por lo que solo se requiere del vector solo con ceros.
- valid_len: consiste en indicar al modelo cual es el tamaño valido de palabras que ingresan al modelo, esto se realiza ya que se aplica valores de relleno para la formación de lotes explicados más adelante.
- labels: consiste en las etiquetas que se obtienen al comparar el corpusOCR con el corpusREF, de aquí es donde se obtendrá la clasificación de palabras correctas (valor 1) y palabras incorrectas (valor 2). Se lo obtiene con el fin de evaluar el modelo durante el entrenamiento y cuál es el valor de la pérdida con las etiquetas verdaderas.

Se inicia creando las etiquetas con las asignaciones de sus valores, por lo que se aplica la función de la Figura 39, donde realiza una división de palabras y se agrega como elementos a nuevas variables, esto para comparar cada palabra de cada par de oraciones y si encuentra un guión, indicado en la sección 2.2, agrega la clasificación correspondiente.

Creación de las etiquetas usando OCR y REF

```python
def definir_etiquetas(corpus_OCR, corpus_REF):
    etiqueta = []
    new_sentences_OCR = []
    new_sentences_REF = []

    for sentence_OCR, sentence_REF in zip(corpus_OCR, corpus_REF):
        fila_resultados = []
        new_sentence_OCR = []
        new_sentence_REF = []

        words_OCR = sentence_OCR.split()
        words_REF = sentence_REF.split()

        words_OCR = [word.replace('-', '').replace('()', '') for word in words_OCR]
        words_REF = [word.replace('-', '').replace('()', '') for word in words_REF]

        for word_OCR, word_REF in zip(words_OCR, words_REF):
            if word_OCR == '':
                continue
            else:
                fila_resultados.append(1 if word_OCR == word_REF else 0)
                new_sentence_OCR.append(word_OCR)
                new_sentence_REF.append(word_REF)

        etiqueta.append(fila_resultados)
        new_sentences_OCR.append(' '.join(new_sentence_OCR))
        new_sentences_REF.append(' '.join(new_sentence_REF))

    return etiqueta, new_sentences_OCR, new_sentences_REF
```

Figura 39. Creación de las etiquetas usando el corpus paralelo

Esta función arroja el resultado indicado en la Figura 40, donde por cada palabra en cada oración asigna una etiqueta correspondiente a si es correcta o incorrecta.

```
etiquetas:  [1, 1, 1, 1]
oración OCR:  in user authorize function
oración REF:  in user authorize function

etiquetas:  [1, 1, 1, 1]
oración OCR:  in user logout function
oración REF:  in user logout function

etiquetas:  [1, 1, 1, 1]
oración OCR:  user email res emailaddress
oración REF:  user email res emailaddress

etiquetas:  [1, 1, 1, 0]
oración OCR:  user firstname res lastname
oración REF:  user firstname res firstname

etiquetas:  [1, 0]
oración OCR:  de deutsche
oración REF:  de deutsch

etiquetas:  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0]
oración OCR:  you can disable comments for the current post using the nocomment
oración REF:  you can disable comments for the current post using the nocomments

etiquetas:  [1, 0, 1, 1]
oración OCR:  if build support app
oración REF:  if builder support app

etiquetas:  [1, 0, 1, 1]
oración OCR:  if build support app
oración REF:  if builder support app

etiquetas:  [1, 1, 0]
oración OCR:  with their exceprts
oración REF:  with their excerpts

etiquetas:  [1, 0, 1, 1, 1, 1, 1, 1]
oración OCR:  added exceprts option to rst dir postlist directive
oración REF:  added excerpts option to rst dir postlist directive
```

Figura 40. Asignación de las etiquetas correspondientes a los errores ortográficos

Al aplicar la clasificación de errores y formar las etiquetas ocurrió un desfase en el número de palabras, por lo que se tiene que verificar la causa y como solucionarlo. Como se muestra en la Figura 41, el error se debe a que existe un doble espacio donde debería estar un elemento vacío, esto a causa de sustituir los guiones anteriormente.

```
Diferencia en la longitud de etiqueta y corpus_REF en el índice 22.
[1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1]
string by building it up taking into account whether the supplied options and whether it is
string by building it up taking into account  the supplied options and whether it is
Diferencia en la longitud de corpus_OCR y corpus_REF en el índice 22.
[1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1]
string by building it up taking into account whether the supplied options and whether it is
string by building it up taking into account  the supplied options and whether it is
```

Figura 41. Desblance en el número de palabras en cada oración.

Para solucionar este desbalance en el número de palabras, como se indica en la Figura 42, se aplica nuevamente una división de palabras y donde encuentra doble espacio se coloca por defecto un espacio en blanco, creando así un elemento vacío. Por temas de tokenización y asignación de los ids, se reemplaza este elemento vacío por un "_".

```python
def igualar_oraciones(corpus_OCR, corpus_REF):
    etiqueta = []
    new_sentences_OCR = []
    new_sentences_REF = []

    for sentence_OCR, sentence_REF in zip(corpus_OCR, corpus_REF):
        fila_resultados = []
        new_sentence_OCR = []
        new_sentence_REF = []

        words_OCR = sentence_OCR.split(" ")
        words_REF = sentence_REF.split(" ")

        for word_OCR, word_REF in zip(words_OCR, words_REF):
            if word_OCR == '':
                new_sentence_OCR.append('')
                new_sentence_REF.append(word_REF)
            elif word_REF == '':
                new_sentence_OCR.append(word_OCR)
                new_sentence_REF.append('_')
            else:
                fila_resultados.append(1 if word_OCR == word_REF else 0)
                new_sentence_OCR.append(word_OCR)
                new_sentence_REF.append(word_REF)

        etiqueta.append(fila_resultados)
        new_sentences_OCR.append(' '.join(new_sentence_OCR))
        new_sentences_REF.append(' '.join(new_sentence_REF))

    return new_sentences_OCR, new_sentences_REF, etiqueta
```

Figura 42. Método para igualar el número de oraciones

Previo a aplicar el ajuste de palabras por oración, se obtuvo el número de palabras y la frecuencia que tiene el corpusOCR, ya que en base a este corpus se tomaran los token_ids.

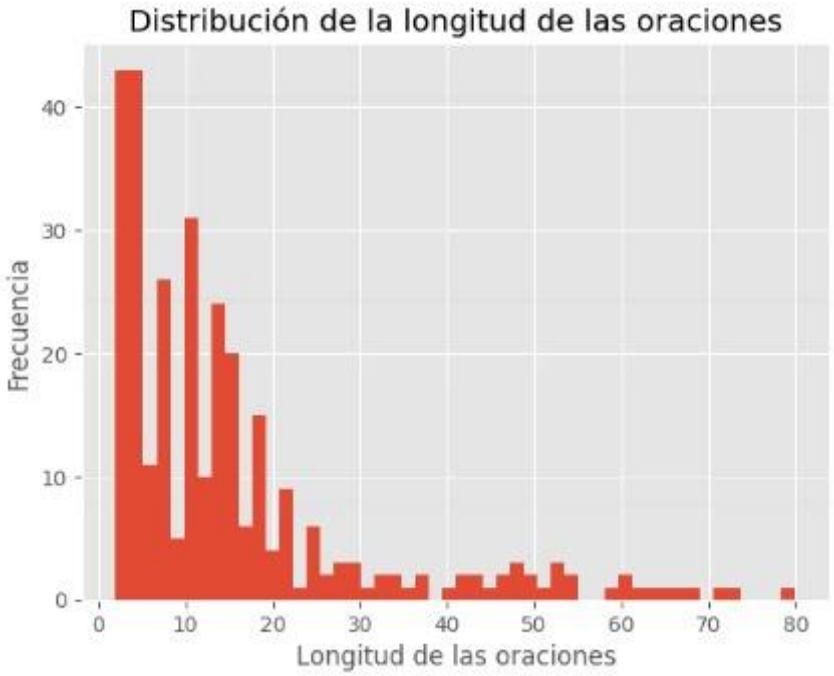


Figura 43. Distribución de longitudes de oraciones del corpusOCR

Al observar la distribución del total de oraciones en el corpusOCR después de la alineación, Figura 43, se sigue observando una distribución similar que la Figura 19, por lo que indica que la mayor parte de las oraciones tiene hasta 20 palabras, valor que será usado para definir cuantas palabras por oración tendrán las entradas del modelo.

Al tener un límite de palabras por oración a usar se considera ya que esto evita la carga masiva al modelo, además de que, si se consideran las oraciones mayores a 100, se tendrá que rellenar las demás palabras para que el bloque pasado al modelo sea uniforme. Debido a esto se hizo un recorte a las oraciones para que formen palabras de hasta 20 palabras.

División de las oraciones

Si la oración es mayor al número definido anteriomente se aplica una división para crear más muestras en el conjunto de entrenamiento

```python
def split_sentences(corpus_OCR, corpus_REF, etiqueta, max_words_len):
    new_sentence1 = []
    new_sentence2 = []
    new_etiqueta = []

    for i in range(0, len(corpus_OCR)):
        sentence1_words = corpus_OCR[i].split()
        sentence2_words = corpus_REF[i].split()
        etiqueta_words = etiqueta[i]

        num_fragments = math.ceil(len(sentence1_words)/max_words_len)
        new_fragment_size = math.ceil(len(sentence1_words)/num_fragments)

        sentence1_fragments = [' '.join(sentence1_words[j:j+new_fragment_size])
                               for j in range(0, len(sentence1_words), new_fragment_size)]
        sentence2_fragments = [' '.join(sentence2_words[j:j+new_fragment_size])
                               for j in range(0, len(sentence2_words), new_fragment_size)]
        etiqueta_fragments = [etiqueta_words[j:j+new_fragment_size]
                              for j in range(0, len(etiqueta_words), new_fragment_size)]

        new_sentence1.extend(sentence1_fragments)
        new_sentence2.extend(sentence2_fragments)
        new_etiqueta.extend(etiqueta_fragments)

    return new_sentence1, new_sentence2, new_etiqueta
```

Figura 44. Función para dividir las oraciones más largas

La Figura 44 indica el recorte de oraciones, OCR y REF, incluidas las etiquetas. Esto se lo realiza para mantener la uniformidad de la distribución de las oraciones, teniendo los resultados mostrados en la Figura 45.

