UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Generación no supervisada de datos para la clasificación de queries en un sistema de diálogo

TRABAJO FINAL DE MÁSTER

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Autor: Francisco de Borja Valero

Tutor: Francisco Casacuberta Nolla

Curso 2019-2020

Resumen

En este trabajo se experimenta con tres técnicas de aumentados de datos para ayudar al clasificador de texto a incrementar su rendimiento.

Para la experimentación se han seleccionado un gran número de corpus donde la mayoría de ellos pertenecen al dominio de la detección de *intents*, ya que, el objetivo de la investigación realizada en este proyecto es aprovechar el conocimiento adquirido para posteriormente aplicarlo en *chatbots* dedicados a servicio al cliente. Estos *chatbots* tendrán que detectar los *intents* de las *queries* enviadas por los usuarios para posteriormente responder consecuentemente.

Los dos modelos empleados en la experimentación son de naturaleza distinta. El primero de ellos es XGBoost que es un modelo de aprendizaje automático clásico y el segundo de ellos utiliza la versión pre-entrenada de RoBERTa que es un modelo aprendizaje automático profundo, el cual actualmente es el estado del arte en la clasificación de texto.

Finalmente, vemos que el uso de estas técnicas no aporta una mejora considerable con respecto de no utilizarlas.

Palabras clave: Sistemas de Diálogo, Aprendizaje Automático, Clasificación de Texto, Redes Neuronales Profundas, Aumentado de datos, Modelos pre-entrenados

Abstract

In this work we experiment with three data augmentation techniques in order to help the classifier to improve its performance.

For the experimentation we chose a big number of corpus where the majority of them belong to the field of intent classification, because the knowledge obtained in this research will be used for applying in a customer service chatbot. This chatbot will answer the queries of the user, but previously it will have to detect the intent correctly.

The two models that we use for the experimentation belong to different nature. The first one is XGBoost that belongs to the classical machine learning models, and the second one uses the pre-trained version of RoBERTa that belongs to the deep learning models, that actually are the state of the art in text classification.

Finally, we will see that the use of these data augmentation techniques in natural language processing do not help to improve considerably the performance of the classifiers.

Key words: Dialog Systems, Machine Learning, Text Classification, Deep Neural Networks, Data augmentation, Pre-trained models

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Este primer capítulo es la introducción del trabajo. En la sección 1.1 se explica el porqué de este proyecto. En la sección 1.2 se presenta el estado del arte en clasificación de texto. En la sección 1.3 se comenta el estado del arte en el aumentado de datos en el campo del procesamiento de lenguaje natural. En la sección 1.4 se indican los objetivos del trabajo. Finalmente, en la sección 1.5 se indica como está estructurado el documento

1.1 Vista general

En este trabajo final de máster vamos a utilizar aproximaciones de aprendizaje automático para construir clasificadores de texto. Uno de los principales problemas que tienen en concreto los modelos de aprendizaje profundo es que requieren de muchas muestras de entrenamiento para obtener un rendimiento elevado. Por ese motivo, en este trabajo se emplearán técnicas de aumentado de datos para ampliar los recursos de entrenamiento disponibles. Por lo tanto, el objetivo principal de esta investigación es demostrar si el uso de estas técnicas ayuda a construir clasificadores más precisos. Adicionalmente, pondremos especial énfasis en la tarea de clasificación de texto especializada en detectar los *intents* de las *queries* de los usuarios, ya que hoy en día los sistemas de diálogo, ya sea en asistentes de voz o en *chatbots* son de los productos que más se están normalizando en nuestras vidas. Además de que para expresar el mismo *intent* se pueden formular *queries* muy diversas, por ese motivo realizar un aumentado en este tipo de corpus podría resultar muy útil para generar ejemplos extra que no son contemplados en el entrenamiento, pero que sí podrían ser de gran ayuda para que el clasificador sea más robusto ante *queries* nuevas de usuarios.

1.2 Clasificación de texto

La clasificación de texto es un problema clásico en el campo del procesamiento del lenguaje natural (PLN). La resolución de este problema consiste en que dado una unidad textual como puede ser una oración, una *query*, un párrafo o un documento se le asigne una etiqueta que trate de sintetizar el significado de dicho texto.

El texto es un recurso que se encuentra constantemente a nuestro alrededor, ya sea en páginas *webs*, en correos electrónicos, en *tickets*, en redes sociales, en opiniones de productos y en muchos más medios. Todas estas fuentes textuales contienen información muy útil y debido a su naturaleza no estructurada es un reto poder extraer la compresión de dicha información sin requerir de una alta dedicación de tiempo. Por lo tanto, la utilización de la clasificación de texto está presente en diversas aplicaciones, ya sea para

realizar análisis de sentimiento, para asignar un tópico a un documento, para categorizar una noticia, para detectar correos de *spam*. En este trabajo en concreto, vamos a poner más énfasis en la detección de *intents* en las *queries* de los usuario de un sistema de atención al cliente.

Inicialmente, las principales aproximaciones utilizadas para la clasificación de texto estaban basadas en reglas [12]. Debido a que los algoritmos empleados en estos sistemas requieren de un conocimiento muy profundo del dominio, ya que utilizan las características lingüísticas específicas de él, no se pueden aprovechar para otro tipo de problemas. Por esa razón, las soluciones que proporcionan estas aproximaciones son totalmente comprensibles en base a las reglas propuestas.

Poco a poco, los sistemas basados en reglas fueron sustituidos por los modelos de aprendizaje automático clásico, debido a que estos modelos no requieren de ningún conocimiento experto, ellos se basan en las observaciones de datos etiquetados siendo capaces de reconocer por sí solos patrones ocultos entre las asociaciones de textos y las correspondientes etiquetas de clase. A diferencia de los sistemas basado en reglas, estos modelos sí que se pueden emplear para resolver diferentes tipos de problemas causando que la interpretación de las soluciones proporcionada sea más difícil de comprender. Entre los diferentes algoritmos de aprendizaje automático clásico cabe destacar los modelos basados en *boosting* [2], como es el caso de AdaBoost que es un *ensamble* de clasificadores débiles. Otro modelo muy popular es el clasificador naive Bayes [26], el cual está basado en el teorema de Bayes y normalmente emplea como características para representar el texto, los vectores TF-IDF extraídos a partir de la frecuencia de las palabras en los documentos. No obstante, el modelo más utilizado por su alto rendimiento está basado en la máquinas de vectores de soporte, con esta aproximación también se suelen utilizar los vectores TF-IDF [45] para representar el texto.

En los últimos años, el auge del aprendizaje automático profundo ha hecho que la mayoría de modelos que se emplean actualmente para la clasificación de texto están basados en Redes Neuronales Artificiales (RNA). El primero de los modelos de aprendizaje profundo utilizado para clasificación de texto fue la Red Neuronal Recurrente (RNR), la cual procesa el texto como si de una secuencia de palabras se tratase, de este modo es capaz de capturar dependencias en la estructura a lo largo del tiempo [7]. Posteriormente, se emplearon modelos basados en Redes Neuronales Convolucionales (RNC), los cuales capturan los patrones en base a como están las representaciones organizados en el espacio, para ello utilizan los *word embeddings* para representar las palabras que constituyen el texto [16]. Tiempo después aparecieron los modelos que empleaban mecanismos de atención, los cuales permitían fijarse en diferentes palabras de la oración en base a la correlación de éstas [44], eso permitió generar unas representaciones más discriminativas que permitieron obtener un rendimiento más alto. Actualmente, el estado del arte en clasificación de texto está basado en el modelo *Transformer*. Este modelo permite procesar secuencias evitando el cuello de botella proporcionado por la RNR al procesar el texto de forma secuencial, para ello se influencia indirectamente de la aproximación de las RNC que es computacionalmente más eficiente, pero en esta caso en vez de utilizar capas convoluciones utiliza capas basadas en el mecanismo de atención. Para que este modelo obtuviera el mayor rendimiento hasta la fecha en clasificación de texto, inicialmente se pre-entrenó con cantidades inmensas de texto, realmente el modelo obtenido era un modelo de lengua contextual, pero posteriormente se extendió y se refinaron sus pesos para una tarea concreta como en este caso es la clasificación de texto [22].

La clasificación de *intents* al ser es un caso particular de la clasificación de texto, los modelos que se han explorado en este dominio concreto son prácticamente los mismos que en el dominio general. Partiendo de las máquinas de vector por soporte [32], pasando

por los modelos basados en RNR [33] [28], siguiendo por los modelos basados en RNC [13] [42] y finalizando en los modelos que emplean mecanismo de atención [21].

1.3 Aumentado de datos

Los modelos basados en el aprendizaje profundo son capaces de construir clasificadores de alto rendimiento, cuando disponen de un número elevado de ejemplos etiquetados para el entrenamiento. Ésto no siempre es así, muchas veces las clases de los conjuntos de datos están desbalanceadas[1] circunstancia que dificulta considerablemente a la hora de obtener un clasificador que obtenga un alto rendimiento para todas las clases del problema a resolver.

Una práctica común para solventar la carencia de ejemplos de entrenamiento consiste en generarlos de forma artificial. Este tipo de prácticas en el campo de la visión por computador han tenido mucho éxito, ya que han permitido obtener mejoras muy notables en las tasas de acierto con respecto de no utilizarlas. No obstante, en el campo del procesamiento de lenguaje natural (PLN) no han tenido tanto impacto, ya que el texto es más sensible a cualquier modificación que una imagen, debido a que esos cambios pueden afectar a que la oración tenga un significado semántico totalmente distinto y eso es un problema para el clasificador.

A continuación, se comentan algunas de las técnicas de aumentado de datos más populares para el PLN:

En el artículo [39] proponen la realización de modificaciones sencillas en el ejemplo original para obtener una nueva muestra. Estas modificaciones consisten en sustituir palabras por sus sinónimas usando *WordNet*, intercambiar el orden de las palabras, eliminar palabras e insertar palabras sinónimas.

Es posible que al sustituir una palabra por una similar se genere un ejemplo nuevo que realmente no pertenece a la misma clase que al ejemplo original del que proviene. Por ese motivo, en la publicación [17] utilizan un modelo de lengua bidireccional basado en Redes Neuronales Recurrentes (RNR), con el cual las palabras que propone para reemplazar las originales sí que tienen en cuenta el "contexto", no obstante al realizar estas nuevas sugerencias basándose únicamente en el modelo de lengua bidireccional puede ser que el ejemplo generado tenga un significado positivo: 'el actor es **fantástico**' cuando originalmente el significado era negativo: 'el actor es **mediocre**'. Para evitar eso habrá que calcular la probabilidad conjunta del nuevo ejemplo, en base al modelo lengua bidireccional y en base también a que ese nuevo ejemplo pertenezca a la misma clase que el ejemplo original. Posteriormente, con el auge de los modelos de lengua contextuales pre-entrenados como es el caso de BERT [6], en el artículo [40] sustituyen el modelo de lengua bidireccional por el *Masked Language Model* (MLM) de BERT que este modelo sí que tiene en cuenta realmente el contexto izquierdo y derecho de la oración, no como sucedía previamente con los modelos de lengua bidireccionales basados en RNR.

En la publicación [11] se influencian de la popular técnica de aumentado de datos utilizada en visión por computador *mixup*. Los autores la aplican tanto a nivel de palabra combinando los correspondientes *word embeddings*, como también a nivel de oración combinando la representación obtenidas al procesar toda la secuencia de *word embeddings* utilizando un modelo neuronal basado en RNR o en RNC.

Back-translation [31] es un método de aumentado de datos muy conocido, el cual utiliza dos traductores automáticos: $A \rightarrow B$ y $B \rightarrow A$. Dado un ejemplo original en el idioma

[1] Eso sucede en los problemas de clasificación cuando las distintas clases no están representadas con un número similar de ejemplos.

A, lo traduce al idioma B, utilizando el traductor $A \rightarrow B$, ese ejemplo traducido al idioma B, lo vuelve a traducir al idioma A esta vez utilizando el traductor $B \rightarrow A$. Siguiendo este procedimiento en la mayoría de los casos se obtiene un nuevo ejemplo diferente al original, pero conservando un significado semántico similar.

Finalmente, en los artículos [1] y [18] se refina un modelo generador de texto como puede ser el GPT-2 [27], para que dado una etiqueta de clase, el modelo sea capaz de generar un nuevo ejemplo que pertenezca a dicha clase. Normalmente, este tipo de modelo se ayuda de un contexto: primeros tokens de un ejemplo real; para generar el nuevo ejemplo sintético.

1.4 Objetivos

Los principales objetivos de este proyecto son:

- Obtener un clasificador de texto mejorado gracias a la aplicación de una técnica de aumentado de datos en el conjunto de entrenamiento.
- Explorar diversas técnicas de aumentado de datos para la clasificación de texto en diferentes corpus públicos y privados.
- Experimentar si el uso de las técnicas de aumentado de datos hace más robusto al clasificador, cuando el número de ejemplos de entrenamiento por clase es "pequeño".
- Encontrar una técnica de aumentado de datos que genere sin mucho coste de recursos y de tiempo, nuevos ejemplos a partir de un número reducido de ejemplos reales de entrenamiento, y al mismo tiempo obtenga un rendimiento competitivo.

1.5 Estructura

La memoria está estructurada de la siguiente manera. En el capítulo 1 se introduce el proyecto. En el capítulo 2 se explican todos los conceptos relacionados con las redes neuronales que utilizamos a lo largo de este trabajo. Siendo éstos los *word embeddings*, las Redes Recurrentes Neuronales, el modelo *Transformer* y los modelos de lengua contextuales. En el capítulo 3 se proponen las diferentes técnicas de aumentado de datos en procesamiento de lenguaje natural que vamos a emplear. En el capítulo 4 se presentan los corpus y los modelos que vamos a usar para la experimentación, como también la configuración de las técnicas de aumentado de datos que vamos a aplicar. En el capítulo 5 se muestran y se analizan los mejores resultados obtenidos para cada uno de los experimentos llevados a cabo. Finalmente, en el capítulo 6 se dan las conclusiones del trabajo realizado y se proponen posibles trabajos a explorar en el futuro siguiendo la misma línea de investigación. Adicionalmente, en el apéndice A se muestran los resultados obtenidos por todos los experimentos realizados.

CAPÍTULO 2
Redes neuronales

Este segundo capítulo proporciona los conceptos relacionados con las redes neuronales que vamos a emplear en este trabajo. El capítulo se divide en dos partes. La primera parte es la introducción donde se explica el modelo *word embeddings* (ver sección 2.1) y el modelo Red Neuronal Recurrente (RNR) (ver sección 2.2). En la segunda parte nos centramos en explicar el modelo *Transformer* (ver sección 2.3) y también los modelos basados en éste que son los modelos de lengua contextuales (ver sección 2.4).

2.1 Word embeddings

El texto en los programas informáticos es representado mediante el tipo de dato llamado *String*. Este formato no se puede introducir en una Red Neuronal Artificial (RNA), ya que este modelo trabaja con vectores. Inicialmente, las palabras dado un vocabulario eran representadas mediante un vector denominado *one-hot*, en el cual solamente el componente del vector asociado al identificador de la correspondiente palabra en el vocabulario tenía valor uno, el resto de componentes tenían valor cero. Estos vectores son muy dispersos y de un tamaño muy elevado, en los cuales dos palabras similares no tienen porqué tener una representación cercana.

Para suplir todas los inconvenientes que aportan las representaciones *one-hot* de las palabras aparecieron los *word embeddings* [24]. Este modelo proporciona a cada palabra una representación vectorial en un espacio continuo de baja dimensionalidad, lo que significa que es más eficiente desde el punto de vista computacional que el *one-hot*. Pero lo innovador de este modelo es que las representaciones compactas que proporciona capturan información oculta del lenguaje, como puede ser el significado contextual, es decir, que las palabras con un significado semántico similar se encuentran en puntos cercanos del espacio. Para conseguir eso, los pesos de la RNA están entrenados en base a la coocurrencia de las palabras en un corpus donde dado un contexto de palabras, el modelo es capaz de predecir las palabras que pueden pertenecer a dicho contexto. De esta manera, se permite captar analogías de palabras y asociaciones semánticas sorprendentes.

Las asociaciones semánticas se pueden obtener a partir de aplicar operaciones aritméticas: *king - man + woman* $\approx$ *queen*, como se muestran en la figura 2.1, donde se perciben las asociaciones desde el punto de vista del género, de la conjugación verbal y de las asociaciones entre palabras como sucede entre los países y sus respectivas capitales.

