

UNIVERSIDAD DE CASTILLA-LA MANCHA ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

**Identificación automática de personajes en textos de ficción
pertenecientes al género fanfiction**

María González Gutiérrez

Febrero, 2021

UNIVERSIDAD DE CASTILLA-LA MANCHA

Departamento de Tecnologías y Sistemas de Información

GRADO EN INGENIERÍA INFORMÁTICA

Computación

**Identificación automática de personajes en textos de ficción
pertenecientes al género fanfiction**

Autor: María González Gutiérrez

Tutor académico: José Ángel Olivas Varela

Febrero, 2021

Índice

1. PÁGINA DE CALIFICACIÓN	4
2. RESUMEN	5
3. ABSTRACT	6
4. AGRADECIMIENTOS	7
5. INTRODUCCIÓN Y OBJETIVOS	8
6. FANFICTION Y ARCHIVE OF OUR OWN	12
7. EXTRACCIÓN DE INFORMACIÓN EN OTROS TRABAJOS	15
8. RECOGIDA Y LIMPIEZA DE DATOS	17
8.1. Creando un scraper para Archive of our Own	17
8.2. Limpieza de datos y creación de datasets	25
9. EXTRACCIÓN DE DATOS A PARTIR DE TEXTO	29
9.1. Algoritmo de identificación de entidades	29
9.1.1. Extracción de entidades con NLTK	29
9.1.2. Extracción de entidades con CoreNLP	34
9.2. Algoritmo de identificación de relaciones	39
9.2.1. Primeras estrategias: Clustering y LDA	41

9.2.2. Correferencia con CoreNLP	45
10. PROGRAMA PRINCIPAL: fic_character_extractor	52
11. EVALUACIÓN DEL SISTEMA	54
11.1. Prueba 1: Funciones básicas del programa con un texto largo (Fanfic 9)	54
11.2. Prueba 2: Género distinto del canon (Fanfic 2856)	56
11.3. Prueba 3: Personajes que no son nombrados (Fanfic 2163)	57
12. CONCLUSIONES	62
12.1. Trabajos futuros	66
A. ANEXO: CÓDIGO DEL SISTEMA DE SCRAPERS	67
A.1. link_scraper	67
A.2. file_scraper	70
B. ANEXO: CÓDIGO DE fanfic_util	74
C. ANEXO: CÓDIGO DEL ALGORITMO DE IDENTIFICACIÓN DE ENTIDADES BASADO EN REGRESIÓN LOGÍSTICA	83
C.1. NER_chunker	83
C.2. NER_trainer	87
C.3. NER_tagger	89
D. ANEXO: CÓDIGO QUE MANEJA CoreNLP	92
D.1. corenlp_wrapper	92

D.2. corenlp_util	96
E. ANEXO: PROGRAMAS DE PRUEBA PARA ALGORITMO DE EXTRACCIÓN DE RELACIONES	109
E.1. toy_relex_kmeans	109
E.2. toy_relex_topic	114
E.3. toy_relex_v2	118
E.4. ner_and_sen_extraction_v2	123
F. ANEXO: PROGRAMA FINAL	135
G. REFERENCIAS	140

1. PÁGINA DE CALIFICACIÓN

TRIBUNAL:

- Presidente:
- Vocal:
- Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE

VOCAL

SECRETARIO

2. RESUMEN

La extracción de información es una tarea consistente en identificar las entidades presentes en un texto y qué relaciones las unen. En este trabajo se aborda una posible aplicación de este proceso en obras literarias, en particular las pertenecientes al género fanfiction, que se caracteriza por que sus autores no crean obras originales: un fanfic es un relato que toma prestados los personajes o la historia de una obra ya existente. La finalidad de este proyecto pues es crear una herramienta que pueda resultar de utilidad a la hora de realizar análisis literarios sobre los temas, conflictos y carencias que los fans de una obra perciben en la misma.

La primera parte del proyecto consiste en descargar una parte de los relatos de la web [Archive of Our Own](#), con el objetivo de crear un corpus básico de textos que utilizar para pruebas y entrenamientos. Esto se realiza mediante un sistema de scrapers, y además se desarrollan las herramientas necesarias para organizar y manejar los archivos del corpus para que los algoritmos de procesamiento de texto puedan entender su contenido. Dichos algoritmos utilizarán tanto el texto en sí como los metadatos. Para ello se explora la estructura de la página web y se echa mano de librerías de python como *BeautifulSoup*, *HTML2Text* y *requests*.

La siguiente parte del proyecto es el procesado de texto natural. Utilizando NLTK, se entrena un modelo de regresión logística con el corpus *Groningen Meaning Bank*, dando como resultado un algoritmo de identificación de entidades nombradas que es capaz de encontrar los nombres de los personajes de un texto y contar cuántas veces se les menciona. Para desarrollar un algoritmo de identificación de relaciones se utilizan librerías como *sci-kit learn*, *gensim* y *Stanza* para explorar diferentes técnicas con el fin de hallar relaciones sociales entre personajes, incluyendo clustering, un modelo LDA y explotando la correferencia pronominal en frases en las que un personaje es mencionado para buscar patrones, pero no se llega a ninguna conclusión satisfactoria. Sin embargo, el uso de CoreNLP (mediante *Stanza*) en estas pruebas resulta en el desarrollo de otro algoritmo de identificación de personajes que tiene las ventajas añadidas de ser capaz de identificar el género del personaje y hallar más menciones que el modelo de regresión logística.

Por último, ambos algoritmos de identificación de personajes llevan a cabo una tarea de canonicalización, que en este proyecto es una tarea definida por las características del género fanfiction: puesto que estos relatos no son originales, sino que están basados en personajes ya existentes, estos algoritmos identificarán cuáles personajes se encontraban en la obra original y cuáles son originales del autor fan. Para reducir el alcance de esta tarea, todos los relatos descargados de [Archive of Our Own](#) están basados en el libro *Good Omens* (Neil Gaiman y Terry Pratchett, 1990).

3. ABSTRACT

Information extraction is the task of recognizing the entities present in a text, and which relationships exist between them. This project attempts an application of this task on literary works, in particular to those belonging the genre of fanfiction, which is defined by not being original works: a fanfic is a story that lends the characters or the plot from an already existing work. The purpose of this project is to create a tool that helps in the literary analysis of the themes, conflicts and oversights that the fans of a particular work find on it.

The first part of the project consists in downloading texts from the website [Archive of Our Own](#), thus creating a basic corpus of stories to be used in tests and training. This is achieved with a scraper system, and developing tools to organise and handle the corpus' files so that the text processing algorithms can understand their contents. Said algorithms will use both the pure text and the metadata of the files. The structure of the website is studied in order to create the scrapers, and python libraries like *BeautifulSoup*, *HTML2Text* and *requests* are used.

The next part of the project is the natural text processing. A named entity recognition algorithm is developed using NLTK, by training a logistic regression model on the *Groningen Meaning Bank* corpus. The result is an algorithm that can recognize a character's name in a text, and count how many times it is mentioned. Several strategies are explored in order to find an algorithm for recognising relationships, and python libraries like *sci-kit learn*, *gensim* and *Stanza* are used to create clustering algorithms, a LDA model and to exploit pronominal coreference in sentences where characters are mentioned in order to find patterns that may signify a relationship, but none of these attempts are successful. However, the use of CoreNLP via *Stanza* ends up creating another named entity recognition algorithm which has the ability of identifying the gender of a character, and can find more mentions than the logistic regression model.

At last, both named entity recognition algorithms perform a task of canonicalization, which in this project is defined by the features of the fanfiction genre: since these stories are not original, but instead they are based in already existing characters, these algorithms identify which characters belong to the original work, and which are created by the fan writer. In order to reduce the scope of this task, all the downloaded works from [Archive of Our Own](#) are based on the book *Good Omens* (Neil Gaiman and Terry Pratchett, 1990).

4. AGRADECIMIENTOS

Esta etapa de mi vida ha sido larga y muy dura para mí, y nunca la habría visto acabar así de bien sin la ayuda de mucha gente. En primer lugar mis padres y mi hermana, que han estado todo este tiempo conmigo y teniendo fe en mis esfuerzos cuando yo no tenía ninguna. También ha habido muchos profesores que me han animado y dado oportunidades, y en especial quiero mencionar a José Ángel Olivas Varela, que siempre me ha apoyado en mis esfuerzos y me ha ayudado cuando lo he necesitado.

También quiero mencionar a mis amigos, fuera y dentro de la universidad, que me han hecho compañía, escuchado mis problemas y levantado el ánimo cuando yo no podía. Nunca podría haber terminado este proyecto si no hubieran estado ahí para tomar cafés, oírme quejarme y descansar de nuestros problemas del día a día, incluso cuando este año hemos estado físicamente separados y los cafés han tenido que ser con la pantalla en medio.

Este trabajo fin de estudios es resultado de la combinación del apoyo y los ánimos de todas estas personas, que me dieron cariño y confianza cuando más los necesitaba. Mis esfuerzos nacen de su compañía.

5. INTRODUCCIÓN Y OBJETIVOS

En este proyecto se desarrollará un sistema para la identificación y extracción de personajes y relaciones en textos de ficción, en particular aquellos pertenecientes al género fanfiction. Consistirá, por un lado, en realizar un scrape parcial de la página web [Archive of our Own](#) para obtener un corpus de textos con los que entrenar y hacer pruebas, y por otro lado se desarrollarán los algoritmos encargados de la identificación de entidades y las relaciones entre ellas. De esta manera se pretende crear una herramienta que ayude en análisis literarios, en particular aquellos que se centren en investigar cuáles son las interpretaciones que los lectores fuera el mundo académico encuentran en una obra particular.

Esta herramienta para análisis literario estará basada en los personajes del texto, y las relaciones entre ellos. Al fin y al cabo, es útil para un académico comprobar cuáles son los personajes que más captan la atención de los lectores, si son los protagonistas o si optan por desarrollar personajes menores a los que el autor original no prestó tanta atención, y también puede aprender mucho de qué relaciones y a qué personajes involucran: cuando un gran número de lectores elige enfrentar a personajes que en la obra original son amigos, o crear amistades y romances entre personajes que son enemigos (o que directamente ni se conocen), un académico puede sacar conclusiones sobre qué encuentran atractivo o repulsivo de un personaje, o qué temas quieren resaltar y explorar mediante el desarrollo de estas relaciones.

Por tanto, es una tarea adecuada para la extracción de información a partir de texto natural. El análisis de lenguaje natural tiene como objetivo ahorrar el trabajo humano de resumir y sacar conclusiones de grandes conjuntos de textos, creando programas capaces de procesar información no estructurada y dotándoles de la habilidad de entender el significado del lenguaje humano. En este proyecto se pretende utilizar esta técnica para ahorrar a un académico el tener que leerse manualmente cientos de relatos para concluir cuáles son los temas, conflictos y personajes que le interesan a una comunidad de escritores en particular.

En primer lugar se va a crear un corpus de relatos pertenecientes al género fanfiction a partir de los textos alojados en la web gratuita [Archive of our Own](#); este corpus será útil para realizar pruebas, entrenamiento y experimentos que ayudarán en el desarrollo de un sistema de procesamiento de texto natural que extraerá personajes nombrados de un texto y qué relaciones sociales les unen, utilizando el libro *Natural Language Processing*, de Jacob Eisenstein ([Eisenstein, 2018](#)) como guía general del proceso.

El proceso de extracción de información que el proyecto va a seguir consistirá en identificar primero las entidades que se corresponden con los personajes del texto, y después identificará las relaciones entre dichos personajes. La identificación de eventos en un texto es también parte habitual del proceso de

extracción de información, pero decidí centrarme sólo en entidades y relaciones para acotar el problema. La intención es aplicar estos dos procesos a un conjunto de textos pertenecientes al género fanfiction para identificar qué personajes aparecen en el mismo y qué relaciones los unen, así como otros datos relevantes al campo del fanfiction como si el personaje aparece en la obra original, o si es un añadido del autor fan. El usuario podrá utilizar esta información para sacar conclusiones sobre la interpretación del autor sobre los personajes de la obra original.

Los textos serán ser extraídos directamente de AO3 utilizando un scraper, en una fase de recogida y limpieza de datos explicada en las secciones 8.1 y 8.2. Por tanto, los **objetivos de este proyecto** consisten en:

1. Extraer un conjunto de relatos alojados en [Archive of Our Own](#) sobre los que utilizar técnicas de extracción de información.
2. Crear módulos y herramientas para manejar y extraer información de los archivos HTML de AO3.
3. Organizar estos archivos de alguna forma y procesarlos para que sean comprensibles para los algoritmos de análisis de texto.
4. Desarrollar un algoritmo que identifique personajes en textos de ficción.
5. Desarrollar un algoritmo que identifique relaciones de carácter social entre dichos personajes.
6. Crear un programa que utilice ambos algoritmos para extraer los personajes y relaciones en los relatos recogidos de AO3, y mostrarlas al usuario.

Para consultar cualquier aspecto del código e implementación del proyecto, se puede visitar el repositorio del mismo en [GitHub](#) (Usuario: *mariaGnlz*, repositorio: *Fanfic_ontology*).

La idea para empezar el proyecto surgió de la cantidad de información que existe en internet sobre fenómenos culturales como *Los Vengadores* o *Harry Potter*, historias que claramente resuenan con mucha gente y que mueven a los fans a reunirse y crear contenido en comunidades digitales. Estas comunidades de fans suelen ser muy activas y producen una vasta cantidad de contenido muy rico en detalle; 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. A veces, estas comunidades crean enormes proyectos de calidad profesional de forma totalmente gratuita, simplemente para mejorar el espacio y las experiencias del resto de miembros. Uno de los mayores ejemplos de este tipo de 'trabajo fan' es la [Organizaton for Transformative Works](#), una organización sin ánimo de lucro creado 'por fans y para fans' que crea y mantiene proyectos como [FanLore](#), una wiki sobre

la cultura fan, o su comité legal, que se encarga tanto de educar sobre leyes de propiedad intelectual y el *fair use* del mismo como de involucrarse en los procesos jurídicos sobre copyright de diversos gobiernos (especialmente Estados Unidos) para defender el derecho del público general a crear obras derivadas.

Uno de sus proyectos más famosos es [Archive of our Own](#), comúnmente acortado a AO3, un sitio web que aloja principalmente relatos pertenecientes al género fanfiction y cuyo objetivo es, por un lado, facilitar la tarea de encontrar fanfiction para aquellos que lo quieran leer y, por otro, funcionar como un archivo que clasifique y documente el fenómeno fanfic a nivel global. En sus datos de mayo de 2020, aparecen más de dos millones de usuarios registrados, y más de seis millones de trabajos alojados. AO3 se ha convertido en una parte fundamental de la cultura fan, especialmente la dedicada a la escritura, y en este proyecto lo utilizaré como fuente de información.

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' ([Barthes, 1968](#)), 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 ([Ellis, 2018](#)), 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 volvieron 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).

6. 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 (Swift, 1998), 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 Weasley, 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

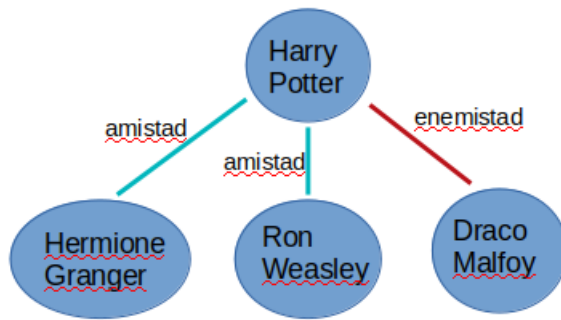
formando, a nivel de comunidad, una interpretación propia de la obra original. Para distinguir la perspectiva del fandom de la que realmente pretende transmitir la obra original, en los fandoms se distingue entre el *fanon* y el canon. Siguiendo el ejemplo de *Harry Potter*, el Ron Weasley del *fanon* es una persona egoísta que sólo es amigo de Harry por interés, mientras que el Ron Weasley 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' (Stuart, 2017).

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*.

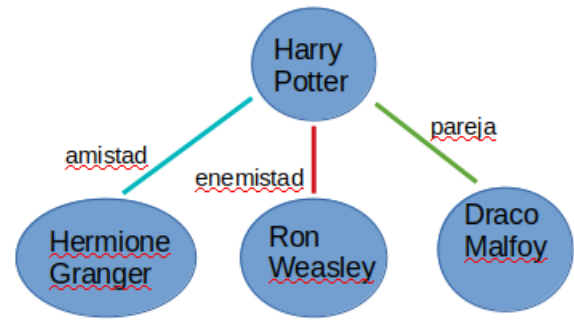
Como se ve en el ejemplo de *Harry Potter*, las relaciones entre personajes son una de las mayores fuentes de especulación entre los fans, especialmente las relaciones románticas. En general, los personajes a los cuales los fans les tienen manía acaban convertidos en villanos (o, como mínimo, enemigo 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 aunque las relaciones canónicas atraen naturalmente la atención de muchos fans, 'inventar' parejas en el *fanon* no sólo es común, sino una de las principales actividades de un fandom. Los fans ven parejas y conflictos amorosos tanto entre amigos como enemigos, y son felices de ignorar todos y cualquiera de los obstáculos que existan en el canon con tal de tener el escenario necesario para que su pareja preferida pueda estar junta, llegando incluso al extremo de sacar a los personajes del universo al que pertenecen para meterlos en otro más amistoso. Un villano que es muy popular entre los fans tiene garantizados fanfics en los que cambia de bando, convirtiéndose en aliado y pareja del protagonista (no necesariamente en ese orden).

Como se ha dicho anteriormente, los fans se organizan en comunidades según la obra que es el objeto de su admiración, y tienen sitios como AO3, dedicados a alojar y compartir sus creaciones fan. Evidentemente, analizar las más de seis millones de obras existentes en AO3 es una tarea imposible con los recursos a mi alcance, de modo que elegí utilizar únicamente los fanfics basados en *Good Omens*, un libro de Terry Pratchett y Neil Gaiman, en parte por mi familiaridad con esa comunidad, pero también porque tenía una cantidad de fanfics extensa pero manejable.



A) Grafo de relaciones entre personajes en los libros de la saga *Harry Potter*



B) Grafo de relaciones entre personajes que se encuentra fácilmente en los fanfics de *Harry Potter*

AO3 no es el único sitio web popular para alojar fanfiction, pero a lo largo de esta década ha desplazado a sitios como [Fanfiction.net](https://www.fanfiction.net) y [Wattpad](https://www.wattpad.com), y la característica que condicionó mi decisión es su extensivo sistema de etiquetas, que como se verá más adelante fue fundamental para filtrar y gestionar los datos del proyecto.

En términos legales y de derechos de autor, la mayoría de legislaciones considera el fanfiction como un tipo de obra derivada (Swift, 1998) y por tanto entra dentro del *fair use*.

7. EXTRACCIÓN DE INFORMACIÓN EN OTROS TRABAJOS

La extracción de información es un problema que se aborda en el análisis de lenguaje natural, cuyo objetivo es ahorrar el trabajo humano de resumir y sacar conclusiones de grandes conjuntos de textos. Para lograrlo, se crean programas capaces de procesar información no estructurada, dotándoles de la habilidad de entender el significado del lenguaje humano para que resuman y saquen conclusiones a partir de grandes volúmenes de textos ya existentes y, por ejemplo, usen esa información para crear una base de datos de conocimiento automáticamente, sin tener que dedicar grandes cantidades de tiempo y esfuerzo a leerlos manualmente. Es especialmente útil en campos como la medicina y la química, para los cuales se han creado distintos modelos (Craven and Kumlien, 1999)(Manica and Auer, 2019) con el objetivo de ayudar a procesar las ingentes cantidades de estudios y artículos que se publican hoy en día.

El primero de los pasos para extraer información de un texto es identificar las entidades nombradas en el mismo. Algunos sistemas, como CoreNLP (Finkel et al., 2005), utilizan un clasificador estadístico (en particular, un campo aleatorio condicional (Lafferty et al., 2001)) para identificar las palabras que denotan un nombre, lugar u organización. La desventaja de éstos métodos es que requieren ser entrenados con datos previamente etiquetados y listas de nombres bien conocidos (nomenclátor), por lo que también se han buscado alternativas no-supervisadas (Nothman et al., 2013) y semi-supervisadas (Lin and Wu, 2009), pero las estrategias que echan mano del *machine learning* y grandes corpus de texto etiquetado para entrenar clasificadores siguen siendo las más populares. Existen por tanto muchas soluciones distintas, algunas dotadas de extractores de características muy sofisticados compuestos por varios algoritmos entrenados para realizar tareas específicas, como capturar el contexto o incluso canonicalizar entidades (Wick et al., 2009).

La extracción de relaciones tradicionalmente se ha realizado tratando de identificar relaciones binarias, bien frase a frase o teniendo en cuenta dos o tres frases consecutivas (Zelenko et al., 2003)(Craven and Kumlien, 1999). Las limitaciones de este sistema no son sólo que no todos los tipos de relaciones son binarias, si no que cuanto más largo es un texto, menos probable es que ambas entidades aparezcan nombradas en una única frase, con lo que se pierde mucha información si se ignoran. Pasar a un sistema que tenga en cuenta varias frases interdependientes viene con sus propios problemas, pues las características sintácticas de una relación son muy dispersas en el texto y tratar de extraerlas automáticamente requiere de mucha memoria y muchos datos; más cuántas más frases tengas en cuenta. Sin embargo, recientemente se han creado algoritmos que identifican una relación n-aria a partir de todas las menciones en las que aparece, como el de Peng et al (Peng et al., 2017), que utiliza grafos LSTM para modelar el contexto e información de cada frase y las conexiones entre ellas, logrando así beneficiarse de la interdependencia de distintas frases sin tanto coste de recursos.

Este proyecto sin embargo busca identificar relaciones de naturaleza social entre distintos personajes de ficción, con lo que se centrará en identificar entidades del tipo 'Persona', y buscará relaciones binarias entre ellas.

Aunque el fanfiction no ha sido muy utilizado para realizar tareas de extracción de información, su disponibilidad online y la cantidad de metadatos asociados a cada obra ha hecho que algunos programadores los aprovechen para crear sistemas de recomendación, como [FanRecs](#), aunque el principal interés parece residir en crear herramientas de descarga ([FicLab](#), [FanfictionDownloader](#)).

8. RECOGIDA Y LIMPIEZA DE DATOS

8.1. 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 relatos, permite alojar todo tipo de archivos multimedia).

Por suerte, AO3 fue creado con la intención específica de funcionar como archivo, por lo que tiene una herramienta de búsqueda y filtrado muy completa y sencilla de usar. Esta herramienta permite filtrar por características como título, autor, idioma y cantidad de palabras, pero su mayor utilidad viene de su sistema de etiquetado. AO3 permite a los autores añadir tantas etiquetas como quieran para que los posibles lectores puedan saber más de su obra a simple vista: temática, personajes principales, parejas en las que se centra, qué medio utiliza, si hay ilustraciones, si trata sobre un evento de la historia original particular... Las etiquetas añaden una gran cantidad de información sobre las historias a las que acompañan, y aunque no es obligatorio poner ninguna, en general los autores se preocupan de etiquetar correctamente sus obras.

AO3 tiene etiquetas específicas para indicar que una obra no es principalmente texto: '*Fanart*', para ilustraciones, y '*Podfic*' para archivos de audio, así que aproveché la herramienta de búsqueda para llevar a cabo un primer filtrado que eliminara todas las obras que las contuvieran, además de todas las que no estuviesen en inglés. El resultado fue un subconjunto de 20190 fanfics, todos en inglés y cuyos autores no habían incluido ninguna etiqueta que indicara que no fuera puro texto. La herramienta además genera un link permanente que siempre lleva a este subconjunto particular, por lo que no es necesario utilizar esta herramienta nada más que una vez.

Una vez localizado el conjunto de textos y el link a los mismos, viene la parte de crear el *scraper* en sí. Utilizando la herramienta de inspeccionar elemento de *Firefox* para explorar la estructura del sitio, y enseguida se hizo obvio que los fanfics estaban organizados en páginas con un máximo de 20 fanfics cada una. En el HTML de la página, cada fanfic se presenta dentro de una clase llamada '*work blurb group*'. No se puede extraer un link de descarga directamente de ésta clase, pero sí el identificador del fanfic.

Exclude ?

- Ratings
- Warnings
- Categories
- Fandoms
- Characters
- Relationships
- Additional Tags

Other tags to exclude

Fanart ☒

Podfic ☒

More Options

- Crossovers
- Completion Status
- Word Count
- Date Updated

Search within results ?

Language

English ▼

Sort and Filter

Figura 1: Herramienta de filtrado de AO3. Permite excluir (o incluir) obras que contengan etiquetas específicas, así como aquellas no escritas en un idioma particular

En AO3, cada fanfic tiene un número que lo identifica de forma única. Es posible acceder a la página de cualquier fanfic simplemente añadiendo ese número al final de 'https://www.archiveofourown.org/works/' en la barra de direcciones, y en esa página sí que se pueden encontrar links de descarga. Por tanto la idea básica para el *scraper* es utilizar las librerías *requests* y *BeautifulSoup* de python para explorar los veinte '*work blurb group*' de cada página, localizar el identificador de cada uno, utilizarlo para acceder a la página del fanfic y extraer el link de descarga. Y así con cada página del listado, hasta llegar a la última. La figura 2 ilustra el proceso con un esquema.

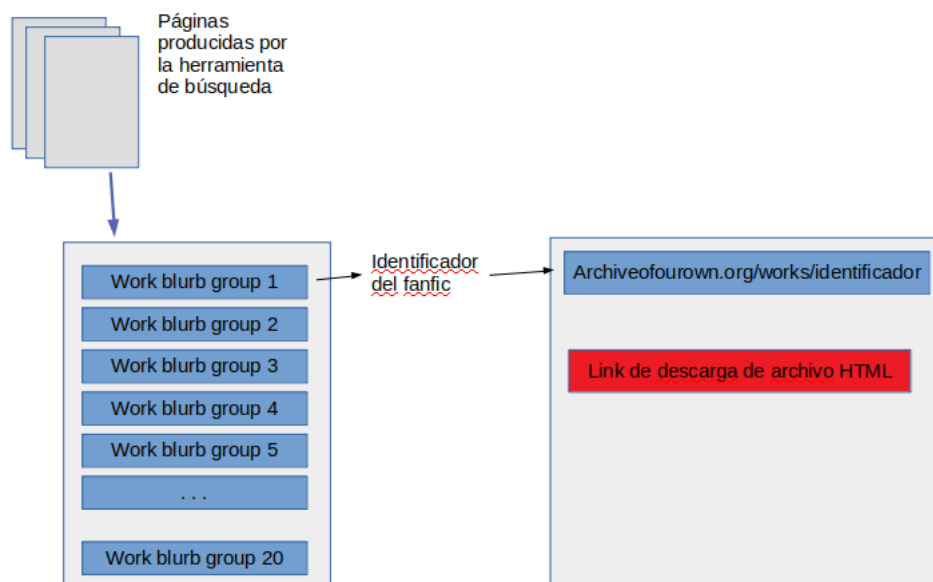


Figura 2: Concepto para el *scraper*. El objetivo es obtener los links de descarga navegando las páginas de búsqueda.

El proceso de descarga de archivos, en principio, tendría estos pasos:

1. Enviar una petición HTTP GET al link permanente del conjunto de datos, generado por la herramienta de búsqueda de AO3.
2. Iterar entre los 20 '*work blurb group*' y extraer el identificador de cada uno.
3. Usar el identificador para acceder a la página de cada fanfic, extraer el link de descarga de la página, y descargar el fanfic como archivo HTML. Hacer esto con los 20 identificadores.
4. Pasar a la siguiente página y repetir, hasta llegar a la última.

Utilizando la librería *requests* de python, el primer paso es trivial, y se puede observar en la figura 5. Encontrar los identificadores tampoco es complicado. Se puede apreciar en 2 que el identificador del fanfic también es el ID del objeto '*work blurb group*' al que pertenece, y expandiendo la clase se puede ver que el identificador completo se puede encontrar dentro del objeto, como un objeto de tipo *h4*. Por tanto, usando *BeautifulSoup* para manejar los datos resultantes de la petición HTTP GET como objeto HTML, se pueden obtener todos los objetos '*work blurb group*' usando la función *find(class_=<nombre clase>)*, cuyo resultado es una lista con los 20 objetos, sobre los cuales se itera para encontrar los identificadores usando de nuevo la función *find()*. En la figura 4 se puede observar un fragmento del código que realiza este trabajo; el código completo se puede consultar en A.