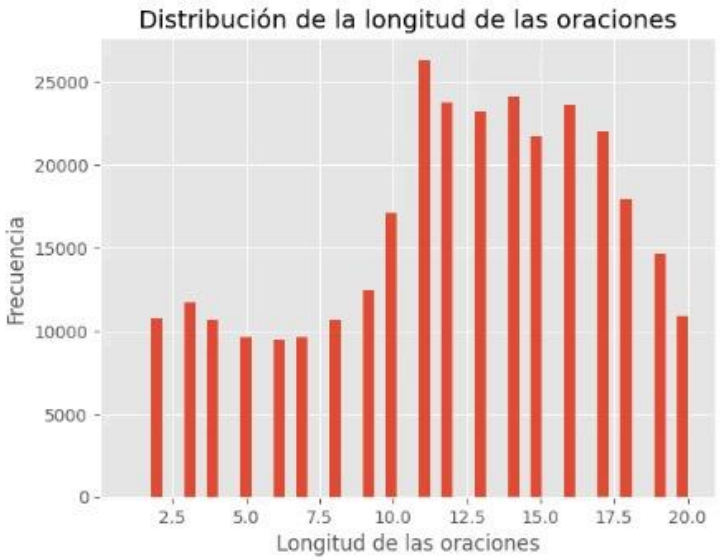


Figura 45. Distribución de oraciones luego de la normalización en el número de palabras.

2.3.2 Formación de los tensores MXNet

Como se mencionó anteriormente en el apartado 2.3.1, se requiere que las oraciones sean transformadas al formato mxnet ya que se está utilizando esta librería para usar el modelo preentrenado; por consecuente se debe primero tokenizar las oraciones anteriores mostradas y transformarlos en ids. La Figura 46 muestra el proceso de transformación a ids.

Función para obtener los ids de los tokens

Obtención de los ids de cada palabra, exclusión de las etiquetas cls y sep. Recorte del conjunto de datos que sean mayor al número de tokens definido anteriormente.

```
model, vocab = nlp.model.get_model('bert_12_768_12', dataset_name='book_corpus_wiki_en_uncased',
                                   ctx=ctx, use_classifier=False, use_decoder=False)

tokenizer = nlp.data.BERTTokenizer(vocab, lower=True)

transform = nlp.data.BERTSentenceTransform(tokenizer, max_len, pair=False, pad=False)

def transform_sentences(transform, corpus_OCR, corpus_REF, etiqueta):

    sentence_transform_OCR = []
    sentence_transform_REF = []
    etiquetas_desc = []

    for (sentence_OCR, sentence_REF, etiquetas) in zip(corpus_OCR, corpus_REF, etiqueta):
        sentenceOCR = transform([sentence_OCR])
        sentenceREF = transform([sentence_REF])

        words_OCR = sentenceOCR[0]
        words_REF = sentenceREF[0]

        sentences_OCR = words_OCR[1:-1]
        sentences_REF = words_REF[1:-1]

        if(len(sentences_OCR) <= max_len and len(sentences_OCR) >= 3):
            sentence_transform_OCR.append(sentences_OCR)
            sentence_transform_REF.append(sentences_REF)
            etiquetas_desc.append(etiquetas)

    return sentence_transform_OCR, sentence_transform_REF, etiquetas_desc
```

Figura 46. Transformación de las oraciones a ids

La función para transformar se la obtiene de la libería GluonNLP, llamada BERTTransform que recibe los parámetros del tokenizador, el cual se lo obtiene también de esta librería usando BERTTokenizador donde esta a su vez recibe el vocabulario obtenido del modelo preentrenado BERT. max_len indica el número de tokens límite a obtener durante la transformación, este valor debe ser mayor al definido en número de palabras por motivos de la técnica Wordpiece indicada en la sección 1.4.8. Pair y pad se colocan con False ya que no se pasará pares de oraciones, si no una sola y no se requiere el uso de etiquetas relleno durante esta implementación. Actúa a nivel de oración, por lo que se pasa una por una y realiza la transformación, se eliminan el primer y el último token generado ya que son

representantes de las etiquetas “[CLS]” (comienzo de oración) y “[SEP]” (separador para dos oraciones), estas etiquetas están dentro del modelo preentrenado, y al descartarlas no serán consideradas en el entrenamiento debido a que será una clasificación a nivel de palabra, o más explícito a nivel de token. Finalmente se realiza una comprobación de que los vectores generados con los ids sean menores a max_len y mayores a 3, esto debido a las oraciones cortas y la falta de contexto explicada en el apartado 2.3.1. El resultado está indicado en la Figura 47.

```
[1999 5310 3166 4697 3853]
[1999 5310 3166 4697 3853]
[1, 1, 1, 1]

[1999 5310 8154 4904 3853]
[1999 5310 8154 4904 3853]
[1, 1, 1, 1]

[ 5310 10373 24501 10373  4215 16200  4757]
[ 5310 10373 24501 10373  4215 16200  4757]
[1, 1, 1, 1]

[ 5310  2034 18442 24501  2197 18442]
[ 5310  2034 18442 24501  2034 18442]
[1, 1, 1, 0]

[ 2017  2064  4487 19150  7928  2005  1996  2783  2695  2478  1996  2053
  9006  3672]
[ 2017  2064  4487 19150  7928  2005  1996  2783  2695  2478  1996  2053
  9006  8163]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0]
```

Figura 47. Resultado de la transformación de token a ids

Se muestran los resultados de la transformación para las oraciones OCR y oraciones REF, además se muestra las etiquetas. Aquí se observa un problema con respecto a las etiquetas, debido a la técnica Wordpiece las palabras se dividieron (ver apartado 1.4.8) generando ahora un desbalance con respecto a los ids y las etiquetas. Para solucionar este se aplicó la división de las etiquetas correspondientes a su clasificación original, ver Figura 48.

Función para la asignación de etiquetas a tokens wordpiece

Al realizar el proceso anterior se obtienen tokens con respecto a wordpiece, a esta separación se le agregan las etiquetas pertenecientes a la palabra completa

```
:
def wordpiece_labels(sentence_transform, vocab, etiqueta):
    new_etiquetas = []
    pos_new_etiquetas = []

    for i, (token_ids, orig_etiquetas) in enumerate(zip(sentence_transform, etiqueta)):
        tokens = [vocab.to_tokens(int(token_id)) for token_id in token_ids]
        current_etiqueta = None
        new_etiqueta = []
        pos_new_etiqueta = []

        etiqueta_idx = 0
        count = 0
        for j, token in enumerate(tokens):
            if etiqueta_idx < len(orig_etiquetas):
                if not re.search(r'^##', token):
                    current_etiqueta = orig_etiquetas[etiqueta_idx]
                    etiqueta_idx += 1
                    count += 1
                    pos = count - 1
            new_etiqueta.append(current_etiqueta)
            pos_new_etiqueta.append(pos)
        new_etiquetas.append(new_etiqueta)
        pos_new_etiquetas.append(pos_new_etiqueta)

    if(len(tokens) != len(new_etiqueta)):
        print(orig_etiquetas)
        print(len(orig_etiquetas))
        print(tokens)
        print(len(tokens))
        print(new_etiqueta)
        print(len(new_etiqueta))
        print("")

    return new_etiquetas, pos_new_etiquetas
```

Figura 48 . Asignación de etiquetas a tokens obtenidos por Wordpiece

El código, en términos generales, toma la etiqueta del anterior token analizado si el token actual tiene en su token los caracteres ## de esta manera se asegura de que todos los tokens que fraccionaron la palabra tengan la misma etiqueta. Además, se obtienen las posiciones de los tokens originales, es decir no se consideran los tokens con ##, sino solamente los válidos como si se trataran de palabras completas, a continuación, un ejemplo en la Figura 49.

```
[1, 1, 1, 1, 1]
[0, 1, 2, 2, 3]

[1, 1, 1, 1, 1]
[0, 1, 2, 2, 3]

[1, 1, 1, 1, 1, 1, 1]
[0, 1, 2, 3, 3, 3, 3]

[1, 1, 1, 1, 0, 0]
[0, 1, 1, 2, 3, 3]

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0]
[0, 1, 2, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10]
```

Figura 49. Distribución de las etiquetas y posiciones originales.

De esta manera, si se requiere unir las palabras nuevamente, se recurrirá a sus posiciones antes de aplicada la técnica Wordpiece.

Luego de todo este proceso al tener organizados los token_ids y las etiquetas, se procede con la creación de los otros dos tensores. Para ello se utiliza la siguiente función (Figura 50).