En la literatura, los *word embeddings* han sido popularmente utilizados por su eficiencia en modelos de clasificación de texto. Principalmente, hay tres modelos: Word2Vec, GloVe y FastText. A continuación, se explicará brevemente cada uno de ellos:

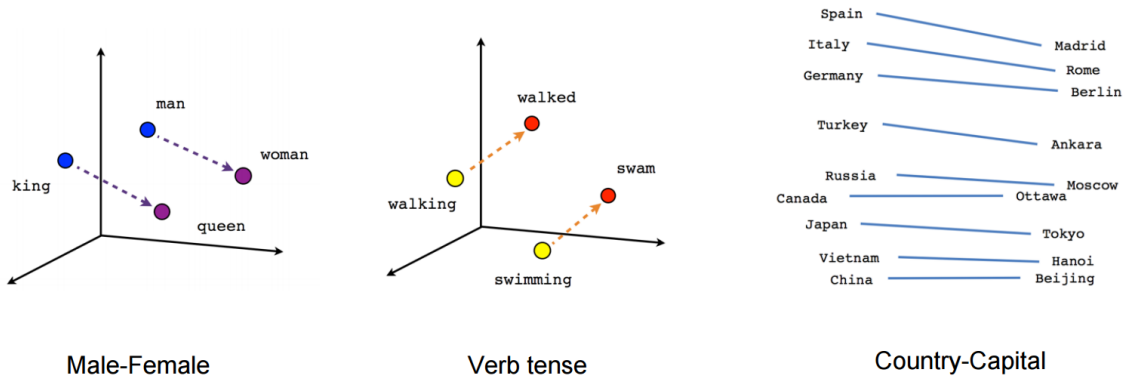

Figura 2.1: Relaciones semánticas en los *word embeddings*.

- **Word2Vec** [23] es una RNA de dos capas entrenada para reconstruir contextos lingüísticos de las palabras. Esta aproximación tiene dos variantes. La primera de ellas es *continuous bag-of-words* (CBOW) que dado el contexto izquierdo y el contexto derecho predice la palabra que se encuentre entre ambos contextos. La segunda aproximación es *skip-gram* (SG) que dada una palabra predice todas las palabras que se encuentran en su contexto izquierdo y derecho.
- **GloVe** [25] obtienen las representaciones de las palabras en un espacio donde las distancia entre ellas está basada en la similaridad semántica. Al entrenamiento, se agregan las estadísticas de co-ocurrencia de palabras presentes en el corpus. El resultado final muestra una interesante relación lineal de las subestructuras de las palabras en el espacio.
- **FastText** [3] está basado en el modelo SG, pero en vez de utilizar palabras usa n-gramas a nivel de carácter. Por lo tanto, las representaciones vectoriales están asociadas a estos n-gramas no a las palabras. La representación de las palabras serán obtenidas a partir de realizar la suma de los correspondientes representación de n-gramas que las constituyen.

Para los tres modelos de *word embeddings* previamente explicados hay versiones pre-entrenados de ellos sobre grandes corpus de entrenamiento. Nosotros en la experimentación recurriremos a estos modelos pre-entrenados, es decir que no entrenaremos ningún modelo desde cero, así será más sencillo poder replicar los experimentos.

2.2 Red Neuronal Recurrente

La Red Neuronal Recurrente (RNR) es un modelo creado para procesar secuencias de elementos. Este modelo comparte los pesos, los estados ocultos y el *bias* en los distintos instantes de tiempo.

En la figura 2.2 se muestra una RNR desplegada en el tiempo, donde el número de unidades del estado oculto es tres, $S = 3$. La RNR en cada instante de tiempo recibe como entrada un nuevo elemento y genera una salida en base a la entrada y al historial. Eso quiere decir, que para una misma entrada puede generar elementos de salida diferentes. Eso es debido, a que el estado oculto s_t puede capturar dependencias a lo largo del tiempo.

En la ecuación 2.1 se muestra como se calcula el estado oculto s_t en el instante t. En el primer instante de tiempo $t = 0$, el estado oculto $s_t \in \mathbb{R}^S$ está inicializado a cero, para el

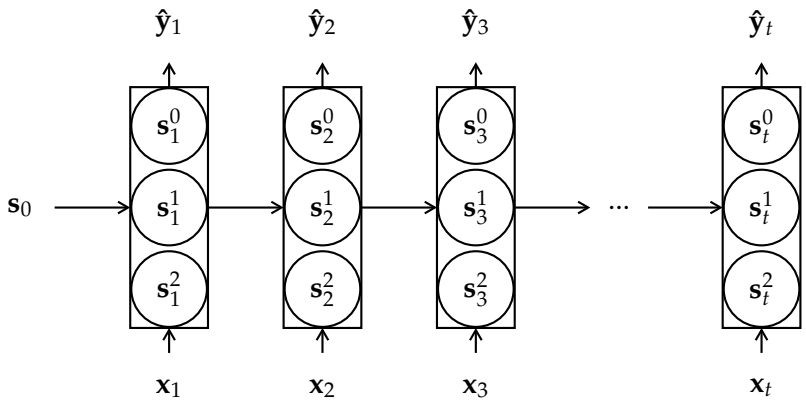

Figura 2.2: Red Neuronal Recurrente desplegada en el tiempo.

resto de instantes de tiempo se utilizan dos fuentes de entrada que son el estado oculto del instante anterior s_{t-1} y el elemento actual $x_t \in \mathbb{R}^D$ de la secuencia a procesar x, donde cada elemento está representado por D características. Cada una de estas entradas es multiplicada por su respectiva matriz de pesos, $W_{ss} \in \mathbb{R}^{S \times S}$ para s_t y $W_{sx} \in \mathbb{R}^{S \times D}$ para x_t. Al resultado obtenido de la suma de las dos multiplicaciones realizadas junto al término *bias* $b_s \in \mathbb{R}^S$ se le aplica una función no lineal g que normalmente es la arcotangente.

$$s_t = \begin{cases} 0, & \text{si } t = 0 \\ g(W_{ss}s_{t-1} + W_{sx}x_t + b_s), & \text{resto} \end{cases} \tag{2.1}$$

En la ecuación 2.2 se indica como se calcula la salida de la RNR $\hat{y}_t \in \mathbb{R}^C$ en el instante de tiempo t, suponiendo que la salida son C elementos[1]. Para ello, se realiza una multiplicación entre el estado oculto actual s_t y la matriz de pesos $W_{\hat{y}_t s} \in \mathbb{R}^{S \times C}$ cuyo producto se le suma el término independiente $b_{\hat{y}} \in \mathbb{R}^C$. A ese resultado se le aplica la función f que en caso de ser una tarea de clasificación será la function *Softmax*, la cual normalizaría la salida de la red para ese instante de tiempo.

$$\hat{y}_t = f(W_{\hat{y}_t s}s_t + b_{\hat{y}}) \tag{2.2}$$

2.3 Transformer

El modelo *Transformer* [37] genera una secuencia de salida a partir de una secuencia de entrada. Este modelo es un transductor secuencial auto-regresivo, ya que primero procesa toda la secuencia de entrada para posteriormente en la decodificación predecir un elemento en cada nuevo instante de tiempo. En las predicciones se tiene en cuenta dos fuentes de información, la propia entrada y la predicción realizada en el instante anterior.

Teniendo en cuenta lo que hemos explicado previamente podemos ver que el *Transformer* sigue la estructura *encoder-decoder*, pero con la diferencia fundamental de que sustituye las capas con recurrencias típicamente basadas en RNR (ver sección 2.2) utilizadas en el modelo *sequence to sequence* (Seq2Seq) [36] por capas basadas únicamente en el mecanismo de atención.

A continuación, se comentan las ecuaciones basadas en el mecanismo de atención a que forman parte del modelo *Transformer*, pero estas ecuaciones se explican desde un punto de vista neuronal. Posteriormente, en la subsección 2.3.1 se explicarán de forma

[1] En un problema de clasificación los elementos son todas las clases de la tarea a resolver.

mucho más detallada desde el punto de vista del campo de la recuperación de la información que es como originalmente se concibieron.

Las ecuaciones 2.5 y 2.4 son las que se utilizan en el bloque $l+1$ del *encoder* e para $1 \leq l \leq$ L, donde L es el número total de bloques que lo constituyen. En la ecuación 2.3 se muestra como se calcula la auto-atención normalizada entre la representaciones h obtenidas en el bloque l del *enconder*, entre los elementos situadas en las posiciones j' y j respectivamente de la secuencia de entrada. En la ecuación 2.4 se utiliza la auto-atención para computar la representación $\mathrm{h}_j^{e,l+1}$ del elemento localizado en la posición j de la secuencia de entrada para el bloque $l+1$ del *encoder*, donde J es la longitud de la secuencia de entrada. Para calcular la representación $\mathrm{h}_j^{e,1}$ del elemento situado en la posición j en el primero bloque, al no haber un bloque previo $\mathrm{h}_{j'}^{e,0}$, se utilizará como entrada la representación inicial correspondiente de cada elemento $x_{j'}$.

$$p_s(j'|j;l+1) = \text{Softmax}(a[\mathrm{h}_j^{e,l}, \mathrm{h}_{j'}^{e,l}]) \tag{2.3}$$

$$\mathrm{h}_j^{e,l+1} = \sum_{j'=1}^{J} p_s(j'|j;l+1)\mathrm{h}_{j'}^{e,l}; \quad \mathrm{h}_j^{e,0} = x_j \quad 1 \leq j \leq J \tag{2.4}$$

Las ecuaciones 2.5, 2.6, 2.7 y 2.8 son las que se utilizan en el bloque $l+1$ del *decoder* d para $0 \leq l \leq$ L-1, donde L es el número total de bloques que lo constituyen.

En la ecuación 2.5 se muestra como se computa la auto-atención normalizada para el bloque $l+1$ entre las representaciones contextuales u obtenidas por el *decoder*, entre los elementos situadas en la posición i e i' respectivamente en la secuencia de salida que se está generando. En la ecuación 2.6 se utiliza la auto-atención para computar la representación $\mathrm{h}_{i'-1}^{d,l+1}$ del elemento localizada en la posición $i'-1$ de la secuencia de salida para el bloque $l+1$ del *decoder*.

$$p_{t,s}(i'|i;l+1) = \text{Softmax}(a[\mathrm{u}_{i-1}^{l}, \mathrm{u}_{i'-1}^{l}]) \tag{2.5}$$

$$\mathrm{h}_{i-1}^{d,l+1} = \sum_{i'=1}^{i-1} p_{t,s}(i'|i;l+1)\mathrm{u}_{i'-1}^{l} \tag{2.6}$$

En la ecuación 2.7 se calcula la atención-cruzada normalizada para el bloque $l+1$ del *decoder* entre las representaciones $h^{e,L}$ del último bloque L del *encoder* y las representaciones $h^{d,l}$ del *decoder* en el bloque l, donde j hace referencia a la posición de un elemento en la secuencia de entrada e i hace referencia a la posición de un elemento en la secuencia de salida. En la ecuación 2.8 se utiliza la atención-cruzada para computar la representación contextual u_i^{l+1} en el bloque $l+1$ del elemento de salida situada en la posición i. Para el primer bloque del *decoder*, la representación contextual u_i^0 del elemento situado en la posición i de la secuencia de salida es la representación inicial correspondiente al elemento previo y_{i-1}.

$$p_{t,c}(j|i;l+1) = \text{Softmax}(a[\mathrm{h}_{i-1}^{d,l}, \mathrm{h}_j^{e,L}]) \tag{2.7}$$

$$\mathrm{u}_i^{l+1} = \sum_{j=1}^{J} p_{t,c}(j|i;l+1)\mathrm{h}_j^{e,L}; \quad \mathrm{u}_i^0 = y_{i-1} \tag{2.8}$$

En las próximas subsecciones, se explica con más detalle los diferentes componentes que constituyen al modelo *Transformer* (ver subsecciones 2.3.1 y 2.3.2). Finalmente, se mostrará como los distintos componentes se comunican entre ellos en la arquitectura final (ver subsección 2.3.3).

2.3.1. Mecanismos de atención

El mejor ejemplo para poder explicar en qué consiste el mecanismo de atención es la traducción automática. En la figura 2.3 se muestra la tabla final de atención obtenida al traducir una oración en francés al inglés. El mecanismo de atención indica en que palabras de la oración fuente (francés) hay que tener en cuenta,"poner atención", para poder generar una nueva palabra en la oración destino (inglés). Dependiendo de la palabra a decodificar los elementos del contexto en los que haya que fijarse serán diferentes, de esta manera se obtienen representaciones del contexto variables y no fijas, como son las que se consiguen cuando una secuencia de texto es procesada por una RNR.

Como se ve en el ejemplo de la figura 2.3, la palabra en inglés *equipment* solamente pone atención en dos palabras en francés. La que deposita la mayoría de su atención es *équipment*, ya que es su correspondiente traducción y la poca atención sobrante la fija en el artículo *l'*, en cambio no pone nada de atención en la palabra *Syrie* o en las demás, ya que no son necesarias para poder generar la palabra *equipment*.

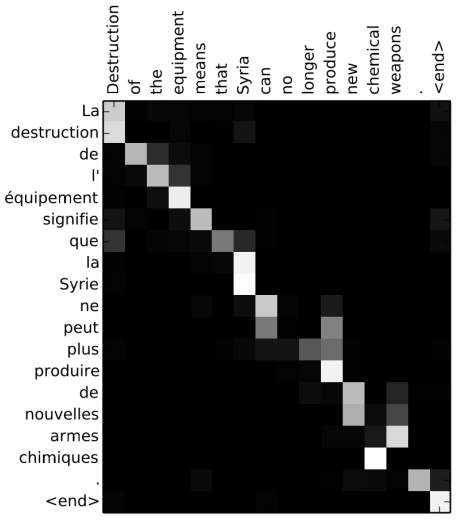

Figura 2.3: Tabla de atención obtenida al traducir una oración del francés al inglés.

En la ecuación 2.9 se muestra como se calcula la atención. La matriz *key* $K \in \mathbb{R}^{m \times d}$ y la matriz *value* $V \in \mathbb{R}^{m \times d}$ ambas representan las m palabras de la oración fuente, donde d es el número de características con las que es representada cada palabra. Estas representaciones son obtenidas por el *encoder*. La matriz *query* $Q \in \mathbb{R}^{n \times d}$ representa las n palabras predichas en la oración destino por el *decoder*. El primer cálculo que se realiza es la similaridad entre las palabras que forman el contexto (*key*) y las palabras predichas (*query*). Este cálculo puede llevarse a cabo realizando el producto escalar de las dos matrices (ver ecuación 2.10) escalándolo por la dimensionalidad de la matriz Q y V que es d. Realmente, el resultado de este cálculo es una tabla de puntuaciones sin normalizar, donde para cada *query* concreta se tiene una puntuación de similaridad con todas las *keys*. Por esa razón, se le aplica la función Softmax a esa tabla para normalizarlas. Finalmente, las puntuaciones normalizadas se multiplican por el contexto V, de esta manera se obtiene

una representación ponderada de éste que ayuda notablemente a realizar la predicción de la próxima palabra en la oración destino.

$$\text{Attention}(K, V, Q) = \text{Softmax}(\text{score}(Q, K)V) \tag{2.9}$$

$$\text{score}(Q, K) = \frac{QK^T}{\sqrt{d}} \tag{2.10}$$

El *multi-head attention* surge para aportar mayor capacidad de generalización al mecanismo de atención. Para ello se realizan H operaciones de atención en vez de una. Estas operaciones se denominan *head* normalmente son 8, $H = 8$. Para cada una de las matrices de entrada: Q, K, V; se aplica una transformación lineal distinta en cada uno de los *heads* (ver ecuación 2.11). Adicionalmente, las matrices de pesos que se utilizan para realizar estas transformaciones lineales: $W_h^K, W_h^V, W_h^Q \in \mathbb{R}^{d \times \frac{d}{H}}$ para $h \in [1, H]$ reducen la dimensión de la matriz de entrada acorde al número de *heads* que se vayan a emplear, ya que sino el número de parámetros a estimar podría ser muy elevado requiriendo de mayor tiempo de entrenamiento y dificultando la convergencia de los pesos en mayor medida. El objetivo principal de aplicar diferentes transformaciones lineales en cada *head*, es para que cada uno de ellos represente un subespacio distinto, donde se focalice en partes diferentes de la oración capturando patrones diversos. La salida de los H *heads* es concatenada, y se le aplica una transformación lineal con la matriz de pesos $W^O \in \mathbb{R}^{d \times d}$ (ver ecuación 2.12). La atención obtenida por *multi-head attention* en la ecuación 2.12 es más efectiva que la calculada utilizando el mecanismo estándar (ver ecuación 2.9), en consecuencia la compresión de los resultados obtenidos es mucho más difícil de interpretar debido a su mayor complejidad.

$$\text{head}_h = \text{Attention}(KW_h^K, VW_h^V, QW_h^Q) \tag{2.11}$$

$$\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.12}$$

Al principio de esta sección se explica el mecanismo de atención a partir de un ejemplo de traducción automática, donde se utiliza una atención-cruzada entre las representaciones proporcionadas por el *encoder* de las palabras de la oración a traducir y las representaciones proporcionadas por el *decoder* de las palabras que ya se han predicho (oración traducida). El *Transformer* destaca principalmente por usar auto-atención que es una atención donde las representaciones de las palabras que constituyen las *queries* y las *keys* pertenecen a la misma oración, es decir que son las mismas. Al aplicar esta auto-atención se podrá desambiguar mejor las palabras que constituyen la oración y de ese modo tener una mejor representación semántica de la oración, otorgando una información más valiosa a las capas posteriores permitiéndoles que puedan discriminar de forma más precisa.

