UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior de Informática

(TITLE)

Autor: María González Gutiérrez

Asignatura:

Grado en Ingeniería Informática

2020, (MONTH)

Índice

1. INTRODUCCIÓN Y OBJETIVOS

Empecé el proyecto porque me gusta el fanfiction y el análisis de datos, y AO3 tiene una base de datos muy grande y accesible. Además, las comunidades de fanfic son muy activas y producen una vasta cantidad de contenido muy detallado; son básicamente una gran discusión entre los fans de una obra sobre qué es lo que dicha obra significa para ellos, y sobretodo, qué es lo que a ellos les hubiese gustado llegar a ver realizado en el texto de la misma.

En literatura comparada se suelen tener en cuenta dos perspectivas a la hora de analizar una obra: la de la teoría del autor, que tiene en cuenta lo que el autor quería comunicar y plasmar en esa obra, y la de la "muerte del autor"[Bar68], que tiene en cuenta el mensaje con el que los lectores se quedan tras leer la obra (independientemente de si coincide con el que el autor quería comunicar). Para entender el mensaje de una obra de forma plena[Ell18], es necesario tener en cuenta tanto la intención comunicativa del autor, como el mensaje que al final los lectores acaban entendiendo. Y como cada lector es hijo de su padre y de su madre, acaban surgiendo muchas posibles interpretaciones distintas a partir de un único texto.

Tradicionalmente, los académicos solamente han tenido en cuenta las opiniones de un grupo reducido de personas (compuesto principalmente por otros académicos) a la hora de analizar una obra desde la perspectiva de la muerte del autor, ya que el lector común no suele tener a su disposición las herramientas necesarias para difundir sus interpretaciones. Sin embargo, desde que Internet y los foros como *LiveJournal* se han vuelto accesibles a grandes partes de la población, miles de comunidades fan empezaron a organizarse justamente con la intención de poner en común sus interpretaciones, de expresar sus críticas y opiniones. No todas estas discusiones tienen lugar en forma de fanfiction, pero es un género muy popular en las comunidades fans, y yo personalmente estoy muy familiarizada con sus estructuras y códigos.

Estas comunidades de Internet están generando una cantidad inmensa de opiniones y perspectivas en torno a un tema común en foros públicamente accesibles, y me pareció interesante la idea de crear un sistema que sea capaz de recoger y procesar toda esta información para crear un .ªbanico"de las distintas interpretaciones que existen en una comunidad fan, especialmente aquellas sobre los personajes y las relaciones entre ellos. El resultado final se podría utilizar como herramienta dentro de la propia comunidad fan, para observar cómo tienden a interpretar a ciertos personajes a nivel de comunidad y cómo estas interpretaciones cambian a lo largo del tiempo, o en distintas subsecciones dentro de la comunidad en general. También se podría

utilizar como herramienta general de análisis literario, aplicándola primero a la obra original y luego a un conjunto de fanfics representativos, y observando cuáles son las diferencias entre la perspectiva del autor original y la de los lectores (convertidos en autores fan).

Al empezar el proyecto, no sabía mucho sobre análisis de texto, por lo que empecé a estudiar sobre análisis de texto natural y extracción de información usando el libro *Natural Language Processing* (Jacob Eisenstein, MIT Press). Tras la fase de recogida y limpieza de datos (explicado en detalle en las secciones 4.1 y 4.2), el proceso de extracción de información que tenía que seguir consistía en:

1. Identificación de entidades
2. Identificación de relaciones entre entidades
3. Identificación de eventos

2. FANFICTION Y ARCHIVE OF OUR OWN

Fanfiction (del inglés *fan fiction*, "ficción del fan", y abreviado como "fanfic") es el nombre que recibe un texto basado en una historia ya existente (normalmente con copyright), en particular cuando el autor es fan de la obra de la cual su texto deriva. Son, por lo tanto, textos de ficción sin ánimo de lucro que los fans escriben como expresión de su creatividad.