Formación de la data para entrenamiento

Creación de las entradas para el modelo y obtención de la distribución de la frecuencia de cada clase.

```python
def data_pretrain(sentences_transform_OCR, sentences_transform_REF, vocab, etiqueta, ctx):
    labels_wordpiece, orig_etiquetas_positions = wordpiece_labels(sentences_transform_OCR, vocab, etiqueta)
    pretrain, total_labels, REF_tokens, orig_etiquetas_positions_excl = [], [], [], []

    for i in range(len(labels_wordpiece)):
        # Si la palabra no contiene al menos un 0 (clase incorrecta) se excluye del entrenamiento
        if 0 in labels_wordpiece[i]:
            tokens_OCR = sentences_transform_OCR[i]
            tokens_REF = sentences_transform_REF[i]

            token_ids = [*tokens_OCR]
            token_ids_ref = [*tokens_REF]
            valid_len = len(token_ids)
            segments = [0] * valid_len
            labels = [*labels_wordpiece[i]]
            pos_etiquetas = [*orig_etiquetas_positions[i]]

            total_labels.append(labels)

            train = (token_ids, segments, valid_len, labels)
            pretrain.append(train)
            REF_tokens.append(token_ids_ref)
            orig_etiquetas_positions_excl.append(pos_etiquetas)

    # Las entradas deben tener un número divisible para el número de dispositivos a usarse
    # Se elimina aleatoriamente el número excedentes si el módulo entre pretrain y los dispositivos no es cero
    num_ctx = len(ctx)
    remainder = len(pretrain) % num_ctx
    if remainder != 0:
        samples_to_remove = remainder
        for _ in range(samples_to_remove):
            random_index = random.randint(0, len(pretrain) - 1)
            del pretrain[random_index]
            del total_labels[random_index]
            del REF_tokens[random_index]
            del orig_etiquetas_positions_excl[random_index]

    # Cálculo de pesos de cada clase
    cont_clase_1 = 0
    cont_clase_0 = 0

    for labels in total_labels:
        for label in labels:
            if label == 1:
                cont_clase_1 += 1
            else:
                cont_clase_0 += 1

    total = cont_clase_1 + cont_clase_0

    peso_clase_0 = cont_clase_0 / total
    peso_clase_1 = cont_clase_1 / total

    class_weights = mx.nd.array([peso_clase_0, peso_clase_1], ctx=mx.cpu(0)) # ([0, 1])

    return pretrain, class_weights, REF_tokens, orig_etiquetas_positions_excl
```

Figura 50. Función para la formación de los tensores segments y valid_len y unificar los tensores en una sola variable

Esta función implica estos procesos:

- Utiliza la función de la Figura 48 para la distribución de las etiquetas con respecto a la transformación de los tokens
- Dentro de una iteración que recorre entorno al tamaño de las etiquetas obtenidas, se verifica que las oraciones tengan al menos un 0 en su contenido, esto para que las oraciones que sean pasadas al modelo tengan al menos un error dentro de su contenido.
- Si la condición es cumplida, se procede con la creación de los tensores segments y valid_len correspondientes con el tamaño de los tokens. Una vez creados se guardan en pretrain para que la función retorne una sola variable, además de las posiciones mencionadas anteriormente.
- Como el modelo está adaptado para trabajar con múltiples dispositivos que se explicará más adelante, se debe adaptar el tamaño de las oraciones a procesarse, por lo que se crea una condición sobre si no es divisible para el número de dispositivos asignados, se eliminarán aleatoriamente las oraciones que excedan el tamaño definido.
- Finalmente, con las etiquetas a usarse, se calcula la probabilidad de que pertenezca a la clase 0 y 1 respectivamente. Esto con el fin de poder replicar las condiciones de aparición de las etiquetas para el modelo Baseline que se usará en la evaluación.
- Como se requiere validar al final del entrenamiento con las oraciones del corpusREF, se aplica el mismo proceso obteniendo tokensREF, esto al aplicar las mismas condiciones que corpusOCR y sus etiquetas.

La Figura 51 muestra un ejemplo de pretrain que será utilizado en el entrenamiento.

```
([5310, 2034, 18442, 24501, 2197, 18442], [0, 0, 0, 0, 0, 0], 6, [1, 1, 1, 1, 0, 0])
([2017, 2064, 4487, 19150, 7928, 2005, 1996, 2783, 2695, 2478, 1996, 2053, 9006, 3672], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0], 14, [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0])
([2065, 3857, 2490, 10439], [0, 0, 0, 0], 4, [1, 0, 1, 1])
([2065, 3857, 2490, 10439], [0, 0, 0, 0], 4, [1, 0, 1, 1])
([2007, 2037, 4654, 3401, 18098, 3215], [0, 0, 0, 0, 0, 0], 6, [1, 1, 0, 0, 0, 0])
```

Figura 51. Ejemplo de la creación y unión de los tensores mxnet

Para la ejecución de todas las funciones presentadas, se crea la siguiente función donde utiliza estas funciones y se procesa en un bloque de código. Figura 52.

Proceso completo para las entradas del modelo

Aplicación de funciones anteriores. Aplicación de FixedBucketSampler para formar grupos de batches definidos por el tamaño de valid_len. Creació
DataLoader.

```
def load_data_wiki(corpus_OCR, corpus_REF, etiqueta, batch_size, max_len, vocab, ctx):
    num_workers = d2l.get_dataloader_workers()
    sentences_transform_OCR, sentences_transform_REF, etiquetas = transform_sentences(transform, corpus_OCR,
                                                                                      corpus_REF, etiqueta)
    pretrain, class_weights, REF_tokens, orig_etiquetas_positions = data_pretrain(sentences_transform_OCR,
                                                                                  sentences_transform_REF, vocab,
                                                                                  etiquetas, ctx)
    data_padded = pad_bert_inputs(pretrain, max_len, vocab)
    dataset = gluon.data.ArrayDataset(*data_padded)

    data_loader = gluon.data.DataLoader(dataset, batch_size=batch_size, num_workers=num_workers, shuffle=False)
    return data_loader, class_weights, REF_tokens, orig_etiquetas_positions, pretrain
```

Figura 52. Compresión del uso de funciones

Al observar la Figura 52, utiliza las funciones para crear el conjunto de oraciones que se utilizarán en el modelo, pero se ha mencionado la parte de etiquetas de relleno y aún no ha sido aplicada. Aquí actúa la función indicada en la Figura 53.

Asignación etiquetas de relleno

Completa cada muestra con el número de etiquetas de relleno faltantes y forma los tensores mxnet

```
def pad_bert_inputs(pretrain, max_len, vocab):
    all_token_ids, all_segments, all_valid_len, all_labels = [], [], [], []

    for (token_ids, segments, valid_len, labels) in pretrain:
        all_token_ids.append(token_ids + [vocab['[PAD]']] * (max_len - len(token_ids)))
        all_segments.append(segments + [0] * (max_len - len(token_ids)))
        all_valid_len.append(valid_len)
        all_labels.append(labels + [-1] * (max_len - len(token_ids)))

    all_token_ids_mx = mx.nd.array(all_token_ids, ctx=mx.cpu(0))
    all_segments_mx = mx.nd.array(all_segments, ctx=mx.cpu(0))
    all_valid_len_mx = mx.nd.array(all_valid_len,  ctx=mx.cpu(0))
    all_labels_mx = mx.nd.array(all_labels, ctx=mx.cpu(0))

    return (all_token_ids_mx, all_segments_mx, all_valid_len_mx, all_labels_mx)
```

Figura 53. Función que asigna las etiquetas de relleno y contextualiza los tensores

Como se ha indicado las etiquetas de relleno son fundamentales en la función de la Figura 52, ya que se creará bloques definidos que contienen cierto número de entradas para el modelo, por lo que si no son aplicadas no se podrá armar los bloques necesarios. Adicional a esto, también se contextualiza los tensores que en este caso antes de usarlos en el modelo serán almacenados en CPU.

2.3.3 Definición del modelo y aplicación de Fine Tuning

Como se indicó en la sección 1.4.12, se aplica un modelo previamente entrenado para poder aprovechar su estructura y realizar el Ajuste Fino (Fine Tuning) que consiste en obtener los datos del modelo BERT usado y encaminar sus resultados para la tarea de detección de errores, colocando las capas conectadas necesarias para crear el clasificador que asignará a cada token su respectiva clasificación.

La Figura 54, muestra la arquitectura aplicada para usar el modelo preentrenado y realizando el Ajuste Fino colocando las capas que serán aplicadas al clasificador.

Definición de la clase aplicando fine tuning para la clasificación binaria.

Clase para la clasificación binaria partiendo de las salidas del modelo preentrenado.

```
class WordClassificationModel(gluon.HybridBlock):
    def __init__(self, bert_model, num_classes):
        super(WordClassificationModel, self).__init__()
        with self.name_scope():
            self.bert_model = bert_model

            self.classifier = gluon.nn.HybridSequential()
            self.classifier.add(gluon.nn.Dense(units=768, flatten=False))
            self.classifier.add(gluon.nn.GELU())
            self.classifier.add(gluon.nn.LayerNorm(epsilon=1e-12))
            self.classifier.add(gluon.nn.Dense(units=num_classes, flatten=False))

    def forward(self, input_ids, segments, valid_len):
        bert_output, _ = self.bert_model(input_ids, segments, valid_len)

        logits = self.classifier(bert_output)

        return logits
```

Figura 54. Arquitectura del clasificador para el modelo de Detección de errores ortográficos

La arquitectura presentada presenta las siguientes capas:

- **Capa BERT (‘bert_model’):**

Como primera capa se aplica el modelo BERT preentrenado, la cual procesa las entradas definidas en la sección 2.3.2, retornado como resultados las representaciones vectoriales contextualizadas con respecto a cada token entregado.

- **Capa Clasificadora (‘classifier’):**

La capa de clasificación está formada por la secuencia de una capa densa directamente conectada realizando una transformación lineal que toma como entradas la salida de la capa BERT y retornará el mismo número (768) de salidas por token, la función de activación

GELU indicada para esta tarea en específico, la capa de normalización que normaliza las activaciones de la capa anterior para que tengan una media cercana a cero y por otro lado un desviación estándar cercana a uno y la última capa densa que retorna la clasificación para cada token.

- **Forward:**

Esta función toma los tensores de entrada y se encarga de pasarlos por la capa BERT, obteniendo las representaciones contextualizadas, para luego ser enviadas al clasificador y retornar la clasificación.

Definida la arquitectura del modelo, se debe inicializar tanto el modelo preentrenado, Figura 46, como las capas agregadas para el clasificador. Además de instanciar el optimizador, la función de pérdida y el entrenador (trainer) y el congelamiento de las capas del modelo BERT, ver Figura 47.