2.3.2. Codificación posicional

El modelo *Transformer* al estar constituido por capas basadas en el mecanismo de atención en vez de por capas que contienen recurrencias, no es capaz de captar la noción del orden debido a que no hay ninguna señal secuencial, ya que el mecanismo de atención está basado en el contenido de la memoria de los pares *key* y *value*, no en la localización.

Para que el mecanismo de atención pueda percibir la localización de las palabras, en el modelo *Transformer* propusieron la codificación posicional, donde dado un número n: posición de la palabra en la oración; se puede generar una representación vectorial de

la misma dimensión D que el *embedding* de la palabra. De esta manera, si se suma esta representación posicional al *embedding* de la palabra, cuando posteriormente se lleve a cabo la atención, esta vez sí que podrá ser sensible al orden en la oración.

En la ecuación 2.13 se muestra como se calcula la codificación posicional dado la posición de la oración n para una representación de D componentes $PE(n) \in \mathbb{R}^D$. Si la posición del componente es par se utiliza la función Seno, en caso contrario se utiliza la función Coseno.

$$PE(n)_d = \begin{cases} \text{Seno}(10000 - \frac{d}{D}n), & \text{si } d \text{ es par} \\ \text{Coseno}(10000 - \frac{d}{D}n), & \text{si } d \text{ es impar} \end{cases} \tag{2.13}$$

2.3.3. Arquitectura

En esta subsección se explica de forma detallada la arquitectura *Transformer* (ver figura 2.4). Esta arquitectura está basada en la estructura *encoder-decoder* donde ambos componentes tienen su propio bloque que se repite N veces, normalmente 6, $N = 6$. La salida del bloque N del *encoder* es una de las señales de información para cualquier bloque i del *decoder*. Cuando el bloque no es el inicial, ya sea en el *encoder* o en el *decoder*, su entrada es la salida del bloque previo $i - 1$, en caso de que sí que sea el bloque inicial será la secuencia de *embeddings* a los cuales se les ha sumado la codificación posicional (ver subsección 2.3.2). La secuencia de *embeddings* del *encoder* proviene de la secuencia de entrada x y en el caso de *decoder* proviene de la secuencia de salida y que se está generando.

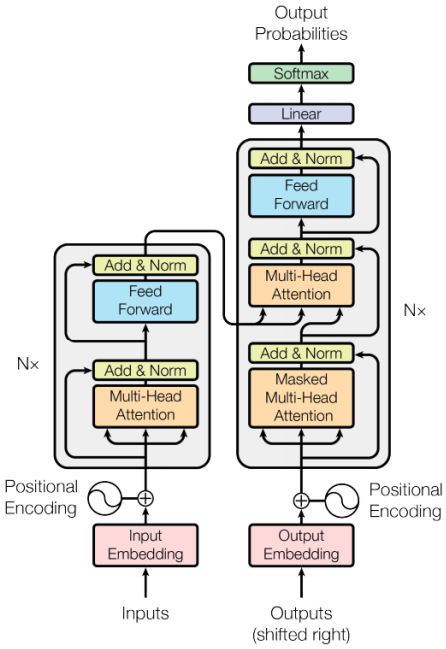

Figura 2.4: Arquitectura completa del modelo Transformer.

En esta arquitectura la secuencia de entrada solo es procesada una vez por el *encoder*, en cambio en el *decoder* se harán varias pasadas de la secuencia de salida, ya que en cada nuevo instante de tiempo se genera un nuevo elemento, por lo tanto, los elementos que constituyen esta secuencia de salida van incrementándose en cada nuevo instante de tiempo. Los nuevos elementos se generan tomando la salida del último bloque del *decoder* e introduciéndose en una capa que realiza una transformación lineal. Al resultado

obtenido se le aplica la función *Softmax*, donde se tomará como elemento aquel que tenga mayor probabilidad. El *decoder* finalizará cuando prediga el elemento final de secuencia.

Una vez que ya se ha dado una descripción general del *Transformer* vamos a centrarnos en explicar con mayor profundidad los bloques concretos que lo constituyen. Primero hablaremos del bloque del *encoder*. Su primera capa es un *multi-head attention* (ver ecuación 2.12) en el que se aplica una auto-atención, posteriormente se realiza una conexión residual entre la entrada y la salida del *multi-head attention* que posteriormente se normaliza *LayerNorm*: $\text{LayerNorm}(x + \text{MultiHeadAttention}(x))$. Seguidamente, se utiliza una RNA donde se vuelve aplicar la conexión residual y la normalización igual que antes, pero esta vez se aplica en la salida de la RNA, en vez de en el de la *multi-head attention*. Finalmente, hablamos del bloque del *decoder* que es algo más complejo que el del *encoder* aunque son muy parecidos. Este bloque tiene dos *multi-head attentions*, el primero en el que se aplica una auto-atención con máscara, ya que solamente se podrán tener en cuenta los elementos anteriores. Para explicarlo mejor utilizaremos un ejemplo. Imaginemos que el *decoder* se encuentra en el instante de tiempo 5 y queremos realizar la auto-atención del elemento generado en el instante 3, las *keys* y *values* que solamente podremos tener en cuenta para ese elemento son los de los instante anteriores aunque conozcamos los elementos generados en los instantes posteriores. En cambio, en el *enconder* se toman todos los elementos de la secuencia al realizar la auto-atención, ya que se conocen desde el primer momento, por eso no se aplica ninguna máscara. El segundo *multi-head attention* del *decoder* realiza una atención-cruzada entre la salida del *encoder* en el último bloque y la salida *multi-head attention* realizada en el paso anterior del mismo bloque del *decoder*. A ambas capas de *multi-head attention* se les aplica la conexión residual y la normalización. Finalmente, las dos última capa del bloque del *decoder* son las mismas que las del *encoder*.

2.4 Modelos basados en Transformer

El modelo *Transformer* cuando se presentó demostró su gran capacidad para codificar secuencias de texto. Si se compara el modelo *Transformer* (ver sección 2.3) con la RNR (ver sección 2.2) vemos que el *Transformer* es completamente paralelizable, no tiene limitaciones de memoria y no es unidireccional, ya que el mecanismo de atención puede atender contextos de palabras que se encuentran lejanos en la oración permitiendo un mayor entendimiento. Por esos motivos, el modelo *Transformer* podría utilizarse como modelo de lengua para representaciones contextuales. Esto nunca podría conseguirse con un modelo basado en RNR.

Las representaciones contextuales proporcionan características semánticas dinámicas de las palabras, eso quiere decir, que para una misma palabra dependiendo del contexto en el que se encuentre tendrá una representación u otra. En los ejemplos que se muestran a continuación: "mobile **cell**" y "prision **cell**", la palabra *cell* tiene un significado diferente en cada una de las oraciones, en el caso de usar representaciones semánticas estáticas como es el caso de los *word embeddings* (ver sección 2.1), la representación de *cell* para ambas será la misma, en cambio, si se utiliza un modelo como BERT la representación será diferente, de esta manera se desambiguará y se obtendrá una mejor representación que permita un mayor entendimiento del sentido de la oración.

Actualmente para la inmensa mayoría de tareas de PLN que consisten en codificar secuencias de texto, los modelos basados en *Transformer* son el estado del arte. La aparición de BERT (ver subsección 2.4.1) causó un gran impacto en el campo del PLN, ya que proporcionó modelos pre-entrenados de forma semi-supervisada utilizando cantidades inmensas de texto, que hasta ese momento no se había propuesto entrenar con corpus de esas dimensiones. Lo realmente importante de estos modelos pre-entrenados es que al

realizar un refinamiento de sus pesos para que se especialicen en una tarea concreta diferente con la que habían sido entrenados previamente, pero sí relacionada, a partir de un entrenamiento supervisado utilizando el correspondiente corpus etiquetado, obtuvieran el mayor rendimiento hasta la fecha para dicha tarea. Esta aproximación que primero preentrena un modelo y posteriormente lo especializa en una tarea concreta (*fine-tunning*) se denomina *transfer learning*.

La aparición de los modelos pre-entrenados de BERT ha significado un antes y un después a la hora de afrontar muchos de los problemas a resolver en el campo del PLN, sobretodo en la clasificación de texto que es el tipo de tarea que nos enfrentamos en este trabajo. BERT ha significado una revolución en su campo de mayor envergadura que la que obtuvieron los modelos pre-entrenados basados en RNC con el conjunto de imágenes de ImageNet[2] en el campo de la visión por computador.

En las subsecciones siguientes se comentan cuales son las mejoras que se propusieron en *Transformer* para que fuera capaz de codificar las oraciones de texto de una forma tan representativa. Finalmente, se explicará las diferencias del entrenamiento de RoBERTa (ver subsección 2.4.2) con respecto del de BERT para que le permitiera obtener un rendimiento superior.

2.4.1. BERT

BERT [6] está basado en la arquitectura *Transformer* (ver sección 2.3.3), pero solamente utiliza los bloques del *encoder*. La bidireccionalidad de BERT le permite ser un modelo de lengua contextual, para conseguirlo, utiliza como función de pérdida para optimizar los pesos del modelo, una que tenga en cuenta tanto el contexto izquierdo como el contexto derecho de las oraciones. Siguiendo el esquema tradicional del modelo de lengua que dado las palabras anteriores (contexto izquierda) predice la palabra siguiente, o dado las palabras posteriores predice la palabra anterior (contexto derecho). Realmente, al concatenar las representaciones obtenidas por esos dos modelos de lengua no se está obteniendo un modelo bidireccional, se necesitaría de una aproximación algo distinta para conseguirlo. Esa aproximación es la que se proponen en BERT la cual la llaman *Masked Language Model* (MLM).

El MLM consiste en que dada una oración: "el médico corrió a la habitación de emergencias a ver a su paciente"; se enmascara aproximadamente el 15 % de los tokens que constituyen dicha oración: "el médico corrió a la `[MASK]` de emergencias a ver a `[MASK]` paciente"; con el objetivo de desenmascararlos. Para ello se tendrá que basar en la información de los tokens visibles, por lo tanto, el modelo estará condicionado tanto por el contexto derecho como por el contexto izquierdo para desenmascarar un token concreto. Para ser más precisos con este método, dentro del 15 % de los tokens que son propuestos para enmascarar, el 80 % son enmascarados, un 10 % son sustituidos por una palabra aleatoria y el 10 % restante no se enmascaran. Con esta función objetivo se obtiene un modelo de lengua contextual.

BERT adicionalmente en el entrenamiento tiene en cuenta otra función objetivo llamada *Next Sentence Prediction* (NSP).Ésta consiste en que para cada oración de entrenamiento A se tiene asociado una oración B. BERT tendrá que discriminar como si de un clasificador binario se tratase indicando si la oración B es posterior a A, es decir, que si guardan alguna relación. Tener en cuenta estas relaciones entre oraciones para algunas tareas de PLN puede ser muy útil como ocurren en *Language inference* y *Question Answering*. Normalmente, el 50 % de las oraciones asociadas eran aleatorias y el 50 % restante sí que existe alguna relación entre ellas. Para representar la pareja de oraciones A y B en el

[2] http://www.image-net.org/

entrenamiento se necesitan dos tokens especiales. El primero token es `[CLS]` que indica el principio de la oración de A y el segundo token es `[SEP]` que separa la oración A de la B, `[SEP]` realmente es el último token de A, por lo tanto, los ejemplos de entrenamiento aparecen representados de la siguiente manera: `[CLS]` A `[SEP]` B.

En la figura 2.5 se muestra como se representan la entrada: `[CLS]` A `[SEP]` B, durante el entrenamiento para el primer bloque de BERT. Como sucedía en el modelo *Transformer* se suma la representación de cada token con la correspondiente representación de la posición (ver subsección 2.3.2), adicionalmente, se suma la representación de que el token pertenezca al segmento A o B, ésto es fundamental para la función de pérdida NSP.

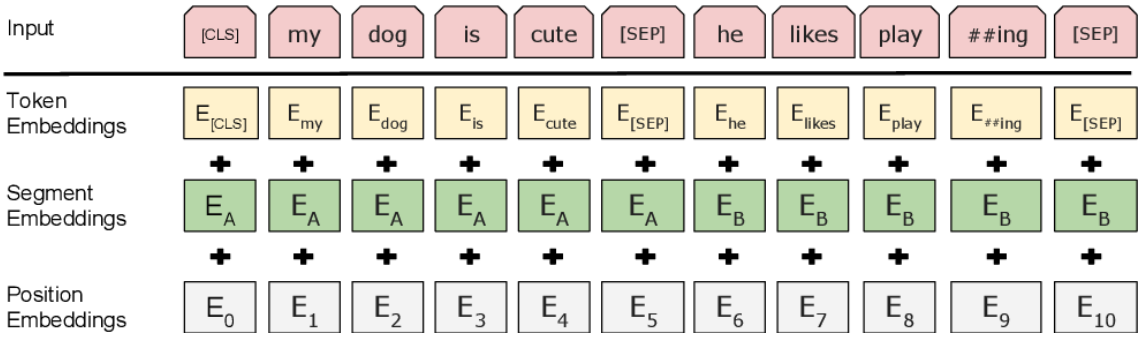

Figura 2.5: Representación de la entrada de BERT.

BERT en vez de aprender los *embeddings* de las palabras las segmenta en unidades más pequeñas llamadas subpalabras. La representación de esas unidades es lo que aprende. Esta aproximación proporciona numerosas ventajas. La primera de ellas es que reduce enormemente el tamaño del vocabulario. La segunda de ellas es que permite manejar de mejor manera las palabras desconocidas. La última de ellas es que las palabras que comparten una estructura similar de subpalabras deben compartir una misma semántica.

Los tokenizadores más populares utilizados para obtener subpalabras son el Byte-Pair Encoding (BPE) [34] y el WordPiece [30]. Ambos tokenizadores emplean un algoritmo iterativo donde el vocabulario es inicializado con todos los símbolos (caracteres) que constituyen el corpus de entrenamiento. En cada iteración el vocabulario es actualizado con los pares de símbolos existentes más frecuentes en él , en caso de utilizar el tokenizador BPE. En el caso de usar el tokenizador WordPiece, el vocabulario será actualizado por aquellos pares que obtienen mejor valor de verosimilitud en el correspondiente corpus de entrenamiento.

Las dos configuraciones disponibles de los modelos de BERT pre-entrenados con cantidades inmensas de texto en inglés son *Base* y *Large*. La configuración *Large* es la que mayor número de parámetros tiene de las dos, y con la que se obtienen resultados ligeramente superiores para las diferentes tareas de PLN con las que se experimentaron. Debido a que la diferencia del rendimiento de utilizar una configuración u otra no es muy notable y la dimensión del modelo en cuanto a número de parámetros sí que lo es (ver tabla 2.1), se utilizará para la experimentación de este trabajo la configuración *Base*, ya que, presenta el mejor equilibrio entre rendimiento y tamaño del modelo.

En la tabla 2.1 se muestra el valor para cada hiperparámetro de las dos configuraciones propuestas para BERT: *Base* y *Large*. Para cada configuración se muestra el número total de parámetros del modelo, el número de bloques B de tipo *encoder* que lo constituyen, la dimensión D de la representación obtenida al procesar una oración y el número de *heads* H empleados en el *multi-head attention*.

En BERT existen dos aproximaciones para representar las oraciones. La más común de ellas consiste en utilizar la representación obtenida por el token `[CLS]` en el último bloque

Configuración	Parámetros	B	D	H
Base	110M	12	768	12
Large	334M	24	1024	16

Tabla 2.1: Configuración de los modelos *Base* y *Large* de BERT.

del modelo pre-entrenado al procesar la oración. La segunda aproximación consiste en utilizar las representaciones obtenidas en el último bloque por cada uno de los tokens que constituyen la oración, para posteriormente calcular el valor medio o valor máximo de cada una de los D componentes de todas las representaciones.

Una de las limitaciones que tiene BERT es que no puede procesar secuencias con más de 512 tokens. Cuando sucede ésto, se tendrá que truncar la secuencia, ya sea, quedándose con los primeros o con los últimos 512 tokens, o optando por una estrategia que solamente tome los tokens del principio y del final de la secuencia, ignorando los tokens del medio.

2.4.2. RoBERTa

RoBERTa [22] es una extension del modelo BERT, el cual aplica algunas modificaciones en varios aspectos del entrenamiento con el objetivo de mejorar el rendimiento obtenido por BERT para algunas tareas concretas.

