Universidad de Deusto
University of Deusto

Deusto

E4

Entrega Final

NLP

lianghao.ramon.diaz@opendeusto.es

Adrián Estoquera Núñez

Álvaro Ruiz Bravo

Entrega 1

(Introducción)

Abstract

El objetivo de nuestro trabajo es desarrollar un sistema capaz de extraer información de interés sobre la personalidad de los personajes de novelas. Nuestro *corpus* estará formado por esas mismas novelas y como *ground-truth* usaremos las personalidades de personajes extraídos de la página web personality database y 16 personalities.

Una de las primeras tareas que tenemos que desarrollar es identificar de manera precisa el sujeto que está narrando la historia en cada momento. De esta forma podemos atribuir a ese personaje texto que hable de él o que muestre su comportamiento. A partir de ese subconjunto del texto total seremos capaces de vincular tokens con personalidades. A la hora de hacer este *entity recognition* tenemos pensado probar y comparar distintos métodos. Un primer acercamiento más tradicional es crear una pila de personajes que añade identificadores de personajes a medida que el texto habla de ellos o ellos hablan durante un texto. De esta forma, por ejemplo, si un personaje empieza a hablar o es descrito su identificador entrará en la pila; pero si pasa a hablar de otro personaje ese entrará en la pila por encima del identificador del primero y será expulsado de la pila en cuanto se deje de hablar de él. Otro acercamiento más clásico es utilizar APIs para desarrolladores que cumplen con esta función como SNER, SpaCy o NLTK (https://monkeylearn.com/blog/named-entity-recognition/). Finalmente probaremos un acercamiento de *deep learning* para poder compararlo con el resto también.

Una vez seamos capaces de vincular texto a personajes utilizaremos el *ground truth* para vincular tokens o texto a personalidades. Vamos a definir una personalidad como un punto dentro de un espacio vectorial de 4 ejes. Estos son Extroverted-Introverted, Sensing-iNtuitive, Thinking-Feeling, Judging-Perceiving. Por tanto la personalidad ENTJ (Extroverted, iNtuitive, Thinking, Judging) o ISFP (Introverted, Sensing, Feeling, Judging) sería la de aquellas personas que tengan más fuerza en esas dimensiones. Para hacer este modelo clasificador creemos de forma preliminar que necesitaremos una red neuronal como un autoencoder u otra forma de reducir la dimensionalidad.

Finalmente, este modelo también podría ser utilizado para obtener la personalidad de un usuario a partir de un pequeño texto que introduzca.

Análisis de datasets

Para llevar a cabo este proyecto vamos a crear nuestros propios conjuntos de datos, debido a que los que hemos encontrado, como mencionamos posteriormente, relacionan contenido de usuarios con etiquetas, y esto no es lo que buscamos. Por lo tanto, utilizaremos los siguientes conjuntos de datos para desarrollar nuestro proyecto:

- *Corpus*:
    - Utilizaremos sagas de novelas de fantasía que conozcamos, formadas por un número extenso de libros y con personajes que aparezcan en todos ellos.
    - Hemos pensado en *Harry Potter* (7 libros) y *Mistborn* (3 libros)
- *Ground-Truth*

  Para obtener las etiquetas que nos servirán para entrenar el modelo hemos creado unas tablas a partir de información publicada en internet.
    - Dataset de PDB (Personality DataBase):
        - Personaje: etiqueta que representa a un personaje concreto de un libro

        - Etiqueta: texto que representa una etiqueta del indicador, e.g: ENTJ
    - Dataset de descripciones:
        - Descripción: texto sin procesar que describe un tipo de personalidad del Indicador Myers-Briggs, incluyendo fortalezas y debilidades, adjetivos, etc ...
        - Etiqueta: texto que representa una etiqueta del indicador, e.g: ENTJ

Para la creación de este último dataset usaremos un scraper para extraer la información de la página de https://www.16personalities.com, ya que contiene descripciones en muchos ámbitos de cada tipo de personalidad. El dataset que relaciona los personajes de cada libro con su personalidad lo tendremos que generar realizando labores de detección de entidades en distintos libros, asociando cada entidad con el texto que la envuelve mediante la extracción de información. Para ello haremos uso, de al menos, Named Entity Recognition, Part Of Speech tagging y Coreference Resolution. Finalmente, para la comprobación de la precisión de nuestro trabajo compararemos con las etiquetas extraídas mediante web scraping de PDB.

Estado del arte

Actualmente en el estado del arte hay muchos análisis que relacionan posts y contenido de usuarios con personalidades definidas mediante el Indicador Myers-Briggs (e.g: https://www.kaggle.com/datasets/datasnaek/mbti-type) , pero este tipo de trabajo se corresponde con la "*tarea final*" de nuestro proyecto: asociar el texto relacionado con un personaje con un tipo de personalidad. Debido a esto hemos buscado información sobre las distintas tareas previas que debemos llevar a cabo, y sobre qué métodos están a la orden del día en proyectos similares.

Hemos encontrado el siguiente paper (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7924459/) en el que se detalla todo el proceso de preparación de los datos en un ámbito similar al que trata nuestro proyecto. En este proyecto se evalúa el performance de distintos sistemas de NER (Named Entity Recognition) a la hora de identificar las entidades en novelas modernas y crear un red de interacciones. En concreto, en este paper se plantean una serie de preguntas con respecto a las herramientas de NER disponibles en la actualidad, como cuáles son las que mejor rendimiento ofrecen, y qué diferencias se pueden encontrar entre redes de personajes extraídas de distintas novelas. Para esto comparan y analizan el uso de 4 herramientas diferentes (BookNLP, Illinois Tagger, IXA-Pipe-NERC y Stanford NER tagger) en 20 novelas modernas y realizan distintos experimentos, obteniendo una serie de buenas prácticas para el proceso. Es por esto que basaremos nuestro modo de proceder a la hora de preparar los datos, y al intentar detectar las entidades presentes en nuestro corpus, en sus conclusiones.

También utilizaremos este paper (https://arxiv.org/pdf/1907.02704.pdf) para guiar el desarrollo de nuestro proyecto, ya que es una recopilación de toda la literatura referente a la creación y el análisis de redes de personajes ficticios, fechado en marzo de 2022. En este paper se analiza la forma en que se construyen las relaciones entre personajes en la ficción, que difiere de la forma en la que ocurren en la realidad. Este estudio trata tanto texto como imagen y video. Nosotros nos centraremos en el procesamiento de texto libre. Esta investigación no plantea los mismos objetivos que nosotros, pero se enfrenta a problemas muy similares, pues tanto para construir una red de interacciones como para extraer las personalidades de los personajes de una obra literaria es necesario procesar el texto con los siguientes fines: detectar en qué momento se hace

referencia a personajes y cuáles de estas referencias se refieren al mismo personaje. Dada la naturaleza similar del procesamiento, nos serviremos de este paper para ver cómo se pueden resolver las diferentes casuísticas. En este estudio se plantean 2 categorías de identificación principales, manual y automática. Nosotros solo nos centraremos en las formas (semi)automáticas. Para detectar personajes de manera automática se proponen varias alternativas: Elaboración de una lista predefinida de personajes para buscar emparejamientos exactos en el texto (se enfrenta al problema de nombrar de formas distintas al mismo personaje), emplear NER (named entity recognition) para asignar categorías a los nombres propios y quedarse únicamente con los etiquetados como "persona". En este segundo método, se proponen algunas formas de evitar errores de etiquetado; eliminar aquellos personajes con muy pocas menciones, buscar honoríficos (e.g. Sir. o Miss.) y buscar alrededor de verbos cuyo actor sea una persona. Otros métodos son utilizar Part-of-Speech tagging para identificar los sujetos de las oraciones o buscar relaciones de posesión. Estos métodos son más robustos, pues son capaces de detectar cualquier elemento que se comporte como una "persona". Un ensemble method utilizando el voto combinado de varios de estos métodos simples es útil para la detección de referencias anafóricas. En lo que se refiere a coreference resolution, esta investigación apunta que es una tarea compleja, especialmente en la narrativa de ficción. Algunos métodos empleados son la clusterización de strings similares junto con su género, para alias. Resolver referencias anafóricas es más complicado, y se utiliza una lista de co-ocurrencias nombre-verbo comunes para detectarlas.

Una de las tareas más importantes de nuestro proyecto es la resolución de correferencia (y anáfora). Este proceso trata de resolver las anáforas (oraciones en el que el sujeto es suprimido en pos de la comprensión humana) y vincular oraciones al sujeto del que hablan. Encontramos información sobre este proceso en este paper:https://www.sciencedirect.com/science/article/pii/S1566253519303677. No obstante, tras analizar la literatura que respecta a este tema, hemos encontrado varias tareas más complicadas relacionadas con este proceso como cambios en el sujeto (personajes que cambian de nombre, apellido, que tienen apodos o casos similares), la coexistencia de la realidad descrita por el narrador y la expresada por los personajes, o cambios léxicos (otras variaciones en las palabras producto del estilo del autor para mejorar el ritmo o la comprensión humana de su obra.). Estos problemas y su resolución están descritas en este paper: https://aclanthology.org/W18-4515.pdf. Finalmente no podemos ignorar esta publicación que se centra en solucionar estos problemas precisamente en el ámbito de la literatura fantástica: https://aclanthology.org/2021.crac-1.3.pdf. Esta publicación propone un nuevo *corpus* para el entrenamiento de la resolución de correferencia en el ámbito de la literatura fantástica completamente etiquetado específicamente diseñado para entrenar modelos que lidien con los problemas mencionados anteriormente. Gracias al proceso de anotación en cuatro etapas que se describe se ha logrado un alto puntaje de IAA (Índice de Acuerdo Interanotador) del 87%. Las cuatro etapas de este método son diseño de pautas de anotación, capacitación de anotadores, anotación de datos y evaluación y resolución de conflictos.