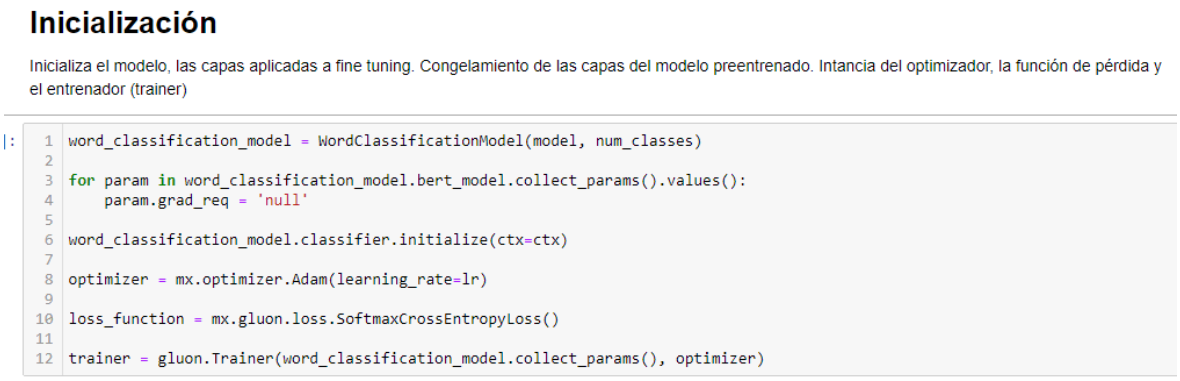


Figura 55. Inicialización e instancia de las capas y parámetros para el entrenamiento

El modelo BERT y la capa ha sido inicializado con respecto al contexto de los dispositivos usados, en este caso se usarán CPUs. El término congelar las capas del modelo se refiere a que las capas del modelo se las establece como no entrenables, pero en este caso no solo se aplicará a las capas sino a los parámetros del modelo, esto ya que no se requiere que el modelo aprenda nuevamente, sino que sea un apoyo para la capa clasificadora que es la que debe entrenarse para cumplir su tarea. El optimizador usado es Adam y la función de pérdida es SoftmaxcCrossEntropyLoss, los más usados para este tipo de modelo. Y finalmente se instancia el entrenador con los parámetros de la arquitectura del modelo y el optimizador.

2.3.4 Rutina de entrenamiento

La rutina de entrenamiento consiste en tres partes, la función para calcular la pérdida por cada bloque de oraciones (Figura 56), la función de evaluación (Figura 57) que actúa al finalizar cada época y el entrenamiento (Figura 58) usando las dos partes anteriores

```
def get_batch_loss(word_classification_model, loss_function, token_ids_shards, segments_shards, valid_len_shards,
                   labels_shards):
    ls = []
    outs = []

    for (token_ids_shard, segments_shard, valid_len_shard, labels_shard) in zip(token_ids_shards, segments_shards,
                                                                                valid_len_shards, labels_shards):
        out = word_classification_model(token_ids_shard, segments_shard, valid_len_shard)

        masked_losses = []
        for i in range(out.shape[0]):
            loss = loss_function(out[i], labels_shard[i])
            mask = (labels_shard[i] != -1).astype(out.dtype)
            masked_loss = ((loss * mask).sum() / mask.sum())
            masked_losses.append(masked_loss)
        # Calcular el promedio de masked_losses y agregarlo a ls
        average_loss = sum(masked_losses) / len(masked_losses)
        ls.append(average_loss)

        outs.append(out)
    mx.nd.waitall()
    return ls, outs
```

Figura 56. Función para el cálculo de la pérdida del modelo al entrenarse

Al dividirse los bloques (batches) de oraciones por dispositivos se calcula tanto las salidas del modelo como la pérdida por cada dispositivo. El modelo BERT no tiene problemas con las etiquetas de relleno, ya que están son excluidas y no interfieren durante el entrenamiento de las capas, caso contrario si interfiere con la obtención de la pérdida. Para este caso se crea una máscara que contiene los ids válidos y excluye las etiquetas de relleno (en este caso -1) obteniendo una pérdida nítida y sin intervención de las etiquetas de relleno. Al obtener la pérdida por cada oración, esto debido a que cada oración tiene un tamaño variable con respecto a los tokens válidos, se realiza un promedio y se agrega a una lista de pérdidas que serán retornadas al igual que las salidas del modelo (se indexan a una lista porque se procesan en distintos dispositivos). Para que los dispositivos logren terminar en conjunto se aplica una sentencia de espera para que todos los dispositivos concluyan.

Función de evaluación del modelo

```
def evaluate_model(model, data_loader, loss_function, ctx):
    metric = d2l.Accumulator(3)
    batch_predictions, batch_labels, batch_probabilities = [], [], []

    for batch in data_loader:
        (token_ids_shards, segments_shards, valid_len_shards, labels_shards) =
        [gluon.utils.split_and_load(elem, ctx, even_split=False) for elem in batch]

        ls, outs = get_batch_loss(model, loss_function, token_ids_shards, segments_shards, valid_len_shards, labels_shards)

        l_mean = sum([float(l.asscalar()) for l in ls]) / len(ls)

        metric.add(l_mean, batch[0].shape[0], 1)

        for (out, labels_shard, valid_len_shard) in zip(outs, labels_shards, valid_len_shards):
            probabilities = mx.nd.softmax(out, axis=-1)
            predictions = probabilities.argmax(axis=-1)

            batch_predictions.append(predictions.asnumpy())
            batch_labels.append(labels_shard.asnumpy())
            batch_probabilities.append(probabilities.asnumpy())

    test_loss = metric[0] / metric[2]

    batch_predictions = np.concatenate(batch_predictions)
    batch_labels = np.concatenate(batch_labels)
    batch_probabilities = np.concatenate(batch_probabilities)

    return test_loss, batch_predictions, batch_labels, batch_probabilities
```

Figura 57. Función de evaluación del modelo

La función de evaluación del modelo tiene una estructura similar a la rutina de entrenamiento, solo que en este caso sirve para evaluar que tan bien aprende el modelo en el transcurso de las épocas, por tal motivo siempre es aplicada al finalizar el entrenamiento en cada época. Retorna la pérdida como un valor escalar y además las etiquetas y las probabilidades usadas para aplicarlas a la evaluación de las métricas al terminar el entrenamiento.

Rutina de entrenamiento del modelo

```
train_epoch_predictions, train_epoch_labels, train_epoch_probabilities = [], [], []
eval_epoch_predictions, eval_epoch_labels, eval_epoch_probabilities = [], [], []

total_timer = d2l.Timer()

animator = d2l.Animator(xlabel='Épocas', ylabel='Métricas', xlim=[1, num_epochs], ylim=[0.0, 1.0],
                        legend=['train_loss', 'eval_loss', 'train_acc'], figsize=(6, 4))

for epoch_id in range(num_epochs):
    metric = d2l.Accumulator(3)
    batch_predictions, batch_labels, batch_probabilities = [], [], []

    for batch in train_loader:
        (token_ids_shards, segments_shards, valid_len_shards, labels_shards) =
        [gluon.utils.split_and_load(elem, ctx, even_split=False) for elem in batch]

        with mx.autograd.record():
            ls, outs = get_batch_loss(word_classification_model, loss_function, token_ids_shards,
                                      segments_shards, valid_len_shards, labels_shards)

        for l in ls:
            l.backward()

        trainer.step(1)

        l_mean = sum([float(l.asscalar()) for l in ls]) / len(ls)

        for (out, labels_shard) in zip(outs, labels_shards):
            probabilities = mx.nd.softmax(out, axis=-1)
            predictions = probabilities.argmax(axis=-1)

            batch_predictions.append(predictions.asnumpy())
            batch_labels.append(labels_shard.asnumpy())
            batch_probabilities.append(probabilities.asnumpy())

        metric.add(l_mean, batch[0].shape[0], 1)

    train_loss = metric[0] / metric[2]

    epoch_predictions = np.concatenate(batch_predictions)
    epoch_labels = np.concatenate(batch_labels)
    epoch_probabilities = np.concatenate(batch_probabilities)

    train_epoch_predictions.append(epoch_predictions)
    train_epoch_labels.append(epoch_labels)
    train_epoch_probabilities.append(epoch_probabilities)

    test_loss, test_predictions, test_labels, test_probabilities = evaluate_model(word_classification_model,
                                                                                   test_loader, loss_function, ctx)
    eval_epoch_predictions.append(test_predictions)
    eval_epoch_labels.append(test_labels)
    eval_epoch_probabilities.append(test_probabilities)

    animator.add(epoch_id + 1, (train_loss, test_loss))

total_timer.stop()
```

Figura 58. Rutina de entrenamiento

La rutina de entrenamiento contiene los siguientes componentes:

- Se guarda en listas todos los valores y resultados obtenidos durante el entrenamiento y la evaluación. Se guardan las etiquetas verdaderas, las predicciones y las probabilidades para posteriores análisis.
- Calcula el tiempo total de entrenamiento y genera una gráfica interactiva que muestra cómo se actualiza la pérdida en el transcurso de las épocas, esto para controlar cual es el progreso de la pérdida del modelo.

- La rutina de entrenamiento empieza en la iteración de las épocas, donde se toman los bloques (batches) generados anteriormente y los divide según la cantidad de dispositivos usados. Y esta división es pasada a la función que calcula la pérdida del entrenamiento del modelo en la función de la Figura 56. Se procesa cada una de las listas de pérdidas con el proceso de retropropagación (backward), siendo la parte esencial del entrenamiento ya que se calculan los gradientes de la función de pérdida con respecto a los pesos de la red y posteriormente realizar una optimización para el ajuste de pesos del modelo en la dirección que minimiza la pérdida.
- Se calcula la pérdida en cada época y este valor es el que se muestra en la gráfica.

2.3.5 Evaluación post-entrenamiento

La evaluación después del entrenamiento consiste en obtener los valores de precisión, exhaustividad (recall), exactitud (accuracy) y medida F1 esto para cada época, por esta razón se guardó las etiquetas verdaderas, probabilidades y predicciones durante el entrenamiento y en cada época. El código implementado se muestra en la Figura 59.

