Universidad Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE COMPUTADORES

Curso Académico 2021/2022

Trabajo Fin de Grado

PROCESAMIENTO DE LENGUAJE NATURAL PARA LA EVALUACIÓN AUTOMÁTICA DE PREGUNTAS ABIERTAS EN EL AULA

Autor: Daniel Cáceres González

Directores: María de Soto Montalvo Herranz

Raúl Cabido Valladolid

Agradecimientos

Muchas gracias a mi familia y amigos por haberme apoyado durante toda mi trayectoria académica y en especial a mis padres, sin ellos no habría sido capaz de la mitad de lo que he logrado.

Gracias.

RESUMEN

En este trabajo se estudiarán y evaluarán diferentes métodos que aplicando técnicas de PLN permitan realizar una evaluación automática de preguntas abiertas en clase.

El problema que se encuentra en las clases donde el profesor tiene un gran volumen de estudiantes, es la imposibilidad de realizar una prueba rápida que otorgue un *feedback* que sitúe tanto al alumno como al profesor en el nivel de aprendizaje adquirido para cada lección.

Hemos considerado que el desarrollo de una herramienta automática de corrección y clasificación ayudarían a resolver el problema anterior sin necesidad de perder un tiempo excesivo en cada parte del temario.

Por ello, en este TFG se pretende desarrollar herramientas que lo permitan, desde dos puntos de vista: supervisado, necesitando datos de entrenamiento y desde un punto de vista no supervisado, donde no sean necesarios y así poder evaluar fácilmente el grado de viabilidad y las posibles ventajas e inconvenientes de cada enfoque.

El código que se use será desarrollado en Python y estará accesible desde el siguiente repositorio GitHub https://github.com/DCaceres2018/TFG

PALABRAS CLAVES

Embedding, modelo supervisado, modelo no supervisado, Bert, Flair, Transformers, procesamiento de lenguaje natural (PLN), conjunto de datos

ABSTRACT

In this document, different methods will be studied and evaluated that applying LNP techniques, allow an automatic evaluation of open questions in class.

The problema found in classes where the teacher has a large volumen of students is the impossibility of carrying out a quick test that provides feedback that places both, the student and the teacher, at the leve lof learning acquired for each lesson.

We have considered that the development of an automatic correction and classification tool would help to solve the previous problema without the need to spend excessive time on each part of the agenda.

For this reason, this paper intends to develop tolos that allow it, from two points of view: supervised, requiring training data and from an unsupervised point of view, where they are not necessary so you can easily assess the degree of feasibility and the possible advantages and disadvantages of each approach.

The code used will be developer at Python and will be accesible on the following GitHub repository https://github.com/DCaceres2018/TFG

KEYWORDS

Embedding, supervised model, unsupervised model, Bert, Flair,Transformers, natural lenguage processing (NLP), dataset

Índice

Índice de figuras

Índice de Tablas

Índice de Bloques de código

1. Introducción

En este primer capítulo del trabajo se hablará de la motivación que desencadenó este estudio y de los objetivos que se pretenden lograr durante el mismo.

1.1 Motivación

Actualmente en la docencia existe un claro punto a mejorar, los docentes no pueden realizar un seguimiento eficaz de los alumnos sin recurrir a pruebas que ponderen en la nota final. Es la forma con la que se ha trabajado desde hace mucho tiempo, pero esto lo que provoca en los alumnos es un grado de insatisfacción y de agobio ante la posibilidad de suspender una asignatura de la cual igual no tienen aprendidos los conceptos principales o simplemente el factor suerte les ha jugado en contra.

Este trabajo se plantea como un modo de estudio de diferentes técnicas que ayuden al equipo docente a realizar preguntas abiertas en clase, con el fin de que estas respuestas puedan ser clasificadas y ordenadas según el grado de acierto.

Esto podría ser útil como herramienta para complementar la antigua forma de evaluación que tenían los docentes y brindar un abanico de posibilidades para procesar el nivel de entendimiento que se presenta durante las clases y actuar en consecuencia. Abriendo una ventana para mejorar la comunicación entre alumnado y docente mediante el uso de un seguimiento más preciso y continuado.

Se usarán técnicas de Procesamiento de Lenguaje Natural (PLN) [1], que es el campo de conocimiento de la Inteligencia Artificial encargado de investigar la manera que tienen de comunicarse las máquinas con las personas usando lenguas naturales, como el español o inglés. Siendo una de las ramas de la inteligencia artificial que más ha evolucionado en los últimos años, se encuentra actualmente en muchísimos dominios, desde el médico hasta el educativo.

1.2 Objetivos

El objetivo principal y general es la elaboración de un sistema que se aplicará para calcular la similitud entre preguntas de los docentes y respuestas de los alumnos, dándole principalmente un enfoque no supervisado.

Otorgar un enfoque no supervisado es presentar un problema a un sistema que evaluará los casos que le demos sin atender a etiquetas o entrenamiento previo, de una manera sistemática.

Para ello, como objetivos secundarios que nos ayudarán a abordar el problema principal, se realizará un estudio del estado del arte de sistemas parecidos, se estudiará la viabilidad de una propuesta basada en un modelo no supervisado y se estudiarán las últimas técnicas de PLN para aplicarlas al problema.

1.3 Estructura de la memoria

El resto de la memoria se estructura en diferentes capítulos con los siguientes contenidos:

- Capítulo 2: Se hace un breve repaso del estado del arte.
- Capítulo 3: Se presenta la metodología de software seguida y las tecnologías usadas.
- Capítulo 4: Se presenta la propuesta que resuelva el problema inicial planteado.
- Capítulo 5: Se presenta la descripción informática de la propuesta.
- Capítulo 6: Se evalúan los resultados obtenidos.
- Capítulo 7: Se presentan las conclusiones y posibles trabajos futuros.

2. Estado del arte

En este capítulo se van a presentar algunos conceptos teóricos y se va a realizar una breve revisión del estado del arte.

2.1 Marco teórico

Para poder entender mejor la revisión del estado del arte que se presenta en la Sección 2.2, aquí se van a introducir algunos conceptos necesarios de forma breve.

Machine Learning [2] es una subcategoría de la inteligencia artificial, que se refiere al proceso por el cual una computadora desarrolla patrones de reconocimiento, o la habilidad de aprender y hacer predicciones basadas en datos y hacer ajustes sin estar específicamente programado para hacer eso. Un modelo de aprendizaje automático es un sistema que analiza datos e identifica patrones, y luego usa esos conocimientos para completar mejor la tarea asignada. Cualquier tarea que dependa de un conjunto de puntos de datos o reglas se puede automatizar mediante el aprendizaje automático.

Dependiendo de la situación, los algoritmos de aprendizaje automático funcionan con más o menos intervención/refuerzo humano. Los cuatro principales modelos de aprendizaje automático son el aprendizaje supervisado, el aprendizaje no supervisado, el aprendizaje semisupervisado y el aprendizaje por refuerzo.

En este trabajo usaremos los dos primeros, el modelo de aprendizaje supervisado y el modelo de aprendizaje no supervisado.

Con el aprendizaje supervisado, el sistema recibe un conjunto de datos etiquetados que le permite aprender cómo realizar una tarea humana.

Con el aprendizaje no supervisado, el sistema recibe datos sin etiquetar y extrae patrones previamente desconocidos de ellos.

2.2 Breve revisión del estado del arte

En la medición de la similitud de texto (en el contexto del procesamiento del lenguaje natural), podemos destacar que los trabajos se realizan mayoritariamente sobre textos en inglés y siguiendo un enfoque supervisado. Para guiar nuestro proyecto y esbozar una visión de las técnicas PLN más usadas para medir similitud entre oraciones, se ha revisado el trabajo de otros equipos para poder esclarecer su forma de trabajar y sus resultados. Los trabajos que vamos a repasar fueron presentados en el SemEval-2013 Task 7, competición que presenta el reto de desarrollo de un sistema con capacidad de medir la similitud entre oraciones bajo un marco educacional [3]. De este reto se ha usado además el conjunto de datos con el que se ha trabajado, por lo tanto, se va a centrar la revisión del estado del arte en los trabajos que fueron presentados allí y con los que se comparará mi propuesta. Entre los equipos, se encuentran CU [4], UKP-BIU [5], ETS [6], SoftCardinality [7], Celi [8], LIMSIILES [9], CoMeT [10] y CNGL [11].

El equipo CU desarrolló un sistema que usa ClearTK toolkit con Eclipse para extraer las características de las respuestas de los estudiantes y de las frases de referencia. Entrenaron un clasificador binario (LibSVM) para catalogar un vector de características en correcto o incorrecto. El resultado fue un sistema que se encuentra estadísticamente en la media de los demás equipos presentados a la competición.