El concepto detrás del fanfiction es, en esencia, una ausencia percibida en la historia original. Uno se termina un libro o un videojuego y siente que le falta algo: el pasado de un protagonista, una perspectiva distinta de un conflicto, una relación que acabó o nunca empezó, qué sucede después del final, o quizás que a la historia le hacían falta doscientas páginas más, o incluso que tendría que haber sido de un género literario distinto... Hay algo en la historia que está ausente. El lector se queda con ganas de explorar más a fondo el mundo y los personajes que el autor ha creado, y de aquí nace el impulso de crear historias propias en las que se exploran dichas ausencias. Por tanto, no es sorprendente descubrir que hay muchos fanfics en los que se cambia el destino de tal o cuál personaje, que exploran qué sucede tras el final, o que llevan a cabo exploraciones exhaustivas de los conflictos, los personajes y sus motivaciones desde perspectivas distintas a las de la obra original.

Todos estos motivos hacen que el fanfiction se considere una obra derivada[Swi98], y está en

su naturaleza el reflejar las opiniones y críticas que el autor tiene de la obra original: qué es lo que le gusta, qué temas siente que faltan en la obra, qué cosas tendrían que haberse explicado desde una perspectiva distinta, etc.

Por ejemplo, es evidente al leer los libros de la saga *Harry Potter* que el texto quiere que pienses que Ron Weasly, el mejor amigo del protagonista, es un chico un poco torpe y bocazas pero con buen corazón, y un buen amigo de Harry. Sin embargo, muchos fans no interpretaron a Ron como torpe y bocazas, sino como egocéntrico e insensible, y hay no pocos fanfics en los que Ron y Harry discuten y dejan de ser amigos, o en los que Ron es directamente un villano aliado con Voldemort.

Cuando los fans de una misma obra se reúnen y organizan, se crean comunidades fan llamadas "fandoms", que suelen crear foros donde intercambiar sus impresiones, teorías y, por supuesto, fanfiction y otras formas de arte fan. Es evidente que existe un intercambio de ideas en foros de discusión y otras comunidades online explícitamente creadas para conversar, pero ya que es totalmente posible inferir las opiniones de un autor a partir de sus fanfics, tanto escribir como leer fanfiction son actividades que contribuyen al discurso general del fandom, ayudando a popularizar algunas teorías y generando las suyas propias.

Cuando un fandom alcanza un cierto nivel de madurez, algunas teorías se consolidan y el fandom acaba formando, a nivel de comunidad, una interpretación propia de la obra original. Para distinguir la perspectiva del fandom de la que realmente pretende transmitir la obra original, en los fandoms se distingue entre el *fanom* y el canon. Siguiendo el ejemplo de *Harry Potter*, el Ron Weasly del *fanon* es una persona egoísta que sólo es amigo de Harry por interés, mientras que el Ron Weasly del canon tiene una amistad sincera con Harry. *Fanon*, por tanto, es el 'conjunto de teorías basadas en el material original que, aunque generalmente parecen ser la interpretación 'obvia' o 'única' de los hechos canónicos, no son realmente parte del canon'[Stu17].

En resumen, las comunidades fan tienen una interpretación propia de la obra original llamada *"fanon"*, que influencia los fanfics que los miembros de dicha comunidad van a escribir y, a su vez, los escritores de fanfic también crean y popularizan interpretaciones que se acaban convirtiendo en parte del *fanon*.

Como se ve en el ejemplo de *Harry Potter*, las relaciones entre personajes son una de las mayores fuentes de especulación entre los fans, especialmente las relaciones románticas. En general,

los personajes a los cuales los fans les tienen manía acaban convertidos en villanos (o, como mínimo, enemigo de los protagonistas) en los fanfics, incluso aunque en la obra original sean aliados. Naturalmente, lo mismo sucede a la inversa: los fans tienden a convertir en amigos y aliados a los personajes que les gustan, incluso aunque en la obra original sean los villanos de la historia. Por tanto, simplemente contrastando las relaciones presentes en un fanfic con las relaciones de la obra original podemos tener una buena idea de cuál es la interpretación del autor del fanfic.

Las relaciones románticas entre personajes son una parte enorme de la especulación fan. El romance es uno de los temas más populares, y no sólo es común interpretar a amigos como parejas, sino también a personajes que son enemigos. Un villano que es muy popular entre los fans tiene garantizado que tendrá fanfic en los que cambia de bando, convirtiéndose en aliado y pareja del protagonista (no necesariamente en ese orden).