Métricas Entrenamiento

Obtención de las métricas anteriores definidas a nivel de época, esto para obtener una gráfica de métricas.

```
precision_list_train, recall_list_train, f1_list_train, accuracy_list_train = [], [], [], []

for i in range(len(train_epoch_labels)):  # Recorrido por época
    TP_epoch, TN_epoch, FP_epoch, FN_epoch = 0, 0, 0, 0

    for j in range(len(train_epoch_labels[i])):  # Recorrido por muestras en cada época
        valid_indices = np.where(train_epoch_labels[i][j] != -1) # exlcusión de etiquetas [PAD] == -1

        valid_train_epoch_labels = train_epoch_labels[i][j][valid_indices]
        valid_train_epoch_predictions = train_epoch_predictions[i][j][valid_indices]

        TP, TN, FP, FN = 0, 0, 0, 0

        # Calcula TP, TN, FP, FN para la época actual
        for idx in range(len(valid_train_epoch_labels)):
            if valid_train_epoch_labels[idx] == 1 and valid_train_epoch_predictions[idx] == 1:
                TP += 1
            elif valid_train_epoch_labels[idx] == 0 and valid_train_epoch_predictions[idx] == 0:
                TN += 1
            elif valid_train_epoch_labels[idx] == 0 and valid_train_epoch_predictions[idx] == 1:
                FP += 1
            elif valid_train_epoch_labels[idx] == 1 and valid_train_epoch_predictions[idx] == 0:
                FN += 1

        TP_epoch += TP
        TN_epoch += TN
        FP_epoch += FP
        FN_epoch += FN

    # Calcula las métricas para la época actual usando fórmulas
    precision = TP_epoch / (TP_epoch + FP_epoch) if (TP_epoch + FP_epoch) != 0 else 0
    recall = TP_epoch / (TP_epoch + FN_epoch) if (TP_epoch + FN_epoch) != 0 else 0
    accuracy = (TP_epoch + TN_epoch) / (TP_epoch + TN_epoch + FP_epoch + FN_epoch)
                if (TP_epoch + TN_epoch+ FP_epoch + FN_epoch) != 0 else 0
    f1 = (2 * precision * recall) / (precision + recall) if (precision + recall) != 0 else 0

    # Almacena las métricas para la época actual
    precision_list_train.append(precision)
    recall_list_train.append(recall)
    f1_list_train.append(f1)
    accuracy_list_train.append(accuracy)
```

Figura 59. Evaluación de métricas

Para a evaluación de las métricas, se obtienen los valores de la matriz de confusión a nivel de época esto para calcular las métricas y graficarlas para un mejor entendimiento. Se obtienen los parámetros que indican Verdaderos Positivos (TP), Falsos Positivos (FP), Verdaderos Negativos (TN) y Falsos Negativos (FN). Esto para posteriormente usar las fórmulas para obtener la precisión, exhaustividad, exactitud y medida F1.

2.3.6 Aplicación de un modelo Baseline para comparar con el modelo de Detección de errores ortográficos

La Figura 60 muestra cómo se estructuró el modelo Baseline tomando como referencia el tamaño de las etiquetas verdaderas del grupo de entrenamiento. Se aplican los pesos obtenidos en la Figura 50, esto para representar la distribución de etiquetas de clase 0 y clase 1, esto para simular un entorno parecido al usado en el entrenamiento del modelo. De manera aleatoria se calcula las probabilidades tomando en cuenta la distribución de etiquetas y con base a estas probabilidades se obtiene las predicciones que el modelo Baseline puede detectar. Como se aplica de manera aleatoria, se debe aplicar una semilla para poder replicar los mismos resultados en cada ejecución del código.

Aplicación de baseline

Usando random choise y los pesos obtenidos anteriormente, definir resultados base para comparar con los resultados del modelo y comparar el rendimiento entre los dos.

```
last_train_labels = train_epoch_labels[-1]
flat_last_train_labels = np.array([label for sublist in last_train_labels for label in sublist])
valid_indices_train = np.where(flat_last_train_labels != -1) # excluye etiquetas [PAD] == -1
filtered_train_labels = flat_last_train_labels[valid_indices_train]

weight_class_0 = total_weights[0].asscalar()
weight_class_1 = total_weights[1].asscalar()

k = filtered_train_labels.size

k_class_0 = int(k * weight_class_0)
k_class_1 = k - k_class_0

np.random.seed(42)  # Cambia la semilla a un valor diferente

probabilities_class_0 = np.random.uniform(0, 0.5, size=k_class_0)
probabilities_class_1 = np.random.uniform(0.5, 1.0, size=k_class_1)

print("probabilities_class_0: ", len(probabilities_class_0), "probabilities_class_1: ", len(probabilities_class_1))
all_probabilities = np.concatenate([probabilities_class_0, probabilities_class_1])
print("all_probabilities: ", len(all_probabilities))
random.shuffle(all_probabilities)

random_probabilities = np.array([[1 - prob, prob] for prob in all_probabilities[:k]])

threshold = 0.5
random_predictions = np.where(random_probabilities[:, 1] <= threshold, 0, 1).astype(float)

random_probabilities = np.column_stack([1 - all_probabilities, all_probabilities])
```

Figura 60. Modelo Baseline

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

Los resultados obtenidos se trabajaron sobre el entorno Jupyter, alojado en los servidores de la Escuela Politécnica por lo que los componentes conocidos y utilizados en este proceso es la cantidad de CPUS disponibles que son 10, de los cuales se utilizó 8 para la asignación de bloques de oraciones para cada dispositivo.

3.1 Resultados

Debido a la falta de recursos, para el entrenamiento del modelo se aplicaron pequeños conjuntos de oraciones con respecto a toda la data completa. El número de oraciones totales, tanto para entrenamiento como para evaluación, se expresa en la Tabla 1.

Tabla 1. Información del conjunto total de oraciones

Nombre del archivo	corpusOCR		corpusREF	
	Entrenamiento	Evaluación	Entrenamiento	Evaluación
Número de oraciones realizado el procesamiento inicial y la alineación	237 097	12 463	237 097	12 463
Número de oraciones al limitar el número de palabras en 20	310 260	16 302	310 260	16 302
Número de oraciones para los tensores mxnet excluyendo aquellas que no contienen errores y tienen menos de 3 palabras.	184 072	9 592	184 072	9 592

Además, hay que considerar la frecuencia de la clase 0 con respecto a la clase 1, donde para la clase 0 le corresponde el 23.16% y para la clase 1 el 76.83%, tomando una relación aproximada de 25-75.

Para el entrenamiento y verificación de la implementación del modelo se aplicó tres conjuntos de oraciones. Antes de indicar estos conjuntos, en la Tabla 2, se muestran los parámetros generales usados para el entrenamiento del modelo con distintos conjuntos de oraciones.

Tabla 2. Parámetros generales usados en el entrenamiento

Parámetro	**Cantidad**
CPUs	8
Número máximo de palabras	20
Número máximo de tokens	25
Tamaño del bloque de oraciones (batch)	128
Número de clases (correcto, incorrecto)	2
Tasa de aprendizaje (learning rate)	0.00001
Número mínimo de tokens	3

Ahora, tomando en cuenta estos parámetros se aplicó los siguientes conjuntos de oraciones y el número de épocas aplicadas en la Tabla 3. Adicional se muestra el tiempo que tardó cada conjunto de oraciones en entrenarse.

Tabla 3. Conjuntos de oraciones usados para el entrenamiento

		Entrenamiento	Evaluación
Conjunto de oraciones 1	Número de oraciones corpus original	300	100
	Número de oraciones con el límite de 20 palabras	399	125

	Número de oraciones transformadas a tensores mxnet	216	80
	Épocas	30	
	Tiempo de ejecución	3334.3520 segundos - 55 minutos	
Conjunto de oraciones 2	Número de oraciones corpus original	3 000	1 000
	Número de oraciones con el límite de 20 palabras	4 093	1 227
	Número de oraciones transformadas a tensores mxnet	2 320	776
	Épocas	30	
	Tiempo de ejecución	34501.0175 segundos – 10 horas	
Conjunto de oraciones 3	Número de oraciones corpus original	10 000	3 000
	Número de oraciones con el límite de 20 palabras	12 940	3 900
	Número de oraciones transformadas a tensores mxnet	7 688	2 304
	Épocas	10	
	Tiempo de ejecución	37524.9645 segundos – 10 horas y media	

Las gráficas de pérdida obtenidas se muestran en la Figura 61, Figura 62 y Figura 63.

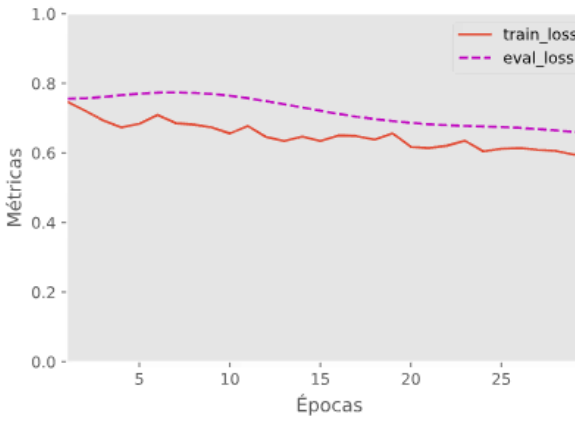


Figura 61. Gráfica de pérdidas de entrenamiento y evaluación del conjunto de oraciones 1

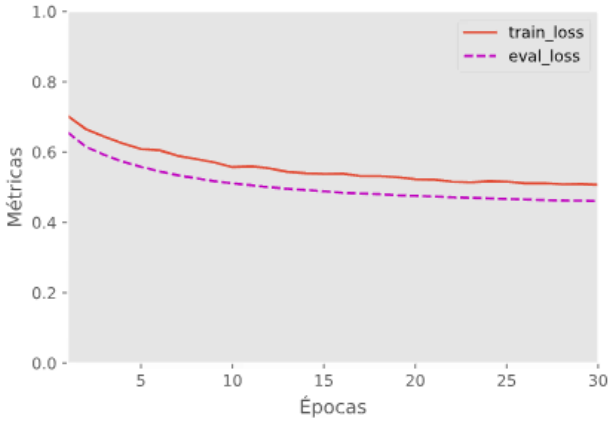


Figura 62. Gráfica de pérdidas de entrenamiento y evaluación del conjunto de oraciones 2