En RoBERTa la función objetivo NSP se elimina, ya que a través de diferentes experimentos demuestran que no es óptima. En consecuencia, los tokens especiales que empleaba BERT para esta función objetivo se eliminan: `[CLS]` y `[SEP]`, no obstante, se emplean otros dos tokens especiales. Uno de los tokens es `<s>`, el cual será el primer token del ejemplo A y el otro es `</s>`, el cual será el último token del ejemplo A, de tal manera que el ejemplo será representado de este modo: `<s>` A `</s>`. Por otro lado, la función objetivo MLM se mantiene pero con una modificación, ya que en BERT emplean un enmascaramiento estático, donde cada ejemplo para las diferentes iteraciones llevadas a cabo durante el entrenamiento tiene aplicado el mismo enmascaramiento, eso en RoBERTa no ocurre, ya que proponen un enmascaramiento dinámico, donde cada ejemplo se le aplican 10 enmascaramiento distintos, causando que el corpus de entrenamiento sea 10 veces mayor de lo que lo era inicialmente. En los diferentes experimentos realizados probando esta aproximación dinámica del MLM obtiene mejoras en el rendimiento con respecto de la aproximación estática.

En cuanto aspectos de hiperparámetros durante el entrenamiento, en RoBERTa emplean tamaños de *mini-batch* y de factor de aprendizaje de mayor valor que los empleados en BERT. Adicionalmente, la longitud de la secuencia de entrada de los ejemplos de entrenamiento también es mayor.

Finalmente, se comentan dos modificaciones más incorporadas por RoBERTa. La primera de ellas consiste en utilizar vocabulario de 50k subpalabras en vez de 30k como hacía BERT. Y la segunda de ellas consiste en emplear un conjunto de entrenamiento enormemente mayor. BERT solamente emplea el corpus BookCorpus (16GB), RoBERTa adicionalmente emplea tres corpus más: CC-NEWS (76G), OpenWebText (38G) y Stories (31G). Por lo tanto, el corpus de entrenamiento de BERT es de 16 GB y el de RoBERTa es de 161 GB, la diferencia es abismal.

Con todos las modificaciones comentadas previamente se consigue una mejora del 15 % con respecto BERT en el SuperGLUE *benchmark* [38], el cual está compuesto por diferentes tareas de PLN relacionadas con el entendimiento del lenguaje, donde una de ellas es la clasificación de texto. Por ese motivo en este trabajo emplearemos el modelo

pre-entrenado de RoBERTa en vez del de BERT, ya que, con las mismas dimensiones del modelo se obtiene un mejor rendimiento. El coste de aplicar *fine-tunning* a un modelo pre-entrenado de BERT o de RoBERTa es prácticamente el mismo, no obstante, hay que remarcar que todas las modificaciones incorporadas por RoBERTa para obtener un mejor rendimiento, hacen que el coste de pre-entrenamiento del modelo sea mucho más costoso que pre-entrenar el correspondiente modelo de BERT.

CAPÍTULO 3
Técnicas de aumentado de datos

En este tercer capítulo explicamos de forma detallada las diferentes técnicas de aumentado de datos que emplearemos para generar nuevos ejemplos sintéticos a partir de los ejemplos originales. Las tres técnicas que hemos seleccionado son las más populares en el campo del PLN para aumentar el conjunto de datos de entrenamiento. La primera que explicaremos será Easy Data Augmentation (ver sección 3.1), la segunda será *Back-translation* (ver sección 3.2) y la última será la técnica basada en aumentado contextual que realiza sustituciones de palabras utilizando modelos pre-entrenados (ver sección 3.3).

3.1 Easy Data Augmentation

La primera técnica con la que se va a experimentar es Easy Data Augmentation (EDA) [39]. De las tres técnicas a explorar ésta es la más rápida y la menos costosa para generar nuevos ejemplos sintéticos, ya que realiza modificaciones muy simples en el ejemplo original. Esta técnica requiere de dos hiperparámetros. El primero es el número de ejemplos sintéticos que se generan a partir de la oración original N y el segundo es α que dependiendo de la operación que se vaya aplicar para generar un nuevo ejemplo tendrá una utilidad u otra, no obstante se aconseja que su valor sea pequeño, inferior a 0.2.

A continuación, explicamos las cuatro operaciones que se llevan a cabo con esta técnica. Antes hay que indicar como se obtiene el valor del parámetro p, donde $p = \alpha$, y el valor del parámetro n, donde $n = \alpha * l$, teniendo en cuenta que l es el número de palabras que constituyen la oración. El valor de n depende de l para evitar que las oraciones compuestas por mayor número de palabras se les aplique mayor ruido que a las oraciones compuestas por menor número de palabras.

- Sustitución de sinónimos (SUS-S): Aleatoriamente se eligen n palabras de la oración original que no sean palabras de parada. Cada una de esas palabras seleccionadas se sustituye aleatoriamente por una de sus correspondientes sinónimos.
- Inserción aleatoria (INS-A): Encontrar aleatoriamente el sinónimo de una palabra aleatoria de la oración que no sea una palabra de parada. Insertar ese sinónimo en una posición aleatoria de la oración. Realizar este procedimiento n veces.
- Intercambio aleatorio (INT-A): Aleatoriamente elegir dos palabras de la oración y realizar el intercambio de posiciones entre ellas dos. Realizar este procedimiento n veces.
- Eliminación aleatoria (ELI-A): Para cada palabra de la oración eliminarla aleatoriamente, si obtiene una probabilidad p.

De las cuatro operaciones listadas previamente, exceptuando la primera: SUS-S, las tres restantes añaden ruido al ejemplo original. En la tabla 3.1 se muestra para cada una de las cuatro operaciones de EDA, ejemplos aumentados obtenidos a partir de aplicarlas.

Operación	Oración
-	A sad, superior human comedy played out on the back roads of life.
SUS-S	A ***lamentable***, superior human comedy played out on the ***backward*** road of life.
INS-A	A sad, superior human comedy played out on ***funniness*** the back roads of life.
INT-A	A sad, superior human comedy played out on ***roads*** back ***the*** of life.
ELI-A	A sad, superior human out on the roads of life.

Tabla 3.1: Ejemplos sintéticos obtenidos a partir de aplicar las diferentes operaciones de la técnica de aumentado de datos EDA.

3.2 Back-translation

La segunda técnica con la que se va a experimentar es *back-translation* [31]. Esta técnica requiere de dos idiomas, el idioma al que pertenecen las oraciones originales A y el idioma pivote B.

El procedimiento llevado a cabo en este método consiste en que dado una oración en el idioma A, utilizando un traductor automático en la dirección $A \rightarrow B$ se obtiene la oración traducida correspondiente al idioma B. Posteriormente, con esa oración traducida al idioma B se utiliza otro traductor automático, esta vez en la dirección opuesta $B \rightarrow A$, de esta manera se obtiene una oración en el mismo idioma que la original A. Ésta última oración obtenida, realmente es una reconstrucción de la oración original, en muchos casos no es igual a la original pero sí que suele conservar el mismo significado semántico, es como si de un parafraseo se tratase. En otras circunstancias dependiendo de la complejidad de la oración y del rendimiento del traductor empleado, la oración obtenida es la misma que la original, en ese caso ese ejemplo aumentado no se incluiría en el entrenamiento ya que no aportaría nada.

En la tabla 3.2 se muestra la oración original y tres oraciones aumentadas a partir de aplicar la técnica *back-translation* (BT). El ejemplo mostrado en la tabla ha sido extraído del siguiente artículo [41]. Ellos utilizan el corpus WMT'14 Inglés-Francés para entrenar los dos traductores automáticos, donde el idioma A es el inglés y el idioma B es el francés. Ellos conservan diferentes traductores, no solamente el mejor obtenido durante el entrenamiento. Para generar las traducciones juegan con los diferentes traductores para cada sentido y con el valor de la temperatura del *decoder* que es el que le permite variar el espacio de búsqueda en la generación de texto, de esta manera produce traducciones algo diferentes a lo esperado. Con esta aproximación son capaces de obtener ejemplos aumentados muy diversos (ver tabla 3.2), pero que conservan el significado semántico original, por lo tanto, la calidad de los ejemplos aumentados obtenidos al emplear esta aproximación es alta.

Original	Given the low budget and production limitations, this movie is very good.
BT	Since it was highly limited in terms of budget, and the production restrictions, the film was cheerful. There are few budget items and production limitations to make this film a really good one. Due to the small dollar amount and production limitations the ouest film is very beautiful.

Tabla 3.2: Ejemplos aumentados obtenidos a partir de aplicar la técnica de aumentado de datos *back-translation*.

El inconveniente de esta técnica es que se requiere de dos traductores automáticos para poder llevarla a cabo. No obstante, existen varías alternativas, la más complicada

consisten en entrenar desde cero tus propios sistemas de traducción automática o directamente utilizar unos que ya estén entrenados, pero para poder utilizarlos se necesita de un equipo con potencia y de un *software* específico que permita disponer de las traducciones en un periodo de tiempo corto. Por esa razón, la opción más sencilla y rápida es utilizar algún servicio en la nube que a través de peticiones responda con las correspondientes traducciones, pero seguramente, estos servicios tendrán algún límite de peticiones por unidad de tiempo, así que requerirán de alguna suscripción de pago. Por lo tanto, dependiendo de la situación de cada usuario, unas opciones serán mejores que otras.

3.3 Sustitución de palabras utilizando modelos pre-entrenados

La tercera y la última técnica de aumentado de datos para PLN con la que se va a experimentar es la utilizada en el artículo TinyBERT [15]. Esta aproximación requiere de un modelo pre-entrenado basado en BERT (ver subsección 2.4.1) y de un modelo pre-entrenado basado en *word embeddings* (ver sección 2.1) como veremos posteriormente.

El método de aumentado de datos propuesto en TinyBERT consiste en enmascarar una palabra de la oración para posteriormente utilizar BERT como modelo de lengua contextual para obtener los M candidatos (normalmente $M = 15$) que tienen más probabilidad de ocupar esa posición en la oración, conservando el resto de palabras intactas. La palabra enmascarada tiene una probabilidad p_t (normalmente $p_t = 0{,}4$) de ser sustituida por una de las candidatas seleccionada de forma aleatoria. Este procedimiento se lleva a cabo para cada palabra de la oración. El ejemplo aumentado se obtendrá cuando se haya iterado por todas las palabras de la oración, por lo tanto, por cada pasada que se realice en la oración aplicando está técnica, solamente se obtendrá un ejemplo aumentado. Por ese motivo, se realizarán N pasadas en la oración para obtener N ejemplos aumentados distintos.

Ellos en sus primeros experimentos se dieron cuenta de que utilizar BERT para las palabras que están subdivididas en varios tokens (subpalabras) no era una buena opción, ya que generaban sustituciones de baja calidad. Por ese motivo, para esas palabras que se dividían en subpalabras, en vez de utilizar BERT propusieron utilizar *word embeddings*. Para obtener los M candidatos utilizaban la similaridad del coseno del *embedding* de la palabra completa con respecto todas las del vocabulario del modelo *word embedding* pre-entrenado. Con esta aproximación se obtenían ejemplos aumentados de mayor calidad, ya que la sustitución se realizaba directamente con la palabra y no con las subpalabras por separado.

Para utilizar esta técnica de aumentado de datos a la hora de obtener los candidatos al enmascarar una palabra de la oración, ya sea usando BERT o los *word embeddings* se requiere de la utilización de un equipo que disponga de tarjeta gráfica, ya que sino el proceso de búsqueda requerirá de mucho tiempo para realizar el computo.

CAPÍTULO 4
Experimentos

En este cuarto capítulo se proporciona la información de los experimentos a realizar. En la sección 4.1 se describe los diferentes corpus empleados con sus respectivas estadísticas. En la sección 4.2 se explican los dos modelos utilizados para la experimentación. Donde el primer modelo está basado en aprendizaje automático clásico (XGBoost) y el segundo está basado en aprendizaje profundo (RoBERTA). Finalmente, en la sección 4.3 se muestran las diferentes configuraciones empleadas para realizar el aumentado de datos en los diferentes corpus.

Adicionalmente, la implementación *open source* de este trabajo final de máster está disponible en GitHub[1] con el objetivo de poder reproducir los experimentos realizados.

4.1 Corpus

Como se explicó en la vista general del primer capítulo (ver sección 1.1), este trabajo tiene como objetivo fundamental explorar la utilidad de aplicar técnicas de aumentados de datos en PLN, con la finalidad de obtener clasificadores de texto que obtengan mayor rendimiento. Adicionalmente, también se propuso como objetivo extra que está fuertemente ligado al objetivo fundamental, el cual consiste en experimentar en conjuntos de datos que pertenecieran al dominio de la atención al cliente. En los cuales, dado la *query* del usuario se tiene que detectar el *intent* correspondiente.

En base a las objetivos del proyecto se han elegido 6 conjuntos de datos idóneos para poder cumplirlos. Los dos primeros conjuntos de datos son los que pretenden satisfacer el objetivo fundamental. Estos corpus son los más populares en la literatura de clasificación de texto. El primero de ellos es el SST-2 [35] que es de análisis de sentimiento (positivo y negativo) de las opiniones en distintas películas. El segundo de ellos es TREC [20] que está basado en clasificación de seis tipos preguntas. El tipo de preguntas puede estar relacionado con información numérica, con personas, con localizaciones, entre otras. El resto de corpus que se van a comentar a continuación están relacionados con el objetivo adicional que consiste en la detección de *intents*. SNIPS [10] contiene 7 *intents* distintos que han sido colectados por un asistente de voz, las *queries* están relacionadas con poner una canción, puntuar un libro, entre otras. Los tres corpus restantes pertenecen al dominio de la atención al cliente, donde se utiliza un corpus que es público que se puede descargar en GitHub Question-Topic[2] y un corpus privado proporcionado por una empresa donde los datos de los clientes están anonimizados. Este corpus privado tiene dos versiones, la primera versión donde las *queries* se clasifican por categorías RAIC y la segunda por

[1] https://github.com/franborjavalero/data-augmentation-for-text-classification
[2] https://github.com/sambit9238/Machine-Learning/blob/master/question_topic.csv

tópicos RAIT. Las categorías es una generalización de los tópicos. Los *intents* de los tres últimos corpus están relacionados con pedidos, con entregas, con productos, con perfiles de usuario, en definitiva con información relacionada con todo el proceso previo y posterior de la compra de un producto en una tienda online.

En la tabla 4.1 se muestran las estadísticas de los diferentes corpus propuestos donde se indica el número de clases *c*, el número de ejemplos totales *N* (test incluido) y el número de ejemplos de test. Hay que remarcar que en los corpus privados de los clientes RAIC y RAIT, inicialmente el número de clases era mucho mayor, pero debido que para alguno de ellos el número de ejemplos de entrenamiento era demasiado reducido solo se conservaron aquellas clases que tuvieran más de 70 ejemplos de entrenamiento, las clases restantes se omitieron para la experimentación.

Corpus	c	N	Test
SST-2	2	9613	1821
TREC-6	6	5952	500
SNIPS	7	14484	700
Question-Topic	7	5017	1008
RAIC	13	9831	1967
RAIT	25	9261	1864

Tabla 4.1: Estadísticas de los corpus propuestos. Donde c es el número de clases, N es el número de ejemplos totales y Test es el número de ejemplos de test.

En la experimentación utilizaremos diferentes tamaños del corpus de entrenamiento para cada uno de los corpus a explorar. Inicialmente, utilizaremos todo el conjunto de entrenamiento disponible. Posteriormente, simularemos escenarios con escasos recursos de datos de entrenamiento en base al número de ejemplos de entrenamiento por clase (NEPC). La empresa que nos ha proporcionado el corpus RAIC y RAIT suele disponer de alrededor de 100 ejemplos por clase, por ese motivo exploraremos para todos los corpus como de efectivas son las técnicas de aumentados de datos en PLN cuando el NEPC de entrenamiento es 20, 50 y hasta un máximo de 80. En consecuencia, el NEPC para el conjunto de validación también sería reducido (ver tabla 4.2).

Escenario	NEPC-E	NEPC-V
A	20	5
B	50	12
C	máx. 80	20

Tabla 4.2: Configuración de los escenarios que disponen de escasos recursos de entrenamiento. Donde NEPC-E es el NEPC para el conjunto de entrenamiento y NEPC-V es el NEPC para el conjunto de validación.

4.2 Modelos

Para realizar un estudio más interesante de la efectividad de las técnicas de aumentado de datos para el PLN hemos utilizado dos modelos de clasificación de diferente naturaleza.