Las complicaciones empiezan una vez se tienen los identificadores. Para formar la dirección completa,

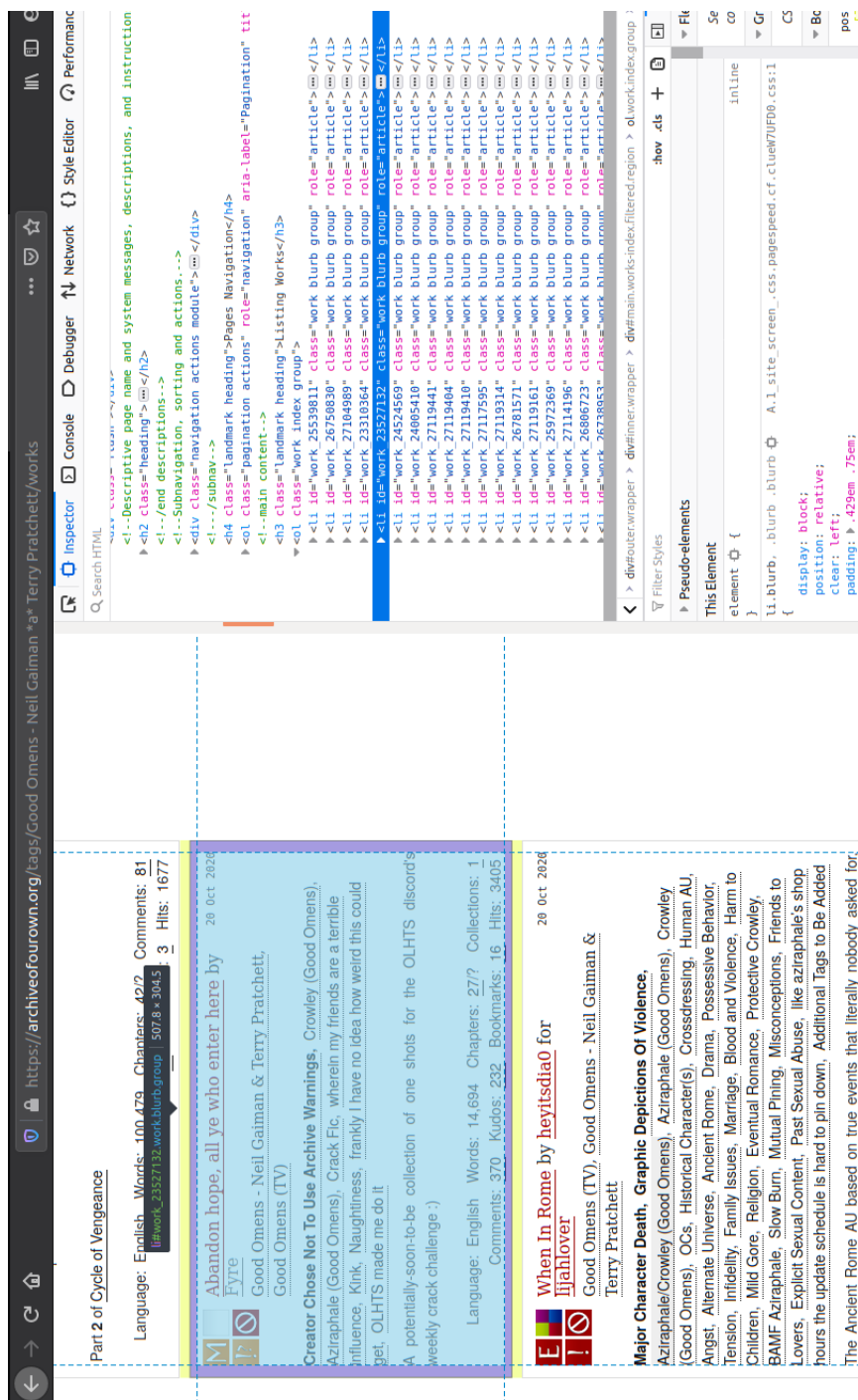


Figura 3: Exploración de la estructura de la página AO3 usando la herramienta 'Inspeccionar elemento' de Firefox. Se puede ver que el sitio utiliza una clase HTML llamada 'work blurb group' para mostrar cada obra.

```

40     current_page = 1
41     while current_page < number_of_pages:
42         blurbs = soup.find_all(class_='work blurb group')
43         #print('current page: ',current_page) #debug
44
45         for blurb in blurbs:
46             #filter out fics that don't contain text
47             contains_text = check_for_text(blurb)
48
49             work_id = (blurb.find('h4')).find('a')
50             if contains_text: work_links.append('https://archiveofourown.org'+work_id['href'])
51             else:
52                 discarded_links.append('https://archiveofourown.org'+work_id['href'])
53                 #print('out:', work_id['href'])
54         #end 'for blurb' loop
55
56         current_page +=1
57         next_page_link = page_link.replace('&page=1&', '&page='+str(current_page)+'&')
58         while True: #wait out if too many requests
59             page = requests.get(next_page_link)
60
61             if page.status_code == 429: #Too Many Requests
62                 print('Sleeping...')
63                 time.sleep(120)
64                 print('Woke up')
65
66             else: break
67
68         soup = BeautifulSoup(page.content, 'html.parser')
69
70     #end while loop

```

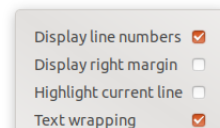


Figura 4: Código perteneciente al *scraper* 'ao3_link_scraper'. Utiliza un bucle *while* para iterar entre las páginas de la búsqueda, y en cada página, usa la librería *BeautifulSoup* para extraer los objetos 'work blurb group' en una lista llamada 'blurbs' (línea 42). De cada 'blurb' extrae el identificador del fanfic y comprueba si tiene texto (líneas 46-49), y si lo contiene forma el enlace a la página del fanfic y lo añade a una lista llamada 'work_links' (línea 50). Si no contiene texto, se añade a otra lista llamada 'discarded_links' (línea 52).

```

29 def get_work_links(page_link):
30     page = requests.get(page_link) #get first page of the archive
31     soup = BeautifulSoup(page.content, 'html.parser')
32
33     #figure out how many pages in total there are
34     page_list = (soup.find(class_='pagination actions')).find_all('li')
35     number_of_pages = int(page_list[len(page_list)-2].text) #there are number_of_pages pages in total
36

```

Figura 5: Código perteneciente al *scraper* 'ao3_link_scraper'. Utiliza la librería *requests* para enviar una petición HTTP GET al link permanente del conjunto de datos (línea 30), y *BeautifulSoup* para navegar el resultado como un objeto HTML del que poder extraer datos útiles, como la cantidad total de páginas (líneas 33-35).

hay que añadir el identificador al final de 'https://www.archiveofourown.org', mandar otra petición HTTP GET a dicha dirección, buscar ahí el link de descarga, solicitarla, esperar a que la descarga termine, y repetir todo esto otras 19 veces hasta tener descargados todos los fanfics de la página. Esto significa que por cada iteración del bucle que explora cada página es necesario introducir otro bucle que haga las descargas.

La última parte, la de pasar a la página siguiente, es más complicada de explicar que de ejecutar. Todas las páginas de resultados de búsqueda de AO3 contienen botones para avanzar, retroceder y saltar a páginas concretas. Es posible saber cuántas páginas en total tiene la búsqueda simplemente observando el texto del botón de la última, tal y como se ve en la figura 7. No se aprecia, pero la clase HTML a la que pertenece dicho botón se llama 'pagination actions', y es posible extraerla gracias a la función

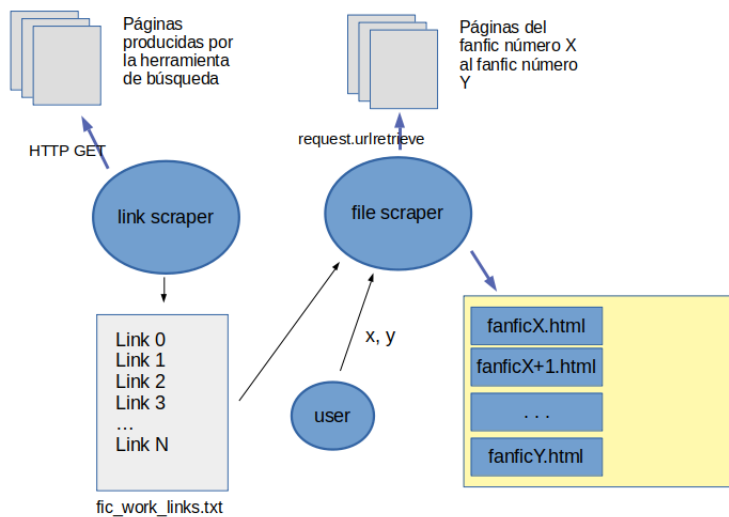


Figura 6: Proceso de descarga de fanfics de AO3 utilizando los programas 'ao3_link_scraper.py' y 'ao3_file_scraper.py'.

find(class_=<nombre_clase>) de *BeautifulSoup*. Y ya con ese objeto, se puede volver a utilizar la función *find()* para buscar todos los objetos hijos de la clase '*pagination actions*' que sean de tipo *li*. El último será el que contenga la cantidad total de páginas, y solicitar la siguiente consiste simplemente en sustituir la referencia en el link a la página 1 por una referencia a la última página. En la figura 5 se ve parte del código que realiza este proceso; el código completo se puede consultar en el anexo A.

Es evidente que la parte de solicitar las descargas en un bucle anidado ralentiza el programa, enturbia el código y además, hace que sea complicado parar o interrumpir el programa si hay algún error de red, pues para reanudar la ejecución por donde se quedó sería necesario almacenar en alguna parte el número de página por el que iba y el número del fanfic dentro de esa página, y programar los bucles para que salten directamente a la iteración deseada.

Ninguna de estas cosas me convenía, ya que descargar más de 20000 archivos ya iba a ser lento de por sí y hacerlo de una sentada sería prácticamente imposible, de modo que decidí dividir el programa en dos: uno que llamé '*link scraper*' y otro '*file scraper*'.

El *link scraper* se ejecutaría una vez y exploraría todas las páginas de búsqueda, extrayendo los links a los fanfics de cada una, y los almacenaría en un archivo de texto. Por tanto, al terminar su ejecución este *scraper* ha generado un archivo llamado '*fic_work_links.txt*' que almacena los enlaces a cada fanfic. El *file scraper* utiliza esta lista para saber dónde buscar las descargas, y el usuario le indica en la línea de comando los índices que acotan el tramo de la lista a descargar, tal y como se ilustra en el esquema de la figura 6. De este modo, es posible indicarle al programa que descargue desde el link 0 al link 1000 de la lista, permitiendo descargar los 20000 archivos en porciones manejables. Además, el programa anuncia en pantalla qué link está siendo descargado en cada momento, por lo que si sucede un error de

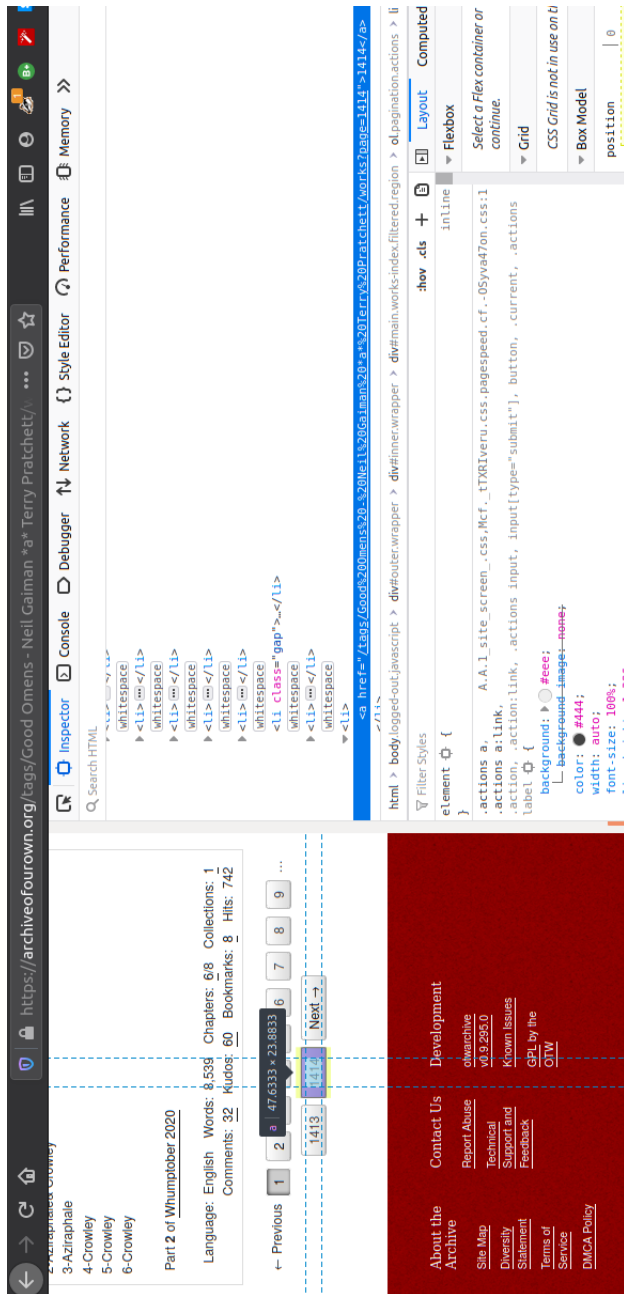


Figura 7: Navegación de páginas de búsqueda de AO3. Todos los botones vienen con su número de página, y se puede ver cuál es la última

red mientras descargaba el link número 866, es posible reanudar el programa fácilmente e indicarle que continúe desde el 866 al 1000.

Esta división del trabajo en dos programas además me daba la oportunidad de introducir con sencillez un segundo filtrado durante el proceso de exploración que realiza el link *scraper*. Si el primer filtrado se encargaba de cribar los fanfics que habían sido etiquetados por sus autores como imágenes o audio, este segundo filtrado pretende detectar los fanfics que tampoco contienen texto, pero no han sido etiquetados como tal por sus autores. Para ello usé el criterio de la relación palabras/capítulo de cada fanfic: si una obra tiene menos de 40 palabras por capítulo, se considera como fanfic "sin texto", y se elimina. Escogí 40 palabras como umbral tras investigar un poco con la herramienta de búsqueda de AO3, que como se puede ver en la figura 1, tiene una opción para filtrar por cantidad total de palabras. Tras probar varios umbrales, 40 parecía ser el que descartaba todas las obras sin texto sin sacrificar muchos microrrelatos en el proceso.

Introducir este filtrado en el *scraper* fue sencillo, puesto que el número de palabras y capítulos de la obra es información que se puede extraer de la clase *work blurb group* de cada fic. Todo esto se realiza desde la función *check_for_text*, y en la figura 4 se puede ver cómo el bucle llama a dicha función; el código completo se puede consultar en A. Por tanto, el link *scraper* realiza estos pasos:

1. Enviar una petición HTTP GET mediante la librería *requests* al link permanente del conjunto de datos, generado por la herramienta de búsqueda de AO3.
2. Iterar entre los 20 '*work blurb group*', comprobar si contienen texto, y descartar los identificadores de los que no.
3. Utilizar cada identificador para generar el link de la página de cada fanfic y almacenarlos en un archivo de texto.
4. Pasar a la siguiente página y repetir, hasta llegar a la última.

Por su parte, el *file scraper* realiza estos pasos:

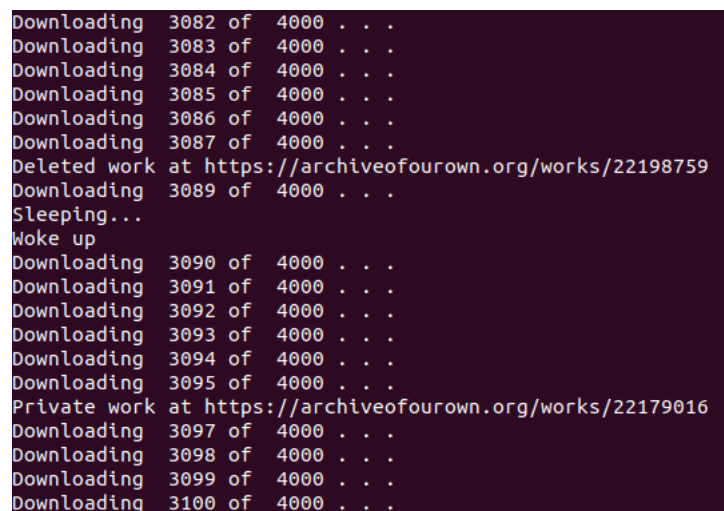
1. Abrir el archivo *fic_work_links.txt* y extraer la lista de links.
2. Mediante la librería *requests*, realizar una petición HTTP GET al primer link, saltando al siguiente si devuelve un código 404.
3. Extraer el link de descarga HTML de cada página.

4. Solicitar la descarga mediante *request.urlretrieve*. Guardar el archivo resultante en la carpeta adecuada en el sistema.
5. Repetir con todos los links de la lista.

El manejo del código de error 404 (Page Not Found) es bastante importante en este *scraper*, puesto que entre el momento en el que se almacenó el link del fanfic mediante el primer *scraper* y el momento en el que el segundo *scraper* lo utiliza para la descarga pueden haber pasado varios días. En ese tiempo, el autor del fanfic puede haber decidido borrar el fanfic de AO3, o haberlo hecho privado, y de ahí que el *scraper* reciba un 404. Un simple *try-catch* detecta el código 404 y simplemente pasa al siguiente link, como se puede consultar en el anexo A.

El otro error que ambos *scrapers* necesitaban manejar es, naturalmente, el error 429 (Too Many Requests). En las líneas 58-66 de la figura 4 se puede ver cómo se utiliza un *try-catch* que envuelve la petición HTTP GET para detectar el status 429 y, en vez de pasar al siguiente link, se lanza una espera de dos minutos tras la cual vuelve a solicitar la página. Antes de incorporar este código a los *scrapers* creé un pequeño programa de prueba, para ver cuánto tardaba AO3 en enviar un 429 y cuánto tiempo de espera requería antes de volver a aceptar solicitudes.

El resultado de la ejecución de estos *scrapers* es una carpeta con 818,8 MB de archivos HTML.



```
Downloading 3082 of 4000 . . .
Downloading 3083 of 4000 . . .
Downloading 3084 of 4000 . . .
Downloading 3085 of 4000 . . .
Downloading 3086 of 4000 . . .
Downloading 3087 of 4000 . . .
Deleted work at https://archiveofourown.org/works/22198759
Downloading 3089 of 4000 . . .
Sleeping...
Woke up
Downloading 3090 of 4000 . . .
Downloading 3091 of 4000 . . .
Downloading 3092 of 4000 . . .
Downloading 3093 of 4000 . . .
Downloading 3094 of 4000 . . .
Downloading 3095 of 4000 . . .
Private work at https://archiveofourown.org/works/22179016
Downloading 3097 of 4000 . . .
Downloading 3098 of 4000 . . .
Downloading 3099 of 4000 . . .
Downloading 3100 of 4000 . . .
```

Figura 8: Ejecución de *ao3_file_scraper.py*

8.2. Limpieza de datos y creación de datasets

Al terminar el proceso de descarga, acabé con un conjunto de archivos HTML y un archivo TXT con una lista de los *path* de todos ellos.

La principal tarea de limpieza de datos es, por tanto, convertir los archivos HTML a texto. Para ello utilicé

la librería *HTML2Text*, que como cuyo nombre dice sirve para eso mismo. Sin embargo, los fanfics en HTML no contienen sólo el texto del fic en sí, sino que también contienen todos los metadatos del mismo: etiquetas, *rating*, resumen, comentarios del autor, entre otros, con lo que tras limpiar el archivo HTML con *HTML2Text* el resultado no era el texto puro del fanfic. Tuve que crear varias funciones y ayudarme de *Beautiful Soup* para limpiar todos estos metadatos y dejar únicamente el texto en sí, sin llevarme por delante parte del texto ni dejarme notas del autor entre capítulos.

Al principio puse estas funciones dentro de cada programa que necesitaba manejar los textos, pero obviamente enseguida se volvió muy aparatoso, por lo que consolidé todas las funciones de limpieza en un único archivo, *fanfic_util.py*, junto con tres clases para encapsular el uso de estas funciones:

- *FanficGetter*, que se encarga de proveer el texto limpio de un fanfic (o lista de fanfics) a demanda. Al principio devolvía los textos como una lista de *string*, pero luego resultó más útil que devolviera una lista de objetos *Fanfic* que pudiera devolver los capítulos por separado o juntos en un mismo *string*, además del identificador del fanfic.
- *FanficHTMLHandler* se encarga de extraer información de los metadatos del archivo HTML de un fanfic, como por ejemplo los personajes principales, las relaciones, las etiquetas, el número de capítulos y su clasificación.
- *Fanfic*, una clase que se utiliza a lo largo del proyecto para encapsular toda la información relevante sobre un fanfic, como sus capítulos, sus personajes, etiquetas, el dataset al que pertenece e incluso los objetos *Document* generados por CoreNLP.

La creación de la clase *Fanfic* surgió a mediados del proyecto, cuando el límite de 100000 caracteres de *CoreNLP* 9.1.2 hizo necesario poder acceder al texto de cada fanfic dividido en capítulos. *Fanfic* por tanto tiene un atributo *chapters*, que es la lista de *string* con los capítulos, y un método *get_string_chapters()* que devuelve todos los capítulos en un único *string*. Según seguía avanzando la clase también demostró ser útil para almacenar en un único elemento toda su información relevante, por lo que termina siendo la unidad básica de trabajo del proyecto.

Sin embargo, el acceso a los datos sigue basándose en una lista de *paths* guardada en un archivo TXT. Es un sistema muy rudimentario (ilustrado en la figura 9), pero no vi necesidad de trasladarlo a una base de datos propiamente dicha, puesto que nada en la creación de una base de datos me ahorra nada del trabajo de crear *fanfic_util*, y desde el punto de vista del resto de programas es igual acceder al texto de un fanfic a través de *FanficGetter* que de una función que recupere el texto de una base de datos. Únicamente intercambiaría el tiempo de limpiar los textos por el de conectar con la base de datos y extraer su información.

El identificador de entidades puede utilizar la lista original con todos los fanfics como fuente para etiquetarlos, pero para la parte de identificación de relaciones del proyecto se necesita filtrar los fanfics

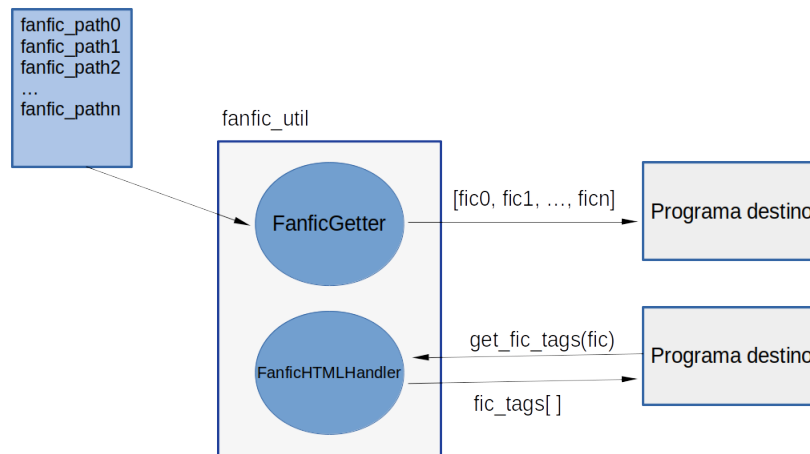


Figura 9: Esquema ilustrando la función de las clases *fanfic_util*

originales en tres subconjuntos: uno centrado en el romance, otro en la amistad, y el último en la enemistad. Por lo tanto, utilicé las funciones de *fanfic_util* para crear un programa, *generate_fic_lists.py*, para crear las listas de *paths* correspondientes a cada grupo.

El criterio utilizado para crear estos tres grupos está basado en la longitud de los textos y sus etiquetas. En el caso de las etiquetas es sencillo: los autores casi siempre etiquetan las relaciones románticas en sus relatos, y a menudo también las amistades, para hacer que sus historias sean más fáciles de encontrar por aquellos que quieran leerlas. En AO3, las etiquetas románticas tienen el formato 'Personaje A/Personaje B', mientras que las etiquetas de amistad son 'Personaje A & Personaje B' o 'Personaje A and Personaje B', con lo que extraer estas etiquetas del archivo HTML es sencillo utilizando expresiones regulares. Además de tener una etiqueta que valide la expresión regular, sólo tuve en cuenta aquellos relatos que sólo tenían un capítulo. De esa forma, esperaba poder eliminar historias largas y elaboradas que contienen romance, pero que principalmente son una historia costumbrista o de aventuras. Limitando la longitud de la historia a un capítulo, todo el romance o la amistad queda condensada en dicho capítulo. Además, como en los relatos que contienen abusos sexuales es común etiquetar al perpetrador y a la víctima con el formato de 'Personaje A/Personaje B', realicé un segundo pase a la lista de romance, eliminando todos aquellos relatos que tuviesen la etiqueta 'Rape/Non-con'.

Crear un conjunto con relaciones de enemistad u odio es una tarea más complicada, ya que los usuarios de AO3 no tienen un formato oficial para las mismas y no se suelen etiquetar. Al final utilicé una lista de las etiquetas que los autores comúnmente utilizan como aviso de que su historia contiene violencia o abusos, como 'Rape/Non-con', 'Torture', 'Graphic Depictions of Violence' o 'Dead Dove: Do Not Eat'.

Esperaba que, entre esas etiquetas y la limitación de longitud, pudiese aislar un conjunto de relatos que sirvieran de modelo para la relación de enemistad.

El resultado fueron 12520 relatos en el conjunto de romance, 784 en el de amistad y 155 en el de enemistad. Para equilibrar los *datasets*, reduje el conjunto de romance a 220 y el de amistad a 180, dando un total de 555 relatos para modelar estas relaciones. A este *dataset* lo llamo 'RFE dataset' (por *Romance, Friendship, Enemy*) y se utiliza en la sección 9.2.1.

9. EXTRACCIÓN DE DATOS A PARTIR DE TEXTO

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.

9.1. Algoritmo de identificación de entidades

9.1.1. Extracción de entidades con NLTK

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. El código utilizado en este proyecto está basado en el tutorial de Ivanov en *Natural Language Processing for Hackers* ([Ivanov, 2016](#)).

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 llamo *NERChunker* (NER por Named Entity 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 incluido 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.

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 figura 10.

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'.

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

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 figura 11.

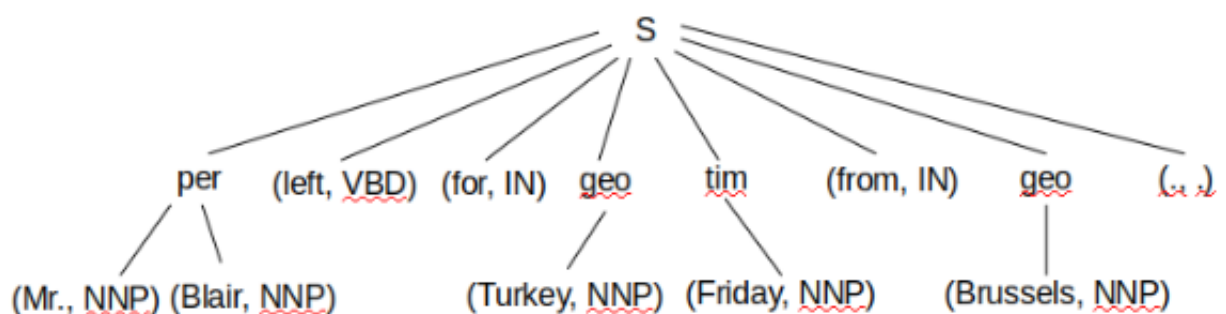


Figura 11: Frase etiquetada por NLTK. Las etiquetas de entidad se encuentran en los nodos, encontrándose todas las palabras pertenecientes a una secuencia de entidad en la profundidad 2 del árbol. Las palabras que no pertenecen a ninguna secuencia de entidad se encuentran en la profundidad 1. Cada hoja del árbol contiene una tupla formada por la palabra y su etiqueta de rol morfológico.

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.

En horizontal queda así:

```

maria@maria-P7815:~/Documents/Fanfic_ontology$ python3 NER_trainer.py
Starting training... go for a walk
optimizing with lambda = 0
NER chunker ( 126.00397671063742 mins)
  ChunkParse score:
    IOB Accuracy: 96.7%%
    Precision:    83.4%%
    Recall:       81.6%%
    F-Measure:    82.5%%
Preparing to pickle. . .
NER_chunker successfully pickled

```

Figura 12: Ejecución final de *NER_trainer.py*, mostrando su evaluación.

(S, [(per, [(‘Mr.’, NNP), (‘Blair’, NNP)]), (‘left’, VBD), (‘for’, IN), (geo, [(‘Turkey’, NNP)]), (tim, [(‘Fri-day’, 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 ser dos procesos muy distintos, y dividirlo parecía la mejor manera de tener un código limpio y claro.

Por lo tanto, el programa *NER_trainer.py* se encarga únicamente de entrenar el *chunker* y guardarlo en un objeto binario con la ayuda de *pickle*, mientras que el programa *NER_tagger.py* carga el objeto binario y lo encapsula en una clase NERTagger. Cuando otro programa quiere usar NERTagger, sólo tiene que importarlo y usar su función *parse(text)*, que requiere el texto completo a analizar, previamente etiquetado con el rol morfológico de cada palabra. En este aspecto, funciona de forma parecida al resto de *taggers* de NLTK.

Sin embargo, al contrario que los modelos de NLTK, NERTagger no devuelve el texto entero con los personajes etiquetados en objetos tipo árbol, sino que devuelve directamente la lista con los nombres de los personajes y el número de veces que es mencionado cada uno. Además, tras la extracción de personajes el programa realiza una canonicalización de personajes.

Vamos a aprovechar el hecho de que los relatos que estamos manejando pertenecen al género fanfic, es decir, son obras basadas en obras ya existentes. Eso quiere decir que hay una alta probabilidad de que la mayoría o incluso todos los personajes del fanfic no sean creación del autor, sino que ya aparecían en la obra original. A estos personajes se les llama ‘personajes canon’.

En este proyecto, la canonicalización de personajes consiste en descubrir qué personajes del fanfic son canon o no. Para ello necesito la lista de personajes que sí que son canon, de modo que, con la ayuda de

Canon ID	Name	Mentions
NO	Lord Beezlebub	2
NO	Lilith	1
14	Did Gabriel	1
NO	Ashmedai	11
4	Aziraphale	42
#end	Laradiri	16
if NO	Every Woman	1
NO	Maruton	1
NO	Joe	1
NO	Amides	1
NO	Dr Dudders	1
NO	History	1
14	Gabriel	1
NO	Butter	1
NO	Alisha	1
4	Aziraphale	1
NO	Mrs Beeton	1
15	God	1
9	Dagon	1
NO	Any	1
NO	So Below	1
NO	Seamus Blackley	1
14	Sir Yes Sir	1
10	Death	1

Figura 13

la página de personajes de la wiki de *Good Omens*¹, creo un archivo llamado *canon_characters.csv* que sirve como base de datos de los personajes de la obra original, asignándole a cada uno un identificador numérico. La base de datos además también contiene el género canónico de cada personaje, y una lista con sus apodos.

Usando la base de datos es sencillo comprobar si un personaje pertenece al canon simplemente observando si su nombre o parte de su nombre coincide con el nombre o algún apodo de un personaje canon. Para no pasar por alto posibles erratas cometidas por los autores al escribir, se considera que dos nombres coinciden cuando la distancia de edición (Levenshtein, 1966) entre ellos es menor que una distancia máxima, cuyo valor se depende de lo largos que sean los nombres. Por ejemplo, un candidato a personaje llamado 'Azriaphale' será enlazado con el personaje canon 'Aziraphale', mientras que para que un candidato sea enlazado con 'War' o 'God' la distancia de edición tendrá que ser 0, puesto que son nombres muy cortos e incluso una distancia de edición de 1 añadiría ruido como 'dog', 'Got', 'warm' o 'ware'. Nombres que tengan varias palabras, como 'Mr. Anderson' y 'Jane Austen', se comparan palabra a palabra con los nombres de los personajes canon, y se escoge la menor distancia de edición.

El resultado final es un diccionario que incluye el nombre del personaje, el identificador numérico de su personaje canon (si lo tiene) y su número de menciones. Un ejemplo se puede observar en la figura 13.