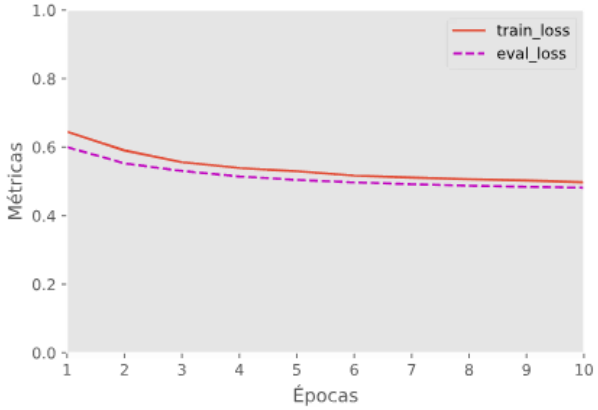


Figura 63. Gráfica de pérdidas de entrenamiento y evaluación del conjunto oraciones 3

En las gráficas de pérdidas de los 3 casos, se observa que, al implementar pocas oraciones, tanto la curva de entrenamiento como la de evaluación presentan variaciones en su formación, por lo que no presenta una disminución totalmente exponencial. Por otro lado, al experimentar con más oraciones las curvas toman una disminución exponencial más pronunciada, indicando un correcto aprendizaje por parte del modelo y en cada época disminuye la pérdida en sus datos.

Las gráficas de las otras métricas calculadas, precisión, exactitud, exhaustividad y medida F1 obtenidas del entrenamiento y evaluación están representadas en la Figura 64, Figura 65, Figura 66, Figura 67, Figura 68 y Figura 69.

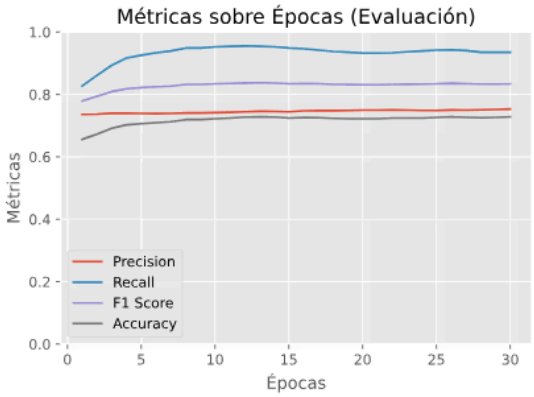


Figura 64. Métricas para el entrenamiento del conjunto de oraciones 1

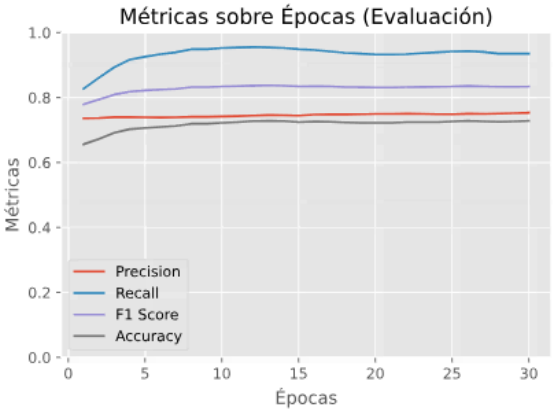


Figura 65. Métricas para la evaluación del conjunto de oraciones 1

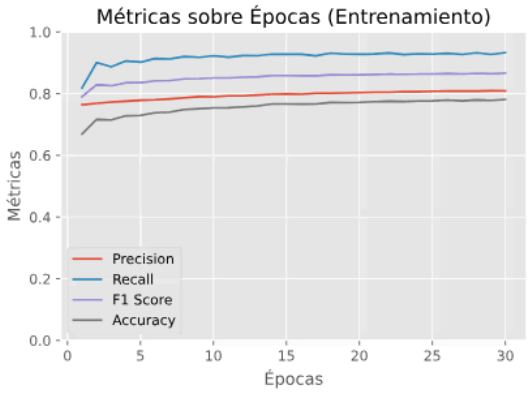


Figura 66. Métricas para el entrenamiento del conjunto de oraciones 2

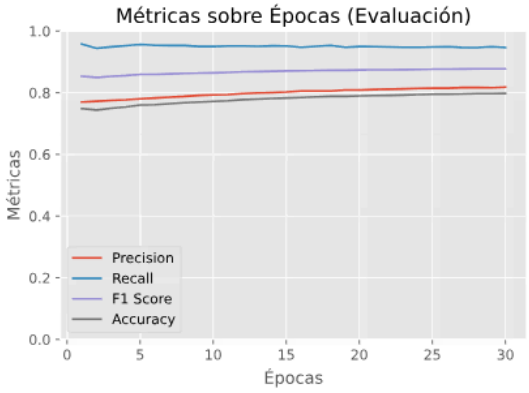


Figura 67. Métricas para la evaluación del conjunto de oraciones 2

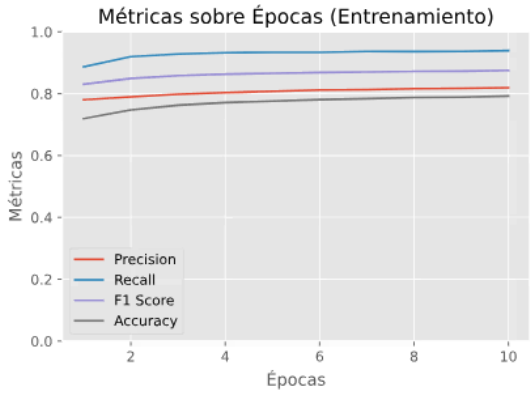


Figura 68. Métricas para el entrenamiento del conjunto de oraciones 3

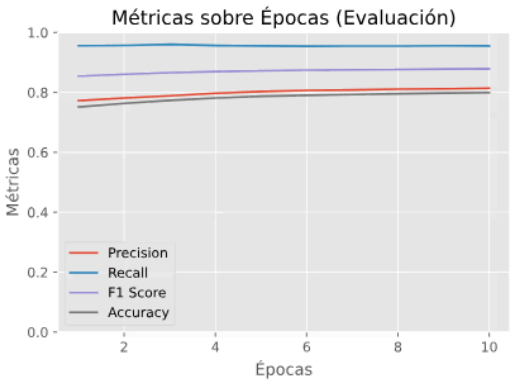


Figura 69. Métricas para la evaluación del conjunto de oraciones

Con respecto a las gráficas de las métricas, se observa que en todos los casos la exhaustividad (recall) obtiene un valor más alto. Esto indica que tiene una alta exhaustividad con respecto al desbalance de clases debido a que la proporción de la clase 0 con respecto a la clase 1 es de 25-75, existiendo una gran mayoría de palabras correctas. En el caso de la precisión presenta una forma constante cerca del 80%, esto indicaría que tiene una alta precisión al detectar correctamente cada una de las etiquetas, teniendo sentido que la exactitud (accuracy) presente un porcentaje más alto en tendencia a crecer, lo que indica que el aprendizaje del modelo es el indicado ya al asignarle más oraciones estos valores podrán aumentar e incluso normalizarse con respecto al desbalance de las clases. La médica F1 está relacionada con la precisión y la exhaustividad indicando que está dentro del rango de estos valores.

Los valores de las métricas de la última época de cada conjunto de oraciones está en la Tabla 4. Estos resultados indican resultados entre los tamaños de los conjuntos indicando que al entrenar con más oraciones y en menos épocas se obtienen resultados satisfactoriamente altos evitando el sobreajuste y que el modelo aprenda y no obtenga resultados memorizados.

Tabla 4. Métricas de la última época de cada conjunto de oraciones

		Entrenamiento	Evaluación
Conjunto de oraciones 1	Precisión	0.7829	0.7532
	Exhaustividad (recall)	0.9029	0.9352
	Medida F1	0.8386	0.8344
	Exactitud (accuracy)	0.7356	0.7282
Conjunto de oraciones 2	Precisión	0.8091	0.8181
	Exhaustividad (recall)	0.9331	0.9465
	Medida F1	0.8667	0.8776
	Exactitud (accuracy)	0.7810	0.7977
Conjunto de oraciones 3	Precisión	0.8190	0.8141
	Exhaustividad (recall)	0.9394	0.9550
	Medida F1	0.8751	0.8790
	Exactitud (accuracy)	0.7923	0.7996

La Tabla 5, Tabla 6, Tabla 7, Tabla 8, Tabla 9 y Tabla 10 muestran las matrices de confusión de los conjuntos de oraciones del entrenamiento y de la evaluación

Tabla 5. Matriz de confusión del entrenamiento del conjunto de oraciones 1

	Predicción Negativa	Predicción Positiva
Actual Negativa	150 (TN)	588 (FP)
Actual Positiva	228 (FN)	2121 (TP)

Tabla 6. Matriz de confusión de la evaluación del conjunto de oraciones 1

	Predicción Negativa	Predicción Positiva
Actual Negativa	44 (TN)	227 (FP)
Actual Positiva	48 (FN)	693 (TP)

Tabla 7. Matriz de confusión del entrenamiento del conjunto de oraciones 2

	Predicción Negativa	Predicción Positiva
Actual Negativa	2254 (TN)	5507 (FP)
Actual Positiva	1672 (FN)	23349 (TP)

Tabla 8. Matriz de confusión de la evaluación del conjunto de oraciones 2

	Predicción Negativa	Predicción Positiva
Actual Negativa	779 (TN)	1746 (FP)
Actual Positiva	444 (FN)	7857 (TP)

Tabla 9. Matriz de confusión del entrenamiento del conjunto de oraciones 3

	Predicción Negativa	Predicción Positiva
Actual Negativa	7029 (TN)	17463 (FP)
Actual Positiva	5095 (FN)	79058 (TP)

Tabla 10. Matriz de confusión de la evaluación del conjunto de oraciones 3

	Predicción Negativa	Predicción Positiva
Actual Negativa	2312 (TN)	5358 (FP)
Actual Positiva	1104 (FN)	23474 (TP)

Con respecto a los valores obtenidos en los verdaderos positivos (clase 1) y los verdaderos negativos (clase 0), se observa un gran porcentaje de predicciones positivas, por lo que es un indicador de que el modelo detecta correctamente las palabras correctas. Por otro lado,

los verdaderos negativos presentan un bajo valor con respecto a los falsos negativos, por lo que es un indicador de que el modelo está colocando incorrectamente la clase 0. Estos resultados obtenidos debido al desbalance de las clases.