El equipo UKP-BIU desarrolló un sistema basado en el *framework* UIMA de Apache y DKPro lab. Usaron el núcleo de DKPro para el preprocesamiento de texto y entrenaron un modelo supervisado (Naive Bayes) usando Weka con extracción de características (Syntactic Features, BOW features, Basis Similarity Features) basado en ClearTK. Los resultados que presentaron en la competición se encuentran en la media de los obtenidos por el resto de los equipos. Destacan que su sistema no decae en escenarios duros o difíciles de similitud de texto.

ETS por su parte, desarrolló un modelo de regresión logístico con regularización (usaron la implementación de regresión lógica de scikit-learn toolkit). Además, su sistema incluye diferentes *features* de similitud de texto que comparan las respuestas de los estudiantes (BLEU y PERP). ETS presentó dos propuestas diferentes, encontrándose una en el grupo de los equipos cuyas propuestas obtuvieron los mejores resultados, mientras que el otro se halla entre los peores equipos de la competición.

El equipo SoftCardinality primero procesaba y limpiaba los datos, tokenizando, convirtiendo las letras mayúsculas en minúsculas y eliminando *stop-words*. Segundo, cada frase preprocesada la representaban en q-grams. Con esto obtenían vectores de datos con los que entrenar un clasificador. Este equipo destaca que rendían muy bien en la mitad del conjunto de datos y mal en el resto. A pesar de esto, fueron el equipo que mejores resultados presentaron a la competición.

El equipo Celi presentó un sistema que calcula la similitud entre sentencias midiendo la distancia obtenida con el algoritmo *WordOverlap*. Con las distancias obtenidas se entrenaba un clasificador binario que determinaba la similitud entre oraciones. Su propuesta se encontraba entre los peores equipos, siendo este el tercer peor resultado.

El equipo LIMSIILES realizaba una etapa de preprocesamiento del texto y calculaban posteriormente una distancia Jaccard para agrupar las respuestas. Determinaban cuanto de cerca se encuentra el vocabulario usado entre dos frases dadas. Este sistema en comparación al del resto de equipos presentados, obtenía peores resultados y se encuentra por debajo de la media.

El equipo CoMeT combinaba tres tipos de subsistemas unidos en un meta-clasificador. CoSeC y CoMiC son dos subsistemas que alinean lingüísticamente las respuestas de los estudiantes con la respuesta de referencia. El último subsistema está inspirado en el concepto *bag-of-words* que clasifica las respuestas de estudiantes en 3 *bags* diferentes. En estos tres

subsistemas realizan previamente un preprocesamiento, donde incluyen una tokenización y una lematización. Los resultados que obtuvieron estaban por encima de la media de la competición.

El equipo de CNGL desarrolló un sistema de traducción referencial en el que parejas de frases eran usadas para entrenar un modelo estadístico MT, cuyo rendimiento es afectado por la cantidad de entrenamiento a la que sometas al modelo. A nivel resultados fueron el equipo que peores obtuvo a nivel general.

Como conclusión, debemos considerar que existen diferentes formas de abordar la detección de similitud entre oraciones usando técnicas de PLN. Atendiendo a los resultados presentados y dado que la mayoría de los equipos han optado por realizar algún tipo de tratamiento de texto, este debe ser un paso fundamental para determinar la similitud entre oraciones.

3. Metodología y tecnologías

En este capítulo se presenta la metodología de software seguida y las tecnologías usadas.

3.1 Metodología empleada

Debido al tipo de proyecto que tenemos por delante, que sigue un esquema repetitivo en el que probaremos diferentes tecnologías y realizaremos las mismas pruebas para todas ellas, se consideró que la mejor forma de organizar el trabajo requerido era siguiendo una metodología en cascada para así poder avanzar y retroceder entre fases además de encajar bien con la forma que se han llevado las reuniones para coordinarse con los tutores, como se puede ver en la Figura 3.1.

"El modelo en cascada es una metodología secuencial para la gestión de proyectos que se divide en fases. Cada fase comienza cuando ha terminado la anterior." [12]

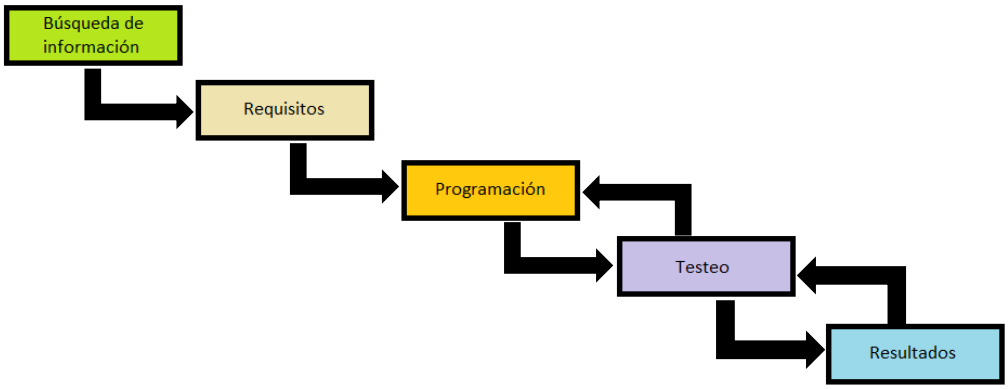

Figura 3.1: Esquema en cascada de la metodología empleada

Esta metodología permitió trabajar de forma escalonada, separando claramente cada etapa del proyecto, lo que facilitó la realización de cada una de las tareas al crear una barrera mental entre la fase actual y la siguiente. Las primeras etapas fueron escalonadas y sin retroceso entre ellas, sin embargo, con el resto se crearon ciclos para cada tipo de tecnología empleada

abarcando desde la programación de esta hasta que se obtienen los resultados.

3.2 Planificación del proyecto

Se presenta un Diagrama de Gantt donde se muestra la distribución en el tiempo de las diferentes fases del proyecto. Como podemos ver en la Figura 3.2, tenemos una primera fase de búsqueda de información. Fue el momento del proyecto en el que encontramos todo lo necesario para empezar a plantearnos los requisitos que íbamos a tener. Fue la fase donde seleccionamos el *dataset* de trabajo y el modelo de estadísticos que vamos a necesitar para comparar los resultados.

La segunda fase corresponde al análisis y selección de requisitos del estudio y del sistema.

Una tercera fase de diseño unida a la programación del código. La organización de la estructura del algoritmo y la adaptación del código proporcionado por los transformadores para nuestro uso con el *dataset* que tenemos.

Una vez tenemos el código, la siguiente parte corresponde a una fase de testeo, cuyo fin será encontrar incompatibilidades o errores de ejecución durante el algoritmo.

Por último, la fase de generación de los resultados y la evaluación de estos, comparándolos con los datos obtenidos en la primera fase previamente comentada.

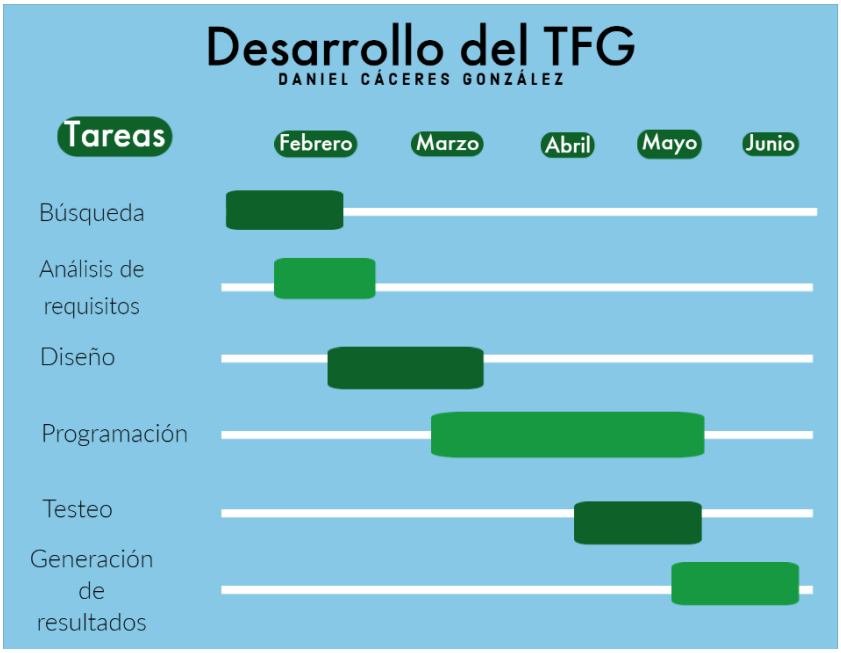

Figura 3.2: Diagrama de Gantt para el desarrollo del trabajo.