¹<https://goodomens.fandom.com/wiki/Category:Characters>

9.1.2. Extracción de entidades con CoreNLP

La decisión de incluir CoreNLP en el proyecto está motivada por la posibilidad de no sólo aprovechar sus capacidades para identificar menciones a una entidad en un texto, y, sobretodo, su función de resolución de correferencia para identificar relaciones entre personajes en un texto. En la sección 9.2.2 está explicado en mayor detalle por qué esto podría ser relevante para el proyecto y cómo utilizarlo en python, pero aquí baste con mencionar que la resolución de correferencia es un método por el cual se enlaza un pronombre con el nombre propio al que se refiere. Por ejemplo, en una frase como *'I love you, Juliet'* se podría aplicar correferencia para enlazar el *you* con *Juliet*. En este ejemplo, *you* es una mención pronominal, mientras que *Juliet* es una mención propia.

Esto hace que CoreNLP sea un complemento atractivo al NERtagger desarrollado en la sección anterior, ya que además del nombre del personaje también recoge información de sus pronombres, posibilitando identificar su género y llevar una cuenta más precisa de cuántas veces se menciona un personaje concreto en el texto (incluso aunque la mención sea sólo un pronombre como *he*). En esta sección se explica cómo utilizar todas estas funciones, además de los metadatos a nuestra disposición, para identificar y extraer el nombre y el género de los personajes presentes en un texto. Además, puesto que la intención del programa es que el fanfic se pueda comparar con el texto original en el que se basa, por cada personaje se identificará si aparece en la obra original o es un añadido del autor fan.

La idea básica para la extracción de personajes es buscar tanto las menciones de entidad como las de correferencia (en particular, las menciones pronominales y propias). Como se explica en su [documentación](#)², CoreNLP tiene dos tipos de menciones:

- *NERMention*, para las menciones relacionadas con identificación de entidades. Hay varios tipos, pero este proyecto se utiliza sobretodo las menciones de tipo 'PERSON'. Cada mención tiene un `entityMentionIndex` que la identifica de forma única, y además también tiene un `canonicalEntityMentionIndex` que identifica a la entidad particular a la que hace referencia (de modo que si una entidad se llama John Smith, todas las menciones que contienen John irían indexadas a una única *NERMention*).
- *Mention*, para las menciones de correferencia. En este proyecto se utilizan sobretodo las de tipo 'PROPER' y 'PRONOMINAL', ya que son las que identifican nombres y pronombres. Cada una tiene un `mentionID` que la identifica de forma única, además de un `corefClusterID` y un `goldenCorefClusterID` que indican los clusters a los que pertenecen.

Una vez procesado un texto, CoreNLP devuelve un objeto *Document* que contiene todos los *NERMention* y *Mention* detectados en el texto. Es sencillo hallar los personajes simplemente creando una lista que contenga todas las *NERMention* con un mismo `canonicalEntityIndex`, pero todas estas menciones con-

²<https://github.com/stanfordnlp/stanza/blob/master/doc/CoreNLP.proto>

tienen el nombre o parte del nombre de un personaje, y mi intención era hallar también los pronombres utilizados para referirse a un personaje. Ahí es donde entra la función de resolución de correferencia, cuyo resultado se almacena en las *Mention*: una mención de tipo *PROPER* contiene el nombre de un personaje, mientras que las de tipo *PRONOMINAL* contienen algún pronombre. CoreNLP organiza todas estas menciones en clusters, de modo que una mención *PROPER* y una mención *PRONOMINAL* comparten el mismo cluster si se refieren a la misma entidad. Para decidir en qué cluster debe ir una mención, CoreNLP tiene en cuenta factores como el género identificado de la entidad, la distancia entre menciones y cuál fue la última entidad mencionada.

La estrategia más evidente es hacer una lista con todas las menciones *PROPER* y *PRONOMINAL*, identificar sus clusters y asociarlos a las entidades de las *NERMentions*, de modo que por cada texto se tenga una lista de personajes únicos con su identificador, su nombre y su género.

Sin embargo, antes incluso de empezar a entender cómo se relacionan las *Mention* y *NERMention* con el texto, hay que tratar el problema de la latencia de CoreNLP, y es que es un servidor que realmente no está preparado para procesar grandes cantidades de información. Mis primeros programas manejando CoreNLP podían tardar alrededor de un minuto por texto, lo cual no es un problema terrible para procesar uno o dos textos, pero puesto que la intención inicial de utilizar CoreNLP era analizar un dataset de casi 400 textos (sección 9.2.2) me vi obligada a buscar una forma eficiente de realizar las peticiones. Además, muchos de los textos exceden el límite de 100000 caracteres del servidor, lo cual hace que la petición expire y el servidor se cierre, terminando el programa. Aunque es posible aumentar dicho límite con los parámetros de configuración del servidor, la mejora en rapidez es insuficiente.

El límite de caracteres tiene fácil solución, puesto que basta con enviar cada fanfic como una lista de capítulos (dividiendo aquellos que aún sigan excediendo el límite), y cada capítulo se envía en una petición separada. Obviamente eso significa que por cada texto se pueden recibir dos o más objetos *Document*, lo que significa que un mismo personaje puede tener distintos identificadores según el *Document* en el que se generó su mención, lo que complica ligeramente el proceso de consolidación de personajes, como se explica más adelante. Sin embargo, esta división del texto en distintas peticiones evita con éxito que expiren por ser demasiado grandes, pero la respuesta sigue siendo muy lenta para listas de más de 9 ó 10 textos, dependiendo de lo largos que sea cada uno. Finalmente la solución fue rediseñar el programa de manera que en vez de abrir y cerrar el servidor por cada texto, se abre una vez y todas las peticiones se envían juntas. Aunque el tiempo que se tarda en abrir el servidor casi siempre es menor que el necesario para procesar el texto en sí, era obviamente la manera más simple de ahorrar tiempo de ejecución.

Para facilitar todo este proceso se crea la clase *CoreWrapper*, que se encarga de todo lo relacionado con la comprobación del límite de caracteres y manejar el servidor. *CoreWrapper* simplemente recibe

una lista de objetos Fanfic y devuelve esencialmente la misma lista, pero ahora cada Fanfic tiene un atributo *annotations* que es una lista de los *Document* asociados a él. CoreWrapper también maneja los errores de servidor y de red que puedan ocurrir durante la ejecución de CoreNLP, avisando si uno se produce, y asegurando que los datos obtenidos hasta el momento sean almacenados. Ésta función resultó muy importante en el procesado del RFE dataset en la sección 9.2.2, puesto que CoreNLP es bastante propenso a errores de red y rara vez podía digerir más de 25 fanfics de golpe.

Una vez solucionado el problema de la latencia, el siguiente reto es entender cómo CoreNLP indexa cada palabra u oración de un texto con sus correspondientes menciones, y cómo éstas se refieren unas a otras. Repasé la documentación de CoreNLP, además de dibujar esquemas y crear varios programas para encontrar el mejor método de agrupar Mention en clusters y NER Mention en listas que contuvieran toda la información necesaria para identificar cada personaje. Tras toda esta experimentación, todas las menciones y su información quedaron resumidas en listas de diccionarios de python, de forma que cada diccionario representa una mención e incluye el texto de la mención (que generalmente es el nombre del personaje), el género del personaje, y otra información como el tipo de mención y sus identificadores (de entidad, de cluster, etc).

Puesto que las menciones de correferencia pertenecen a clusters de menciones, mientras que las menciones de entidad tienen un identificador, en un principio pensé en clasificar todas las menciones de correferencia según cluster, determinar qué personaje representa qué cluster y luego asociar cada uno de estos clusters con un los identificadores de entidad, teniendo en cuenta factores como el género y el nombre de cada uno. Sin embargo, los clusters de correferencia no se corresponden fácilmente con personajes, especialmente si dos personajes del mismo género aparecen mencionados juntos (cosa a la que éste conjunto de relatos es muy susceptible). Por tanto, para consolidar las menciones de correferencia en personajes tuve aprovechar otros patrones en la información proporcionada por CoreNLP:

1. A veces las menciones de entidad y las de correferencia coinciden en una misma palabra. En particular, las menciones de tipo 'PROPER' (que corresponden con sustantivos y nombres propios) como mínimo a veces también serán una mención de entidad.
2. Las menciones de correferencia tienen un atributo *headString*, que es la palabra que CoreNLP identifica como la más importante en la mención, y que suele corresponderse con el nombre propio del personaje (si una mención es 'Mr. Smith', por ejemplo, CoreNLP identifica 'Smith' como el *headString* de la mención).
3. Cuantas más menciones tenga un personaje, más probable es que dicho personaje sea un personaje real en el texto, y no un error de identificación.

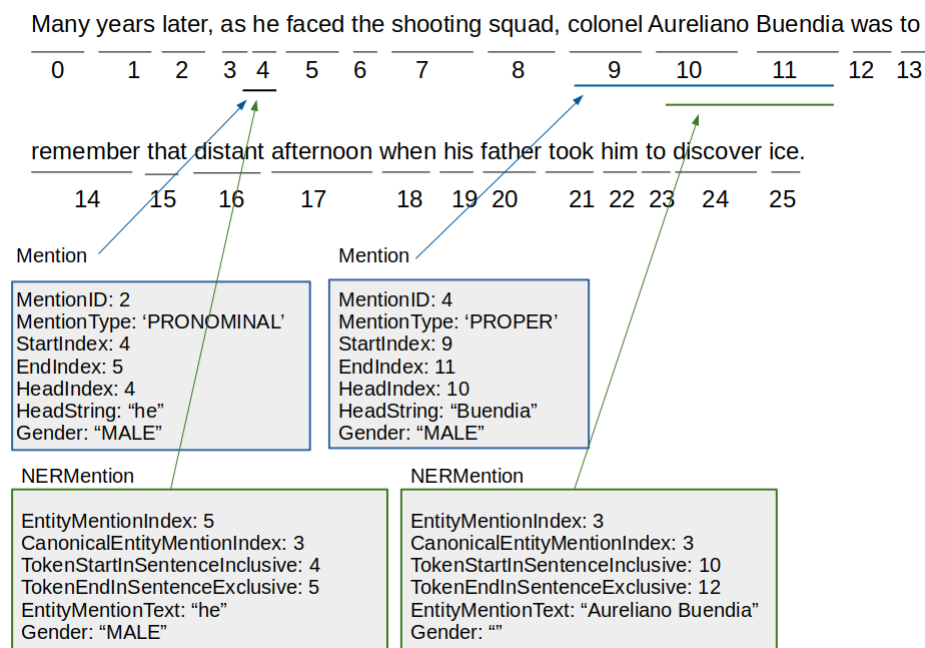


Figura 14: Solapamiento entre menciones de correferencia (Mention) y de entidad (NERMention).

En base a estas observaciones se pueden determinar algunas reglas para decidir qué menciones se refieren a qué personaje. Como en el proceso de canonicalización explicado en la sección 9.1.2, se considera que dos nombres coinciden cuando su distancia de edición ([Levenshtein, 1966](#)) es menor que una distancia máxima, que se fija según lo largo que sea el nombre. Así, tenemos:

1. Todas las NERMention que tengan el mismo *canonicalEntityMentionIndex* se consideran como el mismo personaje, y el atributo *entityMentionText* se considera el nombre del mismo.
2. Identificar las Mention que también son menciones de entidad. Dichas Mention se consideran como el mismo personaje que el de la NERMention si sus nombres coinciden.
3. De las Mention que hayan podido ser identificadas como pertenecientes a un personaje, obtener su cluster de correferencia y considerar a todas las Mention de dicho cluster como pertenecientes a dicho personaje, pero sólo si 1) El atributo *gender* de ambas menciones son el mismo, y 2) El atributo *headString* de ambas Mention coinciden.

El resultado de este proceso es una lista de diccionarios de python que representa a cada personaje, conteniendo su nombre, su género, el número de veces que es mencionado, a qué clusters pertenece, etc.

Esta lista aún es bastante imperfecta. De entrada, los identificadores de entidad y de cluster asignados a cada mención dependen del *Document*, y dada la estructura del programa, la mayoría de relatos van a tener asociados dos o más *Document*, lo que significa que hay personajes con exactamente el mismo

nombre y género que aparecen como dos personajes distintos, porque no comparten el mismo *canonicalEntityMentionIndex* ni ningún cluster de correferencia. Además, esta forma de identificar personajes significa que si alguno tiene un apodo, o si a un personaje se le llama de forma consistente por su apellido en una zona del texto y por su nombre en otro, aparecerá como dos personajes distintos. Para mitigar estos errores, utilicé el proceso de canonicalización de personajes de la sección 9.1.1. También vamos a aprovechar el proceso de canonicalización para determinar el género de un personaje de forma definitiva, y para ello vamos a aprovechar nuevamente que estamos manejando fanfics y que, por lo general, los autores etiquetan sus fanfics. No todos ellos etiquetan el género de sus personajes, pero algunos sí, ya que hay personas a las que le gusta explorar la personalidad o sexualidad de los personajes mediante técnicas como cambiarles el género o darles una expresión ambigua. Algunos ejemplos de las etiquetas que se suelen usar para indicar el género de un personaje son 'He/Him Pronouns for Crowley', 'Androgynous Crowley' o 'Female!Crowley'.

Teniendo en cuenta todos estos factores, la canonicalización de un personaje consiste en estos pasos:

1. Identificar si es canon o no, utilizando los nombres de cada candidato y comparándolos con los nombres y apodos de los personajes canon, igual que en la sección 9.1.1.
2. Identificar el género del personaje:
 - a) Obtener las etiquetas su fanfic y comprobar si hay alguna etiqueta que indique el género del personaje. Si la hay, el personaje se considera de ese género.
 - b) Si no hay ninguna etiqueta, nos quedamos con el género que CoreNLP le haya asignado.
 - c) Si CoreNLP no le ha asignado ningún género ('UNKNOWN', o simplemente el *string* vacío ""), le asignamos el género del personaje canon. Por tanto, si el personaje no es canon, su género seguirá siendo desconocido.

Este proceso resuelve los problemas relacionados con tener los mismos personajes procesados en distintos *Document*, y asegura que una mención será identificada con el personaje canon tanto si menciona el nombre, el apellido o sólo un apodo. Discrimina por género de tal manera que si se le asignan géneros distintos al mismo personaje, se cuentan las menciones de cada uno de modo que al final cada personaje tiene una cuenta de menciones masculinas, femeninas, neutras o desconocidas. De esta manera es más probable identificar correctamente a un personaje con su correspondiente en el canon aunque el género no concuerde.

Todos este proceso de extracción de personajes se lleva a cabo mediante una clase *CoreNLPPDataProcessor*, que se encuentra encapsulada junto a *CoreWrapper* en un programa llamado *corenlp_util.py*.

CoreNLPPDataProcessor, además de extraer personajes, también tiene función de análisis de sentimiento, como se explicará en la sección 10.

En la figura 15 hay un esquema que explica de forma visual todo este proceso de extracción de personajes.

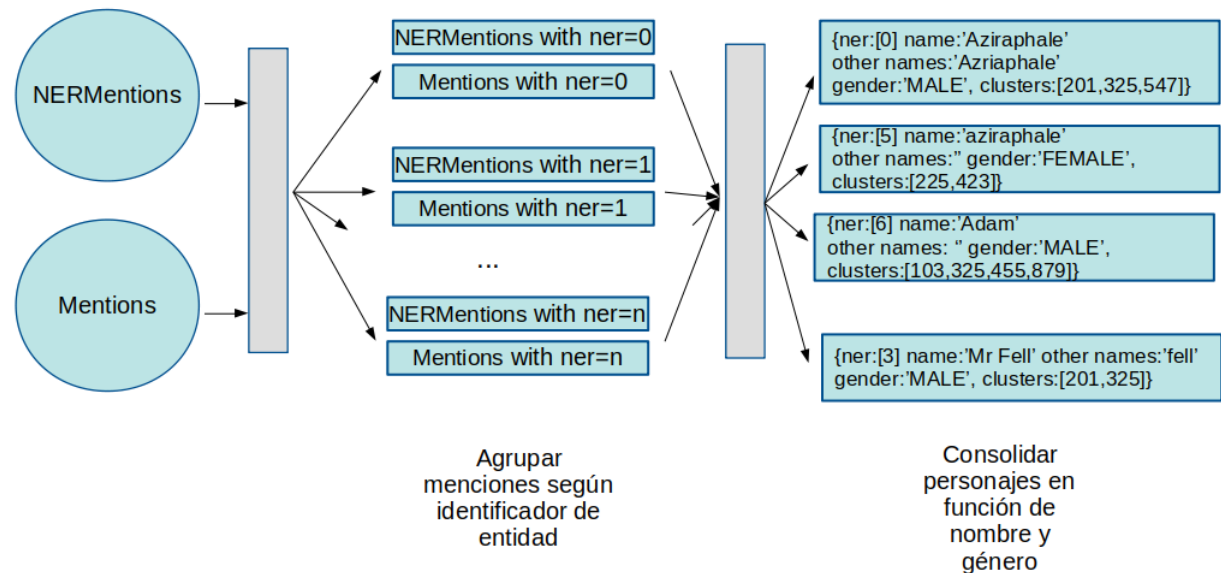


Figura 15: Esquema sobre el proceso de extracción de personajes con CoreNLP.

9.2. Algoritmo de identificación de relaciones

Las relaciones que definen un fanfic son las amistades, los enemigos y, principalmente, los amantes. Extraer relaciones a partir de texto natural es una tarea compleja de por sí, y tratar de detectar este tipo concreto de relaciones en literatura de género puede presentar un reto mayor, puesto que las relaciones sentimentales tienden a representarse de forma implícita, de modo que el lector aprende quiénes son amigos y quiénes enemigos a través de las acciones de los personajes. Además, la ambigüedad, la heterogeneidad y la experimentación son partes naturales de cualquier proceso creativo, por lo que un conjunto de obras no representará una misma relación de forma uniforme, incluso si es entre los mismos personajes.

Encontrar una estrategia para abordar este problema requiere exploración y creatividad, y ya que había empezado el proyecto con NLTK, me pareció natural comenzar la búsqueda por ahí.

El extractor de relaciones de NLTK funciona mediante reglas: después de extraer las entidades nombradas del texto, se puede utilizar el módulo *relextract* para dividir el texto en listas de fragmentos del texto que contienen dichas entidades, y aplicar reglas basadas en expresiones regulares que definan la relación entre las entidades. La regla puede incluir etiquetas de rol morfológico en la expresión regular, y *relextract* permite filtrar por etiqueta IOB, lo que le da algo más de flexibilidad.

Por ejemplo, para extraer una relación de lugar entre una organización y una localización, se puede crear una expresión regular que busque la palabra clave 'in' en el texto, e indicarle a *relextract* que sólo te interesan los fragmentos de texto que tengan una entidad de tipo 'ORG' seguida de una entidad de tipo 'LOC':

```
1 IN = re.compile(r'.*\bin\b(?:\b.+ing\b)')
2
3 for doc in parsed_docs:
4     for rel in nltk.sem.extract_rels('ORG', 'LOC', doc, pattern=IN):
5         print(nltk.sem.show_raw_rtuple(rel))
```

Listado 1: Ejemplo de código que utiliza el módulo *regex* de NLTK para extraer relaciones de lugar y mostrarlas por pantalla. Adaptado del capítulo 7 de Natural Language Processing with Python (Bird, 2012)

Existen proyectos que utilizan este módulo de NLTK para extraer relaciones como *DateOfBirth* y *Has-Parent* (Jose, 2017), pero es evidente que es un método poco adecuado para el tipo de proyecto que estaba intentando hacer.

Estos programas basados en reglas dependen de localizar palabras claves en el texto, y aunque existen palabras clave para identificar relaciones sociales ("love", "kiss", "hug", "friend", "kill", "hate", etc), lo cierto es que la naturaleza de la expresión literaria hace que este método, incluso a simple vista, parezca bastante ingenuo. No sólo es perfectamente posible expresar amor, amistad y odio sin usar "palabras clave".³sociadas con dichos sentimientos, sino que en un texto literario raramente se escribe explícitamente *Romeo loved Juliette*, si no que es más normal encontrar estructuras como '*I love you*', *said Romeo*. En una frase así, no se menciona explícitamente a Julieta, pero un lector humano sabe si se refiere a ella por el contexto de la escena. Pero un programa que únicamente se preocupa de las etiquetas IOB de una frase no será capaz de unir ese *you* con Julieta (ni, ya puestos, el *I* con Romeo).

Descartado el extractor de relaciones de NLTK, empiezo a buscar opciones en otras librerías. El Stanford Natural Language Processing Group publicó un extractor de relaciones como parte de las funciones de CoreNLP, pero las relaciones que está entrenado para detectar (*Live_In*, *Located_In*, *OrgBased_In*, *Work_For*, *None*) no parecen útiles para el proyecto. Por tanto, entrenar mi propio modelo para relaciones sociales parece la única solución.

Crear un modelo de regresión logística con NLTK, similar al identificador de entidades, requería que el texto ya estuviera etiquetado con las relaciones. Los autores de AO3 usan etiquetas que es posible extraer las relaciones a partir del archivo HTML de cada fanfic, pero es una etiqueta a nivel del texto completo, no a nivel de frase, que es como trabaja NLTK. Dejando de lado NLTK por el momento, decidí explorar

soluciones usando clustering y modelado de temas.

9.2.1. Primeras estrategias: Clustering y LDA

Decidir si dos personajes son amigos, enemigos o amantes (sin tener ninguna información previa sobre la obra) puede requerir leer el texto completo, con lo cual es razonable utilizarlo para el análisis y asignar una etiqueta de 'romance', 'amistad' o 'enemistad' al texto en su conjunto, más que etiquetar ciertas palabras y personajes del mismo.

Por tanto, dado un conjunto de textos al azar, podría llegarse a la conclusión de cuáles contienen romance, cuáles amistad y cuáles enemistad observando si hay similitudes entre ellos. Con este enfoque, parece una tarea adecuada para un algoritmo de clustering.

Para comprobar cómo de útil sería esta estrategia utilicé el RFE dataset, cuya creación está explicada en la sección 8.2. Esperaba que teniendo tres conjuntos claros en los que el tema era evidente sirviese para ver cómo de eficaz es el clustering para la tarea general, que sería poder decir si hay romance en un texto incluso si no es el tema central del mismo.

Una vez creados los conjuntos, utilicé la librería *Scikit-Learn* en conjunto con NLTK para crear el programa de clustering. Para preprocesar el texto se utilizan los métodos de NLTK para *tokenizar* y *lemmatizar* los textos, además de crear un conjunto de *stopwords*, palabras comunes en inglés pero que no aportan mucha información sobre el mismo (preposiciones, pronombres, puntuación, demostrativos, etc). Después de preprocesar el texto se procede a extraer las características relevantes del mismo, para lo cual se utiliza el módulo *TfidfVectorizer* de *Scikit-Learn*. Su trabajo es 'vectorizar' el texto de manera que sus características principales queden expresadas en un formato que el algoritmo de clustering pueda entender, cosa que hace asignando un peso a cada palabra dependiendo del número de ocurrencias de la misma (esto se llama *bag of words*). Hay vectorizadores que asignan más peso cuanto más frecuencia, pero esto hace que palabras muy comunes pero con poco valor informativo roben protagonismo a palabras menos frecuentes pero más interesantes. El vectorizador 'Tf-Idf' (*Term frequency-Inverse document frequency*), en cambio, multiplica la frecuencia de una palabra en un documento por un componente *idf* que, como se ve en la fórmula 1, está basado en la frecuencia de ese término en todos los documentos. Los vectores resultantes se normalizan usando la norma de Euclides; más información está disponible en la guía de usuario de *Scikit-Learn* ([skl](#),).

Con los textos ya preprocesados y convertidos en vectores *tf-idf* se puede crear un modelo de clustering, en este caso utilizando el módulo *KMeans* de *Scikit-Learn*. *KMeans* es un algoritmo que crea clusters de

$$\text{idf}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1 \quad (1)$$

Figura 16: Componente idf del vectorizador Tf-Idf. t se refiere al término cuyo peso está determinando, n al número total de documentos y df a la frecuencia de t en este documento.

tal forma que cada uno tenga la misma varianza, minimizando la suma de los cuadrados de las distancias entre los miembros de cada grupo (fórmula 2).

$$\sum_{i=0}^n \min_{\mu_j \in C} ||x_i - \mu_j||^2 \quad (2)$$

Figura 17: Criterio de la suma de cuadrados. n es el total de textos, con x perteneciendo a n . C es el número de clusters, con μ siendo la media de las muestras x de cada cluster. El algoritmo KMeans reduce esta suma todo lo posible.

Además del código necesario para crear el modelo KMeans y procesar los daños, añadí código para evaluar el modelo e imprimir un diagrama de puntos con los clusters. Tras probar dos tokenizers distintos y varias combinaciones con los parámetros del vectorizador y el modelo, los resultados se pueden ver en la figura 18.

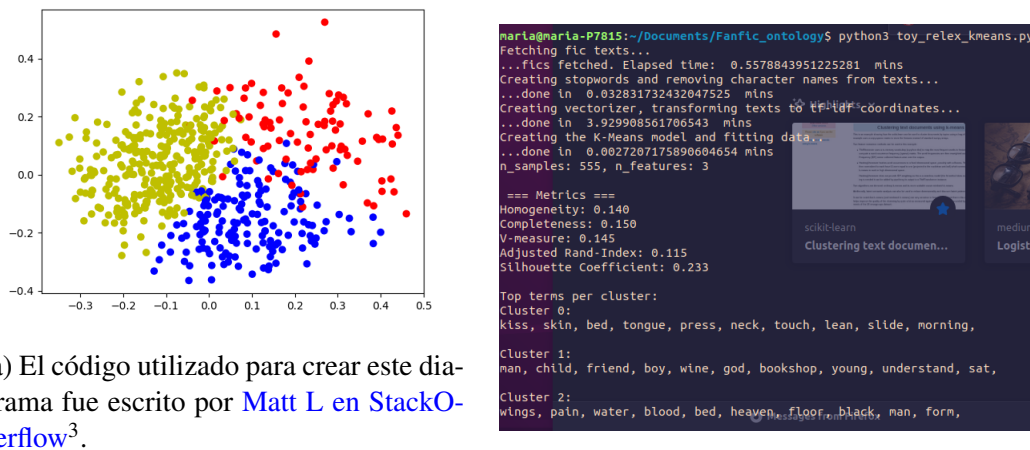


Figura 18: A la derecha, ejecución de *toy_relex_kmeans*, mostrando su evaluación. La homogeneidad se cumple cuando ningún cluster contiene miembros que pertenezcan a categorías distintas en los datos reales. La completitud se satisface si todos los miembros de una de las categorías reales pertenecen al mismo cluster. A la izquierda, el diagrama creado por el programa.

Los resultados no son muy buenos. Aunque los términos de cada cluster parecen prometedores, ninguna métrica sube del 0.1, lo que indica que las categorías del clustering son sólo un poco mejores que haberlas asignado al azar, y que hay mucho solapamiento entre clusters.

³<https://stackoverflow.com/questions/57626286/how-to-plot-text-clusters>

```

maria@maria-P7815:~/Documents/Fanfic_ontology$ python3 toy_relex_lda.py tokv1
tokv1
Fetching fic texts...
...fics fetched. Elapsed time: 0.551144790649414 mins
Preprocessing fanfics and creating dictionaries...
Processing elapsed time: 1.7671716809272766 minutes
Training LDA model...
...LDA elapsed time: 2.681595873832703 minutes
LDA Coherence score: 0.23684957439734447
Topics in LDA model:
(0, '0.034*I" + 0.017"Crowley" + 0.014*He" + 0.013"Aziraphale" + 0.012*say" + 0.007*The" + 0.007*know" + 0.006*You" + 0.006*It" + 0.006*like"')
(1, '0.034*Crowley" + 0.030*Aziraphale" + 0.018*He" + 0.015*I" + 0.008*The" + 0.007*angel" + 0.007*back" + 0.006*It" + 0.006*like" + 0.006*say"')
(2, '0.015*I" + 0.008*He" + 0.008*The" + 0.008*Crowley" + 0.007*Aziraphale" + 0.006*one" + 0.006*angel" + 0.005*know" + 0.005*It" + 0.005*like"')
maria@maria-P7815:~/Documents/Fanfic_ontology$

```

(a) Filtrado: sólo puntuación.

```

maria@maria-P7815:~/Documents/Fanfic_ontology$ python3 toy_relex_lda.py UNITokv2
UNITokv2
Fetching fic texts...
...fics fetched. Elapsed time: 0.5545525550842285 mins
Preprocessing fanfics and creating dictionaries...
Processing elapsed time: 4.393664975961049 minutes
Training LDA model...
...LDA elapsed time: 2.5472853938738504 minutes
LDA Coherence score: 0.25293098120527674
Topics in LDA model:
(0, '0.008*say" + 0.007*know" + 0.006*get" + 0.005*Adam" + 0.005*back" + 0.005*Crawly" + 0.005*look" + 0.004*make" + 0.004*go" + 0.003*time"')
(1, '0.035*Crowley" + 0.032*Aziraphale" + 0.009*say" + 0.007*back" + 0.007*angel" + 0.006*know" + 0.005*look" + 0.005*eyes" + 0.004*demon" + 0.004*get"')
(2, '0.027*Crowley" + 0.019*Aziraphale" + 0.008*say" + 0.008*angel" + 0.007*know" + 0.007*demon" + 0.006*... + 0.006*get" + 0.006*look" + 0.005*back"')
maria@maria-P7815:~/Documents/Fanfic_ontology$

```

(b) Filtrado: puntuación, pronombres, determinantes, etc.

Figura 19: Ejecución de *toy_relex_lda*, con distintos criterios de filtrado. Además se muestran los 10 términos más relevantes de cada tema, y su probabilidad de pertenecer a dicho tema.

Puesto que para crear el modelo he utilizado datos filtrados a propósito para modelar cada categoría tan bien como fuera posible, con la esperanza de poder usarlo como posible semilla para un sistema más general, esto supone un gran problema.

Busqué entonces otra estrategia, utilizando un modelo de temas más que uno de clustering. El modelado de temas con el algoritmo LDA parece también una buena opción para este problema, puesto que al contrario que el clustering clásico, LDA asigna a cada documento una distribución de temas en cada uno. Esto se ajusta a este análisis, puesto que aunque he intentado crear *datasets* 'perfectos' que traten un único tipo de relación en cada uno, lo cierto es que lo normal en un relato es que estén mezcladas.

LDA es un algoritmo que descubre temas de forma no supervisada. Trabaja bajo la asunción de que cada documento es un conjunto de temas, y que cada tema es un conjunto de palabras. Empieza asignando cada a palabra a un conjunto al azar de temas, y en cada iteración mejora la asignación.