Descripción de los datos y carpetas de Drive

Índice de contenidos en Drive:

Aquí tienes un enlace a nuestra carpeta de Drive con todo lo relativo al proyecto: https://drive.google.com/drive/folders/1tbwtJBZplHfm91lceTlpEA9t4RR96dyM?usp=sharing

Esta es la estructura de los contenidos en cada carpeta:

Datasets: Esta carpeta contiene una serie de csv-s, uno con frases de mistborn para entrenar un modelo, y todos los outputs del análisis con TF-IDF del corpus de 16 personalidades. Los csv de p_16_top_words contienen los tipos con una lista de pares palabra valor de importancia en esa personalidad, el "_few" solo está realizado con 3 pestañas de la web por cada personalidad. El archivo _8_dims es lo mismo pero habiendo separado el corpus por dimensiones (E, I, S, N, F, T, P ,J) en lugar de tipos completos, además las palabras y sus scores están en listas diferentes debido a una necesidad de procesado. Ahora contiene la versión actualizada de los resultados de ampliar el corpus de Mistborn y el de 16 personalidades. También contiene la carpeta dataset_clasificador final, que contiene todos los .csv que hemos utilizado para la tarea de clasificación. A parte de esto, hemos incluido la carpeta dataset_rnn_quotes, carpeta en la que están tanto la versión base del dataset de BookNLP con las quotes para la tarea de generación, como la versión filtrada y procesada, el vectorizer y dos modelos de prueba con sus resultados en .txt.

BookNLP: Esta carpeta contiene los outputs en .txt de la librería BookNLP, que hemos utilizado por separado en 2 de los libros de la saga y en el conjunto de la trilogía. Esta carpeta ha sido ampliada con los resultados de BookNLP del segundo libro de la trilogía, así como con la versión limpia de los tres libros juntos.

Corpus: Esta carpeta contiene todos los datos utilizados para esta tarea, están los libros de la saga en txt, pdf y en epub. A parte de los libros están también los archivos con la información de cada personalidad, la completa y la reducida, en "16Personalities" y "Personalities_16_few" respectivamente. Esta carpeta, que contiene nuestro corpus, ha sido ampliada con el segundo libro de Mistborn y con la versión limpia de los tres libros. También se ha añadido la carpeta personalities_16_few_plus, que contiene la información de las personalidades ampliada con la página https://www.truity.com/.

Codigo: La carpeta de código se ha reestructurado en varias subcarpetas, organizadas por las diferentes entregas al largo de la asignatura. El código perteneciente a la cuarta entrega se encuentra en entrega_4.

Embeddings: Esta carpeta contiene los distintos embeddings de Word2Vec que hemos intentado utilizar (incluido google-news). Finalmente, en nuestros modelos usamos los embeddings combinados que hemos construido nosotros (*combined_embeddings*).

Entregas: Esta carpeta contiene los documentos escritos relativos al proyecto.

Enviroment: Esta carpeta contiene la información del environment de python que estamos usando para ejecutar nuestro código (no tiene en cuenta alguna librería que probamos puntualmente en notebooks de prueba).

Actualización y Creación de Datasets:

Para esta entrega hemos extendido algunos datasets utilizados anteriormente y hemos creado algunos nuevos para las tareas que se piden. En lo que se refiere al corpus lo hemos ampliado, tanto para la parte de Mistborn como para la de 16 Personalidades.

Hemos incluido "El pozo de la Ascensión", segundo libro de la trilogía a nuestro corpus de libros, así como un .txt limpio que contiene los tres libros.

Por otro lado, hemos ampliado la información de cada personalidad con la que nos ofrece la página de https://www.truity.com/ , comprobando que esta versión extendida crea mejores representaciones de las personalidades con "Bag Of Words" y "Tf-Idf".

En relación con esto, hemos vuelto a entrenar el "Word 2 Vec" con este corpus actualizado para utilizarlo posteriormente en la clasificación.

- Para la tarea de clasificación hemos creado los siguientes datasets en formato .csv:

- **classify_char_processed**: Esta versión contiene un personaje con 20 modificadores **únicos**, es decir, un mismo personaje, en todas sus apariciones en el dataset tiene un conjunto de 20 modificaciones **disjunto** del del resto de sus apariciones. El split está repartido indistintamente, ya sea una fila de Mistborn o de 16 personalidades.

- **classify_char_raw**: (durante la comparación de modelos nos referiremos a este dataset tambien como *web-más-libro*)Esta versión contiene un personaje con 20 modificadores **repetibles**, es decir, un mismo personaje, en todas sus apariciones en el dataset tiene un conjunto de 20 modificaciones que puede contener palabras que también estén en otras de sus apariciones. El split está repartido indistintamente, ya sea una fila de Mistborn o de 16 personalidades.

- **classify_mistTest_p16Train_raw**: (durante el texto nos referiremos a esta dataset como *solo-web*) Esta versión contiene un personaje con 20 modificadores **repetibles**, es decir, un mismo personaje, en todas sus apariciones en el dataset tiene un conjunto de 20 modificaciones que puede contener palabras que también estén en otras de sus apariciones. El split está repartido **no uniformemente**, las filas de **16 personalidades** corresponden a **Train** y las de **Mistborn** a **Val** y **Test**.

- Todos ellos cuentan con las mismas cuatro columnas:

- **char**: string que representa el **nombre del personaje** al que se le asignan los modificadores, para las filas que vienen de 16 personalidades hemos puesto NA.

- **mods**: lista de **20 modificadores** del personaje en cuestión, es un string separado por espacios.

- **target**: string, **label**, que representan la **personalidad** (1 de las 16 posibles) a la que pertenece ese personaje, todos los personajes con el mismo char tienen asignada la misma personalidad.

- **split**: string que representa el conjunto de entrenamiento al que pertenece esa entrada del dataset, puede ser **train**, **val** o **test**. Se distribuyen siguiendo una proporción de **0.7**, **0.15** y **0.15** respectivamente.

Entrega 2:

Operaciones básicas de procesamiento de texto:

1. Preparación del texto:

Nuestro corpus consiste principalmente de documentos PDF puesto que este formato es el más utilizado para transferir y reproducir textos digitalmente. Para esta primera práctica vamos a operar sobre el texto que forman las dos primeras novelas de la primera trilogía de *Mistborn*, por Brandon Sanderson. Lo primero que debemos hacer es leer el PDF como una lista de palabras.
El primer paso de nuestro proyecto es transcribir los libros descargados en formato PDF a txt. Si bien PDF es utilizado para transferir y visualizar las novelas contiene secciones como el índice, marcas de agua y otros elementos que no nos interesan. Es por ello que quisimos extraer el texto *raw* de estos ficheros. Para ello creamos este *script:*

```
import textract
import ftfy

fileIn = "../Corpus/Epub/TheWellOfAscension.epub"
fileOut = "../Corpus/Txt/TheWellOfAscension.txt"


content= textract.process(fileIn, encoding='utf-8').strip().decode('utf-8',
errors='ignore').replace("OceanofPDF.com", "").replace("Unknown", "")
content = ftfy.fix_text(content)
print(content)
with open(fileOut, 'w', encoding='utf-8') as fout:
   fout.write(content)
```

Si bien la transformación podría parecer trivial gracias a la librería *textract*, tuvimos problemas con caracteres extraños que no se transcribian correctamente y que mermaban los procesos que vamos a describir a continuación. Es por ello que hacemos uso de la librería *ftfy*, que no solamente logra identificar esos caracteres erróneos, sino que además los sustituye por los que se *representaban* en el PDF original. (No serán exactamente *el mismo* caracter, pero sí representará lo mismo).

2. Tokenización

Para esta fase del proyecto vamos a probar 3 métodos de tokenizacion y ver cual es que mejor funciona para nosotros. Los métodos son:

1. Word punct de NLT, que está basado en una expresión regular para separar las palabras en tokens. No es un algoritmo muy sofisticado pero nos permite aislar los signos de puntuación de los tokens.
2. Vamos a tratar de hacer nuestro método propio de tokenización utilizando una expresión regular. La expresión es '[^a-zA-Z0-9]' . Vamos a crear este m-etodo manual porque su comportamiento es completamente predecible para nosotros y podemos utilizarlo para comparar las diferencias entre los dos conjuntos de tokens

3. PUNKT es un método basado en un modelo no supervisado que tiene como objetivo dividir un texto en oraciones. Como nosotros queremos tokens que representen palabras pasamos esas oraciones transcritas por punkt a word_tokenize el cual utiliza expresiones regulares para hacer tokens en forma de *Penn Treebank*.

Los métodos de NLTK separan los signos de puntuación y similares de las palabras mientras que nuestra expresión regular simplemente los ignora y no los salva. A pesar de esto, Word punkt tiene un número de tokens diferentes muy similar a los obtenidos a través de nuestra expresión regular pero punkt tiene unos cuantos (31% más tokens) que los métodos mencionados al principio.