3.3 Estimación de costes

Para poder evaluar los costes de este proyecto hay que separar en el coste económico y el coste en tiempo. El primero de estos, depende de las tecnologías usadas o probadas, en el trascurso del proyecto se probaron tecnologías que posteriormente se acabaron descartando por querer desarrollar código que no acarrease coste económico adicional para que fuese más accesible, aunque fuesen más potentes a nivel ejecución. Se probaron tecnologías como los transformadores proporcionados por OpenAI [13], el cual tenía un límite de prueba de 18€. Según estimaciones por el número de frases procesadas con el volumen de nuestro *dataset*, se estimó que el procesamiento de este rondaría los 2,5€ por ejecución para nuestro caso de estudio o 0,01€ cada 25 frases evaluadas de esta forma.

En cuanto al coste en tiempo, el proyecto comenzó a principios de octubre de 2021 y ha continuado hasta junio de 2022. El número total de horas empleadas para cada fase del proyecto es el siguiente:

- Recolección de información: 5-10h durante todo el proyecto.

- Programación y mantenimiento del código: 20h.

- Ejecuciones y obtención de resultados: 72h.

Esta última etapa duró mucho en comparación a las otras debido al peso computacional que acarrean las tecnologías usadas. A nivel de procesamiento, aunque se ha conseguido ejecutar todo en GPU y disminuye el tiempo de cada ejecución en un 75%, al usar Flair o Bert, se presentan más adelante, tardaban en torno a 2/3 horas por cada prueba que se quisiese hacer.

En total el proyecto ha trascurrido durante 9 meses y ha ocupado un total aproximado de 65-70h.

3.4 Tecnologías empleadas

Python.

Este lenguaje de programación, con el que se ha desarrollado el código, [14] surge a finales de los años 80 creado por el holandés Guido van Rossum, el cual decidió crear este lenguaje como sucesor del lenguaje ABC dotado de la capacidad de manejar excepciones. Se trata de un lenguaje interpretado, de alto nivel, dinámico, multiparadigma (soporta programación orientada a objetos, programación funcional…) y con una filosofía orientada a la legibilidad del código. A destacar también sobre el mismo que se trata de un lenguaje *open source*.

PyCharm

Se trata de un IDE [15] para el lenguaje Python, desarrollado por la empresa *JetBrains*. Este IDE ofrece asistencia durante el desarrollo mediante, por ejemplo, autocompletado, sugerencias y marcado de errores describiendo solamente algunas de las características que ofrece en este

apartado. PyCharm también ofrece integración de diversas plataformas de control de versiones como pueden ser Git y CVS. Otra importante característica de este IDE es que ofrece compatibilidad con sistemas operativos Windows, Linux, y MacOS. Esto ha hecho que el desarrollo del código corra a cargo en este IDE.

NUMPY

Es una librería de Python [16] especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos.

Se ha usado debido a que incorpora una nueva clase de objetos llamados vectores o *arrays* que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación.

La ventaja de Numpy frente a las listas predefinidas en Python es que su procesamiento se realiza mucho más rápido (hasta 50 veces más) que las listas, lo cual la hace ideal para el procesamiento de vectores y matrices de grandes dimensiones.

PANDAS

Es una librería de Python [17] especializada en el manejo y análisis de estructuras de datos.

Los usos de Pandas en este trabajo han sido:

- Definir nuevas estructuras de datos basadas en los *arrays* de la librería NumPy pero con nuevas funcionalidades.
- Leer y escribir ficheros en formato CSV, Excel.
- Acceder a los datos mediante índices o nombres para filas y columnas.
- Realizar todas estas operaciones de manera muy eficiente.

PYTORCH

PyTorch [18], es una librería *open source* basada en Python, enfocada a la realización de cálculos numéricos mediante programación de tensores, lo que facilita su aplicación al desarrollo de aplicaciones de aprendizaje profundo. La sencillez de su interfaz, y su capacidad para ejecutarse en GPUs (lo que acelera el entrenamiento de los modelos), lo convierten en la opción más asequible para crear redes neuronales artificiales.

En nuestro caso de estudio, es usado para normalizar oraciones.

TENSORFLOW

TensorFlow [19] es una biblioteca de código abierto para la computación numérica y *Machine Learning* a gran escala. TensorFlow reúne una serie de modelos y algoritmos de *Machine Learning* y *Deep Learning* y los hace útiles mediante una metáfora común.

TensorFlow se usa para entrenar y ejecutar redes neuronales profundas, el Word embedding y el procesamiento del lenguaje natural.

TRANSFORMERS

Proporciona una API [20] y una serie de modelos pre-entrenados para su uso. Estos modelos sirven para modalidades como:

- Clasificación de texto.
- Extracción de información.
- Responder preguntas.
- Reconocimiento de audio.
- Clasificación de imágenes.
- Modalidades multimodo.

Los Transformers son los encargados en convertir una palabra o un conjunto de palabras en una representación numérica en un espacio vectorial de dimensiones variables o fijas, dependiendo del tipo de transformador. Un *Transformer* se basa en un *Encoder* (encargado de analizar el contexto de la secuencia de entrada) y un *Decoder* (encargado de generar la secuencia de

salda a partir de ese contexto). El resultado de este proceso es un *embedding*. (*Word embedding* es el nombre de un conjunto de modelos de lenguaje y técnicas de aprendizaje en PLN en donde las palabras o frases del lenguaje natural son representadas como vectores de números reales. Conceptualmente implica el encaje matemático de un espacio con una dimensión por palabra a un espacio vectorial continuo con menos dimensiones [21]).

4. Propuesta

En este capítulo se presenta el cómo esta confeccionado nuestro sistema y el enfoque que se le ha querido dar.

4.1 Arquitectura del sistema

En la Figura 4.1. se presenta la arquitectura del sistema propuesto.

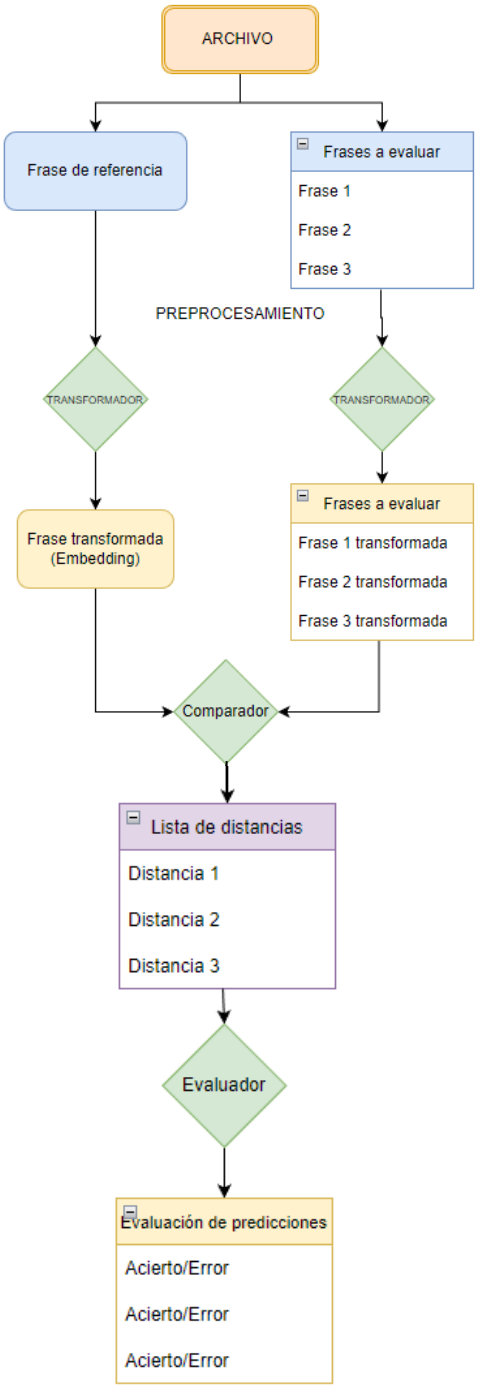

Figura 4.1: Arquitectura para el desarrollo del sistema.

Las partes que componen el sistema son las siguientes:

- Archivo: Corresponde al fichero que contiene las frases que queremos evaluar.
- Transformador: Es el módulo encargado en coger cada frase y convertirla en un vector numérico.
- Comparador: En él, se calcula la similitud existente entre las frases de referencia y las frases a evaluar.
- Evaluador: Compara la etiqueta asociada a cada frase a evaluar con la predicción realizada por nuestro modelo.

4.2 Enfoque

Para la realización de este proyecto se ha optado por darle un primer enfoque no supervisado, debido que, al buscar dentro del estado del arte, nos encontramos mayoritariamente con propuestas cuyo enfoque era supervisado. Para ello, partiendo del conjunto de datos inicial, realizamos un proceso de extracción, en el que obtenemos dos tipos de datos, una frase referencia y una lista de frases a evaluar.