Al igual que en el programa anterior, LDA requiere un preprocesado del texto, con lo que utilizo los dos *tokenizers* del algoritmo de clustering. *Scikit-Learn* carece de modelo LDA, por lo que utilizo el de la librería *gensim*. LDA también trata los textos como un *bag of words*, pero no es necesario vectorizarlos antes de usarlos para entrenar el modelo, que devolverá lista de palabras por tema, junto con la probabilidad de que esa palabra pertenezca a dicho tema.

Tras preprocesar los textos de forma similar a como se hizo con clustering y entrenar el modelo LDA, se observan los resultados en la figura 19a.

```
>>> cdf.sort_values(by=['Coherence'], inplace=True, ascending=False)
>>> cdf[:5]
```

	Validation_Set	Topics	Alpha	Beta	Coherence
48	75% dataset	3	0.9099999999999999	0.9099999999999999	0.355668
313	15% dataset	3	0.61	0.9099999999999999	0.353786
53	75% dataset	3	symmetric	0.9099999999999999	0.316054
318	15% dataset	3	0.9099999999999999	0.9099999999999999	0.313049
307	15% dataset	3	0.61	0.61	0.309767

(a) Mejores puntuaciones de coherencia para 3 temas.

```
AttributeError: 'DataFrame' object has no attribute 'sort_valuse'
>>> df.sort_values(by=['Coherence'], inplace=True, ascending=False)
>>> df[:5]
```

	Validation_Set	Topics	Alpha	Beta	Coherence
233	75% dataset	9	symmetric	0.9099999999999999	0.420302
102	75% dataset	5	0.61	0.61	0.390992
338	15% dataset	4	0.31	0.9099999999999999	0.381947
408	15% dataset	6	0.9099999999999999	0.9099999999999999	0.377929
123	75% dataset	6	0.61	0.9099999999999999	0.374808

(b) Parámetros con las mejores puntuaciones de coherencia de toda la prueba.

Figura 20: Resultados de *topic_evaluate.py*, visualizados y ordenados con la ayuda de *pandas*. Se pueden observar qué número de temas y qué hiperparámetros dan mejores puntuaciones de coherencia para el modelo LDA creado a partir del RFE dataset.

Los resultados son un poco decepcionantes, pues ninguno de los 10 términos más relevantes por tema tiene siquiera un 0.1 % de probabilidad de pertenecer a su tema. Tampoco es sorprendente que no sean muy relevantes, pues aparecen muchos pronombres, determinantes e incluso algún número. Aprovechando el etiquetado de rol morfológico de NLTK, se retiran esas palabras y se crea un nuevo modelo, cuya ejecución está en la figura 19b.

Estos resultados, sin embargo, tampoco son muy convincentes. La puntuación de coherencia, que indica cómo de adecuado es el número de temas para los datos analizados, es 0.23 en el primer caso y 0.25 en el segundo. Es un resultado que se puede mejorar afinando los hiperparámetros de LDA, para lo cual se utiliza el programa *topic_evaluate.py*, que prueba diferentes valores para *alpha* (densidad documento-tema) y *beta* (densidad palabra-tema) del modelo LDA de *gensim*. Los resultados de su ejecución se guardan en el archivo *lda_evaluation.csv*, y en la figura 20a se puede ver que la puntuación de coherencia se puede mejorar bastante para tres temas con los hiperparámetros correctos, pero como evidencia la figura 20b, queda muy lejos del 0.42 de coherencia que se puede conseguir si se permite subir el número de temas a nueve. No parece, por tanto, que este modelo sea el más adecuado para buscar las relaciones que se buscan.

9.2.2. Correferencia con CoreNLP

Tras varias pruebas con los algoritmos de clustering y LDA, encontré *CoreNLP*, un servidor de Stanford NLP Group que realiza diversos tipos de extracción de información y análisis de lenguaje natural, entre ellos, resolución de correferencia.

Volviendo al ejemplo de Romeo y Julieta, una frase como *'I love you', said Romeo*, extraída de un texto más largo en la que el contexto es que Romeo está hablando con Julieta, da poca información a un algoritmo que no es capaz de entender a qué personajes se refieren los pronombres *I* y *you*. Sin embargo, si se aplicase resolución de correferencia sobre el texto, a ojos del algoritmo la frase se convertiría en *'Romeo love Juliette', said Romeo*, con lo que el algoritmo entiende mucho mejor qué está pasando en esta frase y a quiénes afecta.

CoreNLP da acceso a esa posibilidad. Aunque está programado en java, cuenta con un módulo de python llamado *Stanza*, con lo que creé algunos programas para familiarizarme con su funcionamiento y la idea que quería desarrollar. El primero de estos programas, *toy_relex.py*, no hace más que buscar las frases que contengan el verbo *'to love'* y al menos una entidad, elegida de antemano. El segundo programa, *toy_relex_v2.py* expande este concepto usando *CoreNLP*, utilizando tanto su función de identificación de entidades como la de solución de correferencia para obtener las *'cadenas de correferencia'* del texto, utilizándolas para enlazar cada pronombre del texto con la entidad a la que se refiere. La idea sigue siendo seleccionar las frases que tengan el verbo *'to love'*, pero en vez de quedarme únicamente con aquellas en las que se nombren explícitamente a un personaje, también me quedo con aquellas en las que los pronombres formen parte de una cadena de correferencia.

Para poder llevar a cabo todo esto, fue necesario estudiar con atención las propiedades de los objetos que utiliza *Stanza*, que por suerte están bien [documentadas](#), y hacer muchas pruebas y visualización de datos, como se puede ver en las figuras 21 y 22.

Tras analizar toda esta información, entender la indexación de los objetos de *Stanza* y ganar experiencia manejándolos, se hizo evidente que el siguiente paso sería crear un programa cuya función fuese resumir la información proporcionada por CoreNLP de forma que sea útil para identificar relaciones entre personajes. En particular, mi intención era usarlo con el RFE dataset y tratar de identificar relaciones en él. El problema era que el RFE dataset es bastante grande, y enseguida se hizo evidente que era necesario guardar la información proporcionada por CoreNLP en un archivo para no malgastar horas de trabajo. Ya que los objetos de *Stanza* no tienen una forma sencilla de ser almacenados directamente, acabé creando un programa que, tras recoger las respuestas del servidor, resumía la información más relevante y la guardaba en dos archivos csv:

Sentence 92	tokens 0-1	PRONOMINAL	cluster 912	text: He
Sentence 212	tokens 8-9	PROPER	cluster 912	text: Aziraphale
Sentence 272	tokens 14-15	PRONOMINAL	cluster 912	text: his
Sentence 393	tokens 6-7	PROPER	cluster 912	text: Aziraphale
Sentence 333	tokens 3-4	PRONOMINAL	cluster 912	text: He
Sentence 273	tokens 0-1	PRONOMINAL	cluster 912	text: He
Sentence 2	tokens 0-3	PROPER	cluster 912	text: Aziraphale
Sentence 243	tokens 3-4	PRONOMINAL	cluster 912	text: his
Sentence 394	tokens 0-1	PRONOMINAL	cluster 912	text: He
Sentence 303	tokens 12-13	PRONOMINAL	cluster 912	text: he
Sentence 333	tokens 15-16	PRONOMINAL	cluster 912	text: his
Sentence 62	tokens 16-17	PRONOMINAL	cluster 912	text: him
Sentence 334	tokens 0-1	PRONOMINAL	cluster 912	text: His
Sentence 213	tokens 10-11	PRONOMINAL	cluster 912	text: him
Sentence 333	tokens 18-19	PRONOMINAL	cluster 912	text: his
Sentence 304	tokens 0-1	PRONOMINAL	cluster 912	text: He
Sentence 334	tokens 4-5	PRONOMINAL	cluster 912	text: he
Sentence 333	tokens 22-23	PRONOMINAL	cluster 912	text: his
Sentence 244	tokens 0-1	PRONOMINAL	cluster 912	text: He
Sentence 33	tokens 4-5	PRONOMINAL	cluster 912	text: his
Sentence 274	tokens 3-4	PRONOMINAL	cluster 912	text: his
Sentence 304	tokens 6-9	PROPER	cluster 912	text: Aziraphale
Sentence 333	tokens 25-26	PRONOMINAL	cluster 912	text: his
Sentence 274	tokens 6-7	PRONOMINAL	cluster 912	text: him
Sentence 244	tokens 5-6	PRONOMINAL	cluster 912	text: him
Sentence 334	tokens 11-12	PRONOMINAL	cluster 912	text: his
Sentence 425	tokens 0-1	PROPER	cluster 912	text: Aziraphale
Sentence 63	tokens 14-15	PROPER	cluster 912	text: Aziraphale
Sentence 394	tokens 20-21	PRONOMINAL	cluster 912	text: he
Sentence 244	tokens 11-12	PRONOMINAL	cluster 912	text: he
Sentence 335	tokens 0-1	PROPER	cluster 912	text: Aziraphale
Sentence 274	tokens 15-16	PROPER	cluster 912	text: Aziraphale
Sentence 335	tokens 2-3	PRONOMINAL	cluster 912	text: his
Sentence 395	tokens 7-8	PRONOMINAL	cluster 912	text: he
Sentence 335	tokens 6-7	PRONOMINAL	cluster 912	text: his
Sentence 214	tokens 16-17	PRONOMINAL	cluster 912	text: his
Sentence 305	tokens 5-6	PRONOMINAL	cluster 912	text: he
Sentence 394	tokens 29-30	PRONOMINAL	cluster 912	text: his
Sentence 395	tokens 12-13	PRONOMINAL	cluster 912	text: his
Sentence 425	tokens 14-15	PRONOMINAL	cluster 912	text: he
Sentence 4	tokens 8-9	PROPER	cluster 912	text: Aziraphale
Sentence 394	tokens 34-35	PRONOMINAL	cluster 912	text: he
Sentence 396	tokens 0-1	PROPER	cluster 912	text: Aziraphale
Sentence 64	tokens 13-14	PROPER	cluster 912	text: Aziraphale
Sentence 336	tokens 0-1	PRONOMINAL	cluster 912	text: He
Sentence 65	tokens 0-1	PRONOMINAL	cluster 912	text: He
Sentence 95	tokens 2-3	PRONOMINAL	cluster 912	text: his
Sentence 306	tokens 0-1	PROPER	cluster 912	text: Aziraphale

Figura 21: Visualización de la información proporcionada por *CoreNLP*

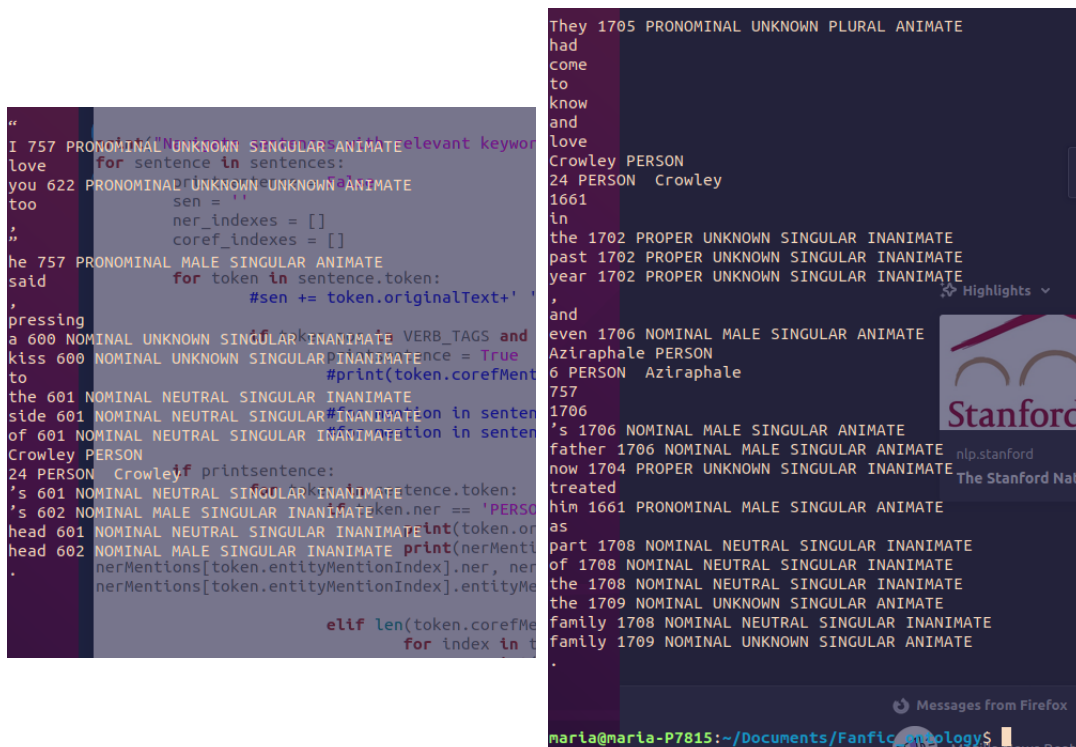


Figura 22: Visualización de frases particulares con anotaciones de correferencia de *CoreNLP*

- fic_characters.csv (tabla 2) almacena los personajes de cada fanfic, junto con su correspondiente en el canon y toda la información necesaria para identificar en qué frases de qué fanfic aparece (clusters, identificador, etc).
- fic_sentences (tabla 1) almacena los fanfics frase a frase, junto con los identificadores y clusters de los personajes mencionados en ellas. Decidí almacenar sólo las frases que mencionan personajes en vez de todas las del fanfic porque estaba orientando este archivo a la identificación de relaciones entre personajes, por lo que parecía razonable asumir que las frases con más información serían aquellas en las que se mencionan personajes.

El desarrollo de este programa no fue trivial, debido a la complejidad de la indexación de *Stanza* y otros problemas derivados de la naturaleza del proyecto, como por ejemplo, si un personaje aparece marcado como masculino en 20 menciones y femenino en 3, ¿Deberían considerarse personajes distintos? ¿Asumo que todas las menciones de un cluster se refieren a un único personaje, incluso si algunos nombres son radicalmente distintos del resto? Para resolver todas estas preguntas y alcanzar un programa funcional fui pasando por varias etapas y distintos programas, que con el tiempo refinaría y acabaría encapsulando en *corenlp_util.py* para su uso posterior en el programa final. En esta parte del proyecto sin embargo utilicé los programas de *corenlp_wrapper.py* y *ner_and_sen_extraction_v2.py*, que se pueden considerar versiones anteriores a las utilizadas en *corenlp_util.py*, y por tanto muchos de los problemas mencionados en la sección 9.1.2 relacionados con la latencia y errores de servidor fueron resueltos durante su desarrollo. Además de estos problemas, hubo que limpiar y revisar el formato de cada *string* y lista, ya que el archivo CSV estaba configurado para procesar cada punto y coma como un separador entre columnas, lo

ficID	Dataset	senID	Sentiment	Verbs	nerIDs	Clusters
57	ROMANCE	2	Positive	Crowley flashed a grin over his shoulder .	0	452
57	ROMANCE	5	Positive	He grabbed the long pole that served as a door handle and pulled it open .	0	452, 18
57	ROMANCE	10	Negative	He cautiously side - stepped around an especially small one .	0	33, 452
57	ROMANCE	11	Neutral	“ They ’re not going to bite you , ” Crowley laughed .	0	452, 36, 37, 113

Cuadro 1: Estructura de *fic_sentences.csv*

ficID	nerID	Name	Gender	Mentions	clusterID	canonID
57	0	ng	UNKNOWN	193	279, 452, 116, 251, 435, 448, 517, 545	-1
57	15, 80, 95	aziraphale	MALE	54	279	4
57	146, 151	crowley	MALE	23	452	8
70	0	crowley	NEUTRAL	9	1	8

Cuadro 2: Estructura de *fic_characters.csv*

cual añadía columnas extra cada vez que encontraba un punto y coma en cualquier parte del texto.

Una vez procesado todo el RFE dataset y completados los CSV, utilicé la información almacenada ellos para realizar varias técnicas de análisis de texto natural, con la esperanza de hallar algún patrón, bigrama o palabra significativa que me ayudara a identificar relaciones entre personajes.

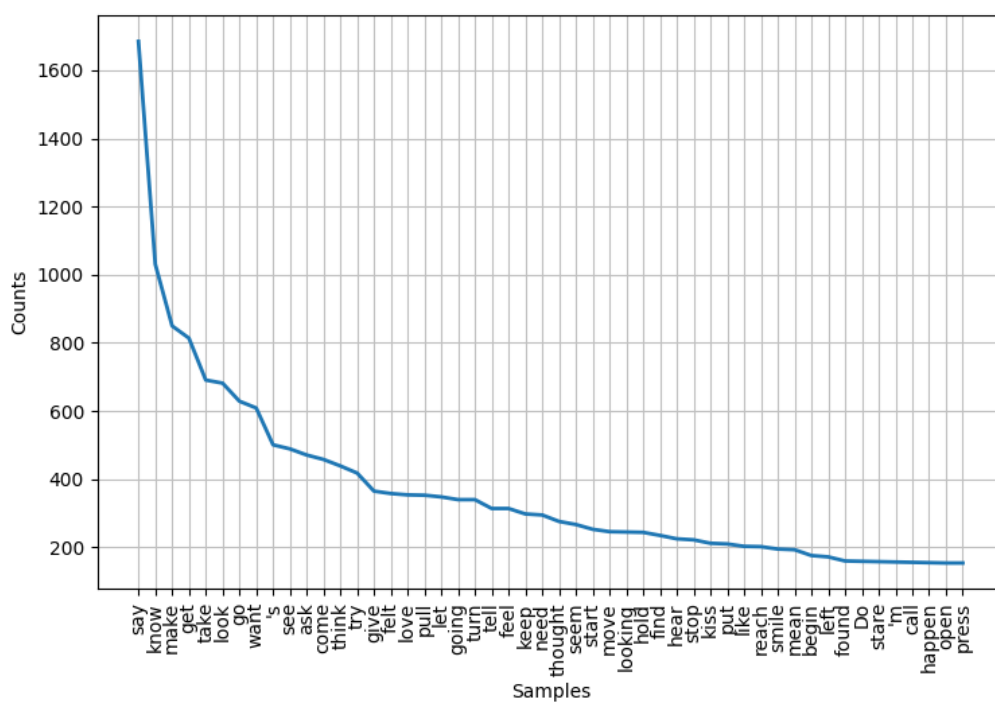
En primer lugar quise buscar si había algún verbo que fuera característico de cada dataset, por lo que extraje y lematicé los verbos de cada frase usando las herramientas de NLTK. En la primera pasada los verbos más repetidos en todos los dataset fueron verbos muy comunes en inglés, como 'say', 'go', 'get', 'make', 'look', etc. de modo que para eliminar ruido hice otro análisis eliminándolos. Para hacer más fácil la comparación y a modo de ejemplo, se muestra la distribución de frecuencias de los verbos en el dataset de romance en la figura 23a-23b.

Desafortunadamente, ni siquiera eliminar los verbos más frecuentes parecía dar un resultado claramente distintivo para cada dataset, por lo que decidí utilizar NLTK para buscar bigramas y trigramas característicos (figuras 24a-24c). Pero tampoco parece haber una alguna combinación distintiva aquí, los n-gramas siguen siendo bastante parecidos de un dataset a otro y utilizan casi las mismas palabras.

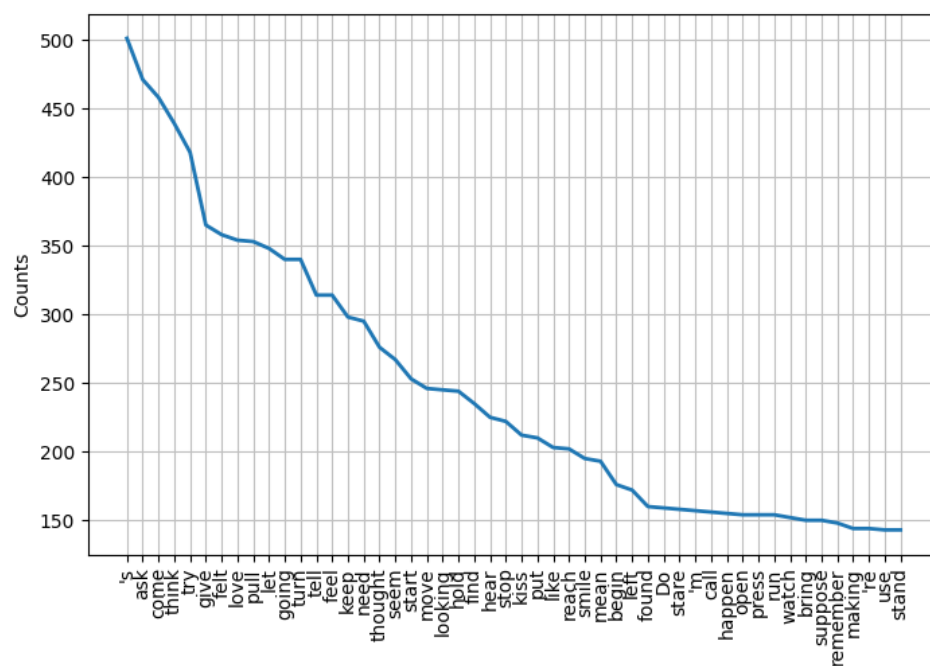
En un último intento de buscar algún patrón, decidí volver extraer la distrución de frecuencia de verbos, bigramas y trigramas, pero esta vez utilizando únicamente las frases en las que se mencione a dos personajes en concreto, en vez de todas las frases del relato. Estos personajes son elegidos de antemano, utilizando los identificadores propocionados por CoreNLP (tal y como se ve en la tabla 1).

La distribución de frecuencias se puede ver en la figura 25.

Sobra decir que los resultados tampoco fueron muy prometedores. Por tanto dejo para un trabajo la parte



(a) Verbos más frecuentes en el dataset de romance.



(b) Verbos más frecuentes en el dataset de romance, tras retirar los más comunes.

de extracción de relaciones, y el resto del proyecto se centra en la extracción de personajes nombrados y determinación de su género.

```

===== Bigram collocations of Romance dataset =====

['had been', 'It was', 'Of course', 'You 're', 'The angel', 'I 'll', 'six thousand', 'I know', 'They were', 'I think', 'The demon', 'Mistress Fell', 'thousand years', 'shook head', 'dear boy', 'You know', 'Mr. Fell', 'I suppose', 'arms around', 'six years']

===== Trigram collocations of Romance dataset =====

['I 'm sorry', 'Crowley 's face', 'Crowley 's eyes', 'Crowley 's hand', 'I 'm afraid', 'Crowley 's head', 'Crowley 's neck', 'Crowley 's hair', 'I 'm going', 'think I 'm', 'Crowley 's shoulder', 'I 'm sure', 'I 'm glad', 'Crowley 's mouth', 'But I 'm', 'Crowley 's lips', 'Crowley 's chest', 'I 'm fine', 'Crowley 's cheek', 'Crowley 's voice']

```

(a) Bigramas y trigramas con mayor likelihood en el dataset de romance.

```

===== Bigram collocations of Friendship dataset =====

['had been', 'It was', 'Of course', 'Aziraphale said', 'Crowley said', 'thousand years', 'He was', 'I think', 'gon na', 'I suppose', 'six thousand', 'holy water', 'shook head', 'six years', 'There was', 'You 're', 'cleared throat', 'did want', 'The angel', 'You know']

===== Trigram collocations of Friendship dataset =====

['I 'm sorry', 'think I 'm', 'I 'm sure', 'I 'm afraid', "I 'm sorry", 'Crowley had been', 'I 'm going', 'But I 'm', "think I 'm", 'I 'm saying', 'I 'm glad', 'know I 'm', 'I 'm getting', 'I 'm warning', 'I 'm supposed', 'thinks I 'm', 'All I 'm', 'What I 'm', 'He had been', "I 'm afraid"]

```

(b) Bigramas y trigramas con mayor likelihood en el dataset de amistad.

```

===== Bigram collocations of Enemy dataset =====

['had been', '- -', 'It was', 'Mr. Fell', 'Port Talbot', '° •', '° °', '• °', '• •', 'Az gon', '## Chapter', 'Chapter 1', '## 1', 'The Teacher', 'I think', 'Of course', 'His eyes', 'The angel', 'At least', 'I 'll']

===== Trigram collocations of Enemy dataset =====

["I 'm sorry", "And I 'm", "I 'm going", "I 'm afraid", "But I 'm", '° • °', '• ° •', "? I 'm", "I 'm asking", "I 'm trying", 'I 'm sorry', 'Crowley 's voice', 'And I 'm', 'He had been', 'Crowley had been', 'I 'm', 'think I 'm', 'Aziraphale had been', 'Crowley 's hand', 'I 'm afraid']

```

(c) Bigramas y trigramas con mayor likelihood en el dataset de enemistad.

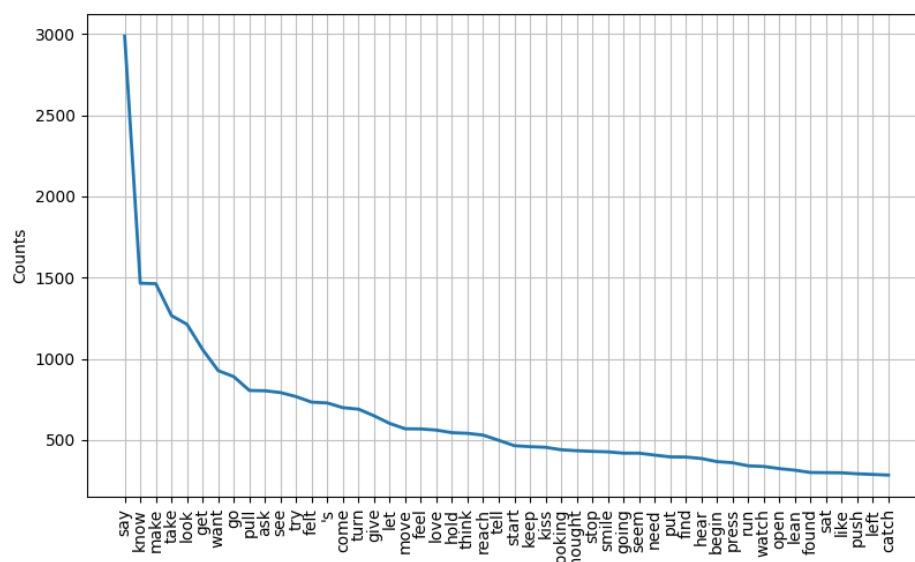


Figura 25: Verbos más frecuentes en el dataset de romance en las frases que mencionan a dos personajes concretos.

10. PROGRAMA PRINCIPAL: `fic_character_extractor`

El programa principal se lanza desde la terminal y tiene dos posibles comandos:

- `fic_character_extractor <fic_index>`, que analizará el fanfic cuyo identificador sea `fic_index`. Por tanto, se comprueba que el usuario no introduzca un identificador menor que 0 ni mayor que 20190.
- `fic_character_extractor`, que analizará un fanfic elegido al azar del total de fanfics disponibles.

Debido a la latencia de CoreNLP, evita elegir un fanfic que tenga más de 50000 caracteres. En ambos casos el programa utiliza la clase `FanficGetter` del módulo `fanfic_util` para extraer el fanfic elegido, obteniendo así el objeto `Fanfic` que encapsula toda la información necesaria del mismo. A continuación, se siguen los siguientes pasos:

1. Identificación de entidades con NERTagger: El texto del fanfic es *tokenizado* en frases y palabras antes de etiquetar cada palabra con su rol morfológico, usando para ello las herramientas de procesamiento de texto de NLTK. A continuación, se utiliza la función `parse()` de la clase `NERTagger` del módulo `NER_tagger(9.1.1)` para extraer los personajes del texto.
2. Identificación de entidades con CoreNLP: se utiliza la clase `CoreWrapper` para enviar el texto al servidor de CoreNLP, y la clase `CoreNLPPDataProcessor` extrae los personajes a partir de la respuesta. Ambas clases pertenecen al módulo `corenlp_util(9.1.2)`.
3. Análisis de sentimiento con CoreNLP: se utiliza la clase `CoreNLPPDataProcessor` de `corenlp_util` para extraer el sentimiento del fanfic y mostrar si es principalmente positivo o negativo.
4. Mostrar en pantalla los resultados, junto con el título y etiquetas de personaje del fanfic.

Las etiquetas de personaje de un fanfic son simplemente la forma del autor de indicar qué personajes aparecen en él. Por motivos técnicos evidentes, los autores se limitan a etiquetar a los personajes más importantes del texto, por lo que es esperable que NERTagger y CoreNLP encuentren más personajes en el texto. Sin embargo, como mínimo no deberían dejar sin identificar ninguno de los personajes etiquetados. Las etiquetas de personaje, como el título, son metadatos del fanfic que se extraen directamente del mismo usando la clase `FanficHTMLHandler` del módulo `fanfic_util`.

Los personajes detectados por NERTagger y CoreNLP serán algo distintos, pero por lo general las coincidencias son bastante consistentes. Las menciones de los personajes de NERTagger siempre serán más bajas que los de CoreNLP, ya que éste último no sólo cuenta cuando al personaje se le menciona por el nombre, sino que también puede detectar menciones puramente pronominales.

Las menciones de CoreNLP aparecerán fraccionadas según género, de modo que un personaje puede

tener 100 menciones masculinas, 3 femeninas, 0 neutras y 12 desconocidas (cuando no ha sido posible asignar ningún género a dicha mención). La intención de mostrar estos datos así es poder cuantificar cómo de seguro está el programa sobre el género de un personaje.

11. EVALUACIÓN DEL SISTEMA

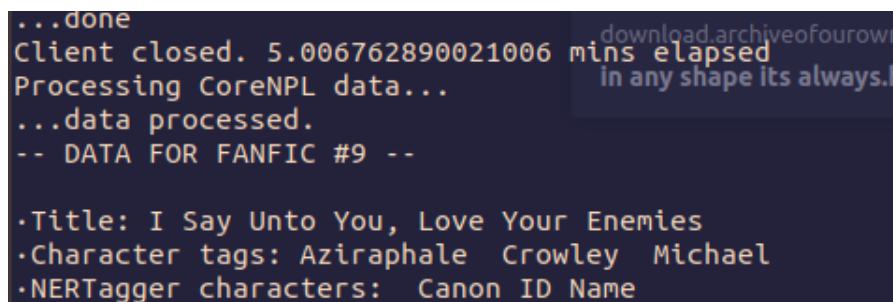
Vamos a evaluar el programa según la cantidad de personajes correctamente identificados y si su género concuerda con el género que el personaje realmente tiene en el texto. Los motivos que me llevaron a elegir cada fanfic para cada prueba serán explicados en la misma, aunque todos tienen en común que no son demasiado largos, lo cual ayuda tanto a que yo como CoreNLP no tardemos mucho en obtener la información que necesitamos de ellos.

La mayoría de información aparece transcrita en tablas vez de ser pantallazos del programa, para ahorrar espacio, pero la información aparece en pantalla con el mismo formato.