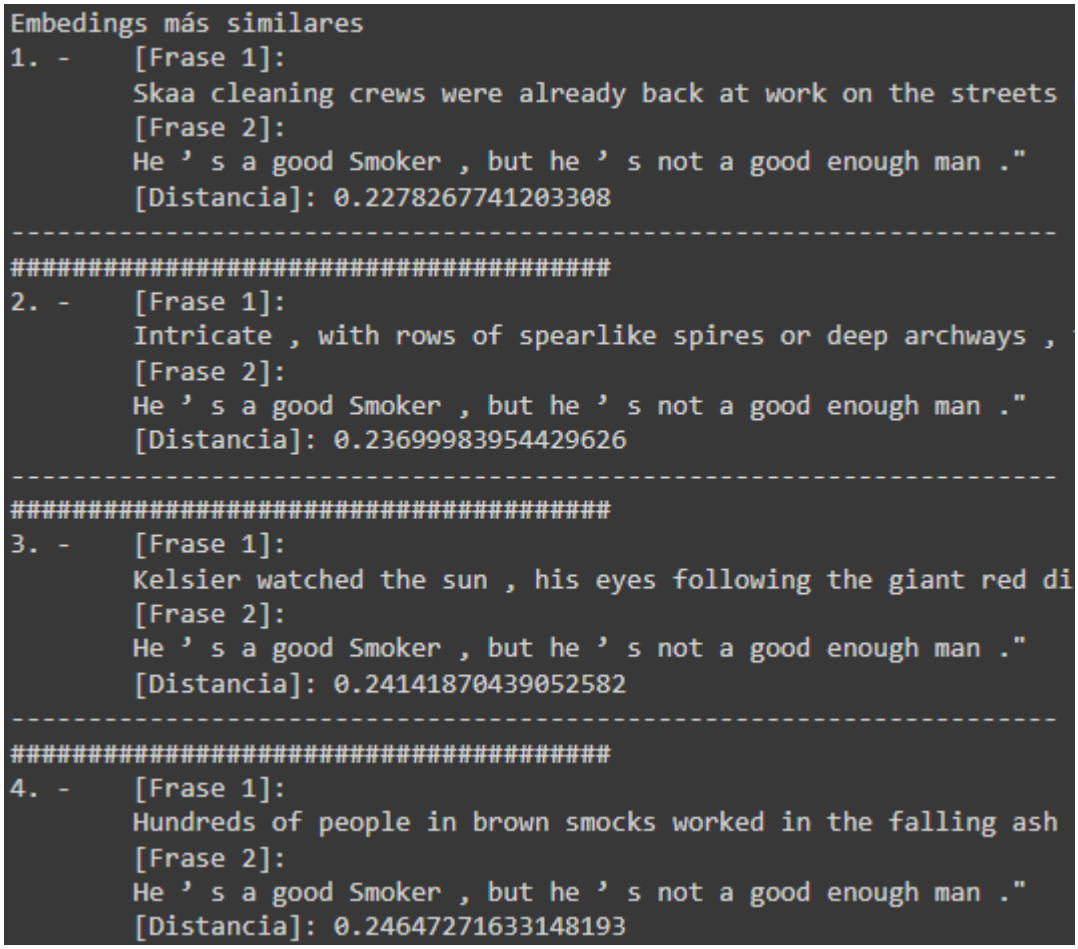
Si analizamos las diferencias entre los conjuntos de tokens, podemos observar que los caracteres que están en word punkt pero que han sido ignorados por nuestra expresión regular, son todos los signos de puntuación y otros símbolos especiales. Por otro lado, nuestra expresión regular ha captado 4 tokens que word punkt ha ignorado, pero lo más probable es que estos tokens sean producto de casos excepcionales de palabras extrañas que al pasar por nuestra expresión regular han perdido letras y ahora no se entienden. En otras palabras, son errores que podemos ignorar y que no determinan qué método es mejor. En cuanto a las diferencias entre word punkt y PUNKT + word_tokenize, hay unas cuantas palabras coherentes que word punkt ha aceptado como tokens pero que PUNKT + word_tokenize ha ignorado. Ejemplos de estas son 'deplorability', 'vaporous', 'ranges' o "paned', Estas son palabras reales que deberían haberse guardado como tokens. Finalmente, PUNKT + word_tokenize ha encontrado una gran cantidad de palabras que word punkt no. Estas son palabras que siguen esta estructura: 'half-breeds', 'else—the', 'person—whoever', o 'enough—you'. Estos *tokens* son en realidad parejas de palabras unidas con un guión, las cuales por separado pertenecen al conjunto de tokens extraidos por el primer método y representarlas por parejas es una representación menos conveniente que hacerlo por separado. Esto se debe a que queremos que nuestros tokens sean palabras sin caracteres especiales y signos de puntuación.

En conclusión, puesto que queremos hacer uso de un método de tokenización sofisticado como los que NLTK ofrece y hemos podido ver que word punkt hace una representación màs adecuada de los tokens que PUNKT + word_token(), hemos decidido representar nuestro texto con los tokens obtenidos con el primer método.

3. Stop words

Para un correcto análisis del texto es necesario eliminar los stop words. Para ello cargamos los stops words en inglés previstos por NLTK y los eliminamos de los vectores de tokens. Esa lista nos resultó insuficiente para eliminar todos los tokens que no aportan información en cualquiera de las tres listas. En las listas de PUNKT + word_tokenize y word_punkt no logramos eliminar los signos de puntuacion y en la lista de tokens extraida mediante expresiones regulares, contracciones como 't, 's son interpretadas como tokens y además son de los más frecuentes.

Es por ello que creamos una *blackList* con signos de puntuación, contracciones y otros simbolos similares.



4. Bag of Words:

Utilizamos un bag of words para encontrar qué palabras fueron las más frecuentes. No utilizamos ninguna libreria, sino que producimos nuestro própio codigo:

```
def getDiccionarioAppearences(listaTokens):
    palabrasDic = {}
    for palabra in listaTokens:
        if palabra in palabrasDic:
            palabrasDic[palabra] +=1
        else:
            palabrasDic[palabra] = 1
    return palabrasDic

def getPalabrasMasFrecuentes(palabrasDic, k):
    palabrasDic = dict(sorted(palabrasDic.items(), key = lambda x:-x[1]))
    return list(palabrasDic)[:k]
```

Con esto, creamos un diccionario que dada una palabra del libro te devuelve el número de veces que aparece esa palabra. Si ordenamos los tokens de cada lista por número de apariciones tras eliminar los stop words obtenemos lo siguiente:

```
Palabras mas frecuentes NLTK: ['said', 'Vin', 'Elend', 'Kelsier', 'Sazed',
Palabras mas frecuentes regex: ['said', 'Vin', 'Elend', 'Kelsier', 'Sazed',
Palabras mas frecuentes PUNKT: ['said', 'Vin', 'Elend', 'Kelsier', 'Sazed',
```

Podemos ver cómo las palabras más repetidas son los nombres de los protagonistas. Esto es muy coherente puesto que son palabras que se repiten muchas veces y encima tienen significado.

Un ejemplo de bag of words:

```
{'ASH': 3,
 'FELL': 2,
 'SKY': 3,
 'Lord': 1198,
 'Tresting': 57,
 'frowned': 298,
 'glancing': 83,
```

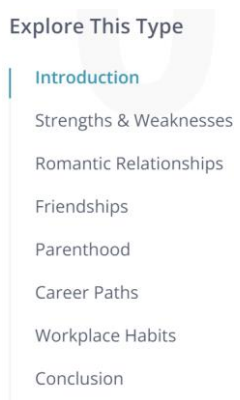
5. Elmo

Tratamos de hacer una representación de embeddings contextual mediante Elmo. Nuestra idea era dividir los dos primeros libros en listas de tokens mediante PUNKT y puncword tokenizer. Puesto que no logramos instalar elmo en nuestra máquina, hemos recurrido a google collab pero como el límite de memoria es de 12.7Gb, no hemos sido capaces de hacer embeddings de más de el 3% de las oraciones del primer texto.

Tratamos de hacer un ranking de las 10 oraciones que más se parecen entre ellas de acuerdo a su representación en embeddings. No obstante, al intentarlo tuvimos un problema: todas las 10 parejas más frecuentes tenían la misma frase como una de sus partes. Tratamos de entender por qué y deducimos que se debía a que esa oración era mucho más corta que la oración más larga con la que se entrenó Elmo y muchos de sus últimas filas son 0. Tratamos de enmascararlo y solucionarlo de alguna manera. Finalmente logramos parchearlo seleccionando frases con entre 17 y 20 tokens. Al representar estas 172 palabras, logramos que las parejas fuesen variadas. No obstante los resultados no son muy enriquecedores.

16 Personalidades

Para caracterizar las 16 personalidades que utilizaremos para clasificar hemos probado diferentes técnicas. Hemos descargado de PersonalityDatabase la información de cada una de ellas a un documento de texto. En esta web cada personalidad tiene 8 páginas con información sobre ella. Hemos realizado todas las pruebas con todas las páginas y solamente con las páginas de "Introduction", "Strengths & Weaknesses" y "Conclusion", pues a lo mejor las otras páginas son sobre temas demasiado específicos y diluyen los resultados.



El primer paso para procesar la información ha sido la tokenización de los documentos, para tener una lista con cada una de las personalidades asociadas a sus tokens. Tras una limpieza de las stopwords, hemos realizado una bag of words para contar la ocurrencia de los tokens en cada personalidad, y los resultados nos indican que el nombre utilizado por la página para designar a las personalidades es siempre la palabra más presente, junto con otras como "personality", "may",

“people” o “type”, que se utilizan en oraciones como “people with this personality type may…” y que en nuestro contexto consideraremos stopwords.

```
enfp,"['campaigners:0.9128815318899555', 'campaigner:0.1615719525468948', 'may:0.10548265083444561', 'people:0.09519166050913384', '
```

Tras este procesado inicial los vectores de tokens que tenemos como resultado tienen estas longitudes.

Solo 3 páginas:

```
El -=<[ ENFJ ]>=- tiene 631 tokens
El -=<[ ENFP ]>=- tiene 646 tokens
El -=<[ ENTJ ]>=- tiene 708 tokens
El -=<[ ENTP ]>=- tiene 749 tokens
El -=<[ ESFJ ]>=- tiene 653 tokens
El -=<[ ESFP ]>=- tiene 660 tokens
El -=<[ ESTJ ]>=- tiene 693 tokens
El -=<[ ESTP ]>=- tiene 754 tokens
El -=<[ INFJ ]>=- tiene 708 tokens
El -=<[ INFP ]>=- tiene 756 tokens
El -=<[ INTJ ]>=- tiene 795 tokens
El -=<[ INTP ]>=- tiene 717 tokens
El -=<[ ISFJ ]>=- tiene 685 tokens
El -=<[ ISFP ]>=- tiene 702 tokens
El -=<[ ISTJ ]>=- tiene 715 tokens
El -=<[ ISTP ]>=- tiene 751 tokens
Entre todos los tipos ah 11323 tokens y 2877 son únicos
```

Todas las páginas:

```
El -=<[ ENFJ ]>=- tiene 1915 tokens
El -=<[ ENFP ]>=- tiene 2037 tokens
El -=<[ ENTJ ]>=- tiene 1888 tokens
El -=<[ ENTP ]>=- tiene 2070 tokens
El -=<[ ESFJ ]>=- tiene 1650 tokens
El -=<[ ESFP ]>=- tiene 1713 tokens
El -=<[ ESTJ ]>=- tiene 1751 tokens
El -=<[ ESTP ]>=- tiene 1798 tokens
El -=<[ INFJ ]>=- tiene 2121 tokens
El -=<[ INFP ]>=- tiene 2191 tokens
El -=<[ INTJ ]>=- tiene 2243 tokens
El -=<[ INTP ]>=- tiene 2295 tokens
El -=<[ ISFJ ]>=- tiene 2034 tokens
El -=<[ ISFP ]>=- tiene 1784 tokens
El -=<[ ISTJ ]>=- tiene 1911 tokens
El -=<[ ISTP ]>=- tiene 1807 tokens
Entre todos los tipos ah 31208 tokens y 4966 son únicos
```