El primero de ellos está basado en un modelo de aprendizaje automático clásico llamado XGBoost [4]. Este modelo es una implementación avanzada y eficiente del algoritmo de *gradient boosting* [8] [9]. XGBoost es un ensamble de muchos predictores que son añadidos secuencialmente, los cuales están entrenados con el error residual obtenido por los predictores previos. A diferencia del algoritmo estándar de *gradient boosting*, éste incluye una función objetivo regularizada. Además de ajustar el nuevo predictor en cada

nueva iteración en base a una aproximación de segundo orden rápida y voraz. Este modelo es popularmente conocido por haber obtenido las mejores soluciones en diferentes competiciones de aprendizaje automático.

Para este trabajo, las características extraídas utilizadas en el modelo XGBoost para realizar la clasificación de texto son obtenidas a partir de utilizar *word embeddings* (ver sección 2.1). Para ello, nosotros hemos utilizado el modelo pre-entrenado con los corpus en inglés de *Common Crawl*[3] y de *Wikipedia*[4] proporcionados por fastText[5], donde la dimensionalidad de los *embeddings* es de 300 componentes. La representación introducida en el modelo XGBoost para cada ejemplo está basada en el método Sec2Vec, el cual obtiene un vector a partir de una oración de texto. Para ello, realiza la suma de los *word embeddings* correspondientes de cada palabra p que constituyen el ejemplo e. Las representaciones de cada palabra son proporcionadas por el modelo pre-entrenado W_{emb} (ver ecuación 4.1). Una vez obtenido el vector v que representa el ejemplo e a partir de la suma de los correspondientes *word embeddings*, éste se normaliza dividiéndolo por la raíz cuadrada de la suma del valor de cada componente de v elevado al cuadrado (ver ecuación 4.2), por lo tanto, ese vector normalizado de 300 componentes v_norm serán las características utilizadas por XGBoost para realizar la clasificación.

$$v(e) = \sum_{p}^{e} W_{emb}[p] \tag{4.1}$$

$$v\_norm(e) = \frac{v(e)}{\sqrt{\sum_{d}^{D} v(e)_d^2}} \tag{4.2}$$

Para obtener un modelo el XGBoost se realiza una optimización en base a diferentes hiperparámetros, donde se guarda el modelo que menor error obtiene en el conjunto de validación. En la tabla 4.3 se proponen los diferentes valores a explorar del modelo para cada hiperparámetro. Donde *n_estimators* es el número de predictores basado en árboles a emplear, *colsample_bytree* es el ratio de características que se tienen en cuenta en cada predictor, *max_depth* es la máxima profundidad permitida para los predictores, *gamma* es el menor error residual tenido en cuenta para generar un nodo hoja en el predictor y *learning_rate* es el parámetro de aprendizaje utilizado en el algoritmo de *gradient boosting*.

Hiperparámetro	**Valor**
n_estimators	50, 100, 200, 250, 500
colsample_bytree	0.6, 0.8, 1.0
max_depth	3, 4, 5, 7
gamma	0.5, 1, 1.5, 2, 5
learning_rate	0.1, 0.01, 0.05

Tabla 4.3: Valores propuestos a explorar para algunos hiperparámetros de XGBoost durante la experimentación.

El segundo de ellos, es un modelo de aprendizaje automático profundo basado en el modelo *Transformer* (ver sección 2.3). Concretamente, vamos a utilizar RoBERTa (ver sección 2.4.2) para clasificar secuencias de palabras, ya que es el estado del arte para este tipo de tarea. Realmente, RoBERTa es un modelo de lengua pre-entrenado. Para este proyecto vamos a utilizar la configuración *base* que está disponible en la librería *Transformers*

[3] https://commoncrawl.org/
[4] https://www.wikipedia.org/
[5] https://fasttext.cc/docs/en/crawl-vectors.html

de HugginFace[6]. Para poder utilizar todo el conocimiento adquirido por RoBERTa en el pre-entrenamiento y de esta manera poder especializarlo en una tarea concreta *transfer learning*, como es la clasificación de texto. Tendremos que expandir el modelo inicial, incorporando una capa basada en RNA que realice la transformación lineal de la representación de 768 componentes obtenida en la última capa de RoBERTa al procesar la oración de texto, en un vector cuya dimensionalidad sea el número de clases de la tarea a aprender. Adicionalmente, en el entrenamiento se aplica un *dropout*[7] del 10 % a la representación obtenida por RoBERTa, de esta manera el clasificador tendrá mayor capacidad de generalización haciéndolo más robusto.

ULMFIT [14] es el primer método de *transfer learning* aplicado PLN. El cual obtuvo para varias tareas el estado del arte. Incluso, con únicamente 100 ejemplos etiquetados obtenía el mismo rendimiento que modelos entrenados con conjuntos de entrenamiento 100 veces mayores. Este método de entrenamiento es muy sencillo de emplear si se utiliza la librería fastai[8]. Nosotros aplicaremos esta aproximación durante el entrenamiento del clasificador de texto. Del método de entrenamiento hay que destacar dos aspectos fundamentales, el primero de ellos es la definición de puntos de separación del modelo para el *fine-tunning* discriminativo y el segundo de ellos es el descongelamiento gradual de los pesos.

Tenemos que recordar que el modelo *base* de RoBERTa tiene 12 bloques (ver tabla 2.1) y que al haberlo utilizado para la clasificación de texto se ha añadido el correspondiente bloque adicional al final del modelo, por lo tanto este clasificador de texto consta de 13 bloques. Cada bloque del modelo captura diferente tipo de información, por lo tanto en cada bloque se debería aplicar un *fine-tunning* distinto, en base al valor del parámetro de aprendizaje. Ellos establecieron que el valor del parámetro de aprendizaje sería más alto cuando más cerca estuviera del bloque de clasificación, por lo tanto los primeros bloques de RoBERTa serán los que tienen fijados el parámetro de aprendizaje a un valor más bajo. Para clasificador de texto basado en RoBERTa hemos establecido tres puntos de separación. El primero de ellos consta solamente del bloque de clasificación. El segundo punto de separación consta de los pesos del primer corte y de los pesos que constituyen el último bloque de RoBERTa. En el tercer y último corte consta de los pesos del segundo corte más los pesos que constituyen el penúltimo bloque de RoBERTa.

El método de entrenamiento propuesto en ULMFIT funciona por ciclos, para nuestro caso requerimos de cuatro. En cada ciclo se parte de los mejores pesos obtenidos en el ciclo anterior en base a un conjunto de datos de validación. En caso de ser el primer ciclo se parte de los pesos proporcionados por el modelo pre-entrenado de RoBERTa, y para el bloque de clasificación se inicializan aleatoriamente. Realmente es como si en cada ciclo se realizara un entrenamiento "nuevo".

Los tres primeros ciclos del entrenamiento están asociados a los tres cortes comentados previamente, donde los pesos que pertenecen al corte se refinarán y el resto se mantendrán congelados, es decir, no se modificarán sus valores. En el cuarto y último ciclo todos los pesos del modelo se refinarán. Como se aprecia, a medida que se avanza en el entrenamiento un mayor número de pesos son tenidos en cuenta para realizar refinamiento, a esto se le llama descongelamiento gradual de los pesos.

Para la implementación del modelo de RoBERTa para la clasificación de texto combinaremos la librería *Transfomers* de HugginFace[9] para la utilización de los modelos preentrenados y tokenizadores correspondientes junto con la librería fastai para realizar el

[6] https://huggingface.co/roberta-base

[7] Dada una probabilidad desactiva aleatoriamente componentes de una matriz de pesos o de un vector de características asignándoles el valor 0 a dichos componentes.

[8] https://docs.fast.ai/

[9] https://huggingface.co/transformers/

entrenamiento utilizando la aproximación propuesta basada en ULMFIT. Para la implementación de XGBoost utilizaremos la correspondiente librería[10] y para la búsqueda de los mejores valores propuestos para cada hiperparármetro (ver tabla 4.3) en base a un conjunto de validación emplearemos Hyperopt[11].

4.3 Aumentado de datos

En esta sección se van a especificar las diferentes configuraciones de las tres técnicas de aumentado de datos empleadas en la experimentación.

4.3.1. Easy Data Augmentation

La primera técnica empleada para realizar el aumentado de datos es Easy Data Augmentation (EDA) (ver sección 3.1). Para esta técnica utilizamos las tres configuraciones recomendadas en el artículo original (ver tabla 4.4). Hay que recordar que el parámetro N es el número de ejemplos aumentados generados a partir del original y que el parámetro α es como la probabilidad de hacer modificaciones en el ejemplo original. Cuando mayor sea el valor de éste, mayor diferencia se obtendrá en los ejemplos aumentados con respecto al ejemplo original del que parten. El parámetro α está explicado con mayor detalle en la sección 3.1.

Configuración	N	α
A	16	0.05
B	8	0.05
C	4	0.1

Tabla 4.4: Configuraciones propuestas de la técnica de aumentado de datos EDA. Donde N es el número de ejemplos aumentados y α es la probabilidad de modificación del ejemplo original.

El código empleado por los autores que proponen esta técnica está público en GitHub, pero debido a que en este trabajo hemos seguido un estándar concreto para tener todos los corpus estructurados de la misma manera hemos modificado la función de lectura del código original. No obstante, la parte del código relacionada con el aumentado de datos no se ha manipulado. El código modificado por nosotros es un *fork* del código original que se encuentra en el siguiente repositorio de GitHub[12].

Para está técnica de aumentado de datos, el número de ejemplos sintéticos será N veces mayor al número de ejemplos originales. Por lo tanto, la configuración C será la que menos ejemplos sintéticos genere y la configuración A la que más. No obstante, todos los ejemplos sintéticos repetidos generados se eliminarán.

Aquellos modelos cuya configuración de aumentado de datos en el conjunto de entrenamiento esté basada en la técnica propuesta en esta subsección aparecerán codificados de la siguiente manera: **EDA-CONF**, donde **CONF** será una de las configuraciones propuestas en la tabla 4.4. En el caso de que la configuración elegida sea A, la codificación correspondiente será **EDA-A**.

[10] https://xgboost.readthedocs.io
[11] http://hyperopt.github.io/hyperopt/
[12] https://github.com/franborjavalero/eda_nlp/tree/fran

4.3.2. Back-translation

La segunda técnica empleada para realizar el aumentado de datos es *back-translation* (ver sección 3.2). Para esta técnica experimentamos con varios idiomas pivote. También realizaremos experimentos donde combinamos las *back-translated* generadas por diferentes idiomas pivotes en vez de solamente utilizar las de uno.

En la tabla 4.5 se muestran los idiomas pivotes empleados, cabe destacar que primero se realizó la experimentación en los escenarios con pocos recursos y en base a los resultados obtenidos se eligieron los idiomas más efectivos que son los que están en negrita. Con ésos se experimentó utilizando todo el conjunto de entrenamiento.

Italiano (it)	**Francés (fr)**	Alemán (de)	**Catalán (ca)**	**Ruso (ru)**
Checo (cs)	**Estonio (et)**	Holandés (nl)	Indonesio (id)	**Bengalí (bn)**

Tabla 4.5: Los idiomas pivotes utilizados en el método de aumentado de datos *back-translation*.

En la tabla 4.6 se muestra tanto para el escenario completo como para los escenarios reducidos las configuraciones donde se combinan en el conjunto de datos de entrenamiento los ejemplos generados por diferentes idiomas pivote (ver tabla 4.5). Como se ha dicho previamente se realizó una mayor experimentación en los escenarios reducidos por eso se proponen cuatro en ellos y solamente uno en el completo.

Escenario	**Combinación de diferentes idiomas**
Completo	fr + ca + ru + cs + et + bg
Reducido	ca + fr
	cs + et + nl + id + bg
	it + fr + de + ca + ru + cs + et + nl + id + bg
	fr + ca + ru + cs + et + bg

Tabla 4.6: Configuración de experimentos de *back-translation* donde se combinan los ejemplos generados por diferentes idiomas pivote.

El código propio desarrollado para realizar el aumentado de datos utiliza la librería *Transformers* de HuggingFace[13], la cual dispone de una funcionalidad basada en la librería rápida y eficiente de traducción automática neuronal MarianNMT[14]. Los idiomas pivotes propuestos en la tabla 4.5 han sido seleccionados a partir de los modelos pre-entrenados de traducción automática neuronal publicados gratuitamente por HuggingFace[15]. Estos modelos de traducción automática han sido entrenados con los correspondientes corpus paralelos disponible en Opus[16].

Para esta técnica de aumentado de datos por cada ejemplo original se genera uno sintético, por lo tanto esta es la aproximación que menos ejemplos aumentados genera, a no ser que se combinen los ejemplos generados por muchos idiomas pivote diferentes.

Un aspecto importante de esta técnica es que dependiendo del idioma pivote que se utilice, tenderá a generar ejemplos con mayor variabilidad al ejemplo original del que proviene. Por ejemplo, el francés y el catalán son los idiomas pivote empleadas que generan más ejemplos reconstruidos que son idénticos al ejemplo original en inglés del que provienen, en cambio para los idiomas pivote como el bengalí y el estonio los ejemplos

[13] https://huggingface.co/transformers/model_doc/marian.html
[14] https://marian-nmt.github.io/
[15] https://huggingface.co/Helsinki-NLP
[16] http://opus.nlpl.eu/

reconstruidos suelen ser mayoritariamente diferentes al ejemplo original en inglés del que provienen. Por lo tanto, con esta técnica el bengalí como pivote generará más ejemplos aumentados que el francés, debido al rendimiento de los traductores a causa de las relaciones entre los idiomas.

Aquellos modelos cuya configuración de aumentado de datos en el conjunto de entrenamiento esté basada en la técnica propuesta en esta subsección aparecerán codificados de la siguiente manera: **BT-LANG**, donde **LANG** será una de las configuraciones de idiomas propuestas (ver tablas 4.5 y 4.6). En caso de que solamente se utilice el bengalí como idioma pivote la codificación será **BT-bn**, en caso de que se utilice más de un idioma pivote como es el catalán y el francés la codificación será **BT-fr+ca**.

4.3.3. Sustitución de palabras utilizando modelos pre-entrenados

La tercera y última técnica empleada para realizar el aumentado de datos está basada en la sustitución de palabras utilizando modelos pre-entrenados (ver sección 3.3). Para esta técnica requerimos de un modelo pre-entrenado basado en BERT (ver subsección 2.4.1) y de un modelo pre-entrenado basado en *word embeddings* (ver sección 2.1).

Para el aumentado de datos de esta aproximación, solamente jugaremos con un parámetro que es el número de ejemplos aumentados N que generamos a partir del ejemplo original. Los valores de N con los que experimentaremos serán 10, 20 y 30. Los otros dos parámetros que tiene esta aproximación los mantendremos con el valor por defecto. En la sección 3.3 ambos parámetros están explicados de forma detallada.

El código empleado por los autores que proponen esta técnica está público en GitHub. Pero al igual que sucede con el código de la técnica EDA (ver subsección 4.3.1), también hemos tenido que modificar la función de lectura del conjunto de datos del código original manteniendo el código relacionada con el aumentado de datos sin tocar. El código modificado por nosotros es un *fork* del código original que se encuentra en el siguiente repositorio de GitHub[17].

Nosotros utilizaremos los mismos modelos pre-entrenados que ellos. En el caso de los *word embeddings* utilizan el modelo GloVe[18], el cual está entrenado con el corpus *Common Crawl* que está compuesto por 42 billones tokens, donde el tamaño del vocabulario es 1.9 millones tokens y la dimensión de los *embeddings* es de 300 componentes. El modelo pre-entrenado que utilizan de BERT es el *base* que está disponible en HuggingFace[19].

Para esta técnica de aumentado de datos, al igual que para EDA (ver subsección 4.3.1), el número de ejemplos sintéticos será N veces mayor al número de ejemplos originales.

Aquellos modelos cuya configuración de aumentado de datos en el conjunto de entrenamiento esté basada en la técnica propuesta en esta subsección aparecerán codificados de la siguiente manera: **SUS-N**, donde **N** será el número de ejemplos aumentados generados a partir del ejemplo original. En caso de que se realicen 10, la codificación será **SUS-10**.

[17] https://github.com/franborjavalero/Pretrained-Language-Model/tree/fran/TinyBERT
[18] http://nlp.stanford.edu/data/glove.42B.300d.zip
[19] https://huggingface.co/bert-base-uncased

CAPÍTULO 5
Resultados

En este quinto capítulo se reportan los resultados obtenidos por las diferentes técnicas de aumentado de datos propuestas para cada uno de los corpus elegidos. Los experimentos fueron realizados utilizando tanto todo el conjunto de datos de entrenamiento (ver sección 5.1), como también simulando varios escenarios distintos donde el número de ejemplos de entrenamiento es reducido (ver sección 5.2). Adicionalmente, se realizan unos tests de significancia estadística sobre los resultados obtenidos (ver sección 5.3).

5.1 Escenario completo

En esta primera sección se muestran los resultados obtenidos por el modelo basado en XGBoost (X) y por el modelo basado en RoBERTa (R) para el correspondiente conjunto de datos de test de cada corpus. La evaluación de los modelos entrenados está basada en las cuatro métricas más populares de la clasificación que son la exactitud (Acc.), la *score* F1 (F1), la precisión (precisión) y la cobertura (recall).