11.1. Prueba 1: Funciones básicas del programa con un texto largo (Fanfic 9)

Escogí este fanfic para la primera prueba porque tiene muchos personajes, algunos de los cuales sólo se les menciona una o dos veces, y con 8821 palabras es un texto medianamente largo, útil para recabar información sobre los personajes. Además, este fic particular tiene varias erratas, lo que también pondrá a prueba la capacidad del programa para identificar un personaje con el nombre ligeramente incorrecto.

Ejecutar el comando `fic_character_extractor 9`. En la figura ?? se muestran los metadatos del fanfic, y en las tablas 3 y 4 se han transcrito los personajes identificados por NERTagger y CoreNLProcessor.



```
...done
Client closed. 5.006762890021006 mins elapsed
Processing CoreNLP data...
...data processed.
-- DATA FOR FANFIC #9 --

·Title: I Say Unto You, Love Your Enemies
·Character tags: Aziraphale Crowley Michael
·NERTagger characters: Canon ID Name
```

Figura 26: Título y etiquetas de personaje del fanfic 9.

De entrada, tenemos los personajes de las etiquetas: Aziraphale, Crowley y Michael. Son los personajes que el autor decidió etiquetar, probablemente por considerarlos los más importantes, y los tres son identificados correctamente tanto por NERTagger como por CoreNLProcessor.

NERTagger ha identificado 35 personajes distintos, de los cuales 8 identifica como canon. Sus resultados, mostrados en la tabla 3, permite observar que lista en entradas distintas a personajes que son claramente el mismo pero con una errata (Ashemedai y Ashmedai, Azriaphale y Aziraphale), pero los identifica correctamente con el mismo personaje (Azriaphale y Aziraphale tienen ambos el identificador 4). Destacan

las entradas 'Lord Beezlebub' y 'Beezlebub': ambas tienen la misma errata, pero el primero no aparece identificado como personaje canon, mientras que el segundo sí. Esto probablemente sea por la distancia de edición y el tamaño de los nombres: 'Lord Beezlebub' son dos palabras y la más corta es Lord, con sólo 4 letras. Esto significa que el algoritmo explicado en la sección 9.1.1 sobre cuándo coinciden dos nombres habrá considerado que la distancia máxima de edición sólo puede ser 1. Cuando hay dos palabras en un nombre, como en este caso, se escoge la que tenga la distancia de edición más pequeña, que en este caso sería la distancia entre 'Beezlebub' y 'Beelzebub', que es 2, excediendo el límite de distancia. Sin embargo, cuando la única palabra en el nombre es 'Beezlebub' y se compara directamente con 'Beelzebub', al ser un nombre más largo la distancia máxima de edición es 3, y NERTagger por tanto lo identifica correctamente como canon.

También hay algunos personajes con nombres 'raros' como 'Did Gabriel' o 'Aziraphale never', en los que claramente NERTagger ha incluido en el nombre del personaje una palabra que no correspondía. Sin embargo, puesto que al menos parte del 'nombre' coincide con un personaje canon, son identificados correctamente.

Los 8 personajes que identifica como canon están correctamente identificados excepto 'Heaven', que confunde un personaje canon llamado 'Raven'. En el texto aparecen otros tres personajes canon que el programa no ha detectado, y sólo dos personajes no canon (mencionados cada uno sólo una vez).

CoreNLProcessor por su parte identifica 9 personajes canon, incluyendo a Ligur, que se le escapó a NERTagger. En la tabla 4 también podemos ver a los personajes Agnes Nutter y Newton Pulsifer, que en realidad no aparecen en el texto pero aparecen incorrectamente identificados como la versión canon de Agares y Beeton (de nuevo, debido a la distancia de edición). Todo el resto de personajes listados en la tabla son personajes reales que aparecen en el texto, y sus géneros están correctamente asignados en todos los casos excepto Beelzebub, que en esta historia tiene pronombres femeninos. También hay que destacar el personaje Laradiri, inventado por el autor y cuyo género no es definido en ningún momento, siendo referido únicamente con el pronombre neutro inglés *they*. Como consecuencia, sus menciones aparecen marcadas como neutras o 'desconocidas'.

Sólo hay un personaje canon y un personaje no canon que ninguno de los dos programas han detectado.

Otro problema claro es que CoreNLProcessor no parece capaz de consolidar correctamente los personajes que no son canon, apareciendo repetidos en vez de en una única entrada que recoja todas las menciones.

11.2. Prueba 2: Género distinto del canon (Fanfic 2856)

En este fanfic, el autor decidió cambiar el género de los protagonistas, por lo que en su historia los personajes Crowley y Aziraphale son dos mujeres. Es un relato con más de 3000 palabras, por lo que no debería de ser poca información.

Cambiar el género de personajes es bastante común en el género fanfic, y es éste uno de los motivos por los cuales analizar fanfiction puede ser tan interesante: los autores juegan con los personajes y exploran su psique desde muchos ángulos, incluido el de género. Nuestro programa debería identificar que, en este texto, la mayoría de menciones a Crowley y Aziraphale son femeninas.

```
...done
Client closed. 1.9697413285573324 mins elapsed
Processing CoreNPL data...
...data processed.
-- DATA FOR FANFIC #2856 --

·Title: Like the Sweet Apple
·Character tags: Aziraphale Crowley
```

Figura 27: Título y etiquetas de personaje del fanfic 2856.

En la tabla 6 podemos ver que la detección de género ha fallado totalmente. A pesar de que no son referidos jamás en masculino en todo el texto, CoreNLProcessor sigue asignándoles menciones masculinas de forma casi exclusiva.

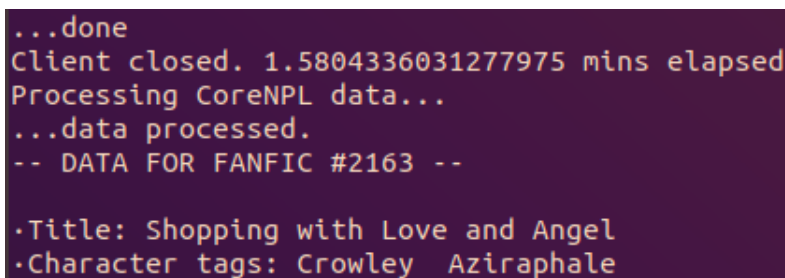
Por lo demás, las identificaciones no son incorrectas. Como se pueden comprobar con los metadatos de la figura ??, Crowley y Aziraphale son correctamente identificados en el texto por ambos programas. En la tabla 5 se muestra que además de los dos protagonistas ha identificado otros 16 nombres, pero sólo 'Eisheth' es realmente otro personaje en el texto ('Mesopotamia' aparece, pero es obviamente un lugar). También confunde 'Rather' con un personaje canon, Aziraphale (Canon ID = 4), probablemente porque uno de sus apodos es '*Brother* Francis', e inexplicablemente, confunde 'Azi Eisheth' con el personaje canon con ID = 38, cuyo nombre es 'Jeremey Wendsleydale' y que no se le menciona por ninguna parte en el texto.

CoreNLProcessor, a pesar del fracaso con el género, identifica correctamente a Aziraphale, Crowley y Adam como personajes canon, e incluso que Crowley en este relato adopta el nombre 'Zadkiel' brevemente. También identifica a Eisheth y a 'Serpent of Eden', que en principio tendría que haber sido vinculado con Crowley, al ser 'Serpent' uno de sus apodos. No añade ningún personaje que no aparezca en el texto.

11.3. Prueba 3: Personajes que no son nombrados (Fanfic 2163)

Decidí probar este fanfic porque los protagonistas Crowley y Aziraphale aparecen, pero son vistos desde la perspectiva de dos extrañas que no les conocen, y por tanto, no saben sus nombres. Además, este fanfic tiene tan sólo 876 palabras, con lo que a la falta de nombres se le añade una información limitada.

Ejecutamos el comando `fic_character_extractor 2163`:



```
...done
Client closed. 1.5804336031277975 mins elapsed
Processing CoreNPL data...
...data processed.
-- DATA FOR FANFIC #2163 --

·Title: Shopping with Love and Angel
·Character tags: Crowley Aziraphale
```

Figura 28: Título y etiquetas de personaje del fanfic 2163.

Ambos programas han identificado correctamente a Lauren y Olivia, los personajes originales del autor desde cuya perspectiva se cuenta este relato. Ambas son también correctamente identificadas como personajes que no son canon (Canon ID es 'NO'), y en 8 podemos ver que todas sus menciones son femeninas excepto las de una de las entradas repetidas de 'Angel Olivia', que tiene 24 menciones 'desconocidas'. Como en la prueba 2, tanto Crowley como Aziraphale son mujeres en esta historia, pero todas las menciones de Aziraphale son identificadas como masculinas. La buena noticia es que tanto NERTagger como CoreNLProcessor han sido capaces de identificar correctamente a Aziraphale por su apodo 'angel', y, curiosamente, en la tabla 7 se puede ver que NERTagger también ha identificado que 'Love' en este relato es un apodo cariñoso para un personaje. Sin embargo, también identifica incorrectamente la palabra 'Well' como refiriéndose a Aziraphale (probablemente por parecido con uno de sus apodos, 'Fell'). Esto significa que NERTagger ha identificado 11 personajes, de los cuáles sólo 4 son correctos (aunque un humano puede detectar fácilmente qué nombres son los falsos positivos), mientras que CoreNLProcessor ha identificado 4 personajes distintos, de los cuales dos son correctos y uno está correctamente identificado con su correspondiente canon, pero el género es incorrecto.

Como se puede ver en los metadatos de la figura ??, ambos programas han identificado correctamente a Aziraphale pero ninguno a Crowley; no es de extrañar, ya que ni su nombre ni ninguno de sus apodos es mencionado en todo el texto, y al propio Aziraphale sólo se le ha identificado por su apodo 'angel'. Sin embargo, NERTagger encuentra el nombre 'Love' y sabe que es un nombre; simplemente no tenía forma de saber que se refería a Crowley. CoreNLProcessor no ha sido capaz de identificar 'Love' como un nombre.

Canon ID	Name	Mentions
NO	Lord Beezlebub	2
NO	Lilith	1
14	Did Gabriel	1
NO	Ashmedai	11
4	Aziraphale	42
NO	Laradiri	16
NO	Every Woman	1
NO	Marut	1
NO	Joe	1
NO	Amides	1
NO	Dr Dudders	1
NO	History	1
14	Gabriel	1
NO	Butter	1
NO	Alisha	1
4	Aziraphale never	1
NO	Mrs Beeton	1
NO	Richard	1
NO	Heavenly	1
4	Azriaphale	1
4	Mr Fell	2
NO	Seamus Blackley	1
24	Michael	1
NO	Do NOT	1
10	Death	1
14	Sir Yes Sir Gabriel Sir	1
13	Heaven	1
NO	Ashemedai	1
9	Dagon	1
NO	Pride	1
8	Crowley	45
NO	So Below	1
NO	Any	1
NO	Inside	1
15	God	1
NO	Got	1
5	Beezlebub	1
NO	Word	1
8	Mr Crowley	1
4	Aziraphale	42
8	Crowley One	1
15	God Herself	1
NO	Mr Solomons	3
NO	Cookery Reformed	1

Cuadro 3: Resultados de la ejecución de fic_character_extractor para analizar fanfic 9

Canon ID	Name	MALE	FEMALE	NEUTRAL	UNKNOWN	Other names
1	Agnes Nutter	0	1	0	0	Agares
4	Aziraphale	147	0	4	0	aziraphale, Azriaphale, Fell, azirphale, azriaphale, consume Aziraphale, Azirphale
5	Beelzebub	4	0	3	0	Beezlebub
8	Crowley	104	0	7	0	crawley, crowley, Shop Crowley, Crawley
14	Gabriel	6	0	0	0	
21	Ligur	1	0	0	0	
24	Michael	2	0	0	0	
25	Newton Pulsifer	0	1			Beeton
36	Uriel	1	0	0	0	
NO	Bentley	1	0	0	1	
NO	Somolons	3	0	0	0	
NO	Laradiri	0	0	0	1	
NO	Eden	0	0	0	1	
NO	Petronius	1	0	0	0	
NO	Ashmedai	0	0	0	1	
NO	Angelo	1	0	0	0	
NO	Ashemedai	0	0	0	4	
NO	Laradiri	0	0	0	1	
NO	Angelo	1	0	0	0	
NO	Richard	1	0	0	0	
NO	Ashmedai	0	0	0	3	
NO	Jane Austen	0	0	0	1	
NO	Richards	1	0	0	0	
NO	Marut	1	0	0	0	
NO	Joe	5	0	0	0	
NO	Laradiri	1	0	0	0	
NO	Alisha	0	2	0	0	
NO	Solomons	1	0	0	0	
NO	Perkins	1	0	0	0	
NO	Hannah Glasse	0	1	0	0	
NO	Jane Austen	0	1	0	0	
NO	Seamus Blackley	1	0	0	0	
NO	Jane Austen	0	2	0	0	
NO	Laradiri	0	0	0	42	
NO	Ashmedai	0	0	0	13	
NO	Ashmedai	0	0	0	2	

Cuadro 4: Resultados de la ejecución de fic_character_extractor para analizar fanfic 9

Canon ID	Name	Mentions
NO	Mesopotamia	1
NO	How	1
NO	Green	3
NO	out	1
NO	Eisheth	3
4	Aziraphale	48
NO	Unhand	1
38	Azi Eisheth	1
NO	Always	1
8	Crowley	62
NO	See	1
NO	No	1
4	Rather	3
NO	gorgeous	1
NO	Just	1
NO	Villain	1
NO	Oh	4
NO	Shouldn	1

Cuadro 5: Resultados de la ejecución de fic_character_extractor para analizar fanfic 2856

Canon ID	Name	MALE	FEMALE	NEUTRAL	UNKNOWN	Other names
0	Adam Young	1	0	0	0	Adam
4	Aziraphale	217	0	1	0	
8	Crowley	445	0	4	0	Crowley Zadkiel
NO	Eisheth	0	0	0	2	
NO	Serpent of Eden	0	0	0	1	

Cuadro 6: Resultados de la ejecución de fic_character_extractor para analizar fanfic 2856

Canon ID	Name	Mentions
NO	Really inaccurate	1
NO	Love	1
4	Angel	8
NO	Yeah	1
4	Well	1
NO	sort	1
NO	Did	1
NO	Oh	3
NO	Really	1
NO	Pardon	1
NO	Olivia	7
NO	Hey	1
NO	Lauren	3
NO	Um	1

Cuadro 7: Resultados de la ejecución de fic_character_extractor para analizar fanfic 2163

Canon ID	Name	MALE	FEMALE	NEUTRAL	UNKNOWN	Other names
4	Aziraphale	7	0	0	0	Angel
NO	Angel Olivia	3	0	0	0	
NO	Olivia	0	7	0	0	
NO	Lauren	0	4	0	0	
NO	Olivia	0	1	0	0	
NO	Olivia	0	1	0	0	
NO	Angel Olivia	0	0	0	24	
NO	Angel Olivia	0	6	0	0	

Cuadro 8: Resultados de la ejecución de fic_character_extractor para analizar fanfic 2163

12. CONCLUSIONES

En este trabajo se ha desarrollado un sistema de extracción y análisis de textos de internet, utilizando técnicas de *scraping* y procesamiento de texto natural.

En primer lugar se ha creado un corpus de relatos pertenecientes al género fanfiction a partir de los textos alojados en la web [Archive of our Own](#), y se ha utilizado este corpus para realizar pruebas, entrenamientos y experimentos que ayudaron en el desarrollo de un sistema de procesamiento de texto natural. Se han seleccionado textos en inglés que se basaran en el libro *Good Omens* (Neil Gaiman y Terry Pratchett, 1990) y que tuvieran una cantidad mínima de palabras para asegurar que la obra estaba compuesta principalmente por texto (en vez de imágenes o audio), y se ha desarrollado un sistema de scrapers capaces de extraer los relatos de la web que tuvieran esas características. El resultado ha sido un corpus compuesto por archivos HTML para cuyo acceso y manejo se han desarrollado diversos objetos y funciones en python, encapsuladas en un módulo que he llamado *fanfic_util*. Además, aprovechando los metadatos de los relatos del corpus se han creado tres datasets, cada uno consistente en relatos de un sólo capítulo centrados en romance, amistad y enemistad, respectivamente. Estos datasets se utilizan para experimentos durante la búsqueda de estrategias para la identificación de relaciones.

En el procesamiento de texto natural se han creado dos algoritmos de identificación de entidades, uno basado en naïve Bayes y otro en un modelo de regresión logística. Ambos fueron desarrollados con la librería NLTK de python (en conjunto con el módulo *megam* en el caso del modelo de regresión logística), y entrenados con el corpus *Groninger Meaning Bank*, que fue creado expresamente para entrenar algoritmos para identificar entidades en textos en inglés. La versión con el modelo de regresión logística consiguió mejores resultados, como se puede ver en la figura 12, con lo que se decide utilizar esta versión bajo el nombre *NER_tagger*.

Para la parte de extracción de relaciones del procesamiento de texto natural, se han explorado varias estrategias. Usando *sci-kit learn* y *gensim* se han creado algoritmos de clustering y un modelo LDA, con la esperanza de que fueran capaces de clasificar correctamente a qué dataset pertenecía cada relato (romance, amistad o enemistad). La idea era que si podían clasificarlos correctamente, las características más relevantes de cada cluster o del modelo podrían utilizarse para refinar el algoritmo y crear una versión que pudiese identificar romance, amistad o enemistad incluso en textos largos donde no fueran el tema central, pero lamentablemente no se obtuvieron resultados mucho mejores que el azar.

La última estrategia para identificar relaciones consistía en explotar la correferencia pronominal en frases donde se nombraran a los personajes, utilizando para ello el servidor de Stanford CoreNLP mediante su biblioteca para python, *Stanza*. La idea era identificar todas las menciones de un personaje en un texto,

incluso aquellas que sólo se les mencionaran por el pronombre (*he, she, they*), aumentando por tanto la cantidad de frases relevantes para el algoritmo. CoreNLP tiene, entre otras funciones, la capacidad de resolver esta correferencia pronominal en textos, por lo que se ha creado un conjunto de funciones para enviar peticiones de forma sencilla y controlada a su servidor, sin inundarlo, y manejar errores de red de forma que se pierda la menor cantidad posible. Todas estas funciones han sido encapsuladas en un módulo que he llamado *corenlp_util*. Mediante este módulo los relatos de los datasets de romance, amistad y enemistad han sido procesados y etiquetados, y los resultados se han guardado en un archivo con el que se realizaron aún más experimentos para buscar alguna forma de identificar relaciones entre entidades. Estos últimos experimentos consistieron en buscar patrones relacionados con los verbos, adjetivos, bigramas o trigramas más frecuentes en las frases que mencionasen al menos a dos personajes, pero tampoco se pudo encontrar ningún patrón que fuese más relevante que el azar.

Decidí entonces dejar el desarrollo de un algoritmo de identificación de relaciones sociales entre personajes para el futuro, y me centré en el algoritmo de identificación de personajes. Ya tenía uno, basado en regresión logística, pero como durante el desarrollo de las pruebas con CoreNLP para realizar correferencia también tuve que manejar las funciones de CoreNLP para identificación de entidades, esto resultó en un segundo algoritmo de identificación de personajes que he llamado *CoreNLPPDataProcessor*, encapsulado también dentro del módulo *corenlp_util*. A diferencia de la versión basada en un modelo de regresión logística, esta versión puede identificar no sólo el nombre sino también el género del personaje, además de aprovechar la resolución de correferencia para identificar más menciones.

El resultado final por tanto, ha sido un programa que he llamado *fic_character extractor*, que utiliza ambos algoritmos de identificación de personajes para extraer información de un texto. Este programa ofrece al usuario la posibilidad de extraer personajes de cualquiera de los relatos del corpus creado en la primera parte del proyecto, dando la posibilidad de analizar uno al azar o introducir el identificador numérico de un relato particular. Entonces, el programa utiliza tanto *NER_tagger* como *CoreNLPPDataProcessor* para analizar el texto y mostrar al usuario el nombre de los personajes, si son canon o no, el número de menciones y, en el caso de este último, su género.

De los objetivos originales expuestos en la introducción (5), se han logrado con éxito cuatro de los seis objetivos, uno que ha fallado y el último, que se ha logrado en parte:

1. Se ha conseguido extraer un conjunto de relatos de la web [Archive of Our Own](#) sobre los que utilizar técnicas de extracción de información, mediante la creación de dos scrapers que seleccionan y descargan los archivos allí alojados.
2. Se ha creado el módulo *fanfic_util* para manejar y extraer información de los archivos HTML de

AO3, de modo que los algoritmos de procesamiento de texto natural pueden acceder tanto al texto puro como a los metadatos de cada archivo de forma simple.

3. Utilizando las herramientas de *fanfic_util*, se han creado tres datasets sobre los que realizar pruebas y experimentos.
4. Se han desarrollado dos algoritmos que identifican personajes en textos de ficción, uno basado en un modelo de regresión logística creado con NLTK y *megam* y entrenado con *Groningen Meaning Bank* para identificar entidades en inglés, y otro que aprovecha las capacidades para identificar entidades de CoreNLP, a través de la biblioteca *Stanza*. Además de identificar sus nombres, también pueden identificar si los personajes son originales del autor fan o si ya existían en la obra original.
5. No se ha conseguido desarrollar un algoritmo de identificación de relaciones entre entidades, aunque se han probado diversas técnicas no-supervisadas y las capacidades de resolución de co-referencia de CoreNLP.
6. No se ha conseguido crear un programa que utilice tanto identificación de entidades y relaciones para los personajes y relaciones en los relatos recogidos de AO3, y mostrarlas al usuario. Sin embargo, sí se ha conseguido un programa que utiliza dos métodos de identificación de entidades para extraer los personajes de un texto, así como información relevante sobre los mismos (número de menciones, género, si son canon). Por lo tanto, este objetivo se ha conseguido parcialmente.

El objetivo final de este proyecto, que era el desarrollo de una herramienta para ayuda al análisis literario, ha quedado conseguido tan sólo en parte. La identificación automática de los personajes de un conjunto de relatos puede ser muy útil para saber qué personajes son más populares entre los fans y pueden dar pie a teorías interesantes según si se corresponden o no con los protagonistas de la obra original, o dividiendo la comunidad fan a estudiar en subcategorías: si se tiene datos sobre los autores, se podría buscar si todos los personajes son populares de forma homogénea en toda la comunidad o si hay algunos que son más populares entre los hombres, si algunos son más populares entre los escritores que además de ser fans de esta obra lo son también de *Harry Potter*, etc. Como el programa final también arroja datos sobre cosas como el género de los personajes y si son canon, un investigador podría utilizar estos datos para explorar, por ejemplo, si hay tendencias entre los fans a interpretar a ciertos personajes con un género diferente al de la obra original, o qué tipos de autores tienen más tendencia a escribir personajes originales. Incluso, se podrían investigar qué personajes no originales se vuelven populares en toda la comunidad, en vez de quedar limitado a los fanfic de un único autor.

Sin embargo, la falta de información sobre las relaciones entre personajes limita mucho los análisis posibles. Sin tener una idea de qué personajes tienen relación de enemistad o romances, es difícil dilucidar

qué personajes son interpretados como los "buenos" o los "malos" de una historia, con lo que un investigador tendría que leer los relatos en más profundidad para sacar una conclusión sobre las tendencias de los fans relacionadas con la redención o corrupción de personajes, y sobre la moralidad que les transmiten ciertos arcos o características de personajes.

Para consultar cualquier aspecto del código e implementación del proyecto, se puede visitar el repositorio del mismo en [GitHub](#) (Usuario: *mariaGnlz*, repositorio: *Fanfic_ontology*).

Durante todo este proceso he tenido que aprender las bases de la extracción de información, técnicas de *machine learning* y cómo crear, organizar y preprocesar un conjunto de archivos para que su información sea comprensible para los algoritmos que los utilizan como entrada. En cierto aspecto este proyecto me ha hecho perderle el miedo al procesado del lenguaje humano, que siempre había visto como algo extremadamente complicado, y aunque ciertamente no es una tarea trivial, he podido ver de primera mano que existen métodos bien establecidos para la extracción de información y que pueden ser aprendidos y entendidos. El modelo de regresión logística utilizado en la sección 9.1.1, por ejemplo, me llevó a repasar funciones y gradientes para poder entender su base matemática, y me di cuenta de que sí, es complejo, pero no es magia.

Mientras me encontraba con que la parte relacionada con el procesado de lenguaje natural en sí no era tan complicada como temía, los problemas relacionados con el manejo de archivos HTML y extraer el texto puro me pillaron por sorpresa en lo retorcidos y frustrantes que podían llegar a ser. Detalles como puntos y coma que destrazan el formato de una base de datos, o la etiqueta HTML que utilizaba para extraer un cierto metadato funciona en la mayoría de archivos pero está misteriosamente ausente en otros, no hizo más que recordarme que la mayoría del esfuerzo en ciencia de datos suele ir a limpiar los datos y darles un formato uniforme.

El diseño de ciertas del proyecto también es algo que hubiese planificado mejor; la clase *Fanfic* de la sección 8.2 acaba siendo la unidad de información básica, pero se desarrolló orgánicamente a medida que el proyecto avanzaba, especialmente mientras buscaba alguna forma de aprovechar las funciones de CoreNLP para identificar relaciones en el texto (ya que es la parte en la que los metadatos de un fanfic adquirieron más importancia). En retrospectiva, crear una clase que encapsule todas las características de un fanfic tenía mucho sentido para el proyecto, y pensar que simplemente con el texto puro de cada obra iba a ser suficiente fue un poco ingenuo. Todo el módulo *fanfic_util* podría haberse beneficiado de haber planificado la clase *Fanfic* desde el principio, por no hablar de ahorrarme trabajo.

12.1. Trabajos futuros

Un añadido evidente para este proyecto sería mejorar el manejo de archivos HTML, introducidos en una base de datos que facilite su filtrado según sus etiquetas o autor o cualquiera de sus metadatos, ya que ahora mismo no tiene un mecanismo de búsqueda generalizado, y los *datasets* fueron creados mediante comandos de python en la terminal. Otra mejora es generalizar la canonicalización de personajes. Este proyecto utiliza una base de datos con los personajes de *Good Omens* para decidir la canonicidad de los personajes de un fanfic dado, por lo que ahora mismo sólo los fanfics de *Good Omens* pueden tener sus personajes marcados como canon. Sin embargo, se podría crear un método que consultara el título de la obra original en los metadatos del fanfic y comprobase si hay alguna wiki dedicada a esta obra en internet, y usar la página de personajes de dicha wiki para decidir qué personajes del fanfic son canon o no. De esta manera el proceso de canonicalización funcionaría para cualquier fanfic cuya obra original tenga una wiki.

El proceso de consolidación de menciones en personajes también podría mejorarse, ya que ahora mismo es un proceso basado en la distancia de edición de los nombres y, en el caso del extractor de personajes que utiliza CoreNLP (9.1.2), también en el género. Un programa más sofisticado podría utilizar más información del contexto de la mención para captar más características de un personaje particular (títulos, nombre y apellidos, especie, país); en otras palabras, adoptar una estrategia que se centre en los personajes como entidad (Wick et al., 2009) que trate de rellenar una "ficha" para cada candidato a personaje podría suponer una mejora para todo el proceso.

Para continuar con los esfuerzos de identificación de relaciones sociales se podría explorar el entrenar un modelo de aprendizaje supervisado, aprovechando las etiquetas de los metadatos de un fanfic para etiquetar la relación principal de un relato. También se podrían explorar técnicas de aprendizaje no supervisado distintas a las ya vistas, como las redes neuronales (Peng et al., 2017), que pueden acceder a una mayor cantidad de contexto a la hora de decidir si existe una relación entre dos entidades particulares.

De cara al usuario, una mejora sería modificar la entrada del programa de modo que acepte un link de una obra de AO3, evitando así que tenga que descargar el archivo y configurar el *path* para que el programa lo encuentre.