Grafos de relaciones entre personajes de un fanfic y mismos personajes en la obra original. Harry Potter?

En términos legales y de derechos de autor, la mayoría de legislaciones considera el fanfiction como un tipo de obra derivada [Swi98] y por tanto entra dentro del *fair use*. A día de hoy, la mayoría de fanfics se publican en sitios web de toda índole, destacando entre ellos Archive Of Our Own, que es un archivo open-source y sin ánimo de lucro creado expresamente para alojar obras creadas por fans. Según sus datos de mayo de 2020, tiene más de dos millones de usuarios registrados y más de seis millones de trabajos alojados. En particular elegí descargar los fanfics basados en *Good Omens*, un libro de Terry Pratchett y Neil Gaiman, tanto por la cantidad de relatos existente como por mi familiaridad con esa comunidad.

Además de la cantidad y variedad de relatos que aloja, los motivos por el que elegí extraer los datos de Archive Of Our Own son su herramienta de búsqueda y filtrado, su sistema de etiquetas y el hecho de que permite descargar el archivo en HTML. Archive Of Our Own permite filtrar fanfics según varios parámetros y genera un enlace a ese subconjunto de relatos particular, muy útil para descargar una gran cantidad de datos.

por qué escogí AO3 sobre FF.net y

3. INTELIGENCIA ARTIFICIAL Y ANÁLISIS DE TEXTO

Lorem ipsum dolor sit amet

consectetur adipiscing elit

mirar intro del libro de eisenstein

explicar bag of words + función que calcula peso en función de vector características

4. RECOGIDA Y LIMPIEZA DE DATOS

4.1. CREANDO UN SCRAPER PARA ARCHIVE OF OUR OWN

En el momento en el que decidí utilizar los fanfics de *Good Omens* para el proyecto, dicho libro tenía unos 22000 fanfics en Archive Of Our Own (AO3 para abreviar). Sin embargo, de todos esos relatos sólo me interesaban los que están en inglés y los que realmente contuvieran texto (puesto que, aunque AO3 se centra en texto, permite todo tipo de archivos multimedia), de modo que utilicé el sistema de filtrado del sitio para seleccionar sólo los fanfics en inglés y que no estuvieran etiquetadas como "fanart", "podfic", etc, ya que indican que la obra no tiene texto. Tras este filtrado, quedaron 20190 relatos.

literaturi esta sección

AO3 crea un enlace que lleva siempre a ese subconjunto de relatos, al cual puedo enviar peticiones HTTP GET y navegar las páginas. Cada página contiene un máximo de 20 fanfics, y antes de descargarlos es necesario llevar a cabo un segundo filtrado para evitar las obras sin texto que no hayan sido etiquetadas como tal por su autor. Decidí crear dos *scrapers* distintos, uno para navegar por las páginas, filtrar y guardar los *links* en un archivo, y otro que simplemente lee dicho archivo y realiza descargas.

- El primer *scraper* envía peticiones a AO3, explora el HTML de cada página para encontrar los objetos que encapsulan la información de cada fanfic, selecciona aquellos que tengan al menos 40 palabras por capítulo y crea una lista con los enlaces, que guarda en un archivo *txt*.
- El segundo *scraper* recorre los enlaces de dicho archivo y los descarga en una carpeta del sistema. Está hecho de tal manera que se le puede indicar qué tramo de la lista tiene que descargar (por ejemplo, del número 5000 al 6000).

Ambos *scrapers* se encontraban de vez en cuando con el error 429 (Too Many Requests) y el 404 (Page Not Found). Se capturan fácilmente con un *try-catch* que detecta el status de la página; para el 429 simplemente lanza una espera de dos minutos tras la cual reanuda su ejecución por donde la dejó, y para el 404 simplemente pasa a la siguiente, pues este error indica que el autor ha borrado su obra de la página y ya no está accesible.

Decidí dividir este proceso en dos programas (el de navegación y filtrado, y el que únicamente descarga) en vez de hacerlo todo en uno porque descargar los más de 20000 archivos de una vez lógicamente tardaría varias horas, y pensé que sería más pragmático si ejecuto una vez el *scraper* crea una lista de enlaces, y luego ya ejecuto todas las veces que sean necesarias el *scraper* que descarga, descargando cada vez un tramo distinto de la lista. De esta forma, pude descargar todos los fanfics en grupos de 2000 (alrededor de una hora y media), pudiendo tener el ordenador libre el resto del tiempo.