Las métricas obtenidas del modelo Baseline para los conjuntos de oraciones se presentan en la Tabla 11, Tabla 12, Tabla 13, Tabla 14, Tabla 15 y Tabla 16.

Tabla 11. Métricas del modelo Baseline del conjunto de oraciones 1

Precisión	Exhaustividad (recall)	Medida F1	Exactitud (accuracy)
0.7535	0.7394	0.7464	0.6177

Tabla 12. Matriz de confusión del modelo Baseline del conjunto de oraciones 1

	Predicción Negativa	Predicción Positiva
Actual Negativa	170 (TN)	568 (FP)
Actual Positiva	612 (FN)	1737 (TP)

Tabla 13. Métricas del modelo Baseline para el conjunto de oraciones 2

Precisión	Exhaustividad (recall)	Medida F1	Exactitud (accuracy)
0.7643	0.7661	0.7652	0.6412

Tabla 14. Matriz de confusión del modelo Baseline para el conjunto de oraciones 2

	Predicción Negativa	Predicción Positiva
Actual Negativa	1851 (TN)	5910 (FP)
Actual Positiva	5852 (FN)	19169 (TP)

Tabla 15. Métricas del modelo Baseline para el conjunto de oraciones 3

Precisión	Exhaustividad (recall)	Medida F1	Exactitud (accuracy)
0.7752	0.7690	0.7720	0.6483

Tabla 16. Matriz de confusión del modelo Baseline para el conjunto de oraciones 3

	Predicción Negativa	Predicción Positiva
Actual Negativa	5727 (TN)	18765 (FP)
Actual Positiva	19439 (FN)	64714 (TP)

Al comparar los resultados de las métricas y de las matrices de confusión del modelo entrenado para detección de errores ortográficos y del modelo Baseline, se observa una mejoría en el modelo entrenado sobre las predicciones que se realizan al azar en un entorno parecido al entramiento (simulando la distribución de clases en la relación 25-75)

Finalmente se muestra la gráfica de la curva ROC-AUC del entrenamiento y de la evaluación, comparadas con el modelo Baseline en la Figura 70, Figura 71, Figura 72, Figura 73, Figura 74 y Figura 75.

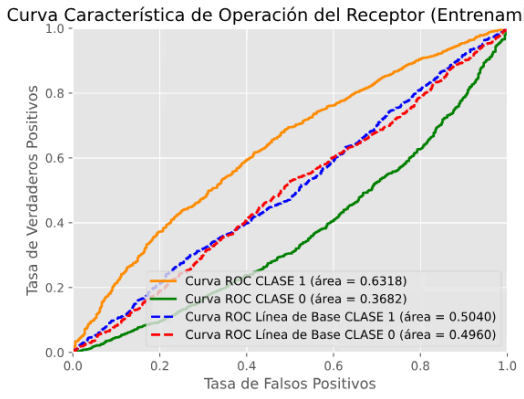


Figura 70. Curva ROC del entrenamiento del conjunto de oraciones 1

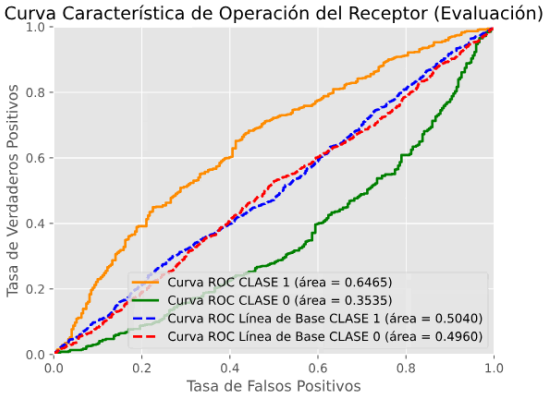


Figura 71. Curva ROC de la evaluación del conjunto de oraciones 1

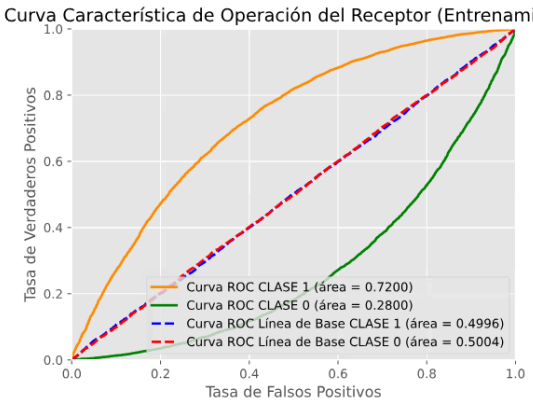


Figura 72. Curva ROC del entrenamiento del conjunto de oraciones 2

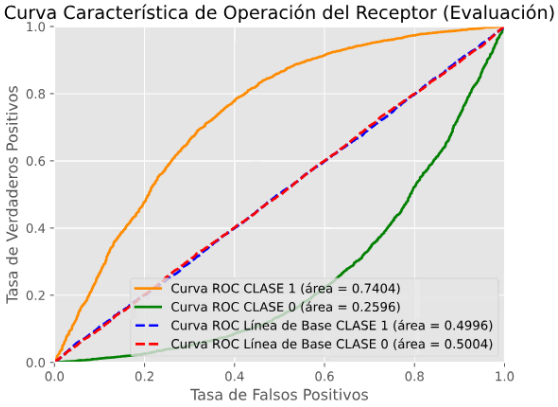


Figura 73. Curva ROC de la evaluación del conjunto de oraciones 2

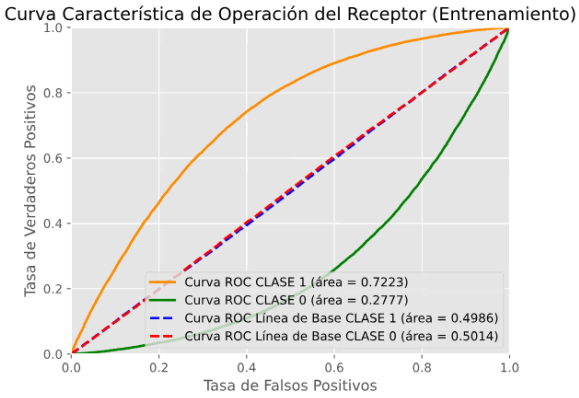


Figura 74. Curva ROC del entrenamiento del conjunto de oraciones 3

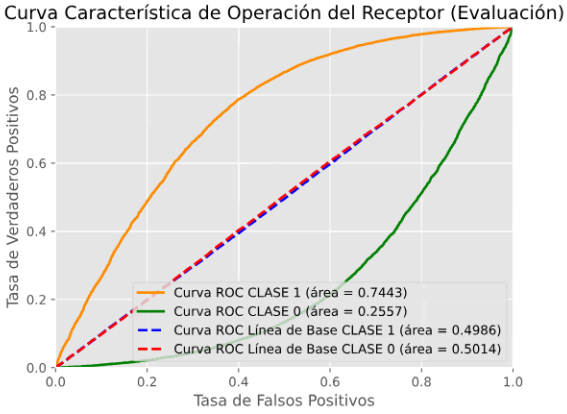


Figura 75. Curva ROC de la evaluación del conjunto de oraciones 3

Revisando las gráficas ROC obtenidas en cada conjunto de oraciones, se observa con respecto a la distribución de clases en relación 25-75, que representa como el rendimiento del modelo actúa con respecto a las clases, al ser un problema binario actúa en relación con detectar la clase 1 y la clase 0. Con la relación indicada el valor de la curva para la clase 1 es cercana al 0.7, considerando que el 75% de las clasificaciones es con respecto a esta distribución se podría considerar que tiene una tasa alta de aprendizaje. Lo mismo actuaría frente a la clase 0, ya que colocándolo en proporción su valor está cerca de 0.25, considerando de igual manera una buena tasa de aprendizaje.

Finalmente, para obtener una mejor visión con respecto al rendimiento de las predicciones del modelo, se compara sus resultados con el corpus de Referencia, dando así una mejor visualización de cómo se están asignando las clases correctas e incorrectas.

La Tabla 17 muestra ejemplo de cómo clasificó el modelo las palabras y como deben ser exactamente.