Un ejemplo del tipo de frases que podemos encontrar podría ser, "El día es caluroso" como frase de referencia, la que el programa va a considerar que es la correcta. Y frases a evaluar, como "Hace calor en la calle" y "El plato está caliente". Produciéndose una evaluación de la similitud entre "El día es caluroso" y "Hace calor en la calle" y por el otro lado "El día es caluroso" y "El plato está caliente".

Una vez obtenemos las distintas frases, realizaremos una etapa de preprocesamiento en la que eliminaremos signos de puntuación y palabras sin valor individual, para así quedarnos con las palabras que aporten significado y puedan indicarnos similitud entre distintas frases. Y una posterior lematización (proceso lingüístico que consiste en, dada una forma flexionada hallar el lema correspondiente. [22]), para simplificar la comparativa de palabras, así varias palabras con mismo significado y contexto aportan lo mismo aun que se encuentren de distinta forma.

Posteriormente, cada frase ya simplificada será transformada, quedando como resultante un vector característico o un *word embedding* asociado.

A nivel ejecución lo que implica el uso de estos *embeddings* es una mayor capacidad para detectar similitud entre ellos al comparar espacios vectoriales, lo que nos facilita la tarea. En caso de no usarlos lo que obtendríamos serían dos frases que tendríamos que comparar palabra a palabra disminuyendo la probabilidad de acierto, debido a que gracias al uso de *embeddings* y de cómo han sido entrenados sus generadores para operar, palabras que guardan relación de significado se encuentran en puntos del espacio vectorial similares o cercanos, por lo que detectaremos similitud entre las palabras gato y felino, por relación conceptual. En cambio, al enfrentar gato y felino por palabra en sí, no existe relación aparente y no la detectaríamos.

Este proceso será realizado por el módulo transformador, con el que se realizará una transformación del texto para convertirlo en vectores de *embeddings*. Esto se realiza de tres formas diferentes:

TF-IDF

Al utilizar *embeddings* de TF-IDF [23], se generará un vector basado en las puntuaciones de frecuencia de términos. El valor TF-IDF aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia de la palabra en la colección de documentos, lo que permite manejar el hecho de que algunas palabras son generalmente más comunes que otras.

BERT

Es un modelo [24] que crea representaciones usando el contexto anterior y posterior de cada palabra y que, una vez entrenado previamente, se puede ajustar (*fine-tuning*) para una tarea específica posterior. BERT tiene las siguientes características:

- Usa un modelo de múltiples capas de transformadores (módulos de *self-attention*). Usando estos módulos, en lugar de LSTMs, se consigue aprender los pesos de atención de cada palabra del contexto anterior y posterior.
- El modelo se pre-entrena en dos tareas no-supervisadas. La primera, ocultando aleatoriamente un porcentaje de los tokens (palabras) de entrada y aprendiendo a predecirlos. La segunda, eligiendo dos frases, que el modelo debe predecir si son consecutivas o no.
- Una vez que el modelo se ha pre-entrenado, se puede ajustar en una tarea posterior y afinar (*fine-tuning*) los parámetros para dicha tarea.

FLAIR

Flair [25] es un *framework* simple para PLN desarrollado por la universidad Humboldt de Berlín.

Flair permite aplicar modelos de PLN de última generación al texto. Lo que permite Flair, entre otras cosas, es la clasificación de texto basado en *labels* y entrenar tus propios modelos.

El siguiente paso de nuestra propuesta consistirá en enfrentar uno a uno, el vector de la frase referencia con los de nuestra lista a evaluar. proporcionándonos así una distancia entre ellos, distancia con la que podremos determinar el grado de similitud y con ello verificar el resultado obtenido mediante un evaluador.

4.3. Comparativa entre vectores

Dado que para el desarrollo del estudio se van a utilizar vectores característicos como resultado de cada frase y al ser estos en esencia, vectores que codifican a frases. Tenemos que atender a las formas que hay de compararlos con el fin de obtener un grado de similitud y poder clasificarlos.

Las diferentes formas de clasificar vectores son [26]:

- **Distancia euclídea.** La raíz cuadrada de la suma de los cuadrados de las diferencias entre los valores de los elementos.

- **Correlación de Pearson**. La correlación producto-momento entre dos vectores de valores.
- **Coseno**. El coseno del ángulo entre dos vectores de valores.
- **Chebychev**. La diferencia absoluta máxima entre los valores de los elementos.
- **Minkowski.** La raíz p-ésima de la suma de las diferencias absolutas elevada a la potencia p-ésima entre los valores de los elementos.

Para decidir cuál es el más apropiado para nuestro caso, probemos con valores reales sobre Python.

Partiendo de estos vectores iniciales,

a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

a2 = [0, 1, 2, 3, 4, 6, 6, 7, 8, 9]

b = [0, 1, 2, 3, 4, 6, 7, 8, 9, 5]

c = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

En este ejemplo, los vectores más similares son el a y a2, debido a que solo tienen un elemento distinto y por lo tanto su distancia total es 1. Los más distantes son a y c debido a que son vectores espejos y el caso intermedio es el que hay entre a y b debido a que la mitad del vector es idéntico y la otra mitad cercano al valor.

Calculando la distancia entre los vectores obtenemos los resultados de la Tabla 4.1 con los diferentes tipos de medidas. Las que podrían reflejar el resultado esperado serían la distancia de coseno y la correlación de Pearson. Pero, la que refleja el resultado esperado de manera más adecuada, es la distancia coseno debido a que con Pearson se obtienen valores negativos.

	ENTRE A Y B	ENTRE A Y C	ENTRE A Y A2
COSENO	0.035	0.57	0.0015
EUCLÍDEA	4.47	18.16	1
PEARSON	0.8787	-1	0.9946
CHEBYSHEV	4	9	1
MINKOWSKI	4.47	18.16	1

Tabla 4.1: Comparativa entre evaluadores de similitud de vectores

5. Descripción informática

Para poder evaluar las diferentes técnicas se ha implementado un conjunto de scripts de Python que nos ayuda a generar los resultados.

5.1. Código

Para la programación de nuestro algoritmo hay que tener en cuenta los diferentes enfoques que podemos darle al código. Como habíamos explicado antes, un enfoque no supervisado implicaba la ejecución sin un entrenamiento previo, mientras que el supervisado tiene que contar con una capa dedicada a enseñar al programa que debería predecir para las frases que tenemos de entrenamiento.

Es por lo que en esta sección debe separarse en estos dos enfoques utilizados para la realización del estudio.

5.1.1 Modelo no supervisado

En este primer enfoque correspondiente al modelo no supervisado, se explica el desarrollo del algoritmo de ejecución capaz de procesar el conjunto de datos usado y transformarlo en un conjunto de *embeddings* los cuales podremos evaluar.

La primera parte de nuestro código corresponde a la extracción de las frases de los ficheros a evaluar. Para ello, el conjunto está dividido en dos carpetas, casos "train" y casos "test" (en el siguiente capítulo se describe con detalle el conjunto de datos utilizado). Para el modelo no supervisado solo utilizaremos la carpeta "test" dado que este enfoque no precisa de un entrenamiento previo como se ha comentado previamente. La carpeta test, con la que trabajaremos contiene 2 subcarpetas divididas a su vez en 2 y 3 carpetas, dando un total de 5 carpetas con archivos "XML" con preguntas, respuestas de referencia y frases a evaluar.

La estructura de estos ficheros es la siguiente:

- Una pregunta etiquetada como <*questionText*>
- Un conjunto de respuestas de referencia etiquetadas como <*referenceAnswer*>
- Un conjunto de respuestas a evaluar etiquetadas como <*studentAnswer*> y con un valor *accuracy* que tiene esa respuesta (correcta, incorrecta)

Sabiendo la estructura del conjunto de datos sobre el que trabajar lo que hay que hacer es obtener todas las respuestas y trabajar sobre ellas en Python.

Por lo que, primero de todo listamos el directorio de las carpetas y almacenamos cada uno de los ficheros en una lista para su posterior uso, tal y como se puede ver en el Bloque 5.1.

```
for x in os.listdir(Ruta):
    route = Ruta + "/" + x
    archivos.append(route)
```

Bloque de código 5.1: listar el directorio raíz.

Una vez tenemos todos los ficheros con los que vamos a trabajar en una lista, tenemos que decidir la forma con la que queremos evaluar los resultados. Anteriormente se había explicado que la mejor forma de determinar el grado de similitud entre dos vectores era con la distancia coseno. Pero, para indicar si esa distancia coseno es suficientemente grande o pequeña como para considerar que dos vectores son similares o no, hace falta el uso de un valor de referencia o cota.

Este valor numérico llamado cota, está comprendido entre valores 0 y 1, siendo 0 el indicativo que dos vectores son completamente diferentes y 1 que dos vectores son iguales. Así es como le indicaremos a nuestro modelo si un valor es suficientemente elevado para ser considerado similar.