El poder ejecutar varias veces el segundo *scraper* sin tener que volver a empezar desde el principio también demostró ser útil para lidiar con errores de red.

Ambos *scrapers* utilizan la librería bs4 y BeautifulSoup para descargar y manejar los archivos HTTP. El resultado de su ejecución es una carpeta con 818,8 MB de archivos HTML

4.2. LIMPIEZA DE DATOS

Los fanfics descargados en HTML vienen con metadatos relacionados con la historia, el autor y cuándo fueron publicados, entre otros. Además, la mayoría de fanfics tienen notas del autor, comentarios y sinopsis que, aunque forman parte del cuerpo del texto, no son relevantes para el análisis que voy a realizar. Me interesa poder recuperar fácilmente los metadatos útiles de cada fanfic, además de convertirlo a texto plano eliminando todos los metadatos y los comentarios del autor, por lo que decidí reunir todas estas funciones en dos clases, FanficCleaner y FanficHTMLHandler, encapsuladas en un programa fanfic_util.py

- FanficCleaner tiene métodos para extraer el cuerpo del texto de un archivo HTML, sin metadatos, comentarios ni notas del autor. También tiene la función de guardar el texto en un archivo txt en un *path* a elegir.
- FanficHTMLHandler tiene métodos que permiten extraer metadatos del archivo HTML de un fanfic, como por ejemplo los personajes principales, el número de capítulos y su clasificación.

Debido a que los archivos HTML están organizados en carpetas, utilizo listas con sus *path* para identificarlos y acceder a ellos. Los métodos de FanficCleaner y FanficHTML reciben tramos de estas listas para acceder y manejar los archivos deseados. Para ello, el programa utiliza las librerías BeautifulSoup y html2text.

5. EXTRACCIÓN DE DATOS A PARTIR DE TEXTO

Explicar los programas que hice para familiarizarme con NLTK

5.1. ALGORITMO DE IDENTIFICACIÓN DE ENTIDADES

En la identificación de entidades, se considera una entidad a los personajes, los lugares y las instituciones, entre otras cosas, que haya sido nombrada en el texto. Un algoritmo capaz de identificar entidades nombradas tiene que poder dividir un texto en tramos y asignarle una etiqueta de entidad ("Persona", "País", etc) a cada uno. Esta tarea además requiere que las palabras del texto hayan sido previamente etiquetadas con su rol morfológico.

Por estos motivos, la librería NLTK parecía la más idónea para la tarea. Es una librería de python que contiene herramientas básicas para el análisis de texto, y en particular me interesaba que venía con un *part of speech tagger* (es decir, un identificador de rol morfológico) ya programado y entrenado. NLTK también viene con un identificador de entidades ya entrenado, pero quería programar uno que fuera más preciso y adaptado a mi conjunto de datos.

Además del identificador de rol morfológico, NLTK también tiene una clase llamada *ChunkParser* cuyo trabajo es dividir un texto en tramos. Todas las funciones de la librería que se encargan de dividir y/o etiquetar texto (como el identificador de rol morfológico) heredan de alguna versión de la clase *ChunkParser*, de modo que la idea para el algoritmo era modificar la clase *ChunkParserI* para convertirla en un identificador de secuencias basado en características.

citar ref al sitio web en cuyo código me he basado

Un identificador de secuencias basado en características trata de asignar un peso a un tramo concreto, y según el peso, le asigna una etiqueta u otra. Este peso se calcula como una función de las características del propio tramo, así como de los tramos que le preceden. El programador puede elegir las características que considere más importantes, pero hay algunas que son bien conocidas como las más importantes para reconocer entidades, como:

- El rol morfológico de la palabra actual, las anteriores y las siguientes.
- La forma de la palabra, las anteriores y las siguientes (si empiezan por mayúscula, si tienen signos de puntuación, si son siglas, etc)
- Los prefijos y/o sufijos de la palabra actual, las anteriores y las siguientes.
- Si la palabra anterior ha sido identificada como una entidad o no.