A. ANEXO: CÓDIGO DEL SISTEMA DE SCRAPERS

A.1. link_scraper

```
1
2  #!/bin/bash/python3
3
4  import requests, time
5  from bs4 import BeautifulSoup
6
7  def check_for_text(blurb): #returns true if the fic contains at least 40 words per chapter on
    average
8  contains_text = True
9
10 #print(blurb.find('dd', class_='words').text) #debug
11 num_words = (blurb.find('dd', class_='words').text).replace(',','') #take away the comma that
    marks the thousands
12
13 if num_words == '': contains_text = False #there's a bug in A03 that makes some works appear
    with no word count (not '0'; it doesn't show a number at all)
14 else:
15     num_words=int(num_words)
16
17 if num_words == 0: contains_text = False
18 else:
19     num_chapters = int(((blurb.find('dd', class_='chapters').text).split('/')[0])) #chapters are
        displayed as '# of current chapters / total # of chapters', we only want the current
        chapters
20     #print(num_words/num_chapters) #debug
21
22 if (num_words/num_chapters) < 40:
23     contains_text = False
24
25
26 #if not contains_text: print('in: ',((blurb.find('h4')).find('a'))['href']) #debug
27 return contains_text
28
29
30 def get_work_links(page_link):
31     page = requests.get(page_link) #get first page of the archive
32     soup = BeautifulSoup(page.content, 'html.parser')
33
34     #figure out how many pages in total there are
35     page_list = (soup.find(class_='pagination actions')).find_all('li')
36     number_of_pages = int(page_list[len(page_list)-2].text) #there are number_of_pages pages in
        total
37
38     #get work links in all pages
39     work_links = []
40     discarded_links = []
```

```

41 current_page = 1
42 while current_page < number_of_pages:
43     blurbs = soup.find_all(class_='work blurb group')
44     #print('current page: ', current_page) #debug
45
46     for blurb in blurbs:
47         #filter out fics that don't contain text
48         contains_text = check_for_text(blurb)
49
50         work_id = (blurb.find('h4')).find('a')
51         if contains_text: work_links.append('https://archiveofourown.org'+work_id['href'])
52         else:
53             discarded_links.append('https://archiveofourown.org'+work_id['href'])
54         #print('out:', work_id['href'])
55     #end 'for blurb' loop
56
57     current_page +=1
58     next_page_link = page_link.replace('&page=1&', '&page='+str(current_page)+'&')
59     while True: #wait out if too many requests
60         page = requests.get(next_page_link)
61
62         if page.status_code == 429: #Too Many Requests
63             print('Sleeping...')
64             time.sleep(120)
65             print('Woke up')
66
67         else: break
68
69     soup = BeautifulSoup(page.content, 'html.parser')
70
71     #end while loop
72
73     #get work links in last page (the loop won't catch it)
74     blurbs = soup.find_all(class_='work blurb group')
75
76     for blurb in blurbs:
77         #filter out fics that don't contain text
78         contains_text = check_for_text(blurb)
79
80         work_id = (blurb.find('h4')).find('a')
81         if contains_text == True: work_links.append('https://archiveofourown.org'+work_id['href'])
82         else: discarded_links.append('https://archiveofourown.org'+work_id['href'])
83     #end 'for blurb' loop
84     print('Current page: ', current_page) #debug
85     return work_links, discarded_links
86
87
88 start = time.time()
89 work_links, discarded_links = get_work_links('https://archiveofourown.org/tags/Good%20mens
    %20-%20Neil%20Gaiman%20*a*%20Terry%20Pratchett/works?commit=Sort+and+Filter&page=1&utf8=%
    E2%9C%93&work_search%5Bcomplete%5D=&work_search%5Bcrossover%5D=&work_search%5Bdate_from%5D

```

```

=&work_search%5Bdate_to%5D=&work_search%5Bexcluded_tag_names%5D=Fanart %2CPodfic&
work_search%5Blanguage_id%5D=en&work_search%5Bother_tag_names%5D=&work_search%5Bquery%5D=&
work_search%5Bsort_column%5D=revised_at&work_search%5Bwords_from%5D=&work_search%5
Bwords_to%5D=')
90 end = time.time()
91
92 print('Time: ', (end-start)/60, ' mins', '\nNumber of fics: ', len(work_links), '\nDiscarded links:
    ', len(discarded_links)) #debug
93
94 fic_work_links = open('./fic_work_links.txt', 'w')
95
96 for link in work_links:
97     fic_work_links.write(link+'\n')
98
99     fic_work_links.close()
100
101 discarded_works = open('./discarded_works.txt', 'w')
102 for link in discarded_links:
103     discarded_works.write(link+'\n')
104
105     discarded_works.close()

```

A.2. file_scraper

```
1  #!/bin/bash/python3
2
3  import requests, time, sys
4  import urllib.request
5  from urllib.error import URLError, HTTPError, ContentTooShortError
6  from bs4 import BeautifulSoup
7
8  ### VARIABLES ###
9  HTML_FICS_PATH = '/home/maria/Documents/Fanfic_ontology/TFG_fics/html/'
10 HTML_FIC_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/html_fic_paths.txt'
11 DELETED_FICS = []
12
13 ### FUNCTIONS ###
14 def get_deleted_fics():
15     f = open(HTML_FICS_PATH+'deleted.txt', 'r')
16     lines = [line[:-1] for line in f.readlines()]
17     f.close()
18
19     index_lines = [line.split(' ') for line in lines if lines.index(line)%2 != 0]
20     DELETED_FICS = [int(line[len(line)-1]) for line in index_lines]
21
22     return DELETED_FICS
23
24 def get_work_links_from_file():
25     link_file = open('fic_work_links.txt', 'r')
26
27     work_links = [line[:-1] for line in link_file.readlines()] #take out the \n at the end of the
28     line
29     link_file.close()
30
31     return work_links
32
33 def write_out_file(link, reason, index):
34     out_file = open(HTML_FICS_PATH+'deleted.txt', 'a')
35     out_file.write(link+'\nReason: '+reason+' Index: '+str(index)+'\n')
36     out_file.close()
37
38 def get_html_link(page):
39     soup = BeautifulSoup(page.content, 'html.parser')
40
41
42     if soup.find('title').text == '\n          New\n          Session\n          |\n          Archive
43         of Our Own\n          ':
44         #the work is private and shouldn't be downloaded
45         html_link=''
46
47     else:
48         all_links = (soup.find(class_='download')).find_all('a')
```



```

48  html_link = all_links[len(all_links)-1]
49
50  return html_link
51
52
53  def download_works_in_range(work_links, start, end):
54  num_deleted = 0
55
56  i = start
57  while i<end:
58  deleted = False
59
60  while True:
61  page = requests.get(work_links[i])
62
63  if page.status_code == 429: #Too Many Requests
64  print('Sleeping...')
65  time.sleep(120)
66  print('Woke up')
67
68  elif page.status_code ==404: #Page Not Found
69  print('Deleted work at '+work_links[i])
70  deleted = True
71  num_deleted += 1
72  write_out_file(work_links[i], 'deleted', i)
73
74  break
75
76  else: break
77
78  if not deleted: #only continue with iteration if the work is still online
79
80  html_link = get_html_link(page)
81
82  if html_link == '': #this work is private
83  num_deleted += 1
84  write_out_file(work_links[i], 'private', i)
85  print('Private work at '+work_links[i])
86
87  else:
88  download_link = 'https://archiveofourown.org'+html_link['href']
89  #print(download_link) #debug
90
91  print('Downloading ',i,'of ',end,'. . .')
92
93  try:
94  fanfic_path, _ = urllib.request.urlretrieve(download_link, HTML_FICS_PATH+'gomensfanfic_'+str(
      i)+'.html')
95  #print(fanfic_path) #debug
96
97  #Write path to fanfic on html_fic_paths.txt

```

```

98  html_list = open(HTML_FIC_LISTING_PATH, 'a')
99  html_list.write(fanfic_path+'\n')
100 html_list.close()
101
102 except HTTPError as e:
103     print('HTTPError ',e.code,': ',e.reason,'\nIn fanfic number ',i)
104     if e.code == 429: #Too Many Requests
105         i-=1 #Will re-attempt iteration from the beginning
106
107 except ContentTooShortError as e:
108     print('ContentTooShortError ',e.code,': ',e.reason,'\nIn fanfic number ',i)
109     write_out_file(work_links[i], e.reason, i)
110
111 except URLError as e:
112     print('URLError ',e.code,': ',e.reason,'\nIn fanfic number ',i)
113     write_out_file(work_links[i], e.reason, i)
114
115 except (IOError, OSError) as e:
116     print('IOError/OSError ',e.errno,': ',e.strerror,'\nIn fanfic number ',i)
117     write_out_file(work_links[i], e.strerror, i)
118
119 except Error as e:
120     print('Error ',e.errno,': ',e.strerror,'\nIn fanfic number ',i)
121     write_out_file(work_links[i], e.strerror, i)
122
123
124 #end if not deleted
125
126 i+=1
127 #end while loop
128
129 num_fics = len((open(HTML_FIC_LISTING_PATH, 'r')).readlines())
130 return num_deleted, num_fics #return number of deleted fics and fics successfully downloaded
131
132
133 ### M A I N ###
134
135 work_links = get_work_links_from_file()
136 num_fics = len((open(HTML_FIC_LISTING_PATH, 'r')).readlines())
137 num_deleted = len(get_deleted_fics())
138
139 if len(sys.argv) == 3:
140     start_index = int(sys.argv[1])
141     end_index = int(sys.argv[2])
142     #print(type(start_index), end_index) #debug
143
144     start = time.time()
145     new_deleted, new_fics = download_works_in_range(work_links, start_index, end_index)
146     end = time.time()
147

```

```

148 print('Successfully downloaded ', (new_fics-num_fics), ' fanfics in ', (end-start)/60, ' minutes
      to '+HTML_FICS_PATH)
149 print('Deleted fics: ', new_deleted, ', total deleted fics: ', new_deleted+num_deleted)
150
151 elif len(sys.argv) == 2:
152 if sys.argv[1] == 'd':
153 print('Downloaded ', num_fics, ' out of ', len(work_links), ' total')
154 print('Deleted fics: ', num_deleted)
155 latest_path = (open(HTML_FIC_LISTING_PATH, 'r')).readlines()
156 print('Path of latest download: ', latest_path[len(latest_path)-1])
157
158 else: print('Error. Correct usage: check_correct.py \ncheck_correct.py [start_index] [
      end_index] \ncheck_correct_py d')
159
160
161 elif len(sys.argv) == 1:
162 start = time.time()
163 #num_deleted, num_fics = download_works_in_range(work_links, 5147, 6000)
164 #num_deleted, num_fics = download_works_in_range(work_links, 0, 10) #debug
165 end = time.time()
166
167 print('Successfully downloaded ', (new_fics-num_fics), ' fanfics in ', (end-start)/60, ' minutes
      to '+HTML_FICS_PATH)
168 print('Deleted fics: ', new_deleted, ', total deleted fics: ', new_deleted+num_deleted)
169
170 else:
171 print('Error. Correct usage: \ncheck_correct.py \ncheck_correct.py [start_index] [end_index] \
      ncheck_correct_py d')

```

B. ANEXO: CÓDIGO DE fanfic_util

```
1  #!/usr/bin/bash/python3
2
3  from bs4 import BeautifulSoup
4  from stanza.server import Document
5
6  import string, html2text, sys
7
8
9  ### VARIABLES ###
10 FIC_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/html_fic_paths.txt'
11 #TXT_LISTING_PATH = '/'
12 #SAVE_TXT_PATH = '/home/maria/Documents/pruebasNLTK/trial_e_fics/'
13 #TXT_LISTING_PATH = '/home/maria/Documents/pruebasNLTK/trial_e_txt_paths.txt'
14
15 ERRORLOG = '/home/maria/Documents/Fanfic_ontology/TFG_logs/fanfic_util_errorlog.txt'
16
17 ### FUNCTIONS ###
18 #def clean_text(text, chapter_titles):
19 def clean_text(text, num_chapters, num_fic):
20     headers_and_footers = ['See the end of the chapter for more notes', 'See the end of the
        chapter for notes', 'Summary', 'Chapter Summary', 'Chapter Notes', 'Notes', 'Chapter End
        Notes']
21     #headers_and_footers.extend(chapter_titles)
22     if num_chapters == 0: num_chapters += 1
23
24     errorstr = '?\n'
25     chapters = []
26     for i in range(0,num_chapters):
27         header1_index = text.find('## ')
28
29         if header1_index < 0: #esto no deberia pasar
30             print("Error on fanfic ", num_fic, ": menos de 0")
31             errorstr = "menos de 0\n"
32         else:
33             header2_index = text[header1_index+3:].find('## ')
34             chapters.append(text[header1_index:header2_index])
35
36     #print(header1_index, header2_index) #debug
37
38     text = text[header2_index:]
39     #print(text[:100]) #debug
40     #print("\n N E X T   C H A P T E R") #debug
41
42     chapter_text = ''
43     clean_chapters = []
44     for chapter in chapters:
45         for line in chapter.splitlines():
46             if line not in headers_and_footers and '>' != line[:2]: chapter_text += line+'\n'
47
```

```

48 clean_chapters.append(chapter_text)
49 chapter_text = ''
50
51 if len(clean_chapters) != num_chapters: #something went wrong
52     print("Chapters of fic number ", num_fic, " were improperly processed") #debug
53     f = open(ERRORLOG, 'a')
54     ficid = "Num fic: "+str(num_fic)+"\n"
55     ficchapters = str(num_chapters)+"\n"
56     f.write(ficid)
57     f.write(ficchapters)
58     f.write(text[:10000])
59     f.write("=====\n")
60     f.close()
61
62
63 #print(fic_text[:10000]) #debug
64
65 return clean_chapters
66
67 def remove_metadata(text):
68     index1 = text.find('See the end of the chapter for more notes')
69     index2 = text.find('See the end of the chapter for notes')
70     index3 = text[3:].find('## ')
71     chapter_header_indexes = [index1, index2, index3]
72
73     #print(text[:1000]) #debug
74
75     while -1 in chapter_header_indexes: chapter_header_indexes.remove(-1) #Remove unvalid indexes
76
77     fic_text = ''
78
79     if len(chapter_header_indexes) == 1: fic_text = text[chapter_header_indexes[0]:]
80     else:
81         index = min(chapter_header_indexes)
82         fic_text = text[index:]
83
84
85     #print(fic_text[:1000]) #debug
86     #print(chapter_header_index1, chapter_header_index2, chapter_header_index3) #debug
87
88
89     """ #debug
90     f = open('no_metadata.txt', 'w')
91     f.write(fic_text)
92     f.close()
93     """
94
95     return fic_text
96
97 def get_chapterised_fic(path, num_fic): #Transforms the HTML file in a list of chapters (a
    list of str)

```

```

98 page = open(path, 'r').read()
99 soup = BeautifulSoup(open(path, 'r'), 'html.parser')
100
101 #Get number of chapters with BeautifulSoup
102 chapter_titles = ['#' + header.text for header in soup.find_all('h2', class_='heading')]
103 num_chapters = len(chapter_titles)
104
105 if num_chapters == 0: #this fic only has one chapter
106 title = soup.find('h1').text
107 chapter_titles = ['#' + title]
108
109 #print(chapter_titles) #debug
110
111
112 #Take HTML tags out with HTML2text
113 to_text = html2text.HTML2Text()
114 to_text.ignore_images = True
115 to_text.ignore_links = True
116
117 text = to_text.handle(page)
118
119 """ #debug
120 f = open('htmlfic.txt', 'w')
121 f.write(text)
122 f.close()
123 """
124
125 #Take author notes and metadata out
126 fic_text = remove_metadata(text)
127 chapterised_fic = clean_text(fic_text, num_chapters, num_fic)
128
129 """
130 print(len(chapterised_fic)) #debug
131 for chapter in chapterised_fic: #debug
132 print(chapter[:100])
133 print(". . .")
134 print(chapter[-100:])
135
136 #debug
137 f = open('text.txt', 'w')
138 f.write(fic_text)
139 f.close()
140 """
141
142 return chapterised_fic
143
144 def get_fanfics(dataset, start, end, slicing): #gets the paths to the fics, opens them
145 #and stores them in chapterised form in a list of Fanfic objects
146 paths_file = open(FIC_LISTING_PATH, 'r')
147 fic_paths = [line[:-1] for line in paths_file.readlines()]
148 paths_file.close()

```

```

149
150     if slicing: fic_paths = fic_paths[start:end] #slices the list, if not all of it is required
151
152     fic_list = []
153     for path in fic_paths:
154         num_fic=int((path.split('_')[3]).split('.')[0])
155         chapterised_fic = get_chapterised_fic(path, num_fic)
156         fic_list.append(Fanfic(num_fic, dataset, chapterised_fic, None, None, None))
157
158     return fic_list
159
160     ### CLASSES ###
161
162     class Fanfic():
163     def __init__(self, index, dataset, chapters, annotations, characters, sentences):
164         self.index = index
165         self.dataset = dataset
166         self.chapters = chapters
167         self.annotations = annotations
168         self.characters = characters
169         self.sentences = sentences
170
171     def set_chapters(self, new_chapters):
172         self.chapters = new_chapters
173
174     def get_chapter(self, index):
175     return self.chapters[index]
176
177     def get_string_chapters(self):
178         chaps = ''
179         for chapter in self.chapters: chaps += chapter+'\n'
180
181
182     return chaps
183
184     def set_annotations(self, ann):
185         self.annotations = ann
186
187     def set_characters(self, new_characters):
188         self.characters = new_characters
189
190     def set_sentences(self, new_sentences):
191         self.sentences = new_sentences
192
193
194     class FanficGetter():
195     def __init__(self):
196         self.dataset = 'GENERAL'
197
198     def get_fanfics_in_range(self, start_index, end_index):
199         fic_list = get_fanfics(self.dataset, start_index, end_index, True)

```

```

200
201
202     return fic_list
203
204     def get_fanfics_in_list(self):
205         fic_list = get_fanfics(self.dataset, 0, 0, False)
206
207         return fic_list
208     def get_fanfic_in_path(self, path):
209         num_fic=int((path.split('_')[3]).split('.')[0])
210         chapterised_fic = get_chapterised_fic(path, num_fic)
211         fic = Fanfic(num_fic, self.dataset, chapterised_fic, None, None, None)
212
213         return fic
214
215     def get_fic_paths_in_range(self, start_index, end_index):
216         paths_file = open(FIC_LISTING_PATH, 'r')
217         fic_paths = [line[:-1] for line in paths_file.readlines()]
218         paths_file.close()
219         fic_paths = fic_paths[start_index:end_index]
220
221         return fic_paths
222
223     def get_fic_paths_in_list(self):
224         paths_file = open(FIC_LISTING_PATH, 'r')
225         fic_paths = [line[:-1] for line in paths_file.readlines()]
226         paths_file.close()
227
228         return fic_paths
229
230     def save_txt_fanfics(fic_list):
231         for fic, path in fic_list:
232             path_tokens = path.split('/')
233             fic_name = path_tokens[7:][0]
234             #print(fic_name)
235             new_path = SAVE_TXT_PATH + fic_name[:-4] + '.txt'
236
237             #print(new_path) #debug
238
239             f = open(new_path, 'w')
240             f.write(fic)
241             f.close()
242
243             f = open(TXT_LISTING_PATH, 'a')
244             f.write(new_path+'\n')
245             f.close()
246
247     def set_fic_listing_path(self, new_fic_listing):
248         global FIC_LISTING_PATH
249         FIC_LISTING_PATH = new_fic_listing
250

```



```

251 if 'romance' in new_fic_listing: self.dataset = 'ROMANCE'
252 elif 'friendship' in new_fic_listing: self.dataset = 'FRIENDSHIP'
253 elif 'enemy' in new_fic_listing: self.dataset = 'ENEMY'
254 elif 'explicit' in new_fic_listing: self.dataset = 'EXPLICIT'
255 elif 'html' in new_fic_listing: self.dataset = 'GENERAL'
256 else: new_fic_listing: self.dataset = 'UNKNOWN'
257
258 def set_save_txt_path(self, new_fic_save_path):
259     global SAVE_TXT_PATH
260     SAVE_TXT_PATH = new_fic_save_path
261
262 def get_fic_listing_path(self): return FIC_LISTING_PATH
263
264 def get_save_txt_path(self): return SAVE_TXT_PATH
265
266
267
268 class FanficHTMLHandler():
269     def get_chapters(self, fic_path):
270         filehandle = open(fic_path, 'r').read()
271         soup = BeautifulSoup(filehandle, 'html.parser')
272         meta_inf = soup.find_all('dd') #chapters are displayed as '# of current chapters / total # of
            chapters'
273
274         meta_chapters = meta_inf[len(meta_inf)-1].text
275         #print(chapters) #debug
276
277         if 'Chapters:' not in meta_chapters: #this means that the fic only has one chapter
278             return [1,1]
279
280         else:
281             lines = meta_chapters.split('\n')
282             for line in lines:
283                 if 'Chapters:' in line:
284                     chapters = line[20:].split('/')
285                     #print(chapters)#debug
286
287             return chapters
288
289     def get_rating(self, fic_path):
290         filehandle = open(fic_path, 'r').read()
291         soup = BeautifulSoup(filehandle, 'html.parser')
292         rating_link = soup.find(class_='tags').find('a')
293         #print(rating_link.text)#debug
294
295         return rating_link.text
296
297     def get_relationships(self, fic_path):
298         filehandle = open(fic_path, 'r').read()
299         soup = BeautifulSoup(filehandle, 'html.parser')
300         dt_inf = soup.find_all('dt')

```

```

301 dd_inf = soup.find_all('dd')
302 #print(len(dt_inf), len(dd_inf)) #debug
303
304 index = 0
305 ships = ''
306 for dt in dt_inf:
307     #print(dt) #debug
308     if 'Relationship:' not in dt.text: index += 1
309     else:
310         ships = dd_inf[index].text
311         break
312
313 #print(ships) #debug
314 ships = ships.split(',')
315 if len(ships) == 1 and ships[0] == '': ships = []
316 else:
317     for i in range(len(ships)):
318         if ' (Good Omens)' in ships[i]: ships[i] = ships[i][:len(' (Good Omens)')]
319
320 return ships
321
322 def get_tags(self, fic_path):
323     filehandle = open(fic_path, 'r').read()
324     soup = BeautifulSoup(filehandle, 'html.parser')
325
326     dt_inf = soup.find_all('dt')
327     dd_inf = soup.find_all('dd')
328
329     tags = ''
330     index = 0
331     for dt in dt_inf:
332         if 'Archive Warning:' not in dt.text: index += 1
333         else:
334             tags = dd_inf[index].text
335
336     tags += ', '
337     index = 0
338     for dt in dt_inf:
339         if 'Additional Tags:' not in dt.text: index += 1
340         else:
341             tags += dd_inf[index].text
342
343     #print(tags) #debug
344     tags = tags.split(',')
345     if len(tags) == 1 and tags[0] == '': tags = [] #I don't think this ever happens, but just in
        case
346     else:
347         tags = [tag.strip() for tag in tags]
348         for i in range(len(tags)):
349             if ' (Good Omens)' in tags[i]: tags[i] = tags[i][:len(' (Good Omens)')]
350

```

```

351     return tags
352
353
354     def get_characters(self, fic_path):
355         filehandle = open(fic_path, 'r').read()
356         soup = BeautifulSoup(filehandle, 'html.parser')
357         dt_inf = soup.find_all('dt')
358         dd_inf = soup.find_all('dd')
359         #print(len(dt_inf), len(dd_inf)) #debug
360
361         index = 0
362         chars = ''
363         for dt in dt_inf:
364             #print(dt) #debug
365             if 'Character:' not in dt.text: index += 1
366             else :
367                 chars = dd_inf[index].text
368                 break
369
370             #print(chars) #debug
371             chars = chars.split(',')
372             if len(chars) == 1 and chars[0] == '': chars = []
373             else:
374                 for i in range(len(chars)):
375                     if ' (Good Omens)' in chars[i]: chars[i] = chars[i][:-len(' (Good Omens)')]
376
377         return chars
378
379     def get_fandoms(self, fic_path):
380         filehandle = open(fic_path, 'r').read()
381         soup = BeautifulSoup(filehandle, 'html.parser')
382         dt_inf = soup.find_all('dt')
383         dd_inf = soup.find_all('dd')
384         #print(len(dt_inf), len(dd_inf)) #debug
385
386         index = 0
387         fandoms = ''
388         for dt in dt_inf:
389             #print(dt) #debug
390             if 'Fandom:' not in dt.text: index += 1
391             else :
392                 fandoms = dd_inf[index].text
393                 break
394
395             #print(chars) #debug
396             fandoms = fandoms.split(',')
397             if len(fandoms) == 1 and fandoms[0] == '': fandoms = [] #does this ever happen?
398             else: fandoms = [fandom.strip() for fandom in fandoms]
399
400         return fandoms
401

```

```
402 def get_title(self, fic_path):
403     filehandle = open(fic_path, 'r').read()
404     soup = BeautifulSoup(filehandle, 'html.parser')
405     h1 = soup.find('h1')
406     #print(len(h1)) #debug
407
408     return h1.text
```

C. ANEXO: CÓDIGO DEL ALGORITMO DE IDENTIFICACIÓN DE ENTIDADES BASADO EN REGRESIÓN LOGÍSTICA

C.1. NER_chunker

```
1  #!/bin/bash/python3
2
3  import string, re
4  import nltk
5  #from nltk.corpus import conll2000
6  from nltk.stem.snowball import SnowballStemmer
7  from nltk.tag import ClassifierBasedTagger
8  from collections import Iterable
9
10 def word_shape(word):
11     shape = 'other'
12
13     if re.match('[0-9]+(\.[0-9]*)?[0-9]*\.[0-9]+$', word): shape = 'number'
14     elif re.match('\W+$', word): shape = 'punct'
15     elif re.match('[A-Z][a-z]+$', word): shape = 'capitalized'
16     elif re.match('[A-Z]+$', word): shape = 'allcaps'
17     elif re.match('[a-z]+$', word): shape = 'alllower'
18     elif re.match('[A-Z][a-z]+[A-Z][a-z]+[A-Za-a]*$', word): shape = 'camelcase'
19     elif re.match('[A-Za-z]+$', word): shape = 'mixedcase'
20     elif re.match('___+__$', word): shape = 'wildcard'
21     elif re.match('[A-Za-z0-9]+\.$', word): shape = 'dot-end'
22     elif re.match('[A-Za-z0-9]+\.[A-Za-z0-9\.]+\.$', word): shape = 'abbreviation'
23     elif re.match('[A-Za-z0-9]+\-[A-Za-z0-9\-\]+\.$', word): shape = 'hyphenated'
24
25     return shape
26
27 def feature_function(sentence, i, history):
28     """
29     sentence: a POS-tagged sentence
30     i: index of the current token
31     history: previous IOB tags
32     """
33
34     stemmer = SnowballStemmer('english')
35
36     #Padding
37     sentence = [('<START2>', '<START2>'), ('<START1>', '<START1>')] + list(sentence) + [('<END1>', '<END1>'), ('<END2>', '<END2>')]
38
39     history = ['<START2>', '<START1>'] + list(history)
40
41     i += 2 #shift the index to accomodate padding
42
43     word, pos = sentence[i]
44     prevword, prevpos = sentence[i-1]
```

```

45     previob = history[i-1]
46     prevprevword, prevprevpos = sentence[i-2]
47     prevpreviob = history[i-2]
48     nextword, nextpos = sentence[i+1]
49     nextnextword, nextnextpos = sentence[i+2]
50
51     """
52     contains_dash = '-' in word
53     contains_dot = '.' in word
54     allascii = all([True for c in word if c in string.ascii_lowercase])
55     allcaps = word == word.capitalize()
56     capitalized = word[0] in string.ascii_uppercase
57     prevallcaps = prevword == prevword.capitalize()
58     prevcapitalized = prevword[0] in string.ascii_uppercase
59     nextallcaps = nextword == nextword.capitalize()
60     nextcapitalized = nextword[0] in string.ascii_uppercase
61     """
62
63     return {
64         'word': word,
65         'lemma': stemmer.stem(word),
66         'shape': word_shape(word),
67         'pos': pos,
68
69         'prev-word': prevword,
70         'prev-lemma': stemmer.stem(prevword),
71         'prev-shape': word_shape(prevword),
72         'prev-pos': prevpos,
73
74         'prevprev-word': prevprevword,
75         'prevprev-lemma': stemmer.stem(prevprevword),
76         'prevprev-shape': word_shape(prevprevword),
77         'prevprev-pos': prevprevpos,
78
79         'prev-iob': previob,
80         'prevprev-iob': prevpreviob,
81
82         'next-word': nextword,
83         'next-lemma': stemmer.stem(nextword),
84         'next-shape': word_shape(nextword),
85         'next-pos': nextpos,
86
87         'nextnext-word': nextnextword,
88         'nextnext-lemma': stemmer.stem(nextnextword),
89         'nextnext-shape': word_shape(nextnextword),
90         'nextnext-pos': nextnextpos,
91     }
92
93
94
95     class NERTagger(nltk.TaggerI):

```

```

96  def __init__(self, train_sents):
97      train_set = []
98
99      for tagged_sent in train_sents:
100         untagged_sent = nltk.tag.untag(tagged_sent)
101         history = []
102
103         for i, (word,tag) in enumerate(tagged_sent):
104             featureset = feature_function(untagged_sent, i, history)
105             train_set.append( (featureset, tag) ) #sentence[1] is the tag of the token
106
107         history.append(tag)
108
109     self.classifier = nltk.MaxentClassifier.train(train_set,
110         algorithm='megam', trace=0)
111
112     def tag(self,sentence):
113         history = []
114         for i, word in enumerate(sentence):
115             featureset = feature_function(sentence, i, history)
116             tag = self.classifier.classify(featureset)
117             history.append(tag)
118
119         return zip(sentence, history)
120
121
122     class NERChunkerv3(nltk.ChunkParserI):
123     def __init__(self, train_sents):
124         tagged_sents = [(w,t,c) for (w,t,c) in train_sents] #transform sentences to a shape that can
125             be understood to the tagger
126         self.tagger=NERTagger(tagged_sents)
127
128     def parse(self, sentence):
129         tagged_sents = self.tagger.tag(sentence)
130         conlltags = [(w,t,c) for ((w,t),c) in tagged_sents]
131
132         return nltk.chunk.conlltags2tree(conlltags)
133
134     class NERChunkerv2(nltk.ChunkParserI):
135     def __init__(self, train_sents, **kwargs):
136         assert isinstance(train_sents, Iterable)
137         tagged_sents = [(((w,t),c) for (w,t,c) in
138             nltk.chunk.tree2conlltags(sent))
139             for sent in train_sents] #transform sentences to a shape that can be understood to the tagger
140
141         self.feature_detector = feature_function
142         self.tagger = ClassifierBasedTagger(train=tagged_sents, feature_detector=feature_function, **
143             kwargs)
144
145     def parse(self, sentence):

```

```

145 tagged_sents = self.tagger.tag(sentence)
146 conlltags = [(w,t,c) for ((w,t),c) in tagged_sents]
147
148 return nltk.chunk.conlltags2tree(conlltags)
149
150
151 class NERChunkerv1(nltk.ChunkParserI):
152     def __init__(self, train_sents):
153         tagged_sents = [[((w,t),c) for (w,t,c) in nltk.chunk.tree2conlltags(
154             sent)]
155             for sent in train_sents]#transform sentences to a shape that can be understood to the tagger
156
157         self.tagger=NERTagger(tagged_sents)
158
159     def parse(self, sentence):
160         tagged_sents = self.tagger.tag(sentence)
161         conlltags = [(w,t,c) for ((w,t),c) in tagged_sents]
162
163         return nltk.chunk.conlltags2tree(conlltags)