Tabla 17. Ejemplos de comparación entre las predicciones del modelo y el corpusREF

Referencia	[1, 1, 1, 1, 0, 0]	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0]	[1, 0, 1, 1]	[1, 0, 1, 1]
Predicción	[1, 1, 0, 0, 1, 0]	[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]	[1, 1, 1, 1]	[1, 0, 1, 0]
TP	2	3	3	2
TN	1	3	0	1
FP	2	8	0	1
FN	1	0	1	0
Token ids (referencia)	[5310, 2034, 18442, 24501, 2034, 18442]	[2017, 2064, 4487, 19150, 7928, 2005, 1996, 2783, 2695, 2478, 1996, 2053, 9006, 8163]	[2065, 12508, 2490, 10439]	[2065, 12508, 2490, 10439]
Token ids (predicción)	[1, 1, 0, 0, 1, 0]	[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]	[1, 1, 1, 1]	[1, 0, 1, 0]
Tokens (referencia)	['user', 'first', '##name', 'res', 'first', '##name']	['you', 'can', 'di', '##sable', 'comments', 'for', 'the', 'current', 'post', 'using', 'the', 'no', '##com', '##ments']	['if', 'builder', 'support', 'app']	['if', 'builder', 'support', 'app']
Tokens (predicción)	['correcto', 'correcto', 'incorrecto', 'incorrecto', 'correcto', 'incorrecto']	['correcto', 'correcto', 'incorrecto', 'incorrecto', 'incorrecto', 'incorrecto', 'incorrecto', 'incorrecto', 'incorrecto', 'incorrecto', 'correcto', 'incorrecto', 'incorrecto', 'incorrecto']	['correcto', 'correcto', 'correcto', 'correcto']	['correcto', 'incorrecto', 'correcto', 'incorrecto']
Palabras (referencia)	user firstname res firstname	you can disable comments for the current	if builder support app	if builder support app

		post using the nocomments		
Palabras (predicción)	correcto incorrecto incorrecto incorrecto	Correcto correcto incorrecto incorrecto incorrecto incorrecto incorrecto incorrecto incorrecto incorrecto	correcto correcto correcto correcto	correcto incorrecto correcto incorrecto
Tokens (OCR)	[5310, 2034, 18442, 24501, 2197, 18442]	[2017, 2064, 4487, 19150, 7928, 2005, 1996, 2783, 2695, 2478, 1996, 2053, 9006, 3672]	[2065, 3857, 2490, 10439]	[2065, 3857, 2490, 10439]
Token ids (OCR)	['user', 'first', '##name', 'res', 'last', '##name']	['you', 'can', 'di', '##sable', 'comments', 'for', 'the', 'current', 'post', 'using', 'the', 'no', '##com', '##ment']	['if', 'build', 'support', 'app']	['if', 'build', 'support', 'app']
Palabras (OCR)	user firstname res lastname	you can disable comments for the current post using the nocomment	if build support app	if build support app

Observando los ejemplos aquí mostrados, existen algunas cosas a considerar, para las oraciones pequeñas, es más precisa la detección de los errores gramaticales debido a que las clases están más balanceadas con respecto a las palabras correctas e incorrecta. Por otro lado, al observar las oraciones más grandes, sobresale el desbalance de la clase 1 por lo que el modelo deberá procesar más información para detectar estos inconvenientes y sea más consistente con el desbalance de clases. Lo cual se logrará realizando un entrenamiento más exhaustivo y con un volumen de oraciones más grande.

3.2 Conclusiones

- Los resultados obtenidos, pese a que se trabajó con un volumen de oraciones considerablemente bajo, se considera bien entrenado debido a los resultados obtenidos con respecto al modelo Baseline, ya que logra mejores resultados que un modelo que se ha configurado para predecir al azar.
- El correcto procesamiento del corpus paralelo desde un principio contribuye a una buena alineación con el algoritmo correcto, ya que, si uno de estos procesos produce errores, la obtención de las etiquetas verdaderas tendrá inconsistencias y debido a esto el modelo podrá malinterpretar los resultados.
- El modelo BERT usado requiere de cierto formato con respecto a las entradas que recibe, por tal motivo la formación de los tensores que contienen la información necesaria como los ids de las oraciones tokenizadas, el tamaño válido que no considera las etiquetas de relleno es fundamentales para que el modelo contextualice correctamente los datos asignados
- La técnica de Ajuste Fino (Fine Tuning) es muy provechosa, ya que al aplicarla se pueden integrar distintas tareas a partir de un modelo que se ha entrenado, ya sea usando capas y parámetros para volver a entrenar el modelo o simplemente usarlo para obtener las representaciones contextualizadas.
- La forma en como el modelo está siendo entrenado es que se disperse en distintos dispositivos ya sean CPUs o GPUs, esto para acelerar el tiempo de ejecución aplicando paralelismo.

3.3 Recomendaciones

- El modelo a pesar de ser entrenado para la clasificación binaria (clase correcta e incorrecta) se lo puede modificar para que detecte otro tipo de errores, ya sean sustitución, transposición, eliminación y agregación de caracteres. Esto se lo puede hacer mediante la correcta detección de estos errores en particular usando los resultados de la alineación.
- Pese a que el modelo se limitó a la detección de errores ortográficos, puede ser modificado para otra tarea que sería la corrección de errores. Esto se puede seguir aplicando con la técnica de Ajuste Fino, ya que el modelo por defecto está

entrenado para predecir palabras mascaradas y adaptando esta tarea podría ajustarse a la predicción de los errores y dar una o más opciones de corrección.

- Este tipo de modelos requieren de un gran componente computacional para ser completamente robusto, para esto se requiere del uso de GPUs avanzados que sean capaces de soportar cantidades masivas de texto y procesarlos.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] "IEEE Xplore Full-Text PDF:" https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9151144

[2] "Text degradations and OCR training," *IEEE Conference Publication | IEEE Xplore*, 2005. https://ieeexplore.ieee.org/abstract/document/1575662

[3] S. S. B. Estrada, G. M. Romero, and P. Pérez, "Los problemas de identificación de caracteres OCR para la recuperación de texto en el libro antiguo: un análisis de caso en el Fondo Antiguo de la Biblioteca Central, UNAM," *Biblioteca Universitaria*, vol. 15, no. 1, pp. 25–34, Jun. 2012, doi: 10.22201/dgb.0187750xp.2012.1.39.

[4] "Eureka | Digitising historical documents." https://www.eurekanetwork.org/blog/digitising-historical-documents

[5] R. Merritt, "¿Qué Es un Modelo Transformer? | Blog de NVIDIA," *Blog Oficial De NVIDIA Latino América*, Apr. 19, 2022. https://la.blogs.nvidia.com/2022/04/19/que-es-un-modelo-transformer/

[6] J. Vig, "Analyzing the structure of attention in a transformer language model," *arXiv.org*, Jun. 07, 2019. https://arxiv.org/abs/1906.04284

[7] N. Heidloff, "Foundation Models, Transformers, BERT and GPT," *Niklas Heidloff*, Feb. 24, 2023. https://heidloff.net/article/foundation-models-transformers-bert-and-gpt/

[8] J. Devlin, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv.org*, Oct. 11, 2018. https://arxiv.org/abs/1810.04805

[9] Getting Started with Google BERT. (n.d.). Google Books. https://books.google.es/books?hl=es&lr=&id=CvsWEAAAQBAJ&oi=fnd&pg=PP1&dq=bert+natural+language+processing&ots=3HeKr2pac5&sig=O9jYbUkL4Kztxce6fMks8eTFxg4#v=onepage&q&f=false

[10] *google-bert/bert-base-uncased · Hugging Face*. (n.d.). https://huggingface.co/google-bert/bert-base-uncased#bert-base-model-uncased

[11] Kumar, A. (2024, January 13). *BERT vs GPT Models: Differences, Examples*. Analytics Yogi. https://vitalflux.com/bert-vs-gpt-differences-real-life-examples/

[12] *BERT tokenization*. (2023, March 14). https://tinkerd.net/blog/machine-learning/bert-tokenization/

[13] *WordPiece tokenization - Hugging Face NLP Course*. (n.d.). https://huggingface.co/learn/nlp-course/chapter6/6?fw=pt

[14] *BERT Embeddings*. (2023b, March 26). https://tinkerd.net/blog/machine-learning/bert-embeddings/

[15] Chen, H. (2018, August 8). *A tutorial on network embeddings*. arXiv.org. https://arxiv.org/abs/1808.02590

[16] "BERT Encoder Layer," Mar. 26, 2023. https://tinkerd.net/blog/machine-learning/bert-encoder/

[17] A. Vaswani, "Attention is all you need," *arXiv.org*, Jun. 12, 2017. https://arxiv.org/abs/1706.03762

[18] T. Whang, "An effective domain adaptive Post-Training method for BERT in response selection," *arXiv.org*, Aug. 13, 2019. https://arxiv.org/abs/1908.04812

[19] "IBM documentation." https://www.ibm.com/docs/es/spss-modeler/saas?topic=dm-crisp-help-overview

[20] R. Salvador,"Aceleración del algoritmo Smith-Waterman mediante dispositivos reconfigurables" *repositorio.uam.es* 2010-2011. https://repositorio.uam.es/bitstream/handle/10486/12965/62216_Salvador_Gradaille_Roberto.pdf?sequence=1

[21] Libretexts, "2.5: El algoritmo de Needleman-Wunsch," *LibreTexts Español*, Nov. 01, 2022. https://espanol.libretexts.org/Biologia/Biolog%C3%ADa_Computacional/Libro%3A_Biolog%C3%ADa_Computacional_-_Genomas%2C_Redes_y_Evoluci%C3%B3n_(Kellis_et_al.)/02%3A_Alineaci%C3%B3n_de_Secuencias_y_Programaci%C3%B3n_Din%C3%A1mica/2.05%3A_El_algoritmo_de_Needleman-Wunsch

[22] "Visualization of Ukkonen's algorithm." https://brenden.github.io/ukkonen-animation/

[23] "11.9. Large-Scale Pretraining with Transformers — Dive into Deep Learning 1.0.3 documentation." https://d2l.ai/chapter_attention-mechanisms-and-transformers/large-pretraining-transformers.html

5 ANEXOS

ANEXO I

Para la implementación y adaptación del código se usó la guía interactiva que ofrece Deep Into Deep Learning disponible en:

https://d2l.ai/

ANEXO II

Para cargar el modelo preentrenado se usó los códigos de ejemplo de GluonNlp disponbles en:

https://nlp.gluon.ai/#

ANEXO III

El corpus paralelo se lo obtuvo del siguiente blog:

http://www.realworldnlpbook.com/blog/unreasonable-effectiveness-of-transformer-spell-checker.html

ANEXO IV

El código desarrollado está disponible en:

https://github.com/michaelchilan/ModeloDetecci-nErroresGramaticalesBERT.git