El conjunto de características de cada tramo se llama vector de características, y se utiliza para

calcular un "peso"que se corresponde con la probabilidad de que un tramo X con un vector de características V tenga una etiqueta Y. El algoritmo al final asigna a cada tramo la etiqueta cuyo peso sea el más alto.

El cómo se calcula exactamente ese peso depende del modelo matemático a utilizar. A la versión modificada de ChunkParserI para la identificación de entidades la llame NERChunker (NER por Named Entitiy Recognition), y tiene tres versiones:

- NERChunkerv1 y NERChunkerv3 utilizan un modelo de regresión logística (también llamado modelo de entropía máxima), a través de la clase MaxentClassifier de NLTK. Para que NLTK pueda utilizar esta clase correctamente, es necesario tener instalado el módulo Megam para python, que no viene incluído en NLTK. La única diferencia entre la versión 1 y la 3 de este chunker es que la 3 maneja las estructuras de NLTK para oraciones y etiquetas de forma ligeramente más rápida.
- NERChunkerv2, que utiliza un modelo de naïve Bayes a través de la clase ClassifierBasedTagger de NLTK.

Las versiones *v1* y *v3* de *NERChunker* obtuvieron los mejores resultados en la evaluación, y la *v3* es algo más rápida, por lo que es la versión definitiva del identificador de entidades. Todas estas versiones, junto con sus funciones auxiliares, se encuentran encapsuladas en el archivo *NERChunkers.py*, para ser utilizadas donde se las necesite.

Evaluaciones de las versiones de NERChunker y el NER de NLTK

Puesto que tanto los clasificadores de regresión logística como los de *naïve* Bayes son algoritmos de aprendizaje supervisado, antes de poder utilizar (o evaluar) cualquiera de las versiones de *NERChunker* era necesario entrenarlas con un conjunto de datos ya etiquetados. El problema aquí es que NLTK, a pesar de incluir un corpus muy extenso en la propia librería, sólo tiene dos conjuntos de datos para identificación de entidades: uno en español y el otro en holandés. Todos los textos a analizar en el proyecto están en inglés, obligándome a buscar un conjunto ajeno a NLTK y finalmente decidiéndome por *Groningen Meaning Bank* (GMB). GMB es un *dataset* para identificación de entidades específicamente en inglés, grande, con una gran variedad de etiquetas de entidad y, sobretodo, con un formato de etiquetado sencillo de entender,cosa

importante puesto que al ser ajeno a NLTK, GMB utiliza etiquetas distintas que son necesario adaptar para que *MaxentClassifier* pueda trabajar con ellas.

GMB utiliza la notación IOB para etiquetar entidades, y separa cada palabra de la siguiente por un carácter de nueva línea, y cada frase, por dos. De modo que la frase *"Mr. Blair left for Turkey Friday from Brussels."* en GMB tendrá el aspecto de la **??**

Mr.	NNP	B-PER
Blair	NNP	B-PER
left	VBD	O
for	IN	O
Turkey	NNP	B-GEO
Friday	NNP	B-TIM
from	IN	I-TIM
Brussels	NNP	I-TIM
.	.	O

Figura 1: Frase etiquetada por GMB. De izquierda a derecha, las columnas representan la palabra a etiquetar, la etiqueta de rol morfológico, y la etiqueta IOB.

Cuando el programa detecta una entidad de tipo persona, etiqueta como 'B-PER' la primera palabra de la secuencia, mientras que el resto de palabras dentro de la secuencia son etiquetadas como 'I-PER'. Similarmente, si la entidad es de tipo geográfico las etiquetas usadas serán 'B-GEO' y 'I-GEO', si es de tiempo serán 'B-TIM' y 'I-TIM', etc. Si una palabra no forma parte de ninguna secuencia de entidad, se etiqueta como 'O'.

NLTK, por su parte, no utiliza la notación IOB ni caracteres de nueva línea, sino que utiliza una estructura de datos propia de tipo árbol que encapsula cada palabra y cada tramo con su etiqueta. La misma frase etiquetada por NTLK tiene el aspecto de la **??**

Como se ve, en vez de usar etiquetas IOB, NLTK organiza las palabras y su etiquetas en una estructura de árbol. La raíz, S, indica el inicio de la frase (Sentence), y las etiquetas de entidad son nodos.