Tras realizar los wordcount-vectors de cada personalidad y calcular la cosine similarity entre ellas, observamos que los resultados son más o menos coherentes, pues, a grandes rasgos, las personalidades más parecidas entre sí son las que difieren en 2 o menos componentes (E/I , S/N, F/T, P/J).

```
Cosine similarity between -=<[ENFJ]>=- and -=<[ENFJ]>=- : 1.00
Cosine similarity between -=<[ENFJ]>=- and -=<[ENFP]>=- : 0.70
Cosine similarity between -=<[ENFJ]>=- and -=<[ENTJ]>=- : 0.54
Cosine similarity between -=<[ENFJ]>=- and -=<[ENTP]>=- : 0.56
Cosine similarity between -=<[ENFJ]>=- and -=<[ESFJ]>=- : 0.61
Cosine similarity between -=<[ENFJ]>=- and -=<[ESFP]>=- : 0.56
Cosine similarity between -=<[ENFJ]>=- and -=<[ESTJ]>=- : 0.57
Cosine similarity between -=<[ENFJ]>=- and -=<[ESTP]>=- : 0.29
Cosine similarity between -=<[ENFJ]>=- and -=<[INFJ]>=- : 0.76
Cosine similarity between -=<[ENFJ]>=- and -=<[INFP]>=- : 0.75
Cosine similarity between -=<[ENFJ]>=- and -=<[INTJ]>=- : 0.65
Cosine similarity between -=<[ENFJ]>=- and -=<[INTP]>=- : 0.66
Cosine similarity between -=<[ENFJ]>=- and -=<[ISFJ]>=- : 0.74
Cosine similarity between -=<[ENFJ]>=- and -=<[ISFP]>=- : 0.59
Cosine similarity between -=<[ENFJ]>=- and -=<[ISTJ]>=- : 0.58
Cosine similarity between -=<[ENFJ]>=- and -=<[ISTP]>=- : 0.52
```

Ahora probaremos con la aplicación de TF-IDF para comparar las personalidades entre sí.

```
Cosine similarity between -=<[ENFJ]>=- and -=<[ENFJ]>=- : 1.00
Cosine similarity between -=<[ENFJ]>=- and -=<[ENFP]>=- : 0.50
Cosine similarity between -=<[ENFJ]>=- and -=<[ENTJ]>=- : 0.35
Cosine similarity between -=<[ENFJ]>=- and -=<[ENTP]>=- : 0.35
Cosine similarity between -=<[ENFJ]>=- and -=<[ESFJ]>=- : 0.43
Cosine similarity between -=<[ENFJ]>=- and -=<[ESFP]>=- : 0.36
Cosine similarity between -=<[ENFJ]>=- and -=<[ESTJ]>=- : 0.37
Cosine similarity between -=<[ENFJ]>=- and -=<[ESTP]>=- : 0.11
Cosine similarity between -=<[ENFJ]>=- and -=<[INFJ]>=- : 0.57
Cosine similarity between -=<[ENFJ]>=- and -=<[INFP]>=- : 0.55
Cosine similarity between -=<[ENFJ]>=- and -=<[INTJ]>=- : 0.43
Cosine similarity between -=<[ENFJ]>=- and -=<[INTP]>=- : 0.47
Cosine similarity between -=<[ENFJ]>=- and -=<[ISFJ]>=- : 0.54
Cosine similarity between -=<[ENFJ]>=- and -=<[ISFP]>=- : 0.39
Cosine similarity between -=<[ENFJ]>=- and -=<[ISTJ]>=- : 0.37
Cosine similarity between -=<[ENFJ]>=- and -=<[ISTP]>=- : 0.33
```

Observamos que pese a tener una escala diferente las relaciones de similitud se mantienen en los vectores creados por el modelo de TF-IDF.

Documento: enfj	Documento: entp	Documento: isfp
right: 0.1449	ideas: 0.1782	need: 0.1562
even: 0.1449	mental: 0.1375	live: 0.1533
others: 0.1366	idea: 0.1326	life: 0.1406
whats: 0.1216	find: 0.1265	artistic: 0.1281
problems: 0.1130	beliefs: 0.1061	moment: 0.1179
persons: 0.1074	many: 0.1043	exciting: 0.1147
speak: 0.1074	even: 0.0984	freedom: 0.1147
rarely: 0.1021	sacred: 0.0882	open: 0.1042
enfj: 0.1011	disparate: 0.0882	enjoyment: 0.0981
walk: 0.1011	questioned: 0.0882	exploratory: 0.0981
shares: 0.1011	exercise: 0.0882	beauty: 0.0957
recognize: 0.0912	arguments: 0.0882	things: 0.0879
greater: 0.0912	sparring: 0.0882	worry: 0.0854
values: 0.0912	contrarianism: 0.0882	spirits: 0.0854
	point: 0.0861	time: 0.0842
	much: 0.0790	whether: 0.0792
	points: 0.0768	creativity: 0.0792

Si analizamos las palabras más representativas de cada uno de los tipos, podemos observar que difieren bastante entre sí, estas palabras son de la versión reducida de 3 páginas. En la otra versión las palabras como "even". "right" o otras como "work" y "children" tenían más importancia, pero realmente esto se debe posiblemente a la estructuración de la propia web a la hora de presentar las secciones "workplace habits" y "parenthood". Con el objetivo de caracterizar las personalidades creemos que es mejor quedarnos con la version de páginas reducidas, aunque en un futuro próximo tal vez intentemos crear los embeddings de los tipos con un corpus más variado y extenso, cogiendo de diferentes webs y plataformas, con el objetivo de deshacerse del bias de 16 Personalities.

Con el objetivo de probar otras representaciones, hemos probado también este método pero separando las categorías en las 4 distintas dicotomías (E/I, S/N, F/T, P/J). Aquí nuestro corpus para cada una de las dimensiones se multiplica, pues se compone de las 8 personalidades que contienen esa característica.

```
Cosine similarity between -=<[E]>=- and -=<[E]>=- : 1.00
Cosine similarity between -=<[E]>=- and -=<[I]>=- : 0.83
Cosine similarity between -=<[E]>=- and -=<[S]>=- : 0.90
Cosine similarity between -=<[E]>=- and -=<[N]>=- : 0.87
Cosine similarity between -=<[E]>=- and -=<[F]>=- : 0.88
Cosine similarity between -=<[E]>=- and -=<[T]>=- : 0.91
Cosine similarity between -=<[E]>=- and -=<[P]>=- : 0.90
Cosine similarity between -=<[E]>=- and -=<[J]>=- : 0.90
```

Vemos que los resultados son los esperados, ya que las características que son polos opuestos de la misma dimensión son las menos similares entre sí. Aún así hay una gran cantidad de palabras que se comparten entre todas o muchas de las categorías, sospechamos que puede ser por lo reducido que es nuestro corpus actualmente. `Hay 9600 palabras que describen a personalidades, de las cuales 2354 son úicas.`

Por lo que decidimos que con el fin de representar mejor cada una de estas 8 dimensiones cogeríamos solo aquellas palabras que no estén también en su categoría opuesta. Observamos que ahora las palabras son mucho mejores descriptoras de las dimensiones que antes.

```
E : length = 629
--------------------------
['together', 'push', 'extraverted', 'qualities', 'leaders', 'group', 'leading', 'tasks', 'experiences', 'passion


I : length = 629
--------------------------
['introverted', 'open', 'integrity', 'freedom', 'giving', 'loyalty', 'imaginative', 'match', 'heart', 'research'


S : length = 721
--------------------------
['roadmap', 'easily', 'hands', 'observant', 'effort', 'versatile', 'communities', 'willing', 'flow', 'experience


N : length = 721
--------------------------
['insight', 'step', 'imagination', 'rationality', 'achieve', 'problems', 'youre', 'inner', 'thought', 'dreams',
```

Tras esta prueba con Bag of Words y TF-IDF probamos también la representación de nuestro corpus de 16 personalidades con Word2Vec. Puesto que los mejores resultados, por el momento, nos han dado con el dataset de personalidades reducido, son esos .txt los que utilizaremos como corpus en el Word2Vec. Primero hicimos la prueba cargando los word_vectors del modelo de GoogleNews, donde las palabras más importantes tienen representaciones que, lejos de ser malas, no son lo que buscamos.

```
extroverted
--------------------------
[('extrovert', 0.7257617115974426), ('introverted', 0.6306266784667969), ('gregarious', 0.625848
--------------------------

introverted
--------------------------
[('introvert', 0.7447564005851746), ('extrovert', 0.6764617562294006), ('loner', 0.64258235692977
--------------------------

observant
--------------------------
[('Observant', 0.5880488157272339), ('attentive', 0.573497474193573), ('conscientious', 0.556541
--------------------------

intuitive
--------------------------
[('intuitive_interface', 0.7640835046768188), ('user_friendly', 0.694433867931366), ('intuitive_
--------------------------

feeling
--------------------------
[('feel', 0.7652187347412109), ('felt', 0.7212082147598267), ('feels', 0.6793736219406128), ('Fe
--------------------------

thinking
--------------------------
[('Thinking', 0.6526099443435669), ('think', 0.6110401153564453), ('dreaming', 0.606206774711608
--------------------------

prospecting
--------------------------
[('Prospecting', 0.7202379703521729), ('exploration', 0.6273497343063354), ('exploratory_drillin
--------------------------

judging
--------------------------
[('Judging', 0.676307201385498), ('judged', 0.6227951049804688), ('Judged', 0.5002167224884033),
--------------------------
```

Podemos ver que el significado que se les está atribuyendo a estas palabras no es exactamente el mismo que se les atribuye en el ámbito de la psicología. El mejor ejemplo de ello es "intuitive", qué se entiende como algo intuitivo y fácil de usar, en lugar de una persona con buena intuición.
A parte de esto, al analizar las palabras de nuestro corpus que no están en el modelo, pese a ser únicamente 38 de 2879, vemos que son las más importantes, dado que son las personalidades que queremos asignar a los personajes, y tener una representación en forma de embedding de estas es esencial para ello. Otras palabras que no aparecen son aquellas referidas a atributos poco comunes o actitudes concretas, las cuales también nos interesan con fin predictivo.

```
38 words of our corpus of 2879 words do not appear in the google news Word2Vec.
['readapt', 'istp', 'enfjs', 'unvetted', 'infjs', 'sherlock', 'openhearted', 'attunement',
```

Hemos pensado que finetunear un modelo pre entrenado sería una buena idea, sin embargo el word_vector de Google news no es re-entrenable. Hemos probado también a realizar nuestro propio modelo de Word2Vec, pero debido, probablemente, a la falta de corpus los embeddings resultantes no son lo que necesitamos, los tipos (entp, esfj, istp...) son agrupados por su significado de "tipo", cuando realmente lo que interesaría sería que cada tipo estuviera bien diferenciado, y con las 8 dimensiones pasa lo mismo, los tensorboard logs que se crean lo demuestran, generando una nube de puntos muy mal diferenciada.