```


C.2. NER_trainer

```
1  #!/bin/bash/python3
2
3
4  #Trainer for an entity recognition process
5
6  import nltk, re, pprint, time, pickle, pandas
7  from nltk.tokenize import word_tokenize
8  from nltk.tag import pos_tag
9  from nltk.classify import megam
10 from NER_chunker import NERChunkerv1
11
12
13
14 ### NOTAS ###
15 """
16 NLTK no tiene un corpus adecuado para el reconocimiento de entidades en ingles,
17 de modo que utilizare una base de datos en csv para NER basada en GMB
18 """
19
20
21 ### VARIABLES ###
22 NER_DATASET_PATH = '/home/maria/Documents/Fanfic_ontology/TFG_train/ner_dataset.csv'
23 megam.config_megam('/home/maria/Downloads/megam_0.92/megam')
24
25 ### FUNCTIONS ###
26
27 def get_tagged_tokens_from_csv():
28     csv_file = pandas.read_csv(NER_DATASET_PATH, encoding='ISO-8859-1')
29
30     i = 0
31     sentences = []
32     current_sent = []
33     while i < len(csv_file['Sentence #']):
34         if i == 0:
35             current_sent.append((csv_file['Word'][i], csv_file['POS'][i], csv_file['Tag'][i]))
36
37         elif ':' in str(csv_file['Sentence #'][i]):
38
39             sentences.append(nltk.chunk.conlltags2tree(current_sent))
40
41             current_sent = []
42             current_sent.append((csv_file['Word'][i], csv_file['POS'][i], csv_file['Tag'][i]))
43
44         else:
45             current_sent.append((csv_file['Word'][i], csv_file['POS'][i], csv_file['Tag'][i]))
46
47         i+=1
48
49     return sentences #returns a list of tagged sentences that NLTK can understand
```

```

50
51 ### M A I N ###
52
53 ### Get train and test sentences:
54 tagged_sentences = get_tagged_tokens_from_csv()
55 #print(tagged_sentences[0], tagged_sentences[1], tagged_sentences[2]) #debug
56 #print(isinstance(tagged_sentences[0], Tree)) #debug
57
58 #Database: 70% for training, 30% for testing (in three 'slices' of 10% each)
59
60 #train_sents = tagged_sentences[:int(len(tagged_sentences)*0.1)] #debug, tarda 20 mins aprox
61
62 train_sents = tagged_sentences[:int(len(tagged_sentences)*0.7)] #aprox 2 horas
63 #test_sents = tagged_sentences[int(len(tagged_sentences)*0.8):int(len(tagged_sentences)*0.7)]
64 #Test A
65 #test_sents = tagged_sentences[int(len(tagged_sentences)*0.9):int(len(tagged_sentences)*0.8)]
66 #Test B
67
68 test_sents = tagged_sentences[int(len(tagged_sentences)*0.9):] #Test C
69
70
71 ### Create and train chunker:
72 print('Starting tranining... go for a walk')
73 start = time.time()
74
75 NER_chunker = NERChunkerv1(train_sents)
76
77 end = time.time()
78
79
80 ### Test our chunker
81 print('NER chunker (', (end-start)/60, 'mins) \n', NER_chunker.evaluate(test_sents))
82
83
84 ### Store trained chunker for later use
85 print('Preparing to pickle. . .')
86 f = open('NER_training.pickle', 'wb')
87 pickle.dump(NER_chunker, f)
88 f.close()
89
90 print('NER_chunker successfully pickled')

```

C.3. NER_tagger

```
1  #!/bin/bash/python3
2
3  #Tagger for NER tags, using a previously trained NER chunker
4
5  import pickle, time, pandas, re, nltk
6
7  ### VARIABLES ###
8  MODEL_PATH = '/home/maria/Documents/Fanfic_ontology/NER_training.pickle'
9  CANON_DB = '/home/maria/Documents/Fanfic_ontology/canon_characters.csv'
10 MAX_EDIT_DISTANCE = 3
11
12 ### FUNCTIONS ###
13
14 def get_max_edit_distance(name):
15     if ' ' in name:
16         subnames = [n.strip() for n in name.split(' ')]
17
18         lengths = [len(name) for name in subnames]
19
20         min_length = min(lengths)
21
22     else: min_length = len(name.strip())
23
24     if min_length > 4: return 3
25     elif min_length == 4: return 2
26     else: return 0
27
28 def get_edit_distance(name1, name2):
29     if ' ' in name1:
30         name_and_surname1 = [n.strip().lower() for n in name1.split(' ')]
31
32         distance = []
33         if ' ' in name2:
34             name_and_surname2 = [n.strip().lower() for n in name2.split(' ')]
35
36             for n1 in name_and_surname1:
37                 for n2 in name_and_surname2: distance.append(nltk.edit_distance(n1, n2))
38
39         else:
40             for n in name_and_surname1: distance.append(nltk.edit_distance(n, name2.lower()))
41
42         return min(distance)
43
44     elif ' ' in name2:
45         name_and_surname2 = [n.strip().lower() for n in name2.split(' ')]
46
47         distance = []
48         for n in name_and_surname2: distance.append(nltk.edit_distance(name1.lower(), n))
49
```

```

50     return min(distance)
51
52     else: return nltk.edit_distance(name1.lower(), name2.lower())
53
54     def link_to_canon(ner_entities):
55         #Get canon_db
56         canon_db = pandas.read_csv(CANON_DB)
57
58         for character in ner_entities:
59             better_fit = (-1, 300) #first member of the tuple is ID, the other is edit distance,
                                   #instantiated to an absolutely ridiculous high one so it can be replaced for the smallest
                                   #one
60
61             for index, canon_character in canon_db.iterrows():
62                 if type(canon_character['Other names']) == str:
63                     other_canon_names = [name.strip().lower() for name in canon_character['Other names'].split(',')
                                           ]
64                     #print(other_canon_names) #debug
65
66                 else: other_canon_names = ['']
67
68             distance = get_edit_distance(canon_character['Name'], character['Name'])
69             max_edit_distance = min(get_max_edit_distance(canon_character['Name']), get_max_edit_distance(
                                   character['Name']))
70
71             if distance == 0:
72                 character['Canon ID'] = index
73                 break
74
75             elif distance < max_edit_distance:
76                 if distance < better_fit[1]: better_fit = (index, distance)
77
78             else:
79                 for other_name in other_canon_names:
80                     distance = get_edit_distance(other_name, character['Name'])
81                     max_edit_distance = min(get_max_edit_distance(other_name), get_max_edit_distance(character['
                                   Name'])))
82
83             if distance == 0:
84                 character['Canon ID'] = index
85                 break
86
87             elif distance < max_edit_distance:
88                 if distance < better_fit[1]: better_fit = (index, distance)
89
90             if character['Canon ID'] == 'NO' and better_fit[0] >= 0:
91                 character['Canon ID'] = better_fit[0]
92
93         return ner_entities
94
95

```

```

96  ### CLASSES ###
97
98  class NERTagger():
99
100     def __init__(self):
101         self.model_path = MODEL_PATH
102
103     def parse(self, tagged_fic): #fanfic must be pos-tagged previously
104         ###Load trained chunker and parse sentences
105         f = open(self.model_path, 'rb')
106         NER_chunker = pickle.load(f)
107         f.close()
108
109         NER_tagged_sentences = [NER_chunker.parse(sent) for sent in tagged_fic] #NER-tagging
110
111         #Explore the tags and create a list of named entities
112         per_entities= []
113         #start = time.time()
114         for sentence in NER_tagged_sentences:
115             for t in sentence.subtrees():
116                 if t.label() == 'per':
117                     leaves = t.leaves()
118                     entity = [word for word, _ in leaves]
119                     #print(entity) #debug
120
121                     entity = ' '.join(entity)
122                     entity = re.sub(r'^[A-Za-z0-9 ]+', '', entity).strip() #Clean name
123
124                     per_entities.append(entity)
125
126                     #print(len(per_entities)) #debug
127
128                     names = set(per_entities)
129                     ner_entities = []
130                     for name in names:
131                         num = per_entities.count(name)
132                         ner_entities.append({'Name':name, 'Mentions':num, 'Canon ID':'NO'})
133
134                     #print(ner_entities) #debug
135
136                     canonized_ner_entities = link_to_canon(ner_entities)
137
138             return canonized_ner_entities
139
140     def get_model_path(self): return self.model_path
141
142     def set_model_path(self, path):
143         self.model_path = path
144
145     ### MAIN ###

```

D. ANEXO: CÓDIGO QUE MANEJA CoreNLP

D.1. corenlp_wrapper

```
1  #!/bin/bash/python3
2
3  from stanza.server import CoreNLPClient
4  from stanza.server.client import TimeoutException
5  from fanfic_util import Fanfic
6  from urllib.error import HTTPError
7
8  import time
9
10  ### VARIABLES ###
11  MAX_CHAPTER_LENGTH = 30000
12
13
14  ### FUNCTIONS ###
15  def split_chapter(chapter):
16      len_chapter = len(chapter)
17
18      if len_chapter % MAX_CHAPTER_LENGTH == 0: divisions = int(len_chapter/MAX_CHAPTER_LENGTH)
19      else: divisions = int(len_chapter/MAX_CHAPTER_LENGTH)+1
20
21      div_chapters = []
22      while len(chapter) > MAX_CHAPTER_LENGTH:
23          div_chapters.append(chapter[:MAX_CHAPTER_LENGTH])
24          chapter = chapter[MAX_CHAPTER_LENGTH:]
25
26      if len(chapter) != 0: div_chapters.append(chapter)
27
28
29      if len(div_chapters) != divisions: raise Exception("[corenlp_wrapper] An error ocurred while
        slicing chapters")
30
31      return div_chapters
32
33  def compress_chapters(fic_chapters):
34      compressed_chapters = []
35
36      processed_chapters = 0
37      while processed_chapters < len(fic_chapters):
38          chapter_text = ''
39          while len(chapter_text) < MAX_CHAPTER_LENGTH:
40              chapter_text += fic_chapters[processed_chapters]
41              processed_chapters += 1
42
43          compressed_chapters.append(chapter_text)
44
45      return processed_chapters
```

```

46
47 def process_fics(fics):
48     if type(fics) == list:
49         #Check that members of list are class Fanfic
50         for i, fic in enumerate(fics):
51             if type(fic) is not Fanfic: raise TypeError("[corenlp_wrapper2] The input for the client must
                    be list of Fanfic, or a single Fanfic")
52             else: #check that all chapters in fanfic are no longer than 100000 characters
53                 fic_chapters = fic.chapters
54
55                 for chapter in fic_chapters:
56                     #print(type(chapter), len(chapter))
57                     if len(chapter) > MAX_CHAPTER_LENGTH:
58                         print('BIG BOY CHAPTER')
59                         extra_chapters = split_chapter(chapter)
60                         fic_chapters.remove(chapter)
61
62                 position = i
63                 for extra_chap in extra_chapters:
64                     #print(len(extra_chap)) #debug
65                     fic_chapters.insert(position, extra_chap)
66                     position += 1
67
68                 #end FOR chapter
69
70                 fic.set_chapters(fic_chapters)
71
72             #end ELSE
73         #end FOR fic
74     #end if TYPE(FICS) == LIST
75     elif type(fics) == Fanfic:
76         fic_chapters = fics.chapters
77
78         for i, chapter in enumerate(fic_chapters):
79             if len(chapter) > MAX_CHAPTER_LENGTH:
80                 extra_chapters = split_chapter(chapter)
81                 fic_chapters.remove(chapter)
82
83             position = i
84             for extra_chap in extra_chapters:
85                 fic_chapters.insert(position, extra_chap)
86                 position += 1
87
88             fics.set_chapters(fic_chapters)
89
90
91     return fics
92
93
94 ### CLASSES ###

```

```

95     class CoreClient(): #This client works with lists of string, or individual string. Returns a
        list of annotations with the data from CoreNLP
96
97     def parse(self, fic_chapters):
98     if type(fic_chapters) == list:
99     #First we check that all chapters are strings, and none of them are longer than 100000
        characters
100    for i, chapter in enumerate(fic_chapters):
101    if not type(chapter) is str: raise TypeError("[corenlp_wrapper] The input for the client must
        be a string")
102    elif len(chapter) > MAX_CHAPTER_LENGTH:
103    extra_chapters = split_chapter(chapter)
104    fic_chapters.remove(chapter)
105
106    position = i
107    for extra_chap in extra_chapters:
108    fic_chapters.insert(position, extra_chap)
109    position += 1
110
111    elif type(fic_chapters) == str:
112    if len(fic_chapters) > MAX_CHAPTER_LENGTH:
113    extra_chapters = split_chapter(fic_chapters)
114    fic_chapters = extra_chapters
115
116    else: fic_chapters = [fic_chapters]
117
118    else: raise TypeError("[corenlp_wrapper] The input for the client must be a list of string or
        a single string")
119
120
121
122
123
124    annotations = []
125
126    with CoreNLPClient(
127    annotators = ['tokenize', 'sentiment', 'ssplit', 'pos', 'lemma', 'ner', 'parse', 'depparse', '
        coref'],
128    timeout=120000,
129    be_quiet = True,
130    memory='4G') as client:
131    print("Annotating data . . .")
132
133    for chapter in fic_chapters: annotations.append(client.annotate(chapter))
134
135
136    print("...done")
137
138    return annotations
139

```



```

140 class CoreClient2(): #This client works with lists of Fanfic objects. Returns the same list,
    but each Fanfic object now has a list of annotations with the data from CoreNLP
141
142 def parse(self, fics):
143     fics = process_fics(fics)
144
145     annotations = []
146
147     try:
148         with CoreNLPClient(
149             annotators = ['tokenize', 'sentiment', 'ssplit', 'pos', 'lemma', 'ner', 'parse', 'depparse',
150                           'coref'],
151             timeout=120000,
152             be_quiet = True,
153             memory='4G') as client:
154                 print("Annotating data . . .")
155
156                 for i, fic in enumerate(fics):
157                     print('fic #', i, 'has ', len(fic.chapters))
158                     annotations = []
159                     for i, chapter in enumerate(fic.chapters):
160                         print('chapter ', i, ': ', len(chapter))
161                         ann = client.annotate(chapter)
162                         time.sleep(4)
163
164                         annotations.append(ann)
165
166                     fic.set_annotations(annotations)
167
168                 print("...done")
169
170             except TimeoutException as e:
171                 print('CoreNLP TimeoutException')
172             return fics, True
173
174
175         return fics, False

```

D.2. corenlp_util

```
1  #!/usr/bin/bash/python3
2
3  import time, pandas, nltk, re, random
4  #from corenlp_wrapper import CoreClient2
5  from stanza.server import Document
6  from stanza.server import CoreNLPClient
7  from stanza.server.client import TimeoutException
8  from urllib.error import HTTPError
9
10 from fanfic_util import *
11
12 ### VARIABLES ###
13 CANON_DB = '/home/maria/Documents/Fanfic_ontology/canon_characters.csv'
14 ERRORLOG = '/home/maria/Documents/Fanfic_ontology/TFG_logs/corenlp_util_errorlog.txt'
15
16 FEMALE_TAGS = ['She/Her Pronouns for ', 'Female ', 'Female!', 'Female-Presenting ']
17 MALE_TAGS = ['He/Him Pronouns for ', 'Male ', 'Male!', 'Male-Presenting ']
18 NEUTRAL_TAGS = ['They/Them Pronouns for ', 'Gender-Neutral Pronouns for ', 'Gender-Neutral ',
19                 'Agender ', 'Genderfluid ', 'Androgynous ', 'Gender Non-Conforming ']
20
21 PRONOUNS = ['he', 'him', 'his', 'she', 'her', 'hers', 'I', 'me', 'mine', 'you', 'your', 'yours', 'we', 'our',
22             'ours', 'they', 'them', 'theirs', 'it', 'its']
23
24 MAX_CHAPTER_LENGTH = 30000
25
26 ### FUNCTIONS ###
27 def normalize_sentiment(sentences):
28     sentiment_info = {'Num sentences':0, 'Very positive':0, 'Positive':0, 'Neutral':0, 'Negative':0, 'Very negative':0}
29     sentiment_info['Num sentences'] = len(sentences)
30
31     sentiment_count = 0
32     for sentence in sentences:
33         sentiment = sentence.sentiment
34         #print(sentiment) #debug
35         sentiment_info[sentiment] += 1
36
37     if sentiment == 'Very positive': sentiment_count += 2
38     elif sentiment == 'Positive': sentiment_count += 1
39     elif sentiment == 'Negative': sentiment_count -= 1
40     elif sentiment == 'Very negative': sentiment_count -= 2
41
42     sentiment_info['Weighted average'] = sentiment_count/len(sentences)
43
44     return sentiment_info
45
46
```

```

47 def make_gender_tags(tags, character_name):
48     for tag in tags:
49         tag = tag+character_name
50         #print(tag) #debug
51
52     return tags
53
54 def decide_gender(characters, canon_db):
55     handler = FanficHTMLHandler()
56
57     for character in characters:
58         fic_link = '/home/maria/Documents/Fanfic_ontology/TFG_fics/html/gomensfanfic_'+str(character['
                    ficID'])+'.html'
59         fic_tags = handler.get_tags(fic_link)
60         character_name = character['Name'].capitalize()
61
62         f_gender = make_gender_tags(FEMALE_TAGS, character_name)
63         m_gender = make_gender_tags(MALE_TAGS, character_name)
64         n_gender = make_gender_tags(NEUTRAL_TAGS, character_name)
65
66         male = female = neutral = False
67
68         if any(tag in fic_tags for tag in f_gender): female = True
69         if any(tag in fic_tags for tag in m_gender): male = True
70         if any(tag in fic_tags for tag in n_gender): neutral = True
71
72         if female and male: character['Gender'] = 'NEUTRAL'
73         elif female: character['Gender'] = 'FEMALE'
74         elif male: character['Gender'] = 'MALE'
75         elif neutral: character['Gender'] = 'NEUTRAL'
76         else: #if there are no gender tags applicable to the character, we'll defer to CoreNLP's
                    opinion on this character's gender
77
78         if character['Gender'] == 'UNKNOWN' or character['Gender'] == '': #if CoreNLP doesn't know this
                    character's gender, and it is canon, we assume the canon gender. If it isn't canon it'll
                    be left unknown
79         #print(character['Name'], character['Gender']) #debug
80
81         if character['canonID'] > -1:
82             canon_gender = canon_db.iloc[character['canonID']]['Gender']
83             character['Gender'] = canon_gender
84
85         elif character['Gender'] == '' or character['Gender'] == 'X': character['Gender'] = 'UNKNOWN'
86
87     return characters
88
89 def merge_sentences(fic_index, fic_dataset, character_sentences):
90     sen_ids = []
91     merged_sentences = []
92
93     for sentence in character_sentences:

```

```

94     sent = {}
95     if sentence['senID'] in sen_ids: #If a dict for this sentence already exists
96     sent = list(filter(lambda s: s['senID'] == sentence['senID'], merged_sentences))
97     #print(len(sent)) #debug
98     sent = sent[0]
99
100    if sentence['nerIDs'] > -1 and sentence['nerIDs'] not in sent['nerIDs']: sent['nerIDs'].append
        (sentence['nerIDs'])
101    if sentence['Clusters'] > -1 and sentence['Clusters'] not in sent['Clusters']: sent['Clusters'
        ].append(sentence['Clusters'])
102
103    else: #Create new sentence dict
104    sen_ids.append(sentence['senID'])
105
106    sent['ficID'] = fic_index
107    sent['ficDataset'] = fic_dataset
108    sent['senID'] = sentence['senID']
109    sent['Sentiment'] = sentence['Sentiment']
110    sent['Verbs'] = sentence['Verbs']
111
112    if sentence['nerIDs'] > -1: sent['nerIDs'] = [sentence['nerIDs']]
113    else: sent['nerIDs'] = []
114
115    if sentence['Clusters'] > -1: sent['Clusters'] = [sentence['Clusters']]
116    else: sent['Clusters'] = []
117
118    merged_sentences.append(sent)
119
120    return merged_sentences
121
122    def get_name_count(ner_chars):
123    names = {}
124    other_names = []
125
126    for char in ner_chars:
127    try:
128    if char['Name'].lower() not in PRONOUNS:
129    names[char['Name']] += 1
130    except KeyError: names[char['Name']] = 1
131
132    return names
133
134    def get_edit_distance(name1, name2):
135    if ' ' in name1:
136    name_and_surname1 = [n.strip().lower() for n in name1.split(' ')]
137
138    distance = []
139    if ' ' in name2:
140    name_and_surname2 = [n.strip().lower() for n in name2.split(' ')]
141
142    for n1 in name_and_surname1:

```

```

143     for n2 in name_and_surname2: distance.append(nltk.edit_distance(n1, n2))
144
145     else:
146         for n in name_and_surname1: distance.append(nltk.edit_distance(n, name2.lower()))
147
148     return min(distance)
149
150     elif ' ' in name2:
151         name_and_surname2 = [n.strip().lower() for n in name2.split(' ')]
152
153         distance = []
154         for n in name_and_surname2: distance.append(nltk.edit_distance(name1.lower(), n))
155
156         return min(distance)
157
158     else: return nltk.edit_distance(name1.lower(), name2.lower())
159
160     def get_max_edit_distance(name):
161         if ' ' in name:
162             subnames = [n.strip() for n in name.split(' ')]
163
164             lengths = [len(name) for name in subnames]
165
166             min_length = min(lengths)
167
168             else: min_length = len(name)
169
170             if min_length > 4: return 3
171             elif min_length == 4: return 2
172             else: return 1
173
174     def merge_character_mentions(fic_index, character_entities, coref_mentions):
175         # create characters from their NER IDs
176         ner_ids = [char['nerID'] for char in character_entities]
177         ner_ids = set(ner_ids) #remove duplicates
178
179         characters = []
180         ## Rule 1 of character merging: Mentions with the same nerID refer to the same character
181         for ner in ner_ids:
182             ner_chars = list(filter(lambda entity: entity['nerID'] == ner, character_entities)) #
                characters with this ner ID
183         #Select gender according to CoreNLP NER extractor
184         gender = []
185         for char in ner_chars: gender.append(char['Gender'])
186         gender = list(set(gender)) #remove duplicates
187         if '' in gender: gender.remove('')
188
189         if len(gender) == 0: gender = 'X'
190         elif len(gender) == 1: gender = gender[0]
191         else:
192             #print(gender) #debug

```

```

193 gender = 'X'
194
195 #Clean and select character's name
196 names = [char['Name'].replace('\n', ' ').strip() for char in ner_chars]
197 names = [re.sub(r'^A-Za-z0-9[ ]+', '', name).strip() for name in names if name.lower() not
198           in PRONOUNS] #strip non-alphanumeric characters, except for spaces
199 names = list(set(names)) #remove duplicates
200 #print(names) #debug
201
202 for name in names:
203     new_character = {'ficID':fic_index,'nerID':[ner],'Name':name,'Other names':[],'Gender':gender,
204                     'clusterID':[-1],'canonID':-1,'Mentions':0}
205     #print(new_character) #debug
206     characters.append(new_character)
207
208 for char in characters:
209     for mention in coref_mentions:
210         if mention['nerID'] == char['nerID'][0]:
211             if mention['MentionT'] == 'PROPER':
212                 #if mention['nerID'] == 0: print(mention) #debug
213                 ## Rule 2 of character merging: Mentions which overlap with a NERmention refer to its same
214                 character
215                 distance = get_edit_distance(char['Name'], mention['Name'])
216                 max_edit_distance = min(get_max_edit_distance(char['Name']),get_max_edit_distance(mention['
217                                         Name']))
218                 if distance < max_edit_distance:
219                     if char['Gender'] == 'X': #gender is added
220                         char['Gender'] = mention['Gender']
221                     elif char['Gender'] != mention['Gender']:
222                         #print('diff gender: ',mention) #debug
223                         diff_gender_chars = list(filter(lambda entity: nltk.edit_distance(char['Name'],entity['Name'])
224                                                         < max_edit_distance and entity['Gender'] == mention['Gender'], characters))
225                         if len(diff_gender_chars) > 0:
226                             #print('len diff gender:',len(diff_gender_chars))#debug
227                             char = diff_gender_chars[0]
228                         else: #Add new entry for the character with different gender
229                             diff_gender_char = char.copy()
230                             diff_gender_char['Gender'] = mention['Gender']
231                             diff_gender_char['Mentions'] = 0
232                             characters.append(diff_gender_char)
233
234                             char = diff_gender_char
235
236                             char['Mentions'] +=1
237
238             if distance > 0: #different names are added

```

```

239 other_names = [name.lower() for name in char['Other names']]
240 if mention['Name'].lower() not in other_names:
241     char['Other names'].append(mention['Name'])
242
243
244 cluster = char['clusterID'] #cluster is added
245 if mention['clusterID'] not in cluster: char['clusterID'].append(mention['clusterID'])
246
247
248 for char in characters:
249     if -1 in char['clusterID']: char['clusterID'].remove(-1) #remove filler
250
251
252 ## Rule 3 of character merging: All the mentions of a coreference cluster refer to the same
    character, but only if the gender is consistent and the names have an edit distance lesser
    than MAX_EDIT_DISTANCE
253 aux = characters.copy()
254 for char in characters:
255     aux.remove(char)
256
257 for c in aux:
258     cluster = c['clusterID']
259
260 if any(cluster) in char['clusterID']:
261     if char['Gender'] == c['Gender']:
262         distance = get_edit_distance(char['Name'], c['Name'])
263         max_edit_distance = min(get_max_edit_distance(char['Name']), get_max_edit_distance(c['Name']))
264
265 if distance == 0:
266     char['Mentions'] = char['Mentions'] + c['Mentions']
267
268     aux.remove(c)
269     characters.remove(c)
270
271 elif distance < max_edit_distance:
272     char['Mentions'] = char['Mentions'] + c['Mentions']
273     char['nerID'].append(c['nerID'][0])
274
275 other_names = [n.lower() for n in char['Other names']]
276 if c['Name'] not in other_names: char['Other names'].append(c['Name'])
277
278     aux.remove(c)
279     characters.remove(c)
280
281
282
283 #print(characters) #debug
284 return characters
285
286 def link_characters_to_canon(characters, canon_db):
287     for character in characters:

```

```

288 better_fit = (-1, 300) #first member of the tuple is ID, the other is edit distance,
    instantiated to an absolutely ridiculous high one so it can be replaced for the smallest
    one
289
290 for index, canon_character in canon_db.iterrows():
291     if type(canon_character['Other names']) == str:
292         other_canon_names = [name.strip().lower() for name in canon_character['Other names'].split(',')
            ]
293         #print(other_canon_names) #debug
294
295
296     else: other_canon_names = ['']
297
298     distance = get_edit_distance(canon_character['Name'], character['Name'])
299     max_edit_distance = min(get_max_edit_distance(canon_character['Name']), get_max_edit_distance(
        character['Name']))
300
301     if distance == 0:
302         character['canonID'] = index
303         break
304
305     elif distance < max_edit_distance:
306         #print(canon_character['Name'].lower(), character['Name'].lower()) #debug
307         if distance < better_fit[1]: better_fit = (index, distance)
308
309     elif character['Name'].lower() in other_canon_names:
310         character['canonID'] = index
311         break
312     else:
313         for name in character['Other names']:
314             distances = [get_edit_distance(name, canon_name) for canon_name in other_canon_names]
315             max_distances = [get_max_edit_distance(canon_name) for canon_name in other_canon_names]
316             max_edit_distance = min(max_distances)
317             if any(distances) == 0:
318                 character['canonID'] = index
319                 break
320
321             elif any(distances) < max_edit_distance:
322                 if distance < better_fit[1]: better_fit = (index, distance)
323
324
325 # end for canon_characters
326 if character['canonID'] < 0 and better_fit[0] >= 0:
327     character['canonID'] = better_fit[0]
328
329 #if a character is not canon its canonID will not be changed from -1
330
331 return characters
332
333 def canonize_characters(characters, canon_db):
334     characters = link_characters_to_canon(characters, canon_db)

```



```

335 characters = decide_gender(characters, canon_db)
336
337 canon_ids = len(canon_db['Name'])
338 #print(canon_ids) #debug
339
340 canonized_characters = []
341 for i in range(canon_ids):
342     canon_name = canon_db.iloc[i]['Name']
343     #print(canon_name) #debug
344     characters_in_canon = list(filter(lambda char: char['canonID'] == i, characters))
345
346     if len(characters_in_canon) > 0:
347         canonized_character = {'Name': canon_name, 'Other names': '', 'Male mentions': 0, 'Female
            mentions': 0, 'Neutral mentions': 0, 'Unknown mentions': 0, 'Canon ID': i}
348
349         canonized_character['Name'] = canon_name
350
351         other_names = []
352         for char in characters_in_canon:
353             if char['Name'].lower() != canon_name.lower(): other_names.append(char['Name'])
354             other_names.extend(char['Other names'])
355
356             if char['Gender'] == 'MALE':
357                 canonized_character['Male mentions'] += char['Mentions']
358
359             elif char['Gender'] == 'FEMALE':
360                 canonized_character['Female mentions'] += char['Mentions']
361
362             elif char['Gender'] == 'NEUTRAL':
363                 canonized_character['Neutral mentions'] += char['Mentions']
364             else:
365                 canonized_character['Unknown mentions'] += char['Mentions']
366
367         other_names = list(set(other_names))
368         canonized_character['Other names'] = ', '.join(other_names)
369
370         canonized_characters.append(canonized_character)
371
372     characters_not_in_canon = list(filter(lambda char: char['canonID'] == -1, characters))
373     noncanon_characters = []
374     for char in characters_not_in_canon:
375         noncanon_char = {'Name': char['Name'], 'Other names': '', 'Male mentions': 0, 'Female mentions'
            : 0, 'Neutral mentions': 0, 'Unknown mentions': 0, 'Canon ID': 'NO'}
376
377         if char['Gender'] == 'MALE':
378             noncanon_char['Male mentions'] += char['Mentions']
379
380         elif char['Gender'] == 'FEMALE':
381             noncanon_char['Female mentions'] += char['Mentions']
382
383         else:

```

```

384 noncanon_char['Unknown mentions'] += char['Mentions']
385
386 noncanon_characters.append(noncanon_char)
387
388 #print(type(canonicalized_characters)) #debug
389
390 return canonicalized_characters+noncanon_characters
391
392 def extract_data_from_annotations(annotation):
393     sentences= annotation.sentence
394     all_ner_mentions = annotation.mentions #NERMention[]
395     all_coref_mentions = annotation.mentionsForCoref #Mention[]
396     chains = annotation.corefChain #CorefChain[], made up of CorefMention[]
397
398     # lists to store the return values in
399     character_entities = []
400     coref_mentions = []
401     character_sentences = []
402
403     ## Extract characters from NER and coreference mentions ##
404     if len(all_ner_mentions) > 0:
405         for ner in all_ner_mentions:
406             if ner.ner == 'PERSON':
407                 sentence = {}
408                 sen = sentences[ner.sentenceIndex]
409
410                 string_sen = ''
411                 for token in sen.token:
412                     string_sen += ' '+token.originalText
413
414                 sentence['senID'] = sen.sentenceIndex
415                 sentence['Sentiment'] = sen.sentiment
416                 #sentence['Verbs'] = get_lemmatized_verbs(string_sen)
417                 sentence['Verbs'] = string_sen
418                 sentence['nerIDs'] = ner.canonicalEntityMentionIndex
419                 sentence['Clusters'] = -1 #filler
420                 character_sentences.append(sentence)
421
422                 character = {}
423                 #token = tokens[ner.tokenStartInSentenceInclusive]
424                 #coref_in_token = token.
425
426                 character['senID'] = sen.sentenceIndex
427                 character['nerID'] = ner.canonicalEntityMentionIndex
428                 character['nerMentionIndex'] = ner.entityMentionIndex
429                 character['Name'] = ner.entityMentionText
430                 character['Gender'] = ner.gender
431                 character['MentionT'] = 'PERSON'
432                 character_entities.append(character)
433
434     #end FOR ner