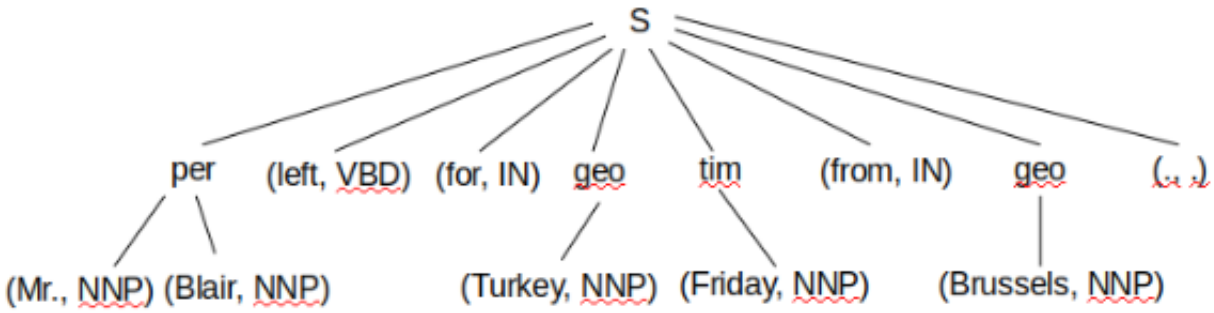


Figura 2: Frase etiquetada por NLTK. Las etiquetas de entidad se encuentran en los nodos, encontrándose todas las palabras pertenecientes a una secuencia de entidad en la profundidad 2 del árbol. Las palabras que no pertenecen a ninguna secuencia de entidad se encuentran en la profundidad 1. Cada hoja del árbol contiene una tupla formada por la palabra y su etiqueta de rol morfológico.

En horizontal queda así:

(S, [(per, [(‘Mr.’, NNP), (‘Blair’, NNP)]), (‘left’, VBD), (‘for’, IN), (geo, [(‘Turkey’, NNP)]), (tim, [(‘Friday’, NNP)]), (‘from’, IN), (geo, [(‘Brussels’, NNP)]), (‘.’,.)])

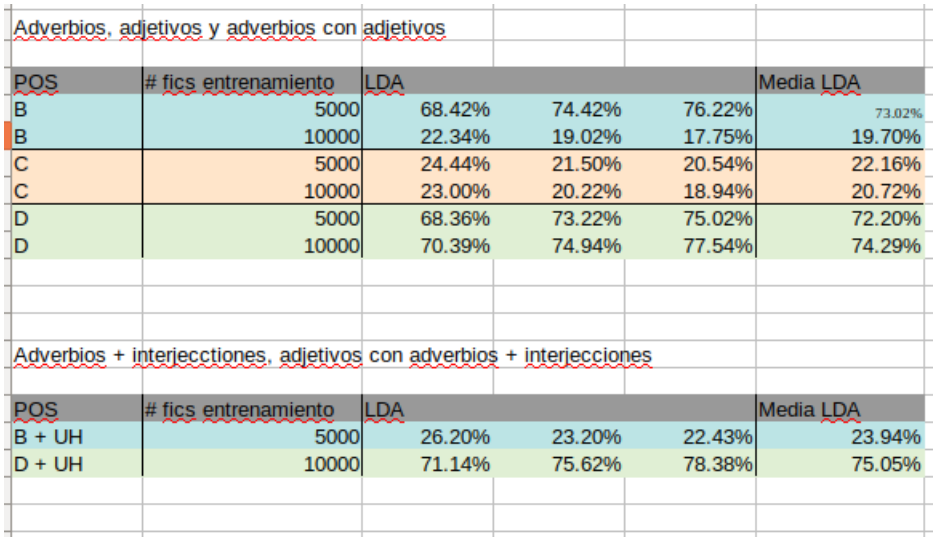
Fue más o menos a estas alturas del proyecto cuando decidí separar el proceso de entrenar el identificador de entidades y el de utilizarlo para etiquetar texto nuevo en dos programas distintos (NERTrainer y NERTagger, respectivamente). Acceder a los textos de GMB y transformar sus etiquetas a un formato que NLTK pueda entender y acceder a los textos de la base de datos de fanfics y preprocesarlos para su posterior etiquetado mediante el programa ya entrenado han resultado serdos procesos muy distintos, y dividirlo parecía la mejor manera de tener un código limpio y claro.