Palabras más parecidas a cada tipo:

```
esfp
--------------------------
[('isfp', 0.8469275832176208), ('entp', 0.6726663112640381),
--------------------------

esfj
--------------------------
[('esfp', 0.6624258160591125), ('entp', 0.6573789715766907),
--------------------------

estp
--------------------------
[('estj', 0.8261793851852417), ('infj', 0.7672327160835266),
--------------------------

estj
--------------------------
[('estp', 0.8261793255805969), ('infj', 0.8256271481513977),
--------------------------
```

Palabras más parecidas a cada dimensión:

```
extraverted
--------------------------
[('intuitive', 0.749420166015625), ('introverted', 0.6240891
--------------------------

introverted
--------------------------
[('extraverted', 0.6240891218185425), ('intuitive', 0.5489251
--------------------------

observant
--------------------------
[('intuitive', 0.7075260877609253), ('extraverted', 0.5423019
--------------------------

intuitive
--------------------------
[('extraverted', 0.7494200468063354), ('judging', 0.71795922
--------------------------
```

Finalmente las pruebas con ELMO no han sido muy exhaustivas, pues nos ha faltado fine tunearlo a nuestro corpus, actualmente los word embeddings que genera son poco precisos para nuestra tarea concreta, además de ser mejor para el análisis de frases, y nosotros tenemos el enfoque puesto en la clasificación.

Pruebas con librerías externas

BookNLP

Tras investigar en el estado del arte descubrimos esta librería especializada en el procesamiento de novelas, la cual nos pareció muy útil a pesar de la escasa documentación. La librería funciona de la siguiente manera:

1. Se le proporciona un corpus en formato txt, así como un pipeline (de tareas de NLP) a seguir y si lo deseas (en nuestro caso fue obligatorio descargar los modelos de BERT a mano para poder arrancarlo) unas rutas con los distintos modelos de lenguaje que utilizará para procesar el texto.
2. La librería procesa el corpus al completo, realizando cada tarea secuencialmente, y saca como resultado varios ficheros.

Fichero .tokens

Incluye cada uno de los tokens de cada línea del texto, así como la siguiente información sobre ellos:

- **paragraph_ID** - Índice del párrafo
- **sentence_ID** - Índice de la línea
- **token_ID_within_sentence** - Índice del token en la linea
- **token_ID_within_document** - Índice del token en el documento
- **word** - Token
- **lemma** - Raíz del token
- **byte_onset** - Índice del primer carácter del token
- **byte_offset** - Índice del último carácter del token
- **POS_tag** - tag del POS de Spacy
- **fine_POS_tag**
- **dependency_relation** - tag DEP de Spacy
- **syntactic_head_ID**
- **event** - Indica si el token origina un evento. Valores → O, NaN, EVENT

Fichero .entities

Proporciona información de las entidades detectadas con el modelo BERT en el corpus:

- **COREF** - Identificador único de la entidad mencionada directamente o mediante correferencia. Este es uno de los mayores retos del NLP, así que no es muy preciso. Este identificador se usa en otros ficheros, como el de quotes.
- **start_token** - Token de inicio de la entidad
- **end_token** - Token final de la entidad, puede ser el mismo que el de inicio
- **prop** - Diferencias entre PROP (proper noun) o PROPN (pronoun), o otras categorías
- **cat** - Tipo de entidad según SpaCy
- **text** - Texto crudo que corresponde a la entidad

Fichero .quotes

Contiene información sobre las citas detectadas en el corpus:

- **quote_start** - Token inicial de la cita
- **quote_end** - Token final de la cita
- **mention_start** - Token inicial de la entidad que habla
- **mention_end** - Token final de la entidad que habla
- **char_id** - Identificador único de la entidad citada; igual que el COREF anterior
- **quote** - Texto crudo de la cita

Fichero .supersense

Fichero similar al de entidades con información detallada sobre tokens detectados por ejemplo como sustancias, percepciones, etc ... Es un fichero único de esta librería:

- **start_token** - Token inicial del texto supersense
- **end_token** - Token final del texto supersense

- **supersense_category** - Categoría POS del texto
- **text** - Texto crudo

Fichero .book

JSON o diccionario con información sobre los personajes del corpus:

- **agent** - Acciones realizadas por el personaje
- **patient** - Acciones recibidas por el personaje
- **mod** - Adjetivos que los describen en el texto
- **poss** - Cosas o entidades relativas o poseídas por la entidad
- **id** - ID único
- **g** - Análisis sobre los pronombres de género utilizados
- **count** - Veces que aparece la entidad
- **mentions** - Menciones del personaje

Fichero .book.html

Fichero estructurado en HTML que analiza el texto al completo, identificando y categorizando cada palabra. También representa un resumen de todos los anteriores.

Conclusiones

A nuestro parecer, la librería puede ahorrarnos gran cantidad de trabajo, especialmente a la hora de detectar las apariciones de cada entidad y los adjetivos que las describen, ya que nuestro objetivo es asociar esos conjuntos de adjetivos a una categoría de personalidad.

Preparación datasets para PyTorch

También hemos realizado algunas pruebas para procesar los ficheros txt que hemos extraído, y transformarlos en un fichero csv (incluyendo stopwords) que incluye cada línea del corpus (más adelante tendremos que transformarlo en cada línea en la que se menciona a un personaje, o a la lista de adjetivos que definen a un personaje) y una división en train, eval y test (70, 15, 15)

Entrega 3:

Durante esta segunda tenemos dos objetivos: la clasificación de personalidades basado en los modificadores de la personalidad y la generación de quotes. Estos dos objetivos los describimos más adelante.

Clasificación de personalidades:

Objetivo

Uno de los principales objetivos de nuestro trabajo era crear un modelo de clasificación de personajes en las 16 personalizadas descritas en las memorias anteriores. De forma adicional, encontramos de interés analizar los tokens que el modelo encuentra de mayor utilidad para clasificar la personalidad de un personaje. Con este objetivo hemos acabado haciendo una clasificación con modelos de shallow ML, dos redes convolucionales (cada una con un modelo de embeddings diferente), y tres perceptrones (uno sin perceptrón y dos con las características que acabamos de describir.)

Obtención de datos

Antes de comenzar con la clasificación, fue necesaria la extracción de las características de cada personaje que servían para definirlo. Estos modificadores son descriptores tanto objetivos, como roles dentro de una sociedad o cualidades físicas (rey, hombre, joven, muerto); o subjetivos, como sentimientos o descripciones de emociones (diferente, asustado, insufrible). Estos modificadores los pudimos obtener gracias a *BookNLP*, el cual nos devuelve un personaje y su lista de modificaciones. En esta lista un mismo modificador puede aparecer varias veces si así lo hace en el libro. En un principio pensamos que teníamos que eliminar las repeticiones (dataset classify_char_processed), pero posteriormente decidimos dejarlas (dataset classify_char_raw) esperando que el modelo interpretase la repetición de un modificador como una forma de enfatizarlo. A continuación, dividimos la lista de modificaciones en grupos de 20 y añadimos dos etiquetas: la personalidad del personaje y si esa fila iba a pertenecer a set de datos de entrenamiento, de evaluación o de prueba. De forma adicional a los personajes y sus modificadores obtenidos de la saga Mistborn, añadimos también las palabras más características de cada una de las 16 personalidades. Para hacer esto, logramos mediante scrapping (a mano), el texto de las páginas web donde se describen cada una de las personalidades y tras hacer TF-IDF logramos las más características de cada una. Del mismo modo que con los personajes, hicimos bolsas de 20 palabras y añadimos al dataset estas entradas. Para ser fieles a nuestra propuesta inicial del problema, pensamos que la red debería ser capaz de aprender a partir de las palabras sacadas de las páginas web y luego ese conocimiento debería ser extrapolable a personajes de libros (al dataset con esta configuración lo llamaremos dataset *solo-web* puesto que todos los ejemplos de entrenamiento provienen de una sola fuente: la página web de las 16 personalidades). Con esto en mente hicimos un dataset en el que todas las rows obtenidas a partir de personajes de libros pertenecían al set de prueba o al de evaluación y las obtenidas de la página web pertenecen a set de entrenamiento. En total terminamos con 1302 filas de personajes y personalidades con 20 modificaciones cada uno, una etiqueta de personalidad y otra del conjunto al que pertenecen (las filas con modificadores obtenidos de la página web no tienen nada en la columna de nombre del personaje). Como nos pareció complicado que la red fuese capaz de identificar la personalidad de un personaje a partir sólo de las palabras genéricas de la web probamos a hacer un set de datos con filas de

entrenamiento tanto de la web como de personajes del libro (a este dataset nos referiremos como dataset *web-más-libro*). Los resultados de todas estas configuraciones las describiremos a continuación.

Shallow Machine learning

Para la clasificación con métodos de machine learning somero, utilizamos dos tipos de vectorizer: countVectorizer y tf-idf vectorizer. Los métodos que utilizamos para hacer la predicción fueron Random Forest, Logistic Regression, Decision Tree, Linear SVM y Non-linear SVM. Además, clasificamos no solo los dos sets de datos que hemos mencionado anteriormente, sino, además uno en el que eliminamos los modificadores repetidos y solamente dejamos uno independientemente del número de veces que haya aparecido. También hemos analizado las palabras más importantes para cada importantes para cada modelo.

Como es de esperar el set de datos que mejor ha sido clasificado es el "web-mas-libro". Esto se debe a que los tokens que han sido visto por los modelos durante la fase de aprendizaje pueden o no estar contenidos en los vectores de prueba. Si bien la representación en modo de count vectorizer o tf-idf parece no tener una repercusión muy significativa en la capacidad de predicción del modelo, los representados mediante count-vectorizer pueden tener un desempeño ligeramente mejor.

Perceptrón Simple

Como ya hemos mencionado, tras el primer acercamiento con los modelos de shallow ML, el primer modelo de predicción que entrenamos fue un perceptrón simple. Este consta de una capa de regresión lineal que tiene una neurona de entrada por cada palabra de nuestro vocabulario y 16 neuronas de salida: una por cada personalidad. Todas las pruebas realizadas con redes neuronales las hicimos con el dataset que dio mejor resultado en las pruebas de shallow ML.