En la tabla 5.1 se muestra la evaluación de los modelos entrenados con todo el conjunto de entrenamiento sin utilizar ninguna técnica de aumentado de datos. Los resultados obtenidos en esta tabla son los que habrá que mejorar aplicando en el conjunto de datos de entrenamiento alguna de las diferentes técnicas de aumentados de datos propuestas en este trabajo (ver capítulo 3). Como es obvio, el rendimiento obtenido por los modelos basados en RoBERTa es muy superior al obtenido por los modelos basados en XGBoost. Realmente, estos dos modelos no son comparables teniendo en cuenta el conocimiento del que parte el clasificador basado en RoBERTa. No obstante, es interesante que para algunos corpus concretos, como es el caso de SNIPS y el de Question-Topic el modelo basado en XGBoost obtenga un rendimiento elevado superando el valor del 0.90 en la métrica F1. También es verdad, que estos dos corpus son los más sencillos, en los cuales los ejemplos que pertenecen a la misma clase siguen una distribución muy similar. Por otro lado, para todos los corpus públicos RoBERTa obtiene un rendimiento superior al 0.93 en la métrica F1, incluso para tres de ellos supera el 0.98. Por lo tanto, parece muy complicado que una técnica de aumentado de datos compense en utilizarla para mejorar el rendimiento de un modelo basado en RoBERTa. En cambio, en los modelos basados en XGBoost hay mucho margen de mejora, por ese motivo las técnicas de aumentado de datos pueden ser más efectivas para ese tipo de modelos.

Hay que puntualizar el bajo rendimiento obtenido en los corpus proporcionados por la empresa: RAIC y RAIT. Esto es debido a que estos dos conjuntos de datos son los que están menos depurados, ya que tienen ejemplos que pertenecen a la misma clase que son muy dispares y eso es causado por haberse realizado un etiquetado a mano sin una excesiva revisión, donde posiblemente hayan bastantes ejemplos etiquetado incorrectamente.

Seguramente, con una selección de ejemplos más restrictiva similar a la llevada a cabo en el corpus Question-Topic el rendimiento obtenido por el clasificador hubiera sido mucho mejor, pero dedicar tiempo al mejor preparamiento del corpus tiene un coste. Pero, para que el rendimiento de un modelo de aprendizaje sea alto es fundamental que la calidad de los ejemplos que se le proporcionan durante el entrenamiento también lo sea.

Corpus	Acc.		F1		Precision		Recall	
	X	R	X	R	X	R	X	R
SST-2	77.3	93.3	0.772	0.933	0.774	0.933	0.773	0.933
SNIPS	91.7	98.9	0.917	0.989	0.920	0.989	0.917	0.989
TREC	71.0	97.6	0.686	0.980	0.732	0.982	0.672	0.980
RAIC	71.4	83.3	0.603	0.748	0.647	0.765	0.583	0.743
RAIT	65.8	79.7	0.592	0.738	0.656	0.745	0.561	0.745
Question-Topic	90.4	99.4	0.909	0.994	0.913	0.994	0.906	0.994

Tabla 5.1: Resultados obtenidos para cada corpus en el escenario completo sin utilizar técnicas de aumentado de datos.

En la tabla 5.2 se muestran los resultados[1] obtenidos aplicando las diferentes configuraciones de las tres técnicas de aumentado de datos (Conf. TAD) a todo el conjunto de entrenamiento. Los modelos mostrados en la tabla son las configuraciones que mayor valor de la métrica F1 obtienen dado una técnica de aumentado de datos y un corpus. Al analizar los resultados obtenidos se ve rápidamente como la mejora que aportan estas técnicas utilizando todo el conjunto de datos de entrenamiento es prácticamente inexistente en el caso del modelo basado en RoBERTa, solamente en el corpus RAIT obtiene un incremento superior al 0.01 en F1. En el caso del modelo basado en XGBoost la mejora es más común para la mayoría de corpus, no obstante sigue siendo baja para los mejores casos, la cual se encuentra alrededor del 0.03 de incremento en la métrica F1. Por lo tanto, el aumentado de datos cuando se dispone de un conjunto de entrenamiento grande no resulta ser muy útil para mejorar el rendimiento del clasificador.

Adicionalmente, en la sección A.1 se muestran los resultados de todos los experimentos realizados para el escenario que utiliza todo el conjunto de datos de entrenamiento.

5.2 Escenario con escasos recursos de entrenamiento

En esta segunda sección se muestran los resultados obtenidos del mismo modo que en la sección anterior (ver sección 5.1), pero esta vez simulando los escenarios propuestos en la tabla 4.2 donde se disponen de "escasos" ejemplos de entrenamiento.

En la tabla 5.3 se muestran los resultados obtenidos en las diferentes métricas, tanto para el modelo basado en XGBoost como para el modelo basado en RoBERTa para los tres escenarios propuestos en cada uno de los corpus. Como es obvio, al reducir el número de ejemplos de entrenamiento el rendimiento se degrada (ver tabla 5.1), no obstante los modelos basados en RoBERTa mantienen un rendimiento muy bueno teniendo en cuenta las pocas muestras de entrenamiento que emplean. En cambio, para XGBoost no sucede lo mismo, el rendimiento empeora en mayor medida. Por lo tanto, podemos ver que el modelo basado en RoBERTa sigue obteniendo en la mayoría de los casos un rendimiento muy competitivo, aunque el número de ejemplos de entrenamiento no sea lo suficientemente grande. Eso es debido, a que parte de un conocimiento previo muy

[1] Las configuraciones de aumentado de datos que tengan el valor de la métrica F1 en negrita significa que mejoran el rendimiento del respectivo modelo que no aplica dichas técnicas, en caso, de que para un mismo modelo haya más de una configuración que la mejore, la que contenga el asterisco * será la mejor de todas.

Corpus	Conf. TAD		Acc.		F1		Precision		Recall	
	X	R	X	R	X	R	X	R	X	R
SST-2	-	-	77.3	93.3	0.772	0.933	0.774	0.933	0.773	0.933
	EDA-1	EDA-0	78.4	93.8	**0.783**	0.938	0.786	0.939	0.784	0.939
	BT-fr	BT-cs	79.4	93.5	***0.794**	0.935	0.795	0.935	0.794	0.935
	SUS-10	SUS-30	78.7	92.8	**0.787**	0.928	0.788	0.928	0.787	0.928
SNIPS	-	-	91.7	98.9	0.917	0.989	0.920	0.989	0.917	0.989
	EDA-0	EDA-1	92.7	99.0	**0.927**	0.990	0.931	0.990	0.927	0.990
	BT-ca	BT-cs	91.7	99.0	0.916	0.990	0.918	0.990	0.917	0.990
	SUS-10	SUS-10	92.3	99.3	0.922	0.993	0.927	0.993	0.923	0.993
TREC	-	-	71.0	97.6	0.686	0.980	0.732	0.982	0.672	0.980
	EDA-0	EDA-2	71.4	96.6	0.689	0.951	0.718	0.952	0.680	0.952
	BT-ru	BT-ru	69.2	96.2	0.675	0.958	0.720	0.969	0.663	0.949
	SUS-20	SUS-10	71.4	97.0	0.689	0.973	0.715	0.977	0.680	0.971
RAIC	-	-	71.4	83.3	0.603	0.748	0.647	0.765	0.583	0.743
	EDA-2	EDA-1	72.4	82.7	**0.621**	0.724	0.680	0.747	0.595	0.720
	BT-ru	BT-ru	73.8	83.1	***0.631**	0.746	0.683	0.752	0.614	0.749
	SUS-20	SUS-10	72.8	81.9	**0.615**	0.711	0.665	0.736	0.604	0.706
RAIT	-	-	65.8	79.7	0.592	0.738	0.656	0.745	0.561	0.745
	EDA-1	EDA-0	68.4	78.1	**0.618**	0.730	0.680	0.727	0.588	0.742
	BT-cs	BT-cs	68.7	80.5	**0.633**	**0.750**	0.678	0.752	0.609	0.758
	SUS-30	SUS-10	70.2	78.7	***0.659**	0.724	0.683	0.726	0.646	0.728
Question-Topic	-	-	90.4	99.4	0.909	0.994	0.913	0.994	0.906	0.994
	EDA-0	EDA-1	90.9	98.7	0.915	0.986	0.915	0.986	0.915	0.986
	BT-et	BT-cs	88.6	98.8	0.894	0.986	0.903	0.986	0.890	0.987
	SUS-10	SUS-10	93.5	99.1	**0.939**	0.991	0.941	0.992	0.938	0.989

Tabla 5.2: Mejores resultados obtenidos para cada corpus aplicando cada técnica de aumentado de datos en el escenario que se emplea el conjunto de entrenamiento completo.

alto, lo que le permite que con unos pequeños ajustes relacione conceptos de una nueva tarea fácilmente.

En la tabla 5.4 se muestran los resultados obtenidos para el escenario de escasos recursos con mayor número de ejemplos de entrenamiento por clase (NEPC) que es 80. Para el modelos basados en RoBERTa se ve como estas técnicas no ayudan a mejorar su rendimiento sino todo lo contrario. La única técnica que suele funcionar es *back-translation* y el incremento de media que obtiene en la métrica F1 es menor al 0.015. En cambio, para el modelo basado en XGBoost todas las aproximaciones suelen funcionar. En la técnica EDA obtiene un incremento de media en la métrica F1 de 0.024, en *back-translation* obtiene un incremento de media del 0.029 y en la técnica basada en la sustitución de palabras obtiene el incremento de media más alto que es de 0.044.

En la tabla 5.5 se muestran los resultados obtenidos para el escenario de escaso recursos que emplea 50 ejemplos de entrenamiento por clase. Igual que sucedía en el escenario previo, para el modelo basado en RoBERTa la técnica más efectiva es *back-translation* que obtiene un incremento de media en F1 del 0.0123. Para XGBoost ocurre igual que antes, donde las tres técnicas suelen obtener mejoras en cada corpus. La mejor de todas sigue siendo la de sustitución que alcanza de media en F1 un incremento del 0.0423. En *back-translation* obtiene un incremento cercano al 0.034 en F1. La técnica EDA es la que peor funciona de las tres, la cual obtiene un incremento de media del 0.023 en la métrica F1.

En la tabla 5.6 se muestra los resultados obtenidos para el escenario más reducido de todos donde solamente se emplean 20 ejemplos de entrenamiento por clase. Los resultados obtenidos en este escenario son muy similares a los obtenidos en los dos anteriores. Para el modelo basado en RoBERTa, *back-translation* es la única técnica que predomina

Corpus	NEPC	Acc.		F1		Precision		Recall	
		X	R	X	R	X	R	X	R
SST-2	80	67.5	84.9	0.675	0.849	0.675	0.849	0.675	0.849
	50	64.4	83.6	0.643	0.836	0.646	0.839	0.644	0.836
	20	59.0	80.8	0.590	0.807	0.591	0.813	0.590	0.808
SNIPS	80	82.6	96.6	0.823	0.966	0.833	0.968	0.826	0.966
	50	78.4	97.4	0.785	0.974	0.794	0.975	0.784	0.974
	20	70.7	93.0	0.704	0.930	0.712	0.932	0.707	0.930
TREC	80	47.4	90.8	0.447	0.888	0.450	0.865	0.560	0.931
	50	41.2	86.4	0.392	0.841	0.404	0.820	0.520	0.889
	20	40.0	79.0	0.382	0.741	0.405	0.742	0.480	0.842
RAIC	80	56.0	73.2	0.500	0.664	0.504	0.652	0.529	0.697
	50	53.7	70.6	0.482	0.634	0.485	0.618	0.509	0.669
	20	42.5	65.2	0.362	0.572	0.362	0.575	0.383	0.597
RAIT	80	54.8	73.8	0.525	0.696	0.511	0.666	0.578	0.752
	50	51.7	69.5	48.9	0.653	0.477	0.632	0.547	0.704
	20	42.2	65.1	0.396	0.611	0.384	0.584	0.454	0.677
Question-Topic	80	76.9	98.1	0.783	0.981	0.784	0.979	0.786	0.983
	50	73.6	97.9	0.751	0.979	0.753	0.978	0.752	0.980
	20	60.8	96.6	0.623	0.965	0.627	0.964	0.621	0.967

Tabla 5.3: Resultados obtenidos para cada corpus en los escenarios reducidos sin utilizar técnicas de aumentado de datos.

Corpus	Conf. TAD		Acc.		F1		Precision		Recall	
	X	R	X	R	X	R	X	R	X	R
SST-2	-	-	67.5	84.9	0.675	0.849	0.675	0.849	0.675	0.849
	EDA-A	EDA-A	69.4	84.5	**0.693**	0.845	0.696	0.848	0.694	0.845
	BT-et	BT-ru	70.8	87.5	**0.708**	**0.875**	0.708	0.875	0.708	0.875
	SUS-30	SUS-30	71.8	84.7	***0.718**	0.847	0.718	0.848	0.718	0.847
SNIPS	-	-	82.6	96.6	0.823	0.966	0.833	0.968	0.826	0.966
	EDA-A	EDA-A	83.4	97.6	**0.832**	***0.976**	0.846	0.977	0.834	0.976
	BT-ca	BT-nl	83.3	97.4	**0.831**	**0.974**	0.840	0.975	0.833	0.974
	SUS-20	SUS-30	85.0	96.7	***0.849**	**0.967**	0.860	0.968	0.850	0.967
TREC	-	-	47.4	90.8	0.447	0.888	0.450	0.865	0.560	0.931
	EDA-A	EDA-C	51.2	84.8	**0.481**	0.858	0.489	0.850	0.569	0.874
	BT-ca	BT-ca	48.4	84.6	**0.469**	0.795	0.470	0.787	0.581	0.885
	SUS-10	SUS-20	52.4	80.4	***0.499**	0.756	0.513	0.747	0.612	0.851
RAIC	-	-	56.0	73.2	0.500	0.664	0.504	0.652	0.529	0.697
	EDA-A	EDA-C	58.9	66.0	**0.530**	0.615	0.526	0.610	0.566	0.660
	BT-ru	BT-et	59.3	68.6	***0.538**	0.640	0.534	0.635	0.575	0.685
	SUS-30	SUS-20	59.4	66.0	**0.534**	0.603	0.527	0.590	0.573	0.647
RAIT	-	-	54.8	73.8	0.525	0.696	0.511	0.666	0.578	0.752
	EDA-A	EDA-C	58.7	68.8	**0.554**	0.656	0.531	0.627	0.619	0.720
	BT-et	BT-fr	59.0	73.6	**0.570**	**0.706**	0.546	0.681	0.631	0.752
	SUS-20	SUS-10	60.9	68.2	***0.580**	0.642	0.550	0.613	0.647	0.701
Question-Topic	-	-	76.9	98.1	0.783	0.981	0.784	0.979	0.786	0.983
	EDA-B	EDA-B	59.4	94.8	0.605	0.948	0.643	0.947	0.627	0.949
	BT-ca	BT-et	81.2	98.5	**0.828**	***0.985**	0.834	0.985	0.829	0.985
	SUS-30	SUS-10	82.5	98.1	***0.838**	**0.982**	0.839	0.982	0.839	0.982

Tabla 5.4: Mejores resultados obtenidos para cada corpus con cada técnica de aumentado de datos en el escenario NEPC=80.