listar en alguna parte las librerías: BeautifulSoup, pandas

5.2. ALGORITMO DE IDENTIFICACIÓN DE RELACIONES

Para este algoritmo decidí crear un algoritmo LDA, que es un algoritmo de aprendizaje no supervisado que se utiliza típicamente en identificación de temas. Para entrenar este algoritmo, seleccioné un conjunto de fanfics que sirviesen de modelo para la relación que quería que aprendiera a detectar. Por ejemplo, para entrenar un modelo LDA que detecte situaciones sexuales, preparé un conjunto de fanfics de un único capítulo en los que sucedía sexo, excluyendo otros que tenían escenas sexuales pero eran sólo una pequeña parte de una historia más larga.

Figura 3: Porcentaje de aciertos de cada modelo. Cada prueba se realizó tres veces.



Adverbios, adjetivos y adverbios con adjetivos

POS	# fics entrenamiento	LDA			Media LDA
B	5000	68.42%	74.42%	76.22%	73.02%
B	10000	22.34%	19.02%	17.75%	19.70%
C	5000	24.44%	21.50%	20.54%	22.16%
C	10000	23.00%	20.22%	18.94%	20.72%
D	5000	68.36%	73.22%	75.02%	72.20%
D	10000	70.39%	74.94%	77.54%	74.29%

Adverbios + interjecctiones, adjetivos con adverbios + interjecciones

POS	# fics entrenamiento	LDA			Media LDA
B + UH	5000	26.20%	23.20%	22.43%	23.94%
D + UH	10000	71.14%	75.62%	78.38%	75.05%

Para buscar el modelo LDA más eficiente, creé tres modelos que tenían en cuenta diferentes categorías morfológicas. Primero utilizaba el *tagger* de NLTK para identificar categorías morfológicas, y el algoritmo sólo tenía en cuenta aquellas que resultaran relevantes.

- El modelo B tenía en cuenta sustantivos, adverbios y verbos.
- El modelo C tenía en cuenta sustantivos, adjetivos y verbos.
- El modelo D tenía en cuenta sustantivos, adverbios, adjetivos y verbos.

Además de utilizar distintas categorías morfológicas, también probé distintos tamaños para el set de entrenamiento, de modo que cada modelo tiene dos versiones: una entrenada con 5000 fanfics y otra entrenada con 10000.

Para la evaluación de los modelos, simplemente los puse a clasificar textos nuevos, contando la cantidad de aciertos de cada uno y calculando el *hit ratio* de cada uno. Los resultados aparecen en la primera tabla de la figura 3.

$$hit\_ratio = \frac{correct\_guesses}{total\_number\_of\_guesses}$$

Las primeras pruebas mostraron que los modelos B y D eran los que arrojaban mejores resultados. Observando qué otras categorías morfológicas el *tagger* de NLTK puede identificar, pensé que añadir interjecciones a los modelos podría aumentar su precisión. Llamé B+UH y D+UH a los modelos resultantes, y repetí las pruebas. Los resultados están en la segunda tabla de la figura 3.

Curiosamente, el modelo D fue mejorado ligeramente teniendo en cuenta las interjecciones, pero el modelo B empeoró considerablemente.

6. EVALUACIÓN DEL SISTEMA

7. REFERENCIAS

Referencias

[Bar68] Ronald Barthes. La mort de l'auteur. *Manteia*, (5), 1968.

[Ell18] Lindsay Ellis. Death of the author. https://www.youtube.com/watch?v=MGn9x4-Y_7A, December 2018. Youtube.

[Stu17] Alasdair Stuart. Dean winchester and commander shepard walk into a bar: Why fanon matters. *Uncanny Magazine*, July/August 2017.

[Swi98] Jonathan Swift. Copyright 101: A brief introduction to copyright for fan fiction writers. http://www.whoosh.org/issue25/lee1a.html#41, October 1998. Woosh Magazine, Birthplace of the International Association of Xena Studies.