En cuanto a las redes neuronales, las entrenamos tanto con el dataset solo-web como con el web-mas-libro. Los resultados del perceptrón no son sorprendentemente buenos. Comenzando con el dataset web-mas-libro, logramos una precisión del 31.6%. No obstante, el interés de esta red es su capacidad para mostrar los tokens más característicos de cada personalidad. Para ver los resultados, haz clic en este enlace. Como podemos observar, muchos de los tokens son modificadores de un personaje en concreto. Por ejemplo "father" que hace referencia a Straff Venture cuya personalidad es ENTJ o "king, scholar y busy", los cuales pertenecen a Elen Venture y su personalidad es INFP. De todas formas, incluso con este dataset hemos logrado un equilibrio entre tokens provenientes de la web y de los libros; no obstante, hemos podido comprobar que a medida que la red cae en overfitting al entrenarla durante demasiadas épocas, esta tiende a utilizar tokens provenientes de modificaciones de personajes mucho más (casi de forma exclusiva) que los provenientes de la web. Hemos llegado a la conclusión de que esto se debe a que los personajes y sus modificaciones son mucho más característicos de una personalidad que los tokens provenientes de la web a pesar de que estos representen la mayoría de las filas de entrenamiento.

Este resultado nos motivó aún más a crear un dataset solo-web cuyas filas de entrenamiento excluyesen a los personajes del libro para poder comprobar si el conocimiento obtenido a partir de la página web podría servir para clasificar los personajes de las novelas a partir de sus modificadores. La precisión de este modelo es de

4,2% lo cual demuestra que con la representación de los modificadores que estábamos utilizando, el modelo no era capaz de ejecutar esta tarea correctamente. No obstante, los tokens que el perceptrón utilizaba para clasificar los personajes son más genéricos que con el modelo anterior. Esto es evidente, puesto que el modelo no habrá visto muchos de los modificadores de los personajes. Además, es destacable que a menudo los tokens más característicos son muy similares entre ellos y aparecen seguidos como "ideas" e "idea" para ENTP o "loyalty" y "loyal" para ISFJ. En conclusión, el aprendizaje más importante que sacamos del entrenamiento del modelo con este dataset es que la representación de las palabras que hacemos no es favorable para que el modelo pueda entrenarse con los tokens extraídos de la web y ser capaz de clasificar modelos con tokens que puede que no haya visto antes.

Embeddings + Perceptrón

Entrenamos nuestro propio modelo de embeddings con el objetivo de hacer una mejor representación de nuestros tokens. El corpus con el que entrenamos este modelo son los modificadores de los personajes de mistborn y el texto obtenido de la página de las 16 personalidades. Esta vez, decidimos utilizar un perceptrón multicapa para incrementar su capacidad predictiva a pesar de que no vayamos a ser capaces de obtener que tokens son los más utilizados para clasificar cada personaje en una personalidad.

Gracias a los embeddings, el modelo logra una precision del 34.37% con el dataset web-mas-libro y un 38% con el solo-web. Estas cifras no solo superan la capacidad predictiva del perceptrón que lo precedía, sino que, además, confirma nuestra hipótesis de que los embeddings dotan al modelo de la capacidad de hacer predicciones en base a tokens que no haya visto antes.

Embeddings + CNN

El siguiente modelo que hicimos fue una CNN con los mismos embeddings para poder hacer una mejor representación de las palabras. Decidimos utilizar una CNN por darle más variedad al ejercicio y probar la capacidad de nuevas redes neuronales en esta tarea, a pesar de que nuestro problema no esté basado en secuencialidad. Para hacer los embeddigns, no pudimos hacer un fine tune de los entrenados con las noticias por lo que hicimos el nuestro propio con los modificadores obtenidos de la página web y los libros. En total contiene 3862 tokens. Desgraciadamente, la capacidad de predicción del modelo no es la esperada: con el dataset web-mas-libro logramos una precisión del 12% y con el solo-web del 9%. No obstante, no es un descenso de precisión tan grande como el de los métodos anteriores sin embeddings.

Embeddings Google news + Perceptrón

No obstante, nuestra determinación por ser capaces de lograr el modelo con mayor capacidad de predicción nos llevó a probar una opción más. Pensamos que nuestros embeddings, dado el poco corpus con el que habían sido entrenados, no lograban tener una representatividad suficiente de la realidad. Es por eso por lo que decidimos utilizar el modelo de embeddings que fue entrenado con noticias de Google. Sorprendentemente la gran mayoría de nuestros modificadores pertenecían al vocabulario con el que ese modelo fue entrenado, incluso las más específicas de nuestro contexto. Tras entrenar la misma red de dos capas de regresión lineal con estos embeddings, logramos una precisión del 38%, que supera ligeramente al perceptrón sin embeddings, pero no a los mejores modelos de machine learning somero.

Embeddings Google news + CNN

Finalmente, probamos este modelo de embeddings con la red convolucional descrita anteriormente y los resultados no fueron los esperados. Con un 37% de capacidad de predicción, si bien no es un modelo nefasto, los predictores que hemos desarrollado tienen más mérito por su capacidad de enumerar qué tokens son más determinantes para clasificar un personaje, que para realmente utilizarlos para clasificar su personalidad.

Tabla de resultados

Modelo	Dataset	Features	Accuracy	Loss (%)
Random Forest	classify_char_processed	countVectorizer	8.00%	No data
	classify_char_processed	TfIdf	3.00%	No data
Random Forest	classify_char_raw (*"web-mas-libro"*)	countVectorizer	36.00%	No data
	classify_char_raw (*"web-mas-libro"*)	TfIdf	29.00%	No data
Random Forest	classify_mistTest_p16Train_raw (*"solo-web"*)	countVectorizer	8.00%	No data
	classify_mistTest_p16Train_raw (*"solo-web"*)	TfIdf	3.00%	No data
Logistic Regression	classify_char_processed	countVectorizer	5.00%	No data
	classify_char_processed	TfIdf	4.00%	No data
Logistic Regression	classify_char_raw	countVectorizer	38.00%	No data
	classify_char_raw	TfIdf	35.00%	No data
Logistic Regression	classify_mistTest_p16Train_raw	countVectorizer	5.00%	No data
	classify_mistTest_p16Train_raw	TfIdf	4.00%	No data
Decision Trees	classify_char_processed	countVectorizer	4.00%	No data

	classify_char_processed	TfIdf	4.00%	No data
Decision Trees	classify_char_raw	countVectorizer	22.00%	No data
	classify_char_raw	TfIdf	22.00%	No data
Decision Trees	classify_mistTest_p16Train_raw	countVectorizer	4.00%	No data
	classify_mistTest_p16Train_raw	TfIdf	3.00%	No data
Linear SVM	classify_char_processed	countVectorizer	4.00%	No data
	classify_char_processed	TfIdf	3.00%	No data
Linear SVM	classify_char_raw	countVectorizer	38.00%	No data
	classify_char_raw	TfIdf	38.00%	No data
Linear SVM	classify_mistTest_p16Train_raw	countVectorizer	4.00%	No data
	classify_mistTest_p16Train_raw	TfIdf	3.00%	No data
Nonlinear SVM	classify_char_processed	countVectorizer	8.00%	No data
	classify_char_processed	TfIdf	11.00%	No data
Nonlinear SVM	classify_char_raw	countVectorizer	28.00%	No data
	classify_char_raw	TfIdf	12.00%	No data
Nonlinear SVM	classify_mistTest_p16Train_raw	countVectorizer	8.00%	No data
	classify_mistTest_p16Train_raw	TfIdf	11.00%	No data
Perceptrón	classify_char_raw	Vocabulary + BoW	31.60%	2.36%
Perceptrón	classify_mistTest_p16Train_raw	Vocabulary + BoW	4.20%	2.78%
MLP (Embeddings propios)	classify_char_raw	Vocabulary + BoW	34.75%	2.07%

MLP (Embeddings Google News)	classify_char_raw	Vocabulary + BoW	38.54%	2.03%
CNN (Embeddings propios)	classify_char_raw	Vocabulary + BoW	22.91%	2.40%
CNN (Embeddings Google News)	classify_char_raw	Vocabulary + BoW	37.50%	1.91%

Generación de Quotes:

Para esta entrega se nos ocurrió que podríamos cubrir la tarea de generación de texto, a pesar de que nuestro objetivo principal fuera clasificar personalidades en base a modificadores. Así que tuvimos la idea de crear un generador de "quotes" o citas, añadiendo además la introducción del nombre del personaje como variable categórica en la hidden layer inicial, para que estas se generen acorde a su forma de expresarse.

Para esto entrenamos una GRU con el dataset de quotes sacadas de BookNLP. Tuvimos que crear un vocabulario nuevo con todas las palabras de ambos corpus, y la red predecirá el siguiente token (palabra) de la cita de un personaje.

```
<ipython-input-17-7620403e50cc>:12: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  epoch_bar = tqdm_notebook(desc='training routine',
training routine: 31% 156/500 [27:38<1:00:35, 10.57s/it]
<ipython-input-17-7620403e50cc>:17: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  train_bar = tqdm_notebook(desc='split=train',
split=train: 98% 59/60 [27:48<00:00, 8.04it/s, acc=12, epoch=156, loss=5.51]
<ipython-input-17-7620403e50cc>:22: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  val_bar = tqdm_notebook(desc='split=val',
split=val: 92% 11/12 [27:49<00:04, 4.62s/it, acc=11.9, epoch=156, loss=5.89]
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
QuoteGenerationModel                     [128, 500, 7813]          --
├─Embedding: 1-1                         [128, 500, 32]            250,016
├─Embedding: 1-2                         [128, 32]                 4,256
├─GRU: 1-3                               [128, 500, 32]            6,336
├─Linear: 1-4                            [64000, 7813]             257,829
==========================================================================================
Total params: 518,437
Trainable params: 518,437
Non-trainable params: 0
Total mult-adds (G): 16.94
==========================================================================================
Input size (MB): 0.51
Forward/backward pass size (MB): 4033.06
Params size (MB): 2.07
Estimated Total Size (MB): 4035.64
==========================================================================================
```

Como vemos nuestras "quotes " tienen una longitud máxima de 500 palabras, este límite hubo que establecerlo, dado que había un par de citas que excedían este número y hacían la red imposible de entrenar por falta de memoria. Además contamos con un vocabulario de 7813 tokens (palabras). Lo entrenamos con un batch size de 128, 32 embedding size, 500 epochs y 5 epochs de early stopping criteria. Con estos valores la red paró a los 156 epochs con los siguientes resultados.

```
Test loss: 5.865879058837891;
Test Accuracy: 11.587594619719516
```

Algunas de las posibles citas generadas se pueden ver en el documento de texto "Sample_GRU_quotes_156_ES_500max.txt". Las citas generadas no tienen la mayor coherencia ni personalización, probablemente debido a la falta de datos y lo básico de la red utilizada, ya que 10975 ejemplos de entrenamiento parecen muchos, pero son insuficientes para generar citas de la variedad y complejidad necesarias con los medios empleados.