Corpus	Conf. TAD		Acc.		F1		Precision		Recall	
	X	R	X	R	X	R	X	R	X	R
SST-2	-	-	64.4	83.6	0.643	0.836	0.646	0.839	0.644	0.836
	EDA-B	EDA-A	64.7	81.8	**0.644**	0.816	0.652	0.827	0.647	0.818
	BT-bg	BT-fr	68.4	85.5	**0.683**	**0.855**	0.684	0.856	0.684	0.855
	SUS-30	SUS-10	70.4	81.4	***0.704**	0.814	0.705	0.816	0.704	0.814
SNIPS	-	-	78.4	97.4	0.785	0.974	0.794	0.975	0.784	0.974
	EDA-A	EDA-B	80.9	97.0	**0.807**	0.970	0.817	0.971	0.809	0.970
	BT-bg	BT-ca	81.9	96.9	***0.817**	0.969	0.824	0.970	0.819	0.969
	SUS-30	SUS-10	80.7	95.6	**0.805**	0.956	0.820	0.958	0.807	0.956
TREC	-	-	41.2	86.4	0.392	0.841	0.404	0.820	0.520	0.889
	EDA-B	EDA-C	46.2	77.8	**0.435**	0.739	0.443	0.734	0.537	0.821
	BT-nl	BT-bg	44.4	81.4	**0.427**	0.796	0.427	0.784	0.542	0.842
	SUS-10	SUS-10	47.4	77.8	***0.448**	0.731	0.459	0.727	0.553	0.820
RAIC	-	-	53.7	70.6	0.482	0.634	0.485	0.618	0.509	0.669
	EDA-A	EDA-B	55.5	67.6	**0.500**	0.606	0.499	0.602	0.540	0.638
	BT-bg	BT-it	55.7	68.3	**0.505**	0.620	0.508	0.620	0.539	0.646
	SUS-30	SUS-20	57.3	63.8	***0.512**	0.571	0.510	0.567	0.555	0.603
RAIT	-	-	51.7	69.5	0.489	0.653	0.477	0.632	0.547	0.704
	EDA-B	EDA-C	52.8	67.9	**0.504**	0.647	0.487	0.623	0.557	0.714
	BT-bg	BT-ca	54.3	69.9	***0.531**	**0.669**	0.509	0.642	0.588	0.726
	SUS-10	SUS-10	54.4	65.4	**0.511**	0.618	0.489	0.593	0.571	0.694
Question-Topic	-	-	73.6	97.9	0.751	0.979	0.753	0.978	0.752	0.980
	EDA-C	EDA-C	41.1	78.8	0.377	0.757	0.392	0.740	0.449	0.800
	BT-cs	BT-de	77.4	98.0	**0.790**	**0.981**	0.791	0.981	0.792	0.981
	SUS-20	SUS-20	80.1	98.0	***0.816**	0.981	0.816	0.982	0.817	0.981

Tabla 5.5: Mejores resultados obtenidos para cada corpus con cada técnica de aumentado de datos en el escenario NEPC=50.

obtener alguna mejora en varios corpus donde el incremento de media en la métrica F1 es de 0.0165. En el modelo basado en XGBoost para el método de sustitución obtiene de media el incremento más alto conseguido hasta el momento que es de 0.0493 en F1. *Back-translation* también obtiene un incremento de media más alto que el conseguido en los dos escenarios previos que es de 0.037 en F1. Finalmente, en EDA la mejora obtenida es inferior al 0.02 en F1.

Después de analizar los resultados obtenidos se ve claramente que las técnicas de aumentados de datos que hemos utilizado no proporcionan una mejora muy importante en el rendimiento de los clasificadores de texto. En el caso del modelo basados en RoBERTa, la mejora solamente es obtenida por la técnica *back-translation* (ver sección 3.2), la cual incrementa ligeramente en 0.01 la métrica F1, por lo tanto, este incremento es insignificante. *Back-translation* es la única técnica que puede funcionar para RoBERTa, ya que puede generar ejemplos con un significado semántico muy similar al original, pero con una variaciones en la estructura de la frase que no se aprenden en el MLM. EDA no funciona para RoBERTa, ya que los ejemplos que genera son puramente ruido. Con el modelo basado en XGBoost, las tres técnicas suelen funcionar y al aplicarlas obtienen mayor mejora que en RoBERTa, pero la mejora sigue siendo muy baja. La técnica que mejores resultados obtiene en XGBoost es la que consiste en aumentar los ejemplos en base a realizar sustituciones de palabras utilizando modelos pre-entrenados (ver sección 3.3). Esta técnica es la más costosa de las tres, pero es la que genera ejemplos de mayor calidad, por esa razón funciona tan bien en XGBoost. En RoBERTa no funciona, ya que esta técnica está basada en el propio modelo, por lo tanto, no le puede aportar nada que no sepa. Aún así, el rendimiento que obtiene XGBoost al aplicar esta técnica sigue siendo bajo si la comparamos directamente con el rendimiento obtenido por RoBERTa sin em-

Corpus	Conf. TAD		Acc.		F1		Precision		Recall	
	X	R	X	R	X	R	X	R	X	R
SST-2	-	-	59.0	80.8	0.590	0.807	0.591	0.813	0.590	0.808
	EDA-A	EDA-A	60.5	83.0	**0.605**	**0.830**	0.605	0.830	0.605	0.830
	BT-it	BT-ru	60.8	80.6	**0.608**	0.805	0.608	0.810	0.608	0.806
	SUS-30	SUS-10	63.4	78.3	***0.633**	0.782	0.635	0.783	0.634	0.783
SNIPS	-	-	70.7	93.0	0.704	0.930	0.712	0.932	0.707	0.930
	EDA-A	EDA-A	72.6	94.3	**0.723**	**0.943**	0.729	0.944	0.726	0.943
	BT-bg	BT-cs	74.7	95.0	***0.744**	**0.950**	0.751	0.953	0.747	0.950
	SUS-30	SUS-20	73.7	95.4	**0.732**	***0.954**	0.755	0.956	0.737	0.954
TREC	-	-	40.0	79.0	0.382	0.741	0.405	0.742	0.480	0.842
	EDA-B	EDA-C	42.8	69.8	***0.402**	0.657	0.414	0.662	0.493	0.756
	BT-et	BT-et	37.0	78.4	0.362	**0.757**	0.386	0.742	0.466	0.820
	SUS-30	SUS-20	41.8	73.0	**0.398**	0.692	0.427	0.679	0.488	0.760
RAIC	-	-	42.5	65.2	0.362	0.572	0.362	0.575	0.383	0.597
	EDA-A	EDA-C	46.1	60.5	**0.395**	0.533	0.394	0.542	0.416	0.565
	BT-bg	BT-ca	46.5	63.4	**0.412**	0.564	0.411	0.569	0.458	0.586
	SUS-20	SUS-20	50.7	56.4	***0.444**	0.496	0.441	0.500	0.483	0.538
RAIT	-	-	42.2	65.1	0.396	0.611	0.384	0.584	0.454	0.677
	EDA-A	EDA-C	42.6	62.4	0.395	0.583	0.388	0.568	0.460	0.651
	BT-ru	BT-nl	43.6	64.5	**0.424**	0.611	0.431	0.587	0.488	0.671
	SUS-10	SUS-10	46.8	55.5	***0.437**	0.517	0.427	0.506	0.497	0.607
Question-Topic	-	-	60.8	96.6	0.623	0.965	0.627	0.964	0.621	0.967
	EDA-B	EDA-C	58.2	96.8	0.594	**0.968**	0.596	0.966	0.610	0.969
	BT-bg	BT-fr	67.7	97.1	**0.683**	***0.972**	0.680	0.973	0.692	0.970
	SUS-30	SUS-20	69.6	96.5	***0.709**	**0.967**	0.706	0.966	0.714	0.967

Tabla 5.6: Mejores resultados obtenidos para cada corpus con cada técnica de aumentado de datos en el escenario NEPC=20.

plearlas. Otro aspecto a destacar, es que en XGBoost cuando el escenario es más reducido, las técnicas de *back-translation* y la basada en sustitución ligeramente son más efectivas.

Adicionalmente, en la sección A.2 se muestran los resultados de todos los experimentos realizados aplicando las diferentes técnicas de aumentado de datos para los tres escenarios reducidos

5.3 Tests de significancia estadística

Previamente, habíamos comentado para cada uno de los experimentos realizados (ver secciones 5.1 y 5.2) que técnica de aumentado de datos predomina funcionar mejor en base a la métrica F1. Pero, para poder formular una conclusión con mayor firmeza se requiere de cierto rigor científico. Por ese motivo en esta sección vamos a realizar tests de significancia estadística [29]. De esta manera sabremos si el rendimiento de los modelos es significativamente diferente.

Para realizar un test de significancia estadística se necesitan de los resultados obtenidos por los dos modelos a comparar para un métrica concreta, en nuestro caso la F1. Para cada modelo se utilizarán los resultados obtenidos en los seis corpus, por lo tanto, se compararán el modelo que emplea una de las tres técnicas de aumentado de datos propuestas en este trabajo (ver capitulo 3) con respecto el correspondiente modelo que no aplica ninguna de esas técnicas.

En la tabla 5.7 se muestran los resultados de todos los tests de significancia estadística realizados en base al valor p. Cuando el valor de p es muy pequeño, en nuestro caso

inferior a 0.05 quiere decir que el rendimiento de los dos modelos comparados es significativamente diferente, cuando el valor es cercano a 1.00, quiere decir que el rendimiento de los dos modelos es prácticamente el mismo.

En el caso del clasificador basado en XGBoost (X) hay bastantes modelos entrenados que obtienen una diferencia significativa en el rendimiento[2] para los diferentes escenarios teniendo en cuenta el número de ejemplos de entrenamiento por clase (NEPC). Para todos esos modelos que utilizan técnicas de aumentado de datos (TAD) su rendimiento es mejor que el correspondiente modelo que no las aplica. En este análisis también se ve que la técnica que predomina funcionar mejor en los diferentes escenario para XGBoost también es SUS (ver sección 3.3), como habíamos comentado en el análisis previo (ver sección 5.2).

En el caso del clasificador basado en el modelo RoBERTa (R), solamente para el modelo que aplica EDA en el escenario con 50 NEPC[3] obtiene una diferencia significativa en el rendimiento. Pero en este caso, si revisamos la tabla 5.5, el modelo que aplica la técnica de aumentado de datos funciona peor que el modelo que no lo aplica. Por lo tanto, para el clasificador basado en RoBERTa no hay ninguna técnica que permita obtener un rendimiento significativo mejor. Previamente, intuíamos que en los escenario con un reducido NEPC, la técnica de aumentado de datos basada en *back-translation* (BT) (ver sección 3.2) si que podía ser efectiva (ver sección 5.2), en este análisis se confirma que no.

NEPC	TAD	X	R
Todos	EDA	**0.028**	0.127
	BT	0.313	0.688
	SUS	**0.032**	0.097
80	EDA	1.000	0.092
	BT	**0.032**	0.753
	SUS	**0.031**	0.164
50	EDA	1.000	*0.031
	BT	**0.033**	0.744
	SUS	**0.029**	0.062
20	EDA	0.320	0.319
	BT	0.097	0.369
	SUS	**0.033**	0.159

Tabla 5.7: Resultados del valor *p* obtenidos en los tests de significancia estadística.

[2]Los modelos que en la columna X de la tabla 5.7 esté en negrita el valor *p*.
[3]El valor *p* del modelo está señalado con un asterisco en la columna R de la tabla 5.7.

CAPÍTULO 6
Conclusiones y trabajos futuros

En este trabajo se ha realizado una experimentación exhaustiva, donde se han explorado varias configuraciones para las distintas técnicas de aumentado de datos propuestas en los diferentes corpus seleccionados. Adicionalmente, también se ha experimentado en escenarios con recursos de entrenamiento reducidos.

Todos los experimentos han sido llevados a cabo utilizando tanto un modelo de aprendizaje automático clásico como es XGBoost, como de un modelo de aprendizaje automático profundo como es RoBERTa que actualmente es el estado del arte en clasificación de texto. Al evaluar todos los experimentos realizados vemos que las técnicas de aumentado de datos son más efectivas en XGBoost que en RoBERTa, ya que en RoBERTa las mejoras no son estadísticamente significativas, no obstante el incremento que se obtiene al aplicar estas técnicas en XGBoost es pequeña.

RoBERTa al ser un modelo pre-entrenado parte de mucho conocimiento a priori, y eso le permite que con leves ajuste de sus pesos pueda obtener un rendimiento muy elevado, incluso en los escenarios con pocos recursos, donde sin requerir de ninguna técnica de aumentado de datos obtiene un rendimiento bastante bueno teniendo en cuenta el tamaño de muestras de entrenamiento que emplea.

Basándonos en los resultados obtenidos, tanto en los conjuntos de datos relacionados puramente con la clasificación de texto como los conjuntos de datos relacionados con la detección de *intents*, el modelo de aprendizaje automático que recomendaríamos utilizar sin ninguna duda sería un modelo pre-entrenado basado en *Transformers* como puede ser BERT, RoBERTa, XLNet [43], ALBERT [19] o ELECTRA [5]. En nuestro caso, la versión pre-entrenada de RoBERTa ha sido lo suficientemente robusta para obtener un rendimiento altamente preciso sin la utilización de ninguna aproximación de aumentado de datos en el conjunto de entrenamiento. En definitiva, no recomendamos el uso de las técnicas de aumentado de datos utilizadas en este trabajo, ya que no aporta una mejora significativa desde el punto de vista estadístico a este tipo de modelos.

Debido al poco éxito que han tenido las tres técnicas de aumentado de datos que hemos utilizado en este proyecto. Sería recomendable en un próximo trabajo refinar un generador de texto, como puede ser la versión pre-entrenada de GPT-2, para que dada una clase o el principio de un ejemplo (contexto) sea capaz de generar nuevos ejemplos de entrenamiento. Con el fin de usar esos ejemplos aumentados junto a los originales para entrenar el modelo de clasificación texto basado en RoBERTa. Esta técnica podría ser muy interesante cuando el número de ejemplos de entrenamiento es reducido, así RoBERTa podría obtener una mejora notable en su rendimiento. Hay dos artículo de investigación comentados en la sección del estado del arte del aumentado de datos en PLN (ver sección 1.3) [1] [18] que emplean aproximaciones de este tipo obteniendo resultados interesantes.

Bibliografía

[1] Ateret Anaby-Tavor, Boaz Carmeli, Esther Goldbraich, Amir Kantor, George Kour, Segev Shlomov, Naama Tepper, and Naama Zwerdling. Do not have enough data? deep learning to the rescue! *arXiv:1911.03118*, 2019.

[2] Stephan Bloehdorn and Andreas Hotho. Boosting for text classification with semantic features. In *International workshop on knowledge discovery on the web*, pages 149–166. Springer, 2004.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[4] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[5] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv:2003.10555*, 2020.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.

[7] Adji B Dieng, Chong Wang, Jianfeng Gao, and John Paisley. Topicrnn: A recurrent neural network with long-range semantic dependency. *arXiv:1611.01702*, 2016.

[8] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[9] Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[10] Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 753–757, 2018.

[11] Hongyu Guo, Yongyi Mao, and Richong Zhang. Augmenting data with mixup for sentence classification: An empirical study. *arXiv:1905.08941*, 2019.

[12] Yannis Haralambous and Philippe Lenca. Text classification using association rules, dependency pruning and hyperonymization. *arXiv:1407.7357*, 2014.

[13] Homa B Hashemi, Amir Asiaee, and Reiner Kraft. Query intent detection using convolutional neural networks. In *International Conference on Web Search and Data Mining, Workshop on Query Understanding*, 2016.

[14] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv:1801.06146*, 2018.

[15] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv:1909.10351*, 2019.

[16] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv:1408.5882*, 2014.

[17] Sosuke Kobayashi. Contextual augmentation: Data augmentation by words with paradigmatic relations. *arXiv:1805.06201*, 2018.

[18] Varun Kumar, Ashutosh Choudhary, and Eunah Cho. Data augmentation using pre-trained transformer models. *arXiv:2003.02245*, 2020.

[19] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv:1909.11942*, 2019.

[20] Xin Li and Dan Roth. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002.

[21] Bing Liu and Ian Lane. Attention-based recurrent neural network models for joint intent detection and slot filling. *arXiv:1609.01454*, 2016.

[22] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv:1907.11692*, 2019.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv:1301.3781*, 2013.

[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[25] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[26] Zhaowei Qu, Xiaomin Song, Shuqiang Zheng, Xiaoru Wang, Xiaohui Song, and Zuquan Li. Improved bayes method based on tf-idf feature and grade factor feature for chinese information classification. In *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 677–680. IEEE, 2018.

[27] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[28] Suman Ravuri and Andreas Stolcke. Recurrent neural network and lstm models for lexical utterance classification. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[29] Stefan Riezler and John T Maxwell III. On some pitfalls in automatic evaluation and significance testing for mt. In *Proceedings of the ACL workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 57–64, 2005.

[30] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[31] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. *arXiv:1511.06709*, 2015.

[32] Imran Sheikh, Irina Illina, Dominique Fohr, and Georges Linares. Learning word importance with the neural bag-of-words model. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 222–229, 2016.

[33] Yangyang Shi, Kaisheng Yao, Le Tian, and Daxin Jiang. Deep lstm based feature mapping for query classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: Human language technologies*, pages 1501–1511, 2016.

[34] Yusuxke Shibata, Takuya Kida, Shuichi Fukamachi, Masayuki Takeda, Ayumi Shinohara, Takeshi Shinohara, and Setsuo Arikawa. Byte pair encoding: A text compression scheme that accelerates pattern matching. Technical report, Technical Report DOI-TR-161, Department of Informatics, Kyushu University, 1999.

[35] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[36] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[38] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, pages 3266–3280, 2019.

[39] Jason W Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv:1901.11196*, 2019.

[40] Xing Wu, Shangwen Lv, Liangjun Zang, Jizhong Han, and Songlin Hu. Conditional bert contextual augmentation. In *International Conference on Computational Science*, pages 84–95, 2019.

[41] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Unsupervised data augmentation for consistency training. *arXiv:1904.12848*, 2019.