```

```

435 #end IF len
436 if len(all_coref_mentions) > 0:
437     for mention in all_coref_mentions:
438         if mention.mentionType in ['PROPER', 'PRONOMINAL']:
439             sentence = {}
440             sen = sentences[mention.sentNum]
441
442             string_sen = ''
443             for token in sen.token:
444                 string_sen += ' ' + token.originalText
445
446             sentence['senID'] = sen.sentenceIndex
447             sentence['Sentiment'] = sen.sentiment
448             #sentence['Verbs'] = get_lemmatized_verbs(string_sen)
449             sentence['Verbs'] = string_sen
450             sentence['nerIDs'] = -1 #filler
451             sentence['Clusters'] = mention.corefClusterID
452
453             character_sentences.append(sentence)
454
455             #if mention.sentNum != sen.sentenceIndex: print('false')
456
457             character = {}
458
459             token = sen.token[mention.headIndex]
460             ner_in_token = all_ner_mentions[token.entityMentionIndex]
461             character['senID'] = sen.sentenceIndex
462             character['nerID'] = ner_in_token.canonicalEntityMentionIndex
463             character['clusterID'] = mention.corefClusterID
464             character['Name'] = mention.headString
465             character['Gender'] = mention.gender
466             character['MentionT'] = mention.mentionType
467             coref_mentions.append(character)
468
469
470 #end FOR mention
471 #end IF len
472
473 return character_entities, coref_mentions, character_sentences
474
475 def split_chapter(chapter):
476     len_chapter = len(chapter)
477
478     if len_chapter % MAX_CHAPTER_LENGTH == 0: divisions = int(len_chapter/MAX_CHAPTER_LENGTH)
479     else: divisions = int(len_chapter/MAX_CHAPTER_LENGTH) + 1
480
481     div_chapters = []
482     while len(chapter) > MAX_CHAPTER_LENGTH:
483         div_chapters.append(chapter[:MAX_CHAPTER_LENGTH])
484         chapter = chapter[MAX_CHAPTER_LENGTH:]
485

```

```

486 if len(chapter) != 0: div_chapters.append(chapter)
487
488
489 if len(div_chapters) != divisions: raise Exception("[corenlp_wrapper] An error occurred while
    slicing chapters")
490
491 return div_chapters
492
493 def process_fics(fics):
494     if type(fics) == list:
495         #Check that members of list are class Fanfic
496         for i, fic in enumerate(fics):
497             if type(fic) is not Fanfic: raise TypeError("[corenlp_wrapper2] The input for the client must
                be list of Fanfic, or a single Fanfic")
498             else: #check that all chapters in fanfic are no longer than 100000 characters
499                 fic_chapters = fic.chapters
500
501                 for chapter in fic_chapters:
502                     #print(type(chapter), len(chapter))
503                     if len(chapter) > MAX_CHAPTER_LENGTH:
504                         #print('BIG BOY CHAPTER') #debug
505                         extra_chapters = split_chapter(chapter)
506                         fic_chapters.remove(chapter)
507
508                         position = i
509                         for extra_chap in extra_chapters:
510                             #print(len(extra_chap)) #debug
511                             fic_chapters.insert(position, extra_chap)
512                             position += 1
513
514                 #end FOR chapter
515
516                 fic.set_chapters(fic_chapters)
517
518             #end ELSE
519         #end FOR fic
520     #end if TYPE(FICS) == LIST
521     elif type(fics) == Fanfic:
522         fic_chapters = fics.chapters
523
524         for i, chapter in enumerate(fic_chapters):
525             if len(chapter) > MAX_CHAPTER_LENGTH:
526                 extra_chapters = split_chapter(chapter)
527                 fic_chapters.remove(chapter)
528
529                 position = i
530                 for extra_chap in extra_chapters:
531                     fic_chapters.insert(position, extra_chap)
532                     position += 1
533
534         fics.set_chapters(fic_chapters)

```

```

535
536
537     return fics
538
539
540     ### CLASSES ###
541     class CoreNLProcessor():
542     def __init__(self, fic): #fic must be annotated with CoreNLP data
543     try:
544     if fic.annotations is None: raise ValueError
545     else: self.fic = fic
546
547     except ValueError: print('This fanfic does not contain CoreNLP annotations')
548
549     def extract_fic_characters(self):
550     #Get canon_db
551     canon_db = pandas.read_csv(CANON_DB)
552
553     # Declarations
554     character_entities = []
555     coref_mentions = []
556     character_sentences = []
557     canonized_characters = [] #final result is stored here
558
559
560     for annotation in self.fic.annotations:
561     entities, mentions, sentences = extract_data_from_annotations(annotation)
562
563     character_entities = character_entities + entities
564     coref_mentions = coref_mentions + mentions
565     character_sentences = character_sentences + sentences
566
567     # Merge all character mentions into unique characters
568     unique_characters = merge_character_mentions(self.fic.index, character_entities,
569     coref_mentions)
570
571     # Link characters to their canon version, if it has one
572     canonized_characters = canonize_characters(unique_characters, canon_db)
573     #print(canonized_characters[:10]) #debug
574
575     self.fic.set_characters(canonized_characters)
576
577     def extract_fic_sentiment(self):
578     fic_sentences = []
579
580     for annotation in self.fic.annotations: fic_sentences.extend(annotation.sentence)
581
582     fic_sentiment = normalize_sentiment(fic_sentences)
583
584     return fic_sentiment

```

```

585 class CoreWrapper(): #This client is like CoreClient2 from corenlp_wrapper
586
587 def parse(self, fics):
588     fics = process_fics(fics)
589     if type(fics) == Fanfic: fics = [fics]
590
591     annotations = []
592
593     try:
594         with CoreNLPClient(
595             annotators = ['tokenize', 'sentiment', 'ssplit', 'pos', 'lemma', 'ner', 'parse', 'depparse', '
                    coref'],
596             timeout=120000,
597             be_quiet = True,
598             memory='4G') as client:
599             print("Annotating data . . .")
600
601             for i, fic in enumerate(fics):
602                 print('fic #', i, 'has ', len(fic.chapters))
603                 annotations = []
604                 for i, chapter in enumerate(fic.chapters):
605                     print('chapter ', i, ': ', len(chapter))
606                     ann = client.annotate(chapter)
607                     time.sleep(4)
608
609                     annotations.append(ann)
610
611                 fic.set_annotations(annotations)
612
613
614             print("...done")
615
616         except TimeoutException as e:
617             print('CoreNLP TimeoutException')
618             return fics, True
619
620
621         return fics, False

```

E. ANEXO: PROGRAMAS DE PRUEBA PARA ALGORITMO DE EXTRACCIÓN DE RELACIONES

E.1. toy_relex_kmeans

```
1  #!/usr/bin/bash/python3
2
3  from nltk import word_tokenize
4  from nltk.tokenize import RegexpTokenizer, sent_tokenize
5  from nltk.tag import pos_tag
6  from nltk.corpus import stopwords
7  from nltk.corpus import wordnet
8  from sklearn import metrics
9  from sklearn.cluster import KMeans
10 from sklearn.feature_extraction.text import TfidfVectorizer
11 from sklearn.decomposition import PCA
12 from fanfic_util import FanficGetter, Fanfic
13 from NER_tagger import NERTagger
14
15 from matplotlib import pyplot as plt
16 from pprint import pprint
17 import string, time, html2text
18 import numpy as np
19
20 ### VARIABLES ###
21 #FIC_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/html_fic_paths.txt'
22 RFIC_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/romance_fic_paths.txt'
23 FFIC_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/friendship_fic_paths.txt'
24 EFIC_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/enemy_fic_paths.txt'
25
26 COLOURS = ["r", "b", "y", "b", "g"]
27 NUM_CLUSTERS = 3
28
29 INTERESTING_POS = ['NN', 'NNS', 'JJ', 'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'] #I choose to
    only use nouns, adjectives and verbs (all kinds of verbs)
30 #INTERESTING_POS = ['NN', 'NNS', 'JJ', 'JJS', 'JJR']
31
32 ### FUNCTIONS ###
33
34 def get_lemma(word):
35     lemma = wordnet.morphy(word)
36
37     if lemma is None:
38         return word
39     else: return lemma
40
41 def get_stopwords():
42     # Tokenize and lemmatize stopwords
43     stop = stopwords.words('english')
44     #tokenizer = RegexpTokenizer(r'\w+')

```

```

45 #stop = [tokenizer.tokenize(word) for word in stop]
46 stop = [word_tokenize(word) for word in stop]
47
48 words = []
49 for item in stop:
50     if type(item) == list:
51         for w in item: words.append(w)
52
53     else: words.append(word)
54
55
56 words = [get_lemma(word) for word in words]
57 #print(words[:10]) #debug
58
59 return words
60
61
62 def fic_tokenizev1(fic):
63     sent_tokens = sent_tokenize(fic)
64     #tokenizer = RegexpTokenizer(r'\w+')
65     word_tokens = [word_tokenize(sen) for sen in sent_tokens]
66
67     #print(character_mentions) #debug
68
69     #Remove character names from text
70     words = []
71     for item in word_tokens:
72         if type(item) == list:
73             for w in item: words.append(w)
74
75         else: words.append(word)
76
77     #Lemmatize words
78     processed_tokens = [get_lemma(word) for word in words]
79
80     return processed_tokens
81
82 def fic_tokenizev2(fic):
83     sent_tokens = sent_tokenize(fic)
84     #tokenizer = RegexpTokenizer(r'\w+')
85     #pos_tokens = [pos_tag(tokenizer.tokenize(sent)) for sent in sent_tokens]
86     pos_tokens = [pos_tag(word_tokenize(sent)) for sent in sent_tokens]
87
88     interesting_words = []
89     for token in pos_tokens:
90         for word, pos in token:
91             #word = word.strip()
92             if pos in INTERESTING_POS: interesting_words.append(word)
93
94     processed_tokens = [get_lemma(word) for word in interesting_words]
95

```



```

96     return processed_tokens
97
98     def remove_characters(fic_texts):
99         ner_tagger = NERTagger()
100
101         for text in fic_texts:
102             sent_tokens = sent_tokenize(text)
103             pos_tokens = [pos_tag(word_tokenize(sent)) for sent in sent_tokens]
104             character_mentions = ner_tagger.parse(pos_tokens)
105
106             for name, _ in character_mentions.items(): text = text.replace(name, '')
107
108         return fic_texts
109
110     def cluster_texts(texts, name_labels):
111         #print(texts[0][:100]) #debug
112
113         print("Creating stopwords and removing character names from texts...")
114         start = time.time()
115
116         stop_words = get_stopwords()
117         #texts = [remove_characters(text) for text in texts]
118
119         end = time.time()
120         print("...done in ", (end-start)/60, " mins")
121
122         print("Creating vectorizer, transforming texts to tf-idf coordinates...")
123         start = time.time()
124         #vectorizer = TfidfVectorizer(tokenizer=fic_tokenizev1, stop_words=stop_words, max_df=0.5,
125                                     min_df=0.3, lowercase=True)
126         vectorizer = TfidfVectorizer(tokenizer=fic_tokenizev2, stop_words=stop_words, max_df=0.5,
127                                     min_df=0.3, lowercase=True)
128
129         #print(type(texts),type(texts[0])) #debug
130         vectorized_data = vectorizer.fit_transform(texts)
131         #print(vectorizer.get_feature_names()) #debug
132
133         end = time.time()
134         print("...done in ", (end-start)/60, " mins")
135
136         print("Creating the K-Means model and fitting data..")
137         start = time.time()
138         k_labels = np.unique(name_labels).shape[0]
139         km_model = KMeans(n_clusters=k_labels, init='k-means++', n_init=10)
140
141         data = km_model.fit_transform(vectorized_data)
142         centroids = km_model.cluster_centers_
143         #print(len(data)) #debug
144         #print(centroids) #debug
145
146         end = time.time()

```

```

145 print("...done in ", (end-start)/60, "mins")
146 print("n_samples: %d, n_features: %d" % data.shape)
147
148 print("\n == Metrics ==")
149 print("Homogeneity: %0.3f" % metrics.homogeneity_score(name_labels, km_model.labels_))
150 print("Completeness: %0.3f" % metrics.completeness_score(name_labels, km_model.labels_))
151 print("V-measure: %0.3f" % metrics.v_measure_score(name_labels, km_model.labels_))
152 print("Adjusted Rand-Index: %0.3f" % metrics.adjusted_rand_score(name_labels, km_model.labels_)
153       )
154 print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(data, km_model.labels_,
155       sample_size=100))
156
157 print("\nTop terms per cluster:")
158 order_centroids = km_model.cluster_centers_.argsort()[:, :-1]
159 terms = vectorizer.get_feature_names()
160 for i in range(k_labels):
161     print('Cluster %d:' % i)
162     all_terms = ''
163     for j in order_centroids[i, :10]: all_terms += terms[j]+' '
164     print(all_terms+'\n')
165
166 ### Plot the data
167
168 model = PCA(n_components=2)
169 scatter_points = model.fit_transform(vectorized_data.toarray())
170 kmean_indices = km_model.fit_predict(vectorized_data)
171
172 x_axis = [x[0] for x in scatter_points]
173 y_axis = [y[1] for y in scatter_points]
174
175 plt.scatter(x_axis, y_axis, c=[COLOURS[d] for d in kmean_indices])
176
177 plt.show()
178
179
180
181 ### MAIN ###
182
183 print("Fetching fic texts...")
184 start = time.time()
185
186 getter = FanficGetter()
187
188 getter.set_fic_listing_path(EFIC_LISTING_PATH)
189 efics = getter.get_fanfics_in_list()
190 elabels = ['enemy'] * len(efics)
191
192 getter.set_fic_listing_path(FFIC_LISTING_PATH)
193 ffics = getter.get_fanfics_in_range(0,180)

```

```

194 flabels = ['friendship'] * len(ffics)
195
196 getter.set_fic_listing_path(RFIC_LISTING_PATH)
197 rfics = getter.get_fanfics_in_range(0,220) # There are a lot of romance fanfics, so we're
      going to tone it down a bit
198 rlabels = ['romance'] * len(rfics)
199
200 fics = efics + ffics + rfics
201 name_labels = elabels + flabels + rlabels
202
203
204 fanfic_texts = [fic.get_string_chapters() for fic in fics]
205 #print(len(fics)) #debug
206 #print(len(fanfic_texts), len(name_labels)) #debug
207
208 end = time.time()
209
210 print "...fics fetched. Elapsed time: ", (end-start)/60, " mins")
211
212 cluster_texts(fanfic_texts, name_labels)

```

E.2. toy_relex_topic

```
1  #!/usr/bin/bash/python3
2
3  from nltk import word_tokenize
4  from nltk.tokenize import RegexpTokenizer, sent_tokenize
5  from nltk.tag import pos_tag
6  from nltk.corpus import stopwords
7  from nltk.corpus import wordnet
8  from gensim import corpora
9
10 import pandas as pd
11 from fanfic_util import FanficGetter, Fanfic
12
13 import string, pickle, gensim, sys, time
14
15 ### VARIABLES ###
16 FIC_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/html_fic_paths.txt'
17 RFIC_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/romance_fic_paths.txt'
18 FFIC_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/friendship_fic_paths.txt'
19 EFIC_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/enemy_fic_paths.txt'
20
21 MODEL_NAME = 'model0.gensim' #default name
22 MODEL_PATH = '/home/maria/Documents/Fanfic_ontology/TFG_models/'
23 DICTIONARY_PATH = '/home/maria/Documents/Fanfic_ontology/TFG_dictionaries/'
24 NUM_TOPICS = 3
25
26 #INTERESTING_POS = ['NN', 'NNS', 'JJ', 'JJS', 'JJR', 'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'] #
    nouns, adjectives and verbs (all kinds) aka C
27 #INTERESTING_POS = ['NN', 'NNS', 'RB', 'RBR', 'RBS', 'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ']
    #nouns, adverbs and verbs (all kinds) aka B
28 #INTERESTING_POS = ['NN', 'NNS', 'RB', 'RBR', 'RBS', 'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ',
    'UH'] #nouns, adverbs and verbs (all kinds) aka BUH
29 #INTERESTING_POS = ['NN', 'NNS', 'JJ', 'JJS', 'JJR', 'RB', 'RBR', 'RBS', 'VB', 'VBD', 'VBG', '
    VBN', 'VBP', 'VBZ'] #nouns, adjectives, adverbs and verbs (all kinds) aka D
30 #INTERESTING_POS = ['NN', 'NNS', 'JJ', 'JJS', 'JJR', 'RB', 'RBR', 'RBS', 'VB', 'VBD', 'VBG', '
    VBN', 'VBP', 'VBZ', 'UH'] #nouns, adjectives, adverbs and verbs (all kinds) aka DUH
31 INTERESTING_POS = ['JJ', 'JJS', 'JJR', 'RB', 'RBR', 'RBS', 'VB', 'VBD', 'VBG', 'VBN', 'VBP', '
    VBZ', 'UH', 'RP', 'IN', 'CC'] #nouns, adjectives, adverbs and verbs (all kinds) aka DUH2
32 #INTERESTING_POS = ['NN', 'NNS', 'JJ', 'JJS', 'JJR', 'RB', 'RBR', 'RBS'] #nouns, adjectives
    and adverbs (all kinds)
33
34 UNINTERESTING_POS = ['PRP', 'PRP$', 'POS', 'CC', 'CD', 'TO', 'DT', 'IN']
35
36 ### FUNCTIONS ###
37
38 def get_lemma(word):
39     lemma = wordnet.morphy(word)
40
41     if lemma is None:
42         return word
```

```

43     else: return lemma
44
45     def get_stopwords():
46         # Tokenize and lemmatize stopwords
47         stop = stopwords.words('english')
48         #tokenizer = RegexpTokenizer(r'\w+')
49         #stop = [tokenizer.tokenize(word) for word in stop]
50         stop = [word_tokenize(word) for word in stop]
51
52         words = []
53         for item in stop:
54             if type(item) == list:
55                 for w in item: words.append(w)
56
57             else: words.append(word)
58
59
60         words = [get_lemma(word) for word in words]
61         #print(words[:10]) #debug
62
63         return words
64
65
66     def fic_tokenizev1(fic):
67         sent_tokens = sent_tokenize(fic)
68         #tokenizer = RegexpTokenizer(r'\w+')
69         #word_tokens = [tokenizer.tokenize(sent) for sent in sent_tokens]
70         word_tokens = [word_tokenize(sent) for sent in sent_tokens]
71
72         words = []
73         for item in word_tokens:
74             if type(item) == list:
75                 for w in item: words.append(w)
76
77             else: words.append(word)
78
79
80         processed_tokens = [get_lemma(word) for word in words]
81
82         return processed_tokens
83
84     def fic_tokenizev2(fic):
85         sent_tokens = sent_tokenize(fic)
86         #tokenizer = RegexpTokenizer(r'\w+')
87         #pos_tokens = [pos_tag(tokenizer.tokenize(sent)) for sent in sent_tokens]
88         pos_tokens = [pos_tag(word_tokenize(sent)) for sent in sent_tokens]
89
90         interesting_words = []
91         for token in pos_tokens:
92             for word, pos in token:
93                 #word = word.strip()

```

```

94     if pos not in UNINTERESTING_POS: interesting_words.append(word)
95
96     processed_tokens = [get_lemma(word) for word in interesting_words]
97
98     return processed_tokens
99
100 def process_text(unprocessed_fics):
101     processed_fics = []
102     for fic in unprocessed_fics:
103         #Get lemmatized tokens
104         #tokens = fic_tokenizev1(fic)
105         tokens = fic_tokenizev2(fic)
106
107         #Filter out stopwords
108         stopwords = get_stopwords()
109
110         word_tokens = [word for word in tokens if word not in stopwords]
111         processed_fics.append(word_tokens)
112
113     return processed_fics
114
115
116 ### MAIN ###
117
118 if len(sys.argv) == 2:
119     start_time = time.time()
120     MODEL_NAME = (sys.argv[1]).strip()
121     print(MODEL_NAME) #debug
122
123
124     print('Fetching fic texts...')
125     start = time.time()
126
127     getter = FanficGetter()
128
129
130     getter.set_fic_listing_path(EFIC_LISTING_PATH)
131     efics = getter.get_fanfics_in_list()
132
133     getter.set_fic_listing_path(FFIC_LISTING_PATH)
134     ffics = getter.get_fanfics_in_range(0,180)
135
136     getter.set_fic_listing_path(RFIC_LISTING_PATH)
137     rfics = getter.get_fanfics_in_range(0,220) # There are a lot of romance fanfics, so we're
        going to tone it down a bit
138
139     fics = efics + ffics + rfics
140
141
142
143     fanfic_texts = [fic.get_string_chapters() for fic in fics]

```

```

144 #print(len(fics)) #debug
145 #print(len(fanfic_texts), len(name_labels)) #debug
146
147 end = time.time()
148
149 print ("...fics fetched. Elapsed time: ", (end-start)/60, " mins")
150
151
152 print ('Preprocessing fanfics and creating dictionaries...')
153 start_time = time.time()
154
155 processed_fics = process_text(fanfic_texts)
156 #print(type(processed_fics), processed_fics[0][:10]) #debug
157
158 dictionary = corpora.Dictionary(processed_fics)
159 corpus = [dictionary.doc2bow(fic) for fic in processed_fics]
160
161 #pickle.dump(corpus, open('corpus.pkl', 'wb'))
162 dictionary.save(DICTIONARY_PATH+MODEL_NAME+'_dictionary.gensim')
163
164 end_time = time.time()
165 print ('Processing elapsed time: ', (end_time-start_time)/60, ' minutes')
166
167 start_time = time.time()
168 print ('Training LDA model. . .')
169 ldamodel = gensim.models.ldamodel.LdaModel(corpus=corpus, num_topics=NUM_TOPICS, id2word=
    dictionary, passes=20)
170 ldamodel.save(MODEL_PATH + 'lda_' + MODEL_NAME + '.gensim')
171 coherence = gensim.models.coherencemodel.CoherenceModel(model=ldamodel, texts=processed_fics,
    dictionary=dictionary, coherence='c_v')
172
173 end_time = time.time()
174 print ('...LDA elapsed time: ', (end_time-start_time)/60, ' minutes')
175
176 print ('LDA Coherence score: ', coherence.get_coherence())
177 topics = ldamodel.print_topics(num_words=10)
178
179 print ('Topics in LDA model:')
180 for topic in topics: print(topic)
181
182
183 else:
184 print ('Error. Correct usage: \ntopic.py <MODEL_NAME>')

```

E.3. toy_relex_v2

```
1  #!/bin/bash/python3
2
3  import sys, time, pickle, pandas, numpy
4  from sklearn.cluster import KMeans
5  from sklearn.feature_extraction import DictVectorizer
6  from sklearn.feature_extraction.text import TfidfTransformer
7  from stanza.server import CoreNLPClient
8
9  from NER_tagger import NERTagger
10 from fanfic_util import FanficGetter, Fanfic
11
12 import matplotlib.pyplot as plt
13
14 ### VARIABLES ###
15
16 VERB_TAGS = ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ']
17 COLORMAP = {0: 'red', 1: 'blue'}
18
19 ### FUNCTIONS ###
20
21 def get_longest_lists(coref_chains): #returns the two longest chains in the coreference graph
22     longest = []
23     longest.append(max(list(coref_chains), key=len))
24
25     coref_chains.remove(longest[0])
26
27     longest.append(max(list(coref_chains), key=len))
28
29     return longest
30
31 def print_coref_mention(mention):
32     print(mention.mentionID, mention.corefClusterID, mention.mentionType, mention.gender, mention.
33           animacy, mention.number)
34
35 def print_ner_mention(mention):
36     print(mention.entityMentionIndex, mention.canonicalEntityMentionIndex, mention.ner, mention.
37           gender, mention.entityMentionText)
38
39 ### CLASSES ###
40
41 class CharacterMention():
42     def __init__(self, ID, word, canonicalName, gender, animacy, number, corefClusterID,
43                 nerEntityID, corefMentions, nerMentions):
44         self.ID = ID
45         self.word = word
46         self.canonicalName = canonicalName
47         self.gender = gender
48         self.animacy = animacy
49         self.number = number
```



```

47
48 self.corefClusterID = corefClusterID
49 self.nerEntityID = nerEntityID
50 self.corefMentions = corefMentions
51 self.nerMentions = nerMentions
52
53
54 def getDictRepresentation(self):
55     return {
56         'name': self.canonicalName,
57         'gender': self.gender,
58         'animacy': self.animacy,
59         'number': self.number,
60
61         'clusterID': self.corefClusterID,
62         'nerID': self.nerEntityID,
63     }
64
65 def getID(self): return self.ID
66
67
68 ### MAIN ###
69
70 fGetter = FanficGetter()
71 NERtagger = NERTagger()
72
73 fic_list = fGetter.get_fanfics_in_range(8,9)
74
75 fic_text = ''
76
77 for i in range(7,10):
78     fic_text += fic_list[0].get_chapter(i)
79
80 #print(fic_text) #debug
81
82
83 print("\n##### Starting client and calling CoreNLP server #####\n")
84 start= time.time()
85
86 sentences = []
87 nerMentions = []
88 corefMentions = []
89 coref_chains = []
90 with CoreNLPClient(
91     annotators = ['tokenize', 'ssplit', 'pos', 'lemma', 'ner', 'parse', 'depparse', 'coref'],
92     timeout=120000,
93     be_quiet = True,
94     memory='4G') as client:
95
96     print("Annotating data . . .")
97     ann = client.annotate(fic_text)

```

```

98
99     print ("...done")
100
101     sentences = ann.sentence
102     nerMentions = ann.mentions #NERMention[]
103     corefMentions = ann.mentionsForCoref #Mention[]
104     chain = ann.corefChain #CorefChain[], made up of CorefMention[]
105
106     for i in range(0, len(chain)): #debug
107         coref_chains.append(chain[i].mention)
108
109
110     #print(len(coref_chains)) #debug
111
112     end = time.time()
113     print ("Client closed. " + str((end-start)/60) + " mins elapsed")
114
115     start = time.time()
116     print ("Processing annotation data...")
117
118
119     coref_chains = get_longest_lists(coref_chains) #debug
120
121     characterMentions = []
122     i = 0
123     for chain in coref_chains:
124         #print("==== CHAIN #" + str(i) + "====") #for visualization purposes
125         for mention in chain:
126             senIndex = mention.sentenceIndex
127             tokBIndex = mention.beginIndex
128             tokEIndex = mention.endIndex
129             clusterID = corefMentions[mention.mentionID].corefClusterID
130             entityID = nerMentions[sentences[mention.sentenceIndex].token[mention.beginIndex].
                entityMentionIndex].canonicalEntityMentionIndex
131             entityName = str(nerMentions[sentences[mention.sentenceIndex].token[mention.beginIndex].
                entityMentionIndex].entityMentionText)
132             mentionText = sentences[mention.sentenceIndex].token[mention.beginIndex].originalText
133
134             character = CharacterMention(i, mentionText, entityName, corefMentions[mention.mentionID].
                gender, corefMentions[mention.mentionID].animacy, corefMentions[mention.mentionID].number,
                clusterID, entityID, [], [])
135             characterMentions.append(character)
136
137             #if i < 10: print(character.getDictRepresentation()) #debug
138
139             i+=1
140
141
142     #print("Sentence " + senIndex + " |          tokens " + tokBIndex + "-" + tokEIndex + " |          "+
        mention.mentionType + "          |          cluster " + clusterID + "          |          entity "+
        entityID + " " + entityName + " |          text: " + mentionText) #for visualization purposes

```

```

143
144 print("CanonicalEntityMentionIndex for nerMentions[0]: ",nerMentions[0].
    canonicalEntityMentionIndex) #debug
145 #print(len(characterMentions)) #debug
146 characterDicts = [char.getDictRepresentation() for char in characterMentions]
147
148 end = time.time()
149 print("Annotations processed. " + str((end-start)/60) + " mins elapsed")
150
151 print("\n##### Starting clustering process #####\n")
152
153 start = time.time()
154 print("Get dictionary and Tfidf vectorization...")
155 vec1 = DictVectorizer()
156 vec2 = TfidfTransformer() #this one is for normalization
157
158 vec_data = vec1.fit_transform(characterDicts) #toarray?
159 #print("before tfidf", vec_data.shape) #debug
160 #vec_data = vec2.fit_transform(vec_data)
161 #print("after tfidf", vec_data.shape) #debug
162 #print(vec1.get_feature_names()) #debug
163
164 data = pandas.DataFrame(vec_data.toarray(), columns = vec1.get_feature_names())
165
166
167 end = time.time()
168 print("...done. " + str((end-start)/60) + " mins elapsed.")
169
170 start = time.time()
171 print("Initializing and fitting KMeans model...")
172 model = KMeans(init='k-means++', n_clusters=2, n_init=5)
173 model.fit(data)
174 predict = model.predict(data)
175
176 data['category'] = model.labels_
177
178 #for i in range(0,1): print(data.iloc[i,:]) #debug
179 end = time.time()
180 print("...done. " + str((end-start)/60) + " mins elapsed.")
181
182 # Print who belongs to which cluster
183 #print(data.shape) #debug
184
185 #print(data[['category','clusterID','nerID']].values) #debug
186
187 print("Navigate sentences with relevant keyword verb 'love'\n")
188 for sentence in sentences:
189     print(sentence = False
190     sen = ''
191     ner_indexes = []
192     coref_indexes = []

```

```

193
194 for token in sentence.token:
195     #sen += token.originalText+' '
196
197     if token.pos in VERB_TAGS and token.originalText == 'love':
198         printsentence = True
199         #print(token.corefMentionIndex)
200
201     #for mention in sentence.mentions: print_ner_mention(mention)
202     #for mention in sentence.mentionsForCoref: print_coref_mention(mention)
203
204     if printsentence:
205         for token in sentence.token:
206             if token.ner == 'PERSON':
207                 print(token.originalText, token.ner, token.gender)
208                 print(nerMentions[token.entityMentionIndex].canonicalEntityMentionIndex, nerMentions[token.
                    entityMentionIndex].ner, nerMentions[token.entityMentionIndex].gender, nerMentions[token.
                    entityMentionIndex].entityMentionText)
209             if len(token.corefMentionIndex) > 0:
210                 for index in token.corefMentionIndex: print(corefMentions[index].corefClusterID)
211
212             elif len(token.corefMentionIndex) > 0:
213                 for index in token.corefMentionIndex:
214                     if corefMentions[index].mentionType == 'PRONOMINAL':
215                         character = list(filter(lambda char: char['clusterID'] == corefMentions[index].corefClusterID,
                            characterDicts)) #find the character with the same cluster ID
216
217                 if len(character) < 1:
218                     print(token.originalText, corefMentions[index].corefClusterID, corefMentions[index].
                        mentionType, corefMentions[index].gender, corefMentions[index].number, corefMentions[index]
                        ].animacy)
219