BookNLP:

Uso de la librería

Hemos utilizado los ficheros generados por la librería en la anterior entrega para crear nuestros propios datasets, que están explicados en la sección de Actualización y Creación de Datasets. En los datasets de clasificación lo que se hace es recoger los modificadores (la librería entiende como modificador un adjetivo o entidad que es influyente en el personaje) de un personaje y crear conjuntos de 20 para intentar clasificar su personalidad. Con las menciones o "*quotes*", simplemente se recogen las que sean de dicho personaje y se unen; con el objetivo de generar nuevas menciones condicionando la generación por personalidad.

Guía de instalación

Para instalar y utilizar la librería en Windows tuvimos que seguir los siguientes pasos:

1. Descargar los modelos requeridos por la librería manualmente en huggingface, debido a un error en la gestión de ficheros de la librería. Esto se debe a que al ejecutar la pipeline para procesar todos los archivos, los modelos se descargan solos supuestamente, pero se introducen en una carpeta que se llama "C:\Users\{Usuario}\booknlp_models". Aparte de esta descarga automática, la librería necesita que crees la siguiente arquitectura de carpetas y ficheros bajo la carpeta "C:\Users\{Usuario}\booknlps" para poder funcionar.

2. Recopilar nuestros corpus en txt, y pasarlos por la "pipeline" de BookNLP con la configuración deseada. Esto se detalla en el fichero *test_booknlp_epub.ipynb.*

Entrega 4:

Objetivos:

Para esta entrega los objetivos se mantienen respecto a la entrega anterior. Es por ello que añadiremos nuevas filas a nuestra tabla para poder comparar el uso de técnicas de *machine learning avanzado* frente a transformers.

Planteamiento de los objetivos

En esta entrega, nuestro principal objetivo era integrar y experimentar con nuestras fuentes de datos utilizando transformers. Para esto, hemos dividido nuestra implementación en las dos tareas de las anteriores entregas: clasificación y generación de quotes.

Con respecto a la tarea de clasificación, primero hemos realizado una exploración y análisis de nuestras fuentes de datos, en la que hemos podido observar el desequilibrio en algunas de las clases de nuestros datasets (debido a la cantidad de registros que representan los protagonistas de las novelas analizadas y sus personalidades). Una vez extraídas dichas estadísticas, hemos implementado un sistema de predicción *multi-class* haciendo uso de dos arquitecturas transformer de Huggingface: DistilBERT y RoBERTa. El procedimiento a seguir con ambas ha sido el mismo: utilizar un modelo de clasificación tradicional (Regresión Logística) con las features extraídas (*hidden states*) usando el modelo transformer sin la "cabeza" de clasificación, y por otra parte, *fine-tunear* el modelo con nuestros datos utilizando dicha "cabeza" para clasificar las personalidades de los personajes. Finalmente, hemos extraído varias estadísticas de desempeño de los modelos, las cuales analizaremos en la siguiente sección.

Por otro lado, para la tarea de generación de quotes hemos seguido un enfoque similar: primero hemos obtenido una serie de características y visualizaciones sobre nuestra fuente de datos, como pueden ser la longitud de las citas, las palabras más frecuentes, etc... Una vez obtenidas, hemos adaptado nuestro dataset a la arquitectura de DistilGPT2, reduciendo la longitud de nuestras citas a algo asequible para el context length de DistilGPT2, y que nos permita agilizar el entrenamiento (citas de 128 tokens de longitud). Finalmente hemos aplicado finetunning al modelo, calculado su métrica de *perplexity* y obtenido algunas citas mediante inferencia. Cabe destacar que para influenciar el output del modelo en base al personaje que dice la cita, hemos optado por introducir dicho personaje como token al inicio de la sentencia, obteniendo resultados satisfactorios.

Desempeño de las soluciones:

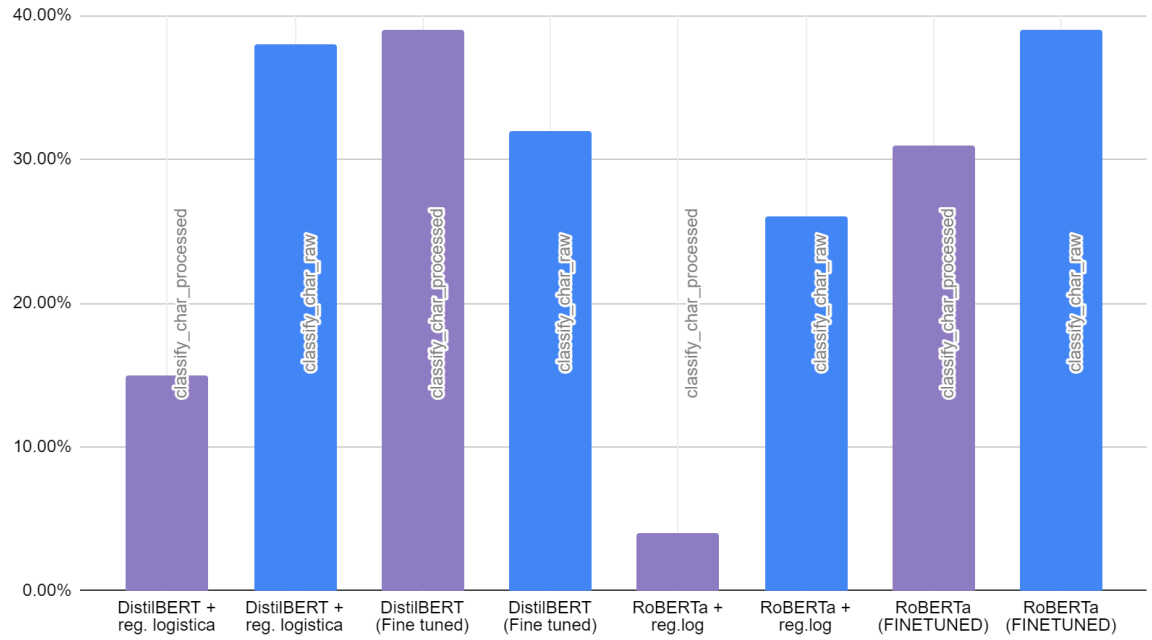
Problema 1: Clasificación de personalidades.

En primer lugar, vamos a comparar los transformers entre ellos antes de comparar todos los métodos de clasificación que hemos utilizado en este proyecto entre todos.
Ejecutamos cada modelo primero sin *finetuning* con las características extraídas de los transformers y haciendo uso del clasificador de regresión logística, pues fue el que mejor resultado nos dio en la fase anterior, y posteriormente haciendo *finetuning* de los modelos. Además de esto, por cada una de las combinaciones

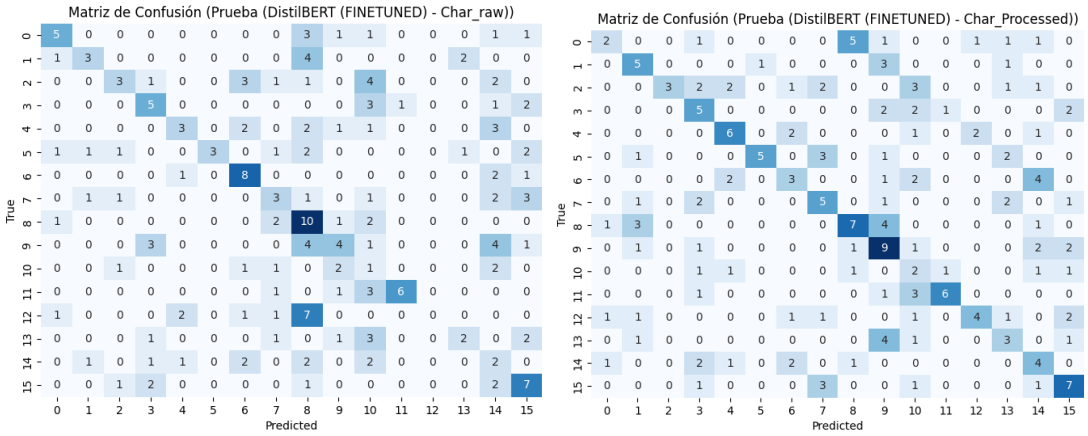
que acabamos de mencionar, probamos también la efectividad del modelo con el dataset "procesado" y sin "procesar" (es decir, sin modificadores repetidos y con modificadores repetidos). Los resultados son los siguientes:

Modelo	Dataset	Features	Accuracy
DistilBERT + reg. logistica	classify_char_processed	Hidden layer DistilBERT	15.00%
DistilBERT + reg. logistica	classify_char_raw	Hidden layer DistilBERT	38.00%
DistilBERT (Fine tuned)	classify_char_processed	Hidden layer DistilBERT	39.00%
DistilBERT (Fine tuned)	classify_char_raw	Hidden layer DistilBERT	32.00%
RoBERTa + reg.log	classify_char_processed	Hidden layer RoBERTa	4.00%
RoBERTa + reg.log	classify_char_raw	Hidden layer RoBERTa	26.00%
RoBERTa (FINETUNED)	classify_char_processed	Hidden layer RoBERTa	31.00%
RoBERTa (FINETUNED)	classify_char_raw	Hidden layer RoBERTa	39.00%

Para poder visualizarlo mejor tenemos el siguiente gráfico:

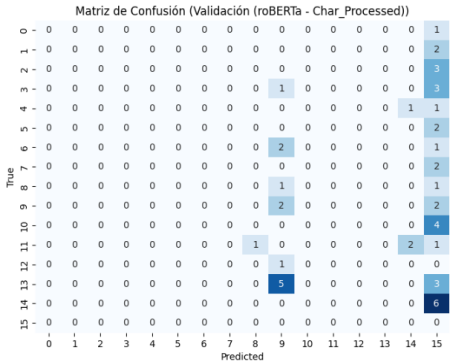


En él lo primero que nos salta la vista es el efecto de los datasets (expresado en el gráfico mediante el color: morado para processed y azul para raw). En todos los casos, menos en *distilBERT* con el dataset *processed* una vez hecho *finetuning*, los resultados son mejores con el raw..