[42] Puyang Xu and Ruhi Sarikaya. Convolutional neural network based triangular crf for joint intent detection and slot filling. In *2013 ieee workshop on automatic speech recognition and understanding*, pages 78–83. IEEE, 2013.

[43] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763, 2019.

[44] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[45] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. Text classification based on multi-word with support vector machine. *Knowledge-Based Systems*, 21(8):879–886, 2008.

APÉNDICE A
Resultados adicionales

En este apéndice se muestran los resultados obtenidos en todos los experimentos realizados en este trabajo. Ya sea, en el escenario que utiliza todo el conjunto de datos de entrenamiento (ver sección A.1) y también en aquellos escenarios donde se emplea un número reducido de ejemplos de entrenamiento por cada una de las clases que constituyen cada corpus (ver sección A.2). Para simplificar la lectura de las tablas mostradas en las dos secciones posteriores, simplemente se mostrarán los valores obtenidos en la métrica F1. No obstante, en el capítulo 5 se muestran para los mejores modelos el valor obtenido en las diferentes métricas más utilizadas para el problema de la clasificación.

A.1 Escenario completo

En la tabla A.1 se muestra el valor de la métrica F1 obtenido por los modelos que han aplicado las distintas configuraciones (Conf. TAD) de la técnica de aumentado de datos EDA (ver subsección 4.3.1) en el conjunto de datos de entrenamiento completo. Los valores que están en negrita corresponden a la configuración de aumentado de datos que ha sido más efectiva. En algunos experimentos no se ha conseguido mejorar el modelo que no aplica ningún aumentado de datos en su conjunto de entrenamiento.

Conf. TAD	SST-2		SNIPS		TREC		RAIC		RAIT		Question-Topic	
	X	R	X	R	X	R	X	R	X	R	X	R
-	0.772	0.933	0.917	0.989	0.686	0.980	0.603	0.748	0.592	0.738	0.909	0.994
EDA-A	0.783	**0.938**	**0.927**	0.986	0.689	0.951	0.615	0.716	0.605	0.730	**0.915**	0.990
EDA-B	**0.783**	0.927	0.914	**0.990**	**0.676**	0.931	0.616	0.724	**0.618**	0.724	0.907	0.986
EDA-C	0.781	0.933	0.914	0.989	0.675	0.951	**0.621**	0.702	0.597	0.715	0.903	0.983

Tabla A.1: Puntuación F1 obtenida en el escenario completo utilizando las diferentes configuraciones de la técnica de aumentado de datos EDA para cada corpus. Donde X es el modelo basado en XGBoost y R es el modelo basado RoBERTa.

En la tabla A.2 se muestra el valor de la métrica F1 obtenido para las distintas configuraciones de la técnica de aumentado de datos *back-translation* (ver subsección 4.3.2). Al igual que en la tabla anterior los valores que están en negrita corresponden a la mejor configuración que ha conseguido mejorar a su correspondiente modelo de partida.

Finalmente, en la tabla A.3 se muestra el valor de la métrica F1 obtenido para las distintas configuraciones de la técnica de aumentado de datos basada en la sustitución de palabras utilizando modelos pre-entrenados (ver subsección 4.3.3). Al igual que en las dos tablas anteriores los valores que están en negrita corresponden a la configuración que ha sido más efectiva.

Conf. TAD	SST-2		SNIPS		TREC		RAIC		RAIT		Question-Topic	
	X	R	X	R	X	R	X	R	X	R	X	R
-	0.772	0.933	0.917	0.989	0.686	0.980	0.603	0.748	0.592	0.738	0.909	0.994
BT-bg	0.769	0.926	0.907	0.987	0.672	0.950	0.621	0.719	0.629	0.742	0.888	0.977
BT-ca	0.781	0.931	0.916	**0.991**	0.665	0.955	0.616	0.726	0.620	0.736	0.884	0.985
BT-cs	0.777	**0.935**	0.907	0.990	0.659	0.953	0.620	0.722	**0.633**	**0.750**	0.889	0.986
BT-et	0.770	0.931	0.915	0.989	0.658	0.941	0.627	0.741	0.631	0.735	0.894	0.984
BT-fr+ca+ru+cs+et+bg	0.769	0.926	0.907	0.987	0.672	0.950	0.621	0.719	0.629	0.742	0.888	0.977
BT-fr	**0.794**	0.926	0.912	0.990	0.662	0.949	0.609	0.716	0.608	0.723	0.877	0.986
BT-ru	0.788	0.934	0.912	0.991	0.675	0.958	**0.631**	0.746	0.622	0.732	0.887	0.979

Tabla A.2: Puntuación F1 obtenida en el escenario completo utilizando las diferentes configuraciones de la técnica de aumentado de datos *back-translation* para cada corpus. Donde X es el modelo basado en XGBoost y R es el modelo basado RoBERTa.

Conf. TAD	SST-2		SNIPS		TREC		RAIC		RAIT		Question-Topic	
	X	R	X	R	X	R	X	R	X	R	X	R
-	0.772	0.933	0.917	0.989	0.686	0.980	0.603	0.748	0.592	0.738	0.909	0.994
SUS-10	**0.787**	0.924	**0.922**	**0.993**	0.684	0.973	0.612	0.711	0.645	0.724	**0.939**	0.991
SUS-20	0.785	0.919	0.912	0.986	**0.689**	0.968	**0.615**	0.694	0.645	0.722	0.939	0.990
SUS-30	0.779	0.928	0.922	0.989	0.687	**0.972**	0.615	0.699	**0.659**	0.723	0.938	0.987

Tabla A.3: Puntuación F1 obtenida en el escenario completo utilizando las diferentes configuraciones de la técnica de aumentado de datos basada en la sustitución de palabras utilizando modelos pre-entrenados para cada corpus. Donde X es el modelo basado en XGBoost y R es el modelo basado RoBERTa.

A.2 Escenario reducido

En la tabla A.4 se muestra el valor de la métrica F1 obtenida para las distintas configuraciones (Conf. TAD) de la técnica de aumentado de datos EDA (ver subsección 4.3.1). Los valores que están en negrita corresponden a la configuración de aumentado de datos que ha sido más efectiva para cada corpus en cada escenario de entrenamiento reducido.

NEPC	Conf. TAD	SST-2		SNIPS		TREC		RAIC		RAIT		Question-Topic	
		X	R	X	R	X	R	X	R	X	R	X	R
80	-	0.675	0.849	0.823	0.966	0.447	0.888	0.500	0.664	0.525	0.696	0.783	0.981
	EDA-A	**0.693**	0.845	**0.832**	**0.976**	**0.481**	0.787	**0.530**	0.597	**0.554**	0.634	0.596	0.948
	EDA-B	0.668	0.839	0.829	0.959	0.474	0.777	0.512	0.607	0.533	0.624	0.605	0.948
	EDA-C	0.680	0.849	0.809	0.973	0.450	0.858	0.518	0.615	0.538	0.656	0.559	0.947
50	-	0.643	0.836	0.785	0.974	0.392	0.841	0.482	0.634	0.489	0.653	0.751	0.979
	EDA-A	0.644	0.816	**0.807**	0.966	0.406	0.729	**0.500**	0.582	0.503	0.609	0.367	0.725
	EDA-B	**0.644**	0.802	0.799	0.970	**0.435**	0.673	0.488	0.606	**0.504**	0.623	0.362	0.750
	EDA-C	0.624	0.809	0.790	0.960	0.405	0.739	0.496	0.594	0.478	0.647	0.377	0.757
20	-	0.590	0.807	0.704	0.930	0.382	0.741	0.362	0.572	0.396	0.611	0.623	0.965
	EDA-A	**0.605**	**0.830**	**0.723**	**0.943**	0.387	0.568	**0.395**	0.506	0.395	0.513	0.597	0.965
	EDA-B	0.591	0.747	0.689	0.941	**0.402**	0.638	0.387	0.530	0.383	0.574	0.594	0.963
	EDA-C	0.593	0.707	0.680	0.923	0.338	0.657	0.382	0.533	0.372	0.583	0.578	**0.968**

Tabla A.4: Puntuación F1 obtenida para los diferentes experimentos realizados en el escenario reducido utilizando la técnica de aumentado de datos EDA. Donde X es el modelo basado en XGBoost y R es el modelo basado RoBERTa.

En la tabla A.5 se muestra el valor de la métrica F1 obtenido para las distintas configuraciones de la técnica de aumentado de datos *back-translation* (ver subsección 4.3.2). Al igual que en la tabla anterior los valores que están en negrita corresponden a la configuración que ha sido más efectiva.

Finalmente, en la tabla A.6 se muestra el valor de la métrica F1 obtenido para las distintas configuraciones de la técnica de aumentado de datos basada en la sustitución de palabras utilizando modelos pre-entrenados (ver subsección 4.3.3). Al igual que en las

NEPC	Conf. TAD	SST-2		SNIPS		TREC		RAIC		RAIT		Question-Topic	
		X	R	X	R	X	R	X	R	X	R	X	R
80	-	0.675	0.849	0.823	0.966	0.447	0.888	0.500	0.664	0.525	0.696	0.783	0.981
	BT-bg	0.702	0.844	0.826	0.963	0.452	0.768	0.535	0.623	0.568	0.675	0.825	0.974
	BT-ca	0.694	0.834	**0.831**	0.970	**0.469**	0.795	0.523	0.612	0.564	0.698	**0.828**	0.982
	BT-ca+fr	0.702	0.844	0.826	0.963	0.452	0.768	0.535	0.623	0.568	0.675	0.825	0.974
	BT-cs+et+nl+id+bg	0.702	0.844	0.826	0.963	0.452	0.768	0.535	0.623	0.568	0.675	0.825	0.974
	BT-cs	0.695	0.838	0.810	0.969	0.438	0.746	0.531	0.600	0.560	0.699	0.817	0.978
	BT-de	0.695	0.857	0.820	0.964	0.439	0.777	0.508	0.590	0.545	0.691	0.809	0.981
	BT-et	**0.708**	0.871	0.817	0.954	0.463	0.763	0.533	0.640	**0.570**	0.675	0.815	**0.985**
	BT-fr	0.687	0.837	0.828	0.967	0.382	0.780	0.515	0.620	0.537	**0.706**	0.794	0.979
	BT-it+fr+de+ca+ru	0.702	0.844	0.826	0.963	0.452	0.768	0.535	0.623	0.568	0.675	0.825	0.974
	BT-it	0.690	0.850	0.824	0.967	0.443	0.728	0.503	0.619	0.534	0.691	0.772	0.982
	BT-nl	0.701	0.831	0.819	**0.974**	0.468	0.775	0.537	0.629	0.568	0.674	0.826	0.967
	BT-ru	0.697	**0.875**	0.827	0.970	0.438	0.769	**0.538**	0.603	0.565	0.695	0.811	0.974
50	-	0.643	0.836	0.785	0.974	0.392	0.841	0.482	0.634	0.489	0.653	0.751	0.979
	BT-bg	**0.683**	0.824	**0.817**	0.951	0.421	0.796	**0.505**	0.584	**0.531**	0.659	0.788	0.968
	BT-ca	0.654	0.817	0.804	0.969	0.404	0.749	0.493	0.584	0.509	**0.669**	0.775	0.980
	BT-ca+fr	0.683	0.824	0.817	0.951	0.421	0.796	0.505	0.584	0.531	0.659	0.788	0.968
	BT-cs+et+nl+id+bg	0.683	0.824	0.817	0.951	0.421	0.796	0.505	0.584	0.531	0.659	0.788	0.968
	BT-cs	0.675	0.840	0.793	0.967	0.401	0.725	0.498	0.570	0.520	0.666	**0.790**	0.980
	BT-de	0.671	0.777	0.786	0.963	0.392	0.719	0.496	0.604	0.500	0.637	0.759	**0.981**
	BT-et	0.680	0.763	0.794	0.957	0.423	0.736	0.498	0.569	0.528	0.668	0.785	0.975
	BT-fr	0.664	**0.855**	0.785	0.958	0.399	0.675	0.482	0.616	0.491	0.659	0.738	0.977
	BT-it+fr+de+ca+ru	0.683	0.824	0.817	0.951	0.421	0.796	0.505	0.584	0.531	0.659	0.788	0.968
	BT-it	0.663	0.806	0.783	0.963	0.375	0.706	0.474	0.620	0.493	0.651	0.750	0.973
	BT-nl	0.668	0.815	0.800	0.963	**0.427**	0.768	0.495	0.553	0.527	0.655	0.771	0.976
	BT-ru	0.665	0.823	0.787	0.953	0.409	0.735	0.504	0.598	0.522	0.680	0.778	0.978
20	-	0.590	0.807	0.704	0.930	0.382	0.741	0.362	0.572	0.396	0.611	0.623	0.965
	BT-bg	0.606	0.788	**0.744**	0.949	0.354	0.616	**0.412**	0.525	0.404	0.582	**0.683**	0.961
	BT-ca	0.591	0.743	0.704	0.914	0.330	0.615	0.391	0.564	0.408	0.600	0.670	0.971
	BT-ca+fr	0.606	0.788	0.744	0.949	0.354	0.616	0.412	0.525	0.404	0.582	0.683	0.961
	BT-cs+et+nl+id+bg	0.606	0.788	0.744	0.949	0.354	0.616	0.412	0.525	0.404	0.582	0.683	0.961
	BT-cs	0.583	0.740	0.729	**0.950**	0.332	0.620	0.412	0.492	0.417	0.568	0.666	0.967
	BT-de	0.600	0.717	0.686	0.901	0.346	0.542	0.381	0.512	0.398	0.561	0.628	0.965
	BT-et	0.594	0.711	0.740	0.936	0.362	**0.757**	0.405	0.503	0.417	0.604	0.659	0.964
	BT-fr	0.596	0.755	0.651	0.930	0.344	0.653	0.384	0.543	0.374	0.590	0.609	**0.972**
	BT-it+fr+de+ca+ru	0.606	0.788	0.744	0.949	0.354	0.616	0.412	0.525	0.404	0.582	0.683	0.961
	BT-it	**0.608**	0.793	0.705	0.925	0.355	0.724	0.378	0.555	0.376	0.604	0.631	0.963
	BT-nl	0.588	0.688	0.738	0.939	0.342	0.687	0.393	0.494	0.409	0.611	0.666	0.966
	BT-ru	0.598	0.805	0.733	0.922	0.340	0.660	0.407	0.505	**0.424**	0.610	0.671	0.967

Tabla A.5: Puntuación F1 obtenida para los diferentes experimentos realizados en el escenario reducido utilizando la técnica de aumentado de datos *back-translation*. Donde X es el modelo basado en XGBoost y R es el modelo basado RoBERTa.

dos tablas anteriores los valores que están en negrita corresponden a la configuración que ha sido más efectiva.

NEPC	Conf. TAD	SST-2		SNIPS		TREC		RAIC		RAIT		Question-Topic	
		X	R	X	R	X	R	X	R	X	R	X	R
80	-	0.675	0.849	0.823	0.966	0.447	0.888	0.500	0.664	0.525	0.696	0.783	0.981
	SUS-10	0.713	0.824	0.835	0.963	**0.499**	0.717	0.529	0.591	0.560	0.642	0.818	**0.982**
	SUS-20	0.711	0.819	**0.849**	0.962	0.484	0.756	0.530	0.603	**0.580**	0.626	0.830	0.970
	SUS-30	**0.718**	0.847	0.830	**0.967**	0.455	0.739	**0.534**	0.583	0.578	0.629	**0.838**	0.978
50	-	0.643	0.836	0.785	0.974	0.392	0.841	0.482	0.634	0.489	0.653	0.751	0.979
	SUS-10	0.666	0.814	0.793	0.956	**0.448**	0.731	0.505	0.560	**0.511**	0.618	0.787	0.981
	SUS-20	0.679	0.787	0.788	0.957	0.438	0.633	0.510	0.571	0.509	0.590	**0.816**	**0.982**
	SUS-30	**0.704**	0.793	**0.805**	0.952	0.429	0.661	**0.512**	0.560	0.518	0.587	0.802	0.972
20	-	0.590	0.807	0.704	0.930	0.382	0.741	0.362	0.572	0.396	0.611	0.623	0.965
	SUS-10	0.613	0.782	0.706	0.915	0.358	0.685	0.421	0.494	**0.437**	0.517	0.700	0.969
	SUS-20	0.625	0.760	0.714	**0.954**	0.384	0.692	**0.444**	0.496	0.413	0.500	0.696	**0.967**
	SUS-30	**0.633**	0.627	**0.732**	0.948	**0.398**	0.584	0.437	0.472	0.429	0.476	**0.705**	0.967

Tabla A.6: Puntuación F1 obtenida para los diferentes experimentos realizados en el escenario reducido utilizando la técnica de aumentado de datos basada en la sustitución de palabras utilizando modelos pre-entrenados. Donde X es el modelo basado en XGBoost y R es el modelo basado RoBERTa.