Poniendo un ejemplo con el fin de que quede claro. Si al ejecutar el algoritmo determinamos que la similitud entre las oraciones "Quiero mucho a mi padre" y "No quiero nada a mi padre" es de 0.63 una cota de 0.7 indicaría que esas dos frases no son iguales y, por tanto, habríamos acertado. Pero, si en vez de 0.7, ponemos como cota 0.5, indicaría que esas dos frases son similares y, por lo tanto, determinaríamos que corresponde a un acierto, pero, estaríamos errando.

Por lo que, el siguiente paso del algoritmo será por cada archivo en la lista y por cada cota que queramos, se ejecutará el algoritmo presentado en el Bloque 5.2 y se determinarán cuántos verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos obtenemos para así evaluar tanto nuestro transformador como la cota empleada en ese caso.

De cada fichero obtenemos las respuestas de referencia y frases de los alumnos a evaluar.

```
doc = minidom.parse(fichero)

respuestaBuena = doc.getElementsByTagName("referenceAnswer")[0]
respuestaBuena = respuestaBuena.firstChild.data
respuestasDadas = doc.getElementsByTagName("studentAnswer")

respuestas.append(respuestaBuena)

for i in respuestasDadas:
    ValoresRespuestas.append(i.getAttribute('accuracy'))
for i in range(len(respuestasDadas)):
    respuestas.append(respuestasDadas[i].firstChild.data)
```

Bloque de código 5.2: Extracción de las respuestas del fichero.

Con esto obtenemos dos listas:

- Una lista de respuestas, encabezado por la respuesta de referencia o respuesta válida.
- Una lista de valores de similitud ("*correct*", "*incorrect*) correspondiente a las frases de la lista superior con relación posición-1.

A la lista de frases o respuestas se le aplica un preprocesamiento. Consiste en realizar una limpieza de las oraciones para convertir las letras

mayúsculas en minúsculas, eliminar *stopwords* (Palabras que carecen de sentido cuando se escriben solas o sin palabra clave) y signos de puntuación. Y una posterior lematización (producir variables morfológicas de una raíz base del lenguaje).

Por lo que de tener la frase "¡Adoramos los girasoles!" se pasaría a tener "[adorar], [girasol]"

Con este vector y dependiendo del transformador lo que se obtiene es un vector numérico resultante de un tamaño N de elementos numéricos que codifican a la frase que teníamos. A continuación, se presentan los tamaños de acuerdo al transformador utilizado:

- Con Bert, se obtienen vectores de 768 elementos.
- Con Flair, se obtienen vectores resultantes de 4196 elementos.
- Con TF_IDF, se obtiene un vector de 2 dimensiones con las filas como instancias y las columnas como características (mide la frecuencia de términos). Su tamaño es variable.

Teniendo ya la lista de vectores codificados y la lista de referencia de los valores de las respuestas, se evalúa una a una la distancia existente entre el vector de referencia (posición 0) y los demás. El valor obtenido (número entre 0 y 1) se compara con la cota que se le había asignado, en caso de ser superior lo contamos como un posible positivo. De no superarlo, como un posible negativo.

El resultando es una lista del mismo tamaño que la lista de valores de similitud, con valores "acierto" o "error".

Una vez tenemos indicado que esa frase debería ser considerada positiva o negativa se comprueba lo que realmente es. En caso de que su clasificación fuese "correct" (en el archivo a evaluar) y se hubiese superado la cota, estaríamos ante un verdadero acierto (TP). De no superarla sería un falso negativo (FN). Si su clasificación fuese "incorrect" y se superase la cota, nos encontraríamos ante un falso positivo (FP). Y, por último, de no superarse, sería un verdadero negativo (TN).

Una vez hemos evaluado todas las frases existentes en el fichero obtenemos los valores de aciertos y errores, tanto positivos como negativos. Para poder evaluar el modelo entero, se ejecuta este algoritmo en todos los ficheros de la carpeta test, agregando los valores obtenidos de aciertos y errores a una variable global.

Por ejemplo, para la cota 0,4 con el trasformador Bert, obtenemos total de 3613 aciertos y 3488 errores. O lo que es lo mismo, 2839 verdaderos positivos, 774 verdaderos negativos, 3354 falsos positivos y 134 falsos negativos. Pudiendo determinar, entre otras cosas, que con estos valores el *accuracy* (en el capítulo siguiente se presentan las medidas de evaluación empleadas) de esta cota con este transformador es de 0.508%.

5.1.2 Modelo supervisado

Como ya habíamos comentado, para realizar un enfoque supervisado era necesario contar con una etapa previa a la evaluación del modelo, correspondiente a una etapa de entrenamiento.

Se opto por trabajar de manera diferente con el conjunto de datos para este modelo y se decidió crear un fichero "CSV" con una estructura "Frase de referencia"- "Frase a evaluar"- "Similitud", para así facilitarnos la ejecución. Así, obtenemos dos ficheros, uno correspondiente a las frases extraídas de la carpeta "*train*" y otro de la carpeta "*test*".

Con estas listas, las preprocesamos y lematizamos (al igual que en anterior modelo) y hacemos un *fine-tuning* de Bert.

Fine-tuning [27] en PLN hace referencia al procedimiento de reentrenar un modelo que ya haya sido entrenado, usando tus propios datos. Como resultado del *fine-tuning*, los pesos del modelo original han sido actualizados para amoldarse a las características del dominio de datos y la tarea en la que estás interesado. Para realizar todo este procedimiento, nuestro algoritmo

cogerá las frases con su índice de un fichero “CSV “, creado previamente (tal y como se puede ver en el Bloque 5.3).

```
test_df = pd.read_csv("../datasetTest.csv")
test_df["label"] = test_df["similarity"].apply(
    lambda x: 0 if x == 0 else 1
)
y_test = tf.keras.utils.to_categorical(test_df.label, num_classes=2)
```

Bloque de código 5.3: Extracción de las respuestas del fichero.

Realizando esto para el entrenamiento y para el test obtenemos 2 listas de frases y dos listas de índices (con valores 1 o 0 en función de si estas fuesen parecidas o no (acierto o error)).

Posteriormente, cargamos el modelo de Bert entrenado (Bloque 5.4).

```
# Loading pretrained BERT model.
bert_model = transformers.TFBertModel.from_pretrained("bert-base-
uncased")
```

Bloque de código 5.4: Modelo Bert precargado.

Y cargamos el modelo completo de Keras y lo compilamos (Bloque 5.5).

```
model = tf.keras.models.Model(
    inputs=[input_ids, attention_masks, token_type_ids],
outputs=output
)

model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss="categorical_crossentropy",
    metrics=["acc"],
)
```

Bloque de código 5.5: Utilización modelo Keras y su compilación.

Teniendo ya en modelo compilado, preparamos las frases de entrenamiento y test (Bloque 5.6).

```
train_data = BertSemanticDataGenerator(
    train_df[["sentence1", "sentence2"]].values.astype("str"),
    y_train,
    batch_size=batch_size,
    shuffle=True,
)
test_data = BertSemanticDataGenerator(
    test_df[["sentence1", "sentence2"]].values.astype("str"),
    y_test,
    batch_size=batch_size,
    shuffle=False,
)
```

Bloque de código 5.6: Preparación de los conjuntos de datos.

Y procedemos a entrenar el modelo con los datos que tenemos en *"train_data"*, tantas veces como épocas (*epochs*) queramos (Bloque 5.7).

```
history = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=epochs,
    use_multiprocessing=True,
    workers=-1,
)
```

Bloque de código 5.7: Entrenamiento del modelo con nuestros datos.

Posteriormente para evaluar el resultado con los datos de test (Bloque 5.8).

```
model.evaluate(test_data, verbose=1)
```

Bloque de código 5.8: Evaluación del modelo.

Y obtendríamos los resultados del modelo supervisado usando un *fine-tuning* de Bert.

6. Resultados

En este capítulo se habla del conjunto de datos usado durante la realización del proyecto, de los resultados obtenidos y de sus medidas de evaluación.

6.1 Conjunto de datos

Nuestro conjunto de datos y los resultados de referencia que tenemos fueron obtenidos del SemEval-2013 Task 7 [28] y de los trabajos publicados a raíz de esta tarea [3].

Nuestro conjunto de datos cuenta con respuestas de alumnos debidamente etiquetadas manualmente, respuestas a las preguntas de explicaciones y definiciones que aparecen típicamente en ejercicios prácticos, *tests* o preguntas vistas en clase.

Específicamente, dada una pregunta y una respuesta conocida o de referencia, las respuestas de los alumnos serán clasificadas en "*Correct*" o "*Incorrect*".

El conjunto de datos de análisis de respuestas de estudiantes consta de dos subconjuntos de datos distintos: Datos de BEETLE, basados en transcripciones de estudiantes que interactúan con el sistema de diálogo tutorial de BEETLE II (Dzikovska, 2010) y datos SCIENTSBANKS, basado en el corpus de respuestas de los estudiantes a las preguntas de evaluación (Nielsen, 2008b).

El subconjunto de datos BEETLE, consta de 56 preguntas del dominio de la electricidad y la electrónica, con 1.258 respuestas de estudiantes. SCIENTBANK contiene 5.843 respuestas de alumnos de 197 preguntas de evaluación diferentes de 15 dominios de la ciencia. El conjunto de datos de BEETLE fue etiquetado manualmente mientras que SCIENTBANK usó un esquema más detallado donde primero se etiquetaron las respuestas automáticamente con una posterior revisión manual.

La razón para el uso de este conjunto de datos es que se trata de un conjunto de sentencias ya etiquetadas, que encajan en el problema planteado en este trabajo. Además, se trata de un conjunto de datos accesible, que se ha utilizado en un reto al que se presentaron diferentes participantes, permitiendo marcar un punto de referencia y comparar los resultados obtenidos con los presentados a ese reto.

6.2 Medidas de evaluación

Para poder valorar la calidad de nuestros resultados, vamos a compararlo tanto entre ellos, como entre los que presentaron su propuesta a la tarea "SemEval-2013 Task 7: The Joint Student Response Analysis and 8th Recognizing Textual Entailment Challenge" [3]

En esta tarea o reto los resultados son expresados en función de su F1-score. Como se puede ver en la Figura 6.1, para calcular el valor de F1, debemos conocer previamente el de la precisión y el del *recall*.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Figura 6.1: Fórmula del F1-Score.

La precisión nos muestra la calidad del modelo, indica que cantidad de los casos detectados como positivos eran realmente positivos. Se calcula con la fórmula de la Figura 6.2, el número de verdaderos positivos detectados dividido entre la suma de todos los positivos.

$$precision = \frac{TP}{TP + FP}$$

Figura 6.2: Fórmula de la precisión.

El *recall* o exhaustividad es la métrica que mide la cantidad de positivos somos capaces de identificar, se calcula como se puede ver en la Figura 6.3, dividiendo el número de verdaderos positivos detectados entre la suma de verdaderos positivos y falsos negativos.

$$recall = \frac{TP}{TP + FN}$$

Figura 6.3: Fórmula del *recall*.

La última métrica que se usa es el *accuracy* o exactitud. Mide el porcentaje de casos que el modelo ha acertado. Su fórmula es la que encontramos en la Figura 6.4, que suma el total de aciertos y lo divide entre el número total de casos.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Figura 6.4: Fórmula del *accuracy*.

En el caso de los equipos que compitieron sobre el conjunto de datos de test, obtuvieron los resultados mostrados en la Tabla 6.1 (Datos obtenidos de la tabla original del *paper* que describía la tarea de investigación, la Tabla 6.2).

EQUIPOS	MACRO-AVERAGE F1
CELI	0,619
CNGL2	0,581
CoMeT	0,669
CU	0,667
ETS$_1$	0,694
ETS$_2$	0,599
LIMSIILES	0,642
SoftCardinality	0,707
UKP-BIU	0,648

Tabla 6.1: Representación de los resultados de la competición

Dataset:	BEETLE		SCIENTSBANK			
Run	UA	UQ	UA	UQ	UD	Mean
$CELI_1$	0.640	0.656	0.588	0.619	0.615	0.624
$CNGL_2$	**0.800**	0.666	0.591_1	0.561	0.556	0.635
$CoMeT_1$	***0.833***	**0.695**	***0.768***	0.579	*0.670*	0.709
CU_1	0.778	**0.689**	0.603	0.638	*0.673*	0.676
ETS_1	**0.802**	**0.720**	*0.705*	*0.688*	*0.683*	0.720
ETS_2	***0.833***	**0.702**	***0.762***	0.602	0.543	0.688
$LIMSIILES_1$	0.723	0.641	0.583	0.629	0.648	0.645
$SoftCardinality_1$	0.774	0.635	*0.715*	***0.737***	***0.705***	0.713
$UKP\text{-}BIU_1$	0.608	0.481	0.726	0.669	0.666_2	0.630
Median	0.778	0.666	0.705	0.629	0.666	0.676
Baselines:						
Lexical	0.788	**0.725**	0.617	0.630	0.650	0.682
Majority	0.375	0.367	0.362	0.371	0.367	0.368

Tabla 6.2: Two-way task macro-average F1

6.3 Resultados

Teniendo como referencia los valores de la Tabla 6.1, se van a presentar los resultados obtenidos para cada uno de nuestros transformadores y cada uno de nuestros modelos.

6.3.1 Modelo no supervisado

Como se ha mencionado previamente, contamos con 3 tipos de transformadores, cuyos resultados se presentan a continuación.

TF-IDF

En esta primera ejecución, usamos el transformador TF-IDF [29], y los resultados que se obtienen son los mostrados en la Tabla 6.3.

	PRECISION		**RECALL**		**F1-SCORE**		**ACCURACY**
TF-IDF	MACRO AVERAGE	WEIGHTED AVERAGE	MACRO AVERAGE	WEIGHTED AVERAGE	MACRO AVERAGE	WEIGHTED AVERAGE	
0.0	0.79	0.76	0.51	0.59	0.39	0.45	0.59
0.1	0.79	0.76	0.51	0.59	0.39	0.45	0.59
0.2	0.77	0.74	0.52	0.59	0.40	0.46	0.59
0.3	0.77	0.74	0.52	0.59	0.42	0.47	0.60
0.4	0.76	0.74	0.53	0.61	0.43	0.48	0.61
0.5	0.75	0.73	0.54	0.61	0.45	0.50	0.61
0.6	0.73	0.71	0.55	0.62	0.48	0.53	0.62
0.7	0.71	0.69	0.58	0.64	0.53	0.57	0.64
0.8	0.67	0.67	0.61	0.65	0.59	0.62	0.65
0.9	0.66	0.67	0.66	0.67	0.66	0.67	0.67
1.0	0.65	0.68	0.61	0.57	0.55	0.54	0.57

Tabla 6.3: Datos obtenidos al ejecutar el algoritmo con el transformador TF-IDF

Los mejores resultados obtenidos en esta ejecución se asemejan a los mostrados en la Tabla 6.1, y el umbral que más nos interesa es el de 0.9 debido a que el F1 obtenido en esa cota es de 0.66. En la competición la media del F1-score se encontraba en 0.64, situándonos entre los equipos que mejores resultados obtenían.

FLAIR

Como segundo transformador nos encontramos con Flair, que trabajaba con vectores más grandes que TF-IDF y cuyo tiempo de ejecución era considerablemente mayor.

Los resultados obtenidos son representados en la Tabla 6.4.

	PRECISION		**RECALL**		**F1-SCORE**		**ACCURACY**
FLAIR	MACRO AVERAGE	WEIGHTED AVERAGE	MACRO AVERAGE	WEIGHTED AVERAGE	MACRO AVERAGE	WEIGHTED AVERAGE	
0.0	0.56	0.59	0.50	0.42	0.30	0.25	0.42
0.1	0.57	0.59	0.50	0.42	0.30	0.25	0.42
0.2	0.60	0.63	0.50	0.42	0.30	0.26	0.42
0.3	0.63	0.66	0.51	0.43	0.32	0.28	0.43
0.4	0.63	0.67	0.52	0.45	0.36	0.32	0.45
0.5	0.60	0.63	0.54	0.47	0.42	0.39	0.47
0.6	0.61	0.63	0.57	0.52	0.50	0.48	0.52
0.7	0.61	0.63	0.61	0.59	0.59	0.58	0.58
0.8	0.64	0.64	0.63	0.65	0.63	0.64	0.65
0.9	0.71	0.69	0.57	0.63	0.52	0.56	0.63
1.0	0.29	0.34	0.50	0.58	0.37	0.43	0.58

Tabla 6.4: Datos obtenidos al ejecutar el algoritmo con el transformador de Flair

Observando los valores y los umbrales, se puede determinar que es en cotas altas como 0.8 con las que obtenemos los mejores resultados. Los estadísticos más prometedores de Flair corresponden a los obtenidos con el umbral 0.8, encontrándose un poco por debajo de la media obtenida en la competición y algo inferiores tanto en *accuracy* como en F1 a los que encontramos en la Tabla 6.3, lo que indica que es algo menos preciso que TF-IDF.

BERT

El último transformador que probaremos será el de Bert, cuyos resultados son mostrados en la Tabla 6.5.

	PRECISION		RECALL		F1-SCORE		ACCURACY
BERT	MACRO AVERAGE	WEIGHTED AVERAGE	MACRO AVERAGE	WEIGHTED AVERAGE	MACRO AVERAGE	WEIGHTED AVERAGE	
0.0	0.21	0.18	0.50	0.42	0.30	0.25	0.42
0.1	0.60	0.63	0.50	0.42	0.30	0.25	0.42
0.2	0.60	0.63	0.51	0.43	0.32	0.27	0.43
0.3	0.64	0.68	0.53	0.46	0.37	0.33	0.45
0.4	0.66	0.69	0.57	0.51	0.46	0.44	0.51
0.5	0.66	0.69	0.63	0.59	0.58	0.57	0.58
0.6	0.67	0.68	0.67	0.65	0.65	0.65	0.65
0.7	0.66	0.67	0.66	0.67	0.66	0.67	0.67
0.8	0.70	0.69	0.63	0.68	0.62	0.65	0.68
0.9	0.74	0.72	0.56	0.63	0.49	0.53	0.62
1.0	0.29	0.34	0.50	0.58	0.37	0.43	0.58

Tabla 6.5: Datos obtenidos al ejecutar el algoritmo con el transformador de Bert

Comparando los resultados obtenidos con los representados en la Tabla 6.1, observamos que al igual que los 2 anteriores casos, presentamos resultados similares y rondando la media de la competición en ciertos umbrales. En este caso, la cota que mejor se ajusta al objetivo es diferente en función de la medida de evaluación que uses. Así es, si hablamos de F1, la mejor cota sería 0.7 y los resultados sería idénticos a los obtenidos con TF-IDF y presentados en la Tabla 6.3 y algo superiores a los obtenidos con Flair. Sin embargo, si la medida de evaluación que usamos es el *accuracy*, la cota con mayor valor sería 0.8 con un 0.68, obteniendo una mínima mejora respecto al obtenido por TF-IDF.

Debido a que los resultados obtenidos en los 3 casos rondan valores similares entre ellos, hay que evaluar cada caso (Tabla 6.6) y tener en cuenta los valores de precisión y *recall* obtenidos en cada uno para ver cuál se ajusta mejor al objetivo final.

		PRECISION		RECALL		F1-SCORE		ACCURACY
	MEJOR UMBRAL	MACRO AVERAGE	WEIGHTED AVERAGE	MACRO AVERAGE	WEIGHTED AVERAGE	MACRO AVERAGE	WEIGHTED AVERAGE	
TF-IDF	0.9	0.66	0.67	0.66	0.67	0.66	0.67	0.67
FLAIR	0.8	0.64	0.64	0.63	0.65	0.63	0.64	0.65
BERT	0.7	0.66	0.67	0.66	0.67	0.66	0.67	0.67
BERT	0.8	0.70	0.69	0.63	0.68	0.62	0.65	0.68

Tabla 6.6: Comparativa de mejores umbrales.

Separando del resto los umbrales que mejor se han comportado durante la ejecución, se puede ver que los valores entre ellos son muy similares. Para nuestro caso, lo que más interesa es maximizar la cantidad de positivos que somos capaces de detectar. Entonces la medida de evaluación secundaria que hay que tener en cuenta es el *recall* (en caso de similitud en F1 y *accuracy)*.

Comparando estos 4 casos, descartamos la cota 0.8 de Bert debido a que el valor de F1 es inferior al resto además de contar con el *recall* más bajo. Ocurre lo mismo con la cota 0.8 de Flair, cuyos estadísticos son notablemente inferiores a las otras dos opciones disponibles. Nos quedan entonces dos casos que estadísticamente son idénticos, tanto el obtenido con un umbral de 0.9 con TF-IDF como el obtenido con Bert con un umbral de 0.7 obtienen exactamente los mismos estadísticos, que se encuentra entre los presentados en la competición y mostrados en la Tabla 6.1.

6.3.2 Modelo supervisado

Para este modelo el modo de trabajo cambia, como se menciona en el apartado 5.1.2, vamos a introducir una primera capa de entrenamiento y unas sucesivas épocas que van a definir nuestro modelo y lo irán moldeando tras cada ejecución. Es por ello, por lo que se presentaron varios escenarios para evaluar. Un primer escenario con pocas épocas de entrenamiento y uno con muchas.

Durante la ejecución del entrenamiento y de la comprobación de los resultados se obtuvieron las siguientes líneas aprendizaje.

1ª época:

278/278 [==========] - 1159s 4s/step - *loss*: 0.5821 - *accuracy*: 0.6849

2ª época:

278/278 [==========] - 1250s 4s/step - *loss*: 0.5086 - *accuracy*: 0.7485

Evaluación sobre el test:

221/221 [==========] - 781s 4s/step - *loss*: 0.6064 - *accuracy*: 0.6993

Mientras, que cuando se quiso introducir un mayor número de épocas de aprendizaje lo que se acabó obteniendo fue la siguiente información:

8ª época:

278/278 [==========] - 1143s 4s/step - *loss*: 0.3845 - *accuracy*: 0.8286

Evaluación sobre el test:

221/221 [==========] - 773s 3s/step - *loss*: 0.6777 - *accuracy*: 0.6804

Se observa que el modelo supervisado mejora el resultado del entrenamiento a medida que avanzan las diferentes épocas, pero el resultado posterior del test no sigue el mismo progreso y en este caso, su *accuracy* disminuye mientras que aumenta su *loss*. Esto indica que nos encontramos ante un sobreentrenamiento del modelo, se ha convertido en un predictor

demasiado específico para las preguntas existentes del conjunto de datos de entrenamiento y por lo tanto el clasificador se ajusta mejor al objetivo con pocas épocas de entrenamiento.

6.4 Discusión de los resultados

De todos los modelos y transformadores que se han utilizado podemos llegar a ciertas conclusiones de cada uno de ellos.

TF-IDF

Con este transformador en un modelo no supervisado podemos determinar que el umbral que mejor resultados obtiene es 0.9, como se puede ver en la Tabla 6.3, obtenemos un valor de F1 del 67% y un *accuracy* del 67%. Resultados que se encuentran entre los equipos que mejores resultados obtuvieron en la competición. Este transformador encuentra una correlación muy grande entre frases similares debido a que el umbral para determinar que dos frases son similares se encuentra muy próximo al 1.

FLAIR

Mirando los datos representados en la Tabla 6.4, se puede apreciar un esquema similar respecto al anterior transformador. En este, la cota con la que mejor F1-score obtenemos es 0.8, cuyo valor de F1 ronda el 64% y con un *accuracy* del 65%. Se obtiene un resultado inferior respecto al primero presentado, pero, aumenta el margen en el que una frase es considerada correcta, permitiendo mayor maniobra para determinar la similitud entre las oraciones. Aun así, considerando el tiempo de ejecución y los resultados obtenidos no se considera una opción viable.

BERT

Como se puede apreciar en la Tabla 6.5, correspondiente a los estadísticos obtenidos con Bert, este funciona de manera similar a Flair y a TF-IDF, permite usar una cota menor y tener más margen al determinar que una oración es similar a otra y observando su mejor cota (0.7) vemos que el valor del F1-score ronda el 67%, al igual que el *accuracy*, exactamente igual que los

obtenidos con TF-IDF con cota 0.9. Esto nos indica una pequeña mejora respecto a Flair y resultados idénticos obtenidos con el primero. Es por este detalle por lo que se decidió usar BERT y no Flair o TF-IDF para desarrollar un modelo supervisado, cuyos resultados discutiremos a continuación.

Modelo Bert Supervisado

Observando los datos expuestos en el apartado 6.3.2, introducir una capa de aprendizaje nos otorga una mayor capacidad de detección de las frases ya valoradas, es tanto esto que después de una 8ª época de aprendizaje el *accuracy* que se obtiene es del 83%. Sin embargo, respecto a la evaluación del conjunto de datos test, se puede ver cómo puede llegar a ser hasta contraproducente, produciéndose un descenso del 2% entre estos dos tipos de entrenamiento, reflejando que nos encontramos ante un modelo sobreentrenado.

También se observan resultados similares a los del modelo no supervisado, lo que nos quiere indicar que entrenar una pequeña capa de aprendizaje nos podría ser útil para valorar la similitud de texto y obtener una mayor precisión en nuestras predicciones.

7. Conclusiones y trabajo a futuro

En este séptimo capítulo se presentan las conclusiones y posibles trabajos futuros, además de hacer una revisión de la satisfacción de requisitos.

7.1 Conclusiones del trabajo desarrollado

En conclusión, respecto al proyecto, se ha logrado hacer un estudio de diferentes técnicas de procesamiento del lenguaje natural y a nivel personal me queda una enorme sensación de aprendizaje en cosas no vistas o vistas por encima durante la carrera y que he considerado extremadamente interesantes. Considero que los resultados obtenidos podrían ser más precisos para la tarea que se quería desarrollar, pero hubiese sido necesario el uso de herramientas más potentes que en un principio no se quisieron utilizar o durante el desarrollo del proceso se valoró y se acabó decidiendo su no utilización (como los transformadores de OpenAI).

Personalmente después de realizar este estudio, considero que la mejor opción para el desarrollo de un programa de evaluación automática de preguntas abiertas en clase es bajo un enfoque supervisado siempre y cuando se tenga una pequeña base de datos ya existente, debido a que con un modelo no supervisado no se podrían garantizar los resultados con otro conjunto de datos diferente y habría que ver su funcionamiento previamente para determinar el valor de cota óptimo en preguntas de otro campo distinto. Sin embargo, con un pequeño entrenamiento de un modelo supervisado ya se puede garantizar conseguir resultados aceptables y a medida que pase el tiempo y se fuese ampliando la base de datos, este clasificador sería más preciso.

Como conclusión al resultado final, las técnicas de procesamiento de lenguaje natural trabajan mejor bajo un marco supervisado y, además, reduce en parte el trabajo empleado para desarrollarlas además de crear un marco más general y válido para todo tipo de campos.

7.2 Satisfacción de objetivos

En cuanto a la satisfacción de objetivos, el principal era la elaboración de un sistema de cálculo de similitud de frases, dándole principalmente un enfoque no supervisado e intentar replicar los resultados que obtuvieron equipos anteriores. Este objetivo considero que se ha cumplido, dado que se han encontrado dos modelos basados en lenguaje no supervisado cuyos resultados no son dispares a los que teníamos como referencia. Además, el hecho de haber probado también con un modelo supervisado amplía este objetivo principal y otorga una comparativa entre el uso de un modelo supervisado y no supervisado.

7.3 Trabajo futuro

Como trabajo futuro quedaría seguir atento a los trabajos relacionados con la similitud entre oraciones y a una futura ampliación de estas técnicas usadas, dado que es un campo con mucha capacidad de mejora.

Además, un siguiente paso para estudiar podría ser evaluar con los mismos algoritmos, un conjunto de datos diferente, debido a que al obtener los resultados y compararlos con los presentados en la competición (cabe destacar que la competición tiene 9 años), me queda una pequeña sensación de que el conjunto de datos no es todo lo preciso que podría ser para ser evaluado correctamente y que el modelo no supervisado podría tener un comportamiento diferente dependiendo del conjunto de datos que se le presente.

Y un último trabajo a futuro es implementar en el código un traductor español-inglés para así poder evaluar también conjuntos de datos en español.

Bibliografía

[1] Antonio Moreno. "Procesamiento del lenguaje natural ¿qué es?" Accessed: Jul. 12, 2022 [Online]. Available: https://www.iic.uam.es/inteligencia/que-es-procesamiento-del-lenguaje-natural/

[2] "What is Machine Learning?" Accessed: May. 10, 2022. [Online] Available: https://www.hpe.com/es/es/what-is/machine-learning.html

[3] Myroslava Dzikovska, Rodney Nielsen, Chris Brew, Claudia Leacock, Danilo Giampiccolo, Luisa Bentivogli, Peter Clark, Ido Dagan, and Hoa Trang Dang. 2013. SemEval-2013 Task 7: The Joint Student Response Analysis and 8th Recognizing Textual Entailment Challenge. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 263–274, Atlanta, Georgia, USA. Association for Computational Linguistics.

[4] Ifeyinwa Okoye, Steven Bethard, Tamara Sumner. "CU : Computational Assessment of Short Free Text Answers - A Tool for Evaluating Students' Understanding" Proceedings of Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 603-607.

[5] Torsten Zesch, Omer Levy Iryna, Gurevych, Ido Dagan. " UKP-BIU: Similarity and Entailment Metrics for Student Response Analysis". Proceedings of Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 285-289.

[6] Michael Heilman, Nitin Madnani." ETS: Domain Adaptation and Stacking for Short Answer Scoring∗". Proceedings of Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 275-279.

[7] Sergio Jimenez, Claudia Becerra, Alexander Gelbukh. "SOFTCARDINALITY: Hierarchical Text Overlap for Student Response Analysis" Proceedings of Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 280-284.

[8] Ergun Biçici, Josef van Genabith. "CNGL: Grading Student Answers by Acts of Translation". Proceedings of Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 585–591.

[9] Milen Kouylekov, Luca Dini, Alessio Bosca, Marco Trevisan. "Celi: EDITS and Generic Text Pair Classification". Proceedings of Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 592–597.

[10] Martin Gleize, Brigitte Grau. "LIMSIILES: Basic English Substitution for Student Answer Assessment at SemEval 2013". Proceedings of Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 598–602.

[11] Niels Ott, Ramon Ziai, Michael Hahn, Detmar Meurers. "CoMeT: Integrating different levels of linguistic modeling for meaning assessment". Proceedings of Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 608–616.

[12] Sarah Laoyan. "Todo lo que necesitas saber acerca de la gestión de proyectos en cascada" Accessed: Jun. 05, 2022. [Online]. Available: https://asana.com/es/resources/waterfall-project-management-methodology

[13] Arvind Neelakantan, Lilian WengBoris, PowerJoanne Jang. " Introducing Text and Code Embeddings in the OpenAI API". Accessed: Jun. 21, 2022. [Online]. Available: https://openai.com/blog/introducing-text-and-code-embeddings/

[14] «Python Documentation by Version». python.org. Accessed: Jun. 8, 2022. [Online]. Available: https://www.python.org/doc/versions/

[15] Dmitry Jemerov. "PyCharm 3.0 Community Edition source code now available". Accessed: Jun. 22, 2022. [Online]. Available: https://blog.jetbrains.com/pycharm/2013/10/pycharm-3-0-community-edition-source-code-now-available/

[16] "La librería Numpy". Accessed: Feb. 02, 2022. [Online]. Available: https://aprendeconalf.es/docencia/python/manual/numpy/

[17] "La librería Pandas". Accessed: Feb. 02, 2022. [Online]. Available: https://aprendeconalf.es/docencia/python/manual/pandas/

[18] "Así puedes aprender a usar PyTorch, la herramienta más accesible para crear redes neuronales". Accessed: Feb. 02, 2022. [Online]. Available: https://www.genbeta.com/desarrollo/asi-puedes-aprender-a-usar-pytorch-herramienta-accesible-para-crear-redes-neuronales

[19] Oliver Holloway. "De Resolver Ecuaciones a Aprendizaje Profundo: Un Tutorial de TensorFlow Python". Accessed: May. 12, 2022. [Online]. Available: https://www.toptal.com/machine-learning/de-resolver-ecuaciones-a-aprendizaje-profundo-un-tutorial-de-tensorflow-python

[20] "PYTORCH-TRANSFORMERS". Accessed: Apr. 05, 2022. [Online]. Available: https://pytorch.org/hub/huggingface_pytorch-transformers/

[21] Li, Yitan; Xu, Linli (2015). «Word Embedding Revisited: A New Representation Learning and Explicit Matrix Factorization Perspective" Accessed: Jun. 28, 2022. [Online]. Available: https://www.ijcai.org/Proceedings/15/Papers/513.pdf

[22] Molino de Ideas. "Lematizador". Accessed: Jun. 19, 2022. [Online]. Available: https://www.molinolabs.com/lematizador.html

[23] Intellica.AI. "Comparison of different Word Embeddings on Text Similarity – A use case in NLP" Accessed: Dec. 10, 2021. [Online]. Available: https://intellica-ai.medium.com/comparison-of-different-word-embeddings-on-text-similarity-a-use-case-in-nlp-e83e08469c1c

[24] Adrián Hernández. "Procesamiento de lenguaje, de los word embeddings hasta BERT" Accessed: Jan. 21, 2022. [Online]. Available: https://mlearninglab.com/2019/12/01/procesamiento-de-lenguaje-de-los-word-embeddings-hasta-bert/

[25] Akbik, Alan and Bergmann, Tanja and Blythe, Duncan and Rasul, Kashif and Schweter, Stefan and Vollgraf, Roland. Accessed: Apr. 27, 2022. [Online]. Available: https://github.com/flairNLP/flair

[26] IBM Corporation. "Análisis de clústeres jerárquico: Medidas para datos de intervalo". Accessed: Jun. 17, 2022. [Online]. Available: https://www.ibm.com/docs/es/spss-statistics/SaaS?topic=method-hierarchical-cluster-analysis-measures-interval-data

[27] Serdar Cellat. "Fine-Tuning Transformer-Based Language Models". Accessed: May. 26, 2022. [Online]. Available: https://ymeadows.com/en-articles/fine-tuning-transformer-based-language-models

[28] leocomelli. "score-freetext-answer". Accessed: Nov. 11, 2021. [Online]. Available: https://github.com/leocomelli/score-freetext-answer

[29] "sklearn.feature_extraction.text.TfidfTransformer". Accessed: Jun. 21, 2022. [Online] Available: TdidfTransformer