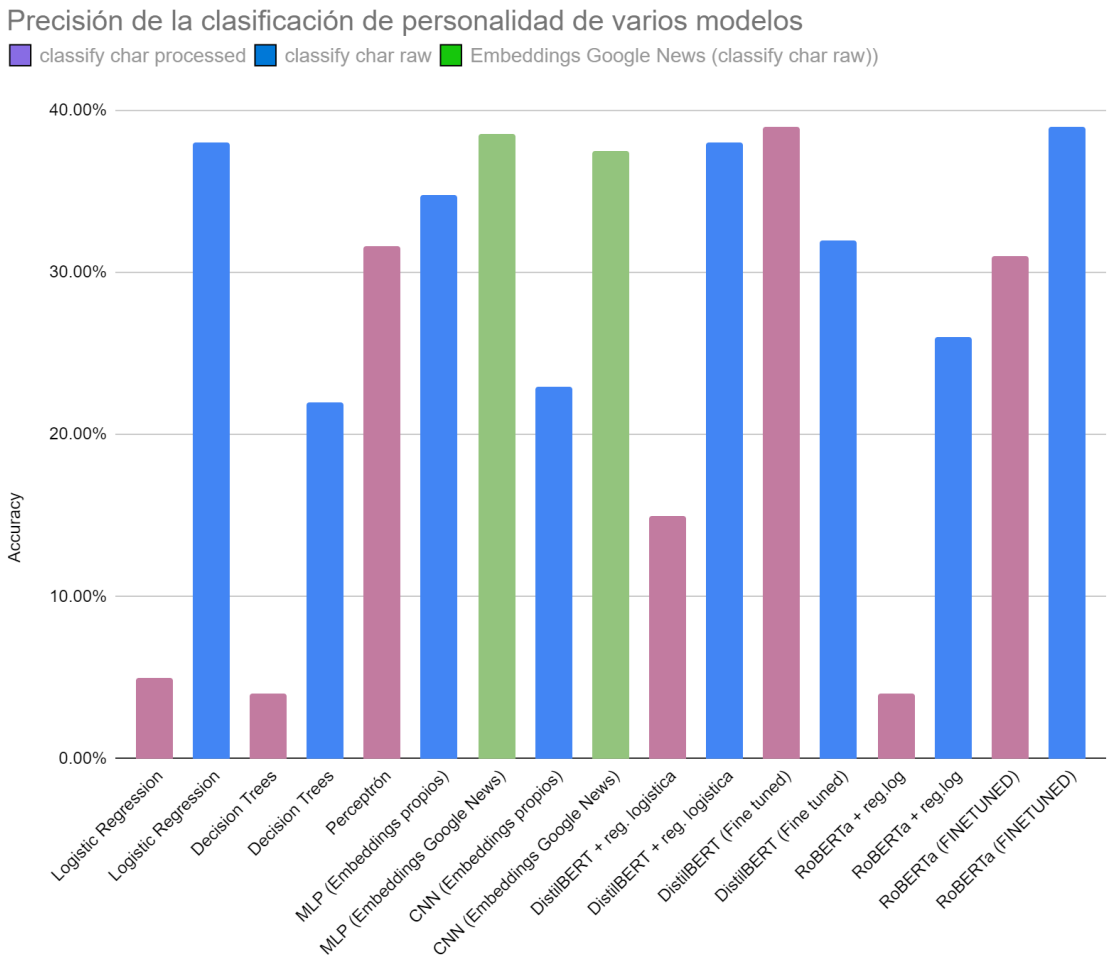


Si comparamos las matrices de confusión veremos que no son tán diferentes. Lo cierto es que la diferencia de precisión es de tan solo un 6% lo cual, teniendo en cuenta que nuestra tarea consiste en una clasificación de 16 clases tampoco es un gran error.

Otro caso destacable es el de *roBERTa + regresión logística* para el dataset *processed:* Su baja capacidad predictiva se explica fácilmente al observar su matriz de confusión ya que podemos ver que el regresor logístico ha predicho la mayoría de clases como la misma:



Si observamos el gráfico de barras nuevamente podremos observar que el efecto de hacer finetunning también tiene un efecto positivo muy notorio (a excepción de *distilBERT* con el dataset *raw,* cuya contraparte con regresor logístico lo hace mucho mejor. Es destacable mencionar que *distilBERT* con el dataset *raw* también lo hace peor que el regresor logístico con los feature TF-IDF que utilizamos en la entrega anterior).



Si comparamos la capacidad predictiva de los transformers con los mejores modelos tanto de *machine learning* avanzado como simple podremos observar que los transformers con *finetuning* son los mejores modelos predictivos de todos, seguidos de MLP con los *Embeddings* de google News y el regresor logístico con los features TF-IDF. Es decir, utilizar un transformer para obtener los features que luego daremos a un regresor logístico no mejora en nuestro caso la capacidad predictiva frente a un procesamiento más simple como es TF-IDF (Logistic Regression tiene una precisión de 38.00%, DistilBERT + reg. Logistica de 38.00% y RoBERTa + reg.log solo de 26.00%). Así nos quedaría la tabla utilizada para crear la visualización anterior ordenada de acuerdo a la precisión: Es mencionable destacar como salvo el caso mencionado anteriormente, los dataset raw logran los puestos más altos.

Model	Dataset	Accuracy
DistilBERT (Fine tuned)	classify_char_processed	39.00%
RoBERTa (FINETUNED)	classify_char_raw	39.00%
MLP (Embeddings Google News)	classify_char_raw	38.54%
Logistic Regression	classify_char_raw	38.00%
DistilBERT + reg. logistica	classify_char_raw	38.00%
CNN (Embeddings Google News)	classify_char_raw	37.50%
MLP (Embeddings propios)	classify_char_raw	34.75%
DistilBERT (Fine tuned)	classify_char_raw	32.00%
Perceptrón	classify_char_raw	31.60%
RoBERTa (FINETUNED)	classify_char_processed	31.00%
RoBERTa + reg.log	classify_char_raw	26.00%
CNN (Embeddings propios)	classify_char_raw	22.91%
Decision Trees	classify_char_raw	22.00%
DistilBERT + reg. logistica	classify_char_processed	15.00%
Logistic Regression	classify_char_processed	5.00%
Decision Trees	classify_char_processed	4.00%
RoBERTa + reg.log	classify_char_processed	4.00%

En azul: Deep learning, Verde: Machine Learning, Morado: Transformers

Finalmente, añadir que las matrices de confusión de los clasificadores con transformers pueden encontrarse en este documento, así como más estadísticas.

Por último concluir diciendo que pese a haber probado un gran conjunto de modelos predictivos y representaciones de datos, no hemos logrado una gran precisión. Esto se debe tanto a la dificultad de nuestra tarea como al hecho de que tenemos que clasificar correctamente 16 clases diferentes. No obstante, nuestro ejercicio si que tiene una conclusión clara que es ofrecer una comparación profunda entre distintos modelos de clasificación y representaciones de texto.

Problema 2: Generación de *quotes:*

El desempeño de *GPT* en comparación con el método utilizado en la entrega 2 es muy sorprendente. En nuestra primera iteración del la solución ya obtuvimos frases gramaticalmente correctas y ligeramente relacionadas con nuestros textos. Es por ello por lo que decidimos ir mas allá y tratar que el generador nos devolviese frases basandose en las que un personaje en concreto generaría. Es decir, quisimos que al pedirle

una frase de "Vin" nos la devolviese diferente a la que nos devolveria si le pidiesemops una de "Kelsier". Ademas cada una de las frases deberia estar relacionada con las que estos personajes dicen durante las novelas.
Tratamos de añadir el personaje como informacion adicional a la red, pero no logramos la forma de hacerlo. Es por ello por lo que decidimos, añadir al inicio de cada una de nuestras quotes, que iban a ser utilizadas para entrenar al modelo, un "token/indicador" de quién era el personaje que estaba hablando. De esta forma la red podria relacionar oraciones con personajes.

```
def preprocess_function(batch):
    return tokenizer([f"[{c}] {''.join(x)}" for x, c in
```

También tratamos de añadir un indicador más similar a lo que escribimos en prompts para redes como *chatGPT* y en vez de escribir "[*nombreDelPersonaje*] *oracion*" antes de cada frase escribimos: "You are *nombreDelPersonaje* and this is your quote: '*frase*'". En cualquier caso, el resultado es muy similar. Las oraciones si que son diferentes y relacionadas con cada uno de los personajes, lo cual es un exito. Pero, como hemos añadido el token a las frases con las que entrenamos la red, esto resulta en que las frases que generan la red también tienen el token en mitad de la frase. Puede ser que ese token en mitad de la frase signifique que en la oracion deberia ir el nombre del personaje y la red ha decidido poner el token en su lugar.

Conclusiones

Resumidamente, hemos obtenido una serie de modelos extrapolables los cuales son capaces de clasificar la personalidad de personajes ficticios en base al indicador Myers-Briggs, y también pueden generar una serie de frases o citas teniendo en cuenta el personaje que habla. Además hemos creado varias fuentes de datos que podrían ser de utilidad para futuros experimentos, haciendo uso de los recursos de la asignatura, así como de una librería externa especializada en el procesamiento del lenguaje, como puede ser BookNLP.

En definitiva, el enfoque alternativo que hemos utilizado para la obtención de nuestras fuentes de datos y la orientación de las tareas a seguir ha resultado satisfactorio. Hemos podido obtener unos resultados aceptables en base a la naturaleza de nuestras tareas, y la experiencia ha resultado ser a la vez desafiante y muy interesante.

Bibliografia

Info sobre las 16 personalidades:

- 16 Personalities: https://www.16personalities.com/es
- Truity: https://www.truity.com/
- Personality DataBase: https://www.personality-database.com/

Corpus de Mistborn:

- https://epdf.mx/mistborn-trilogy.html

BookNLP:

- Repositorio: https://github.com/booknlp/booknlp
- Introduction to BookNLP: https://booknlp.pythonhumanities.com/intro.html

Huggingface y Transformers:

- Text Classification: https://huggingface.co/docs/transformers/tasks/sequence_classification
- Causal Language Modeling: https://huggingface.co/docs/transformers/tasks/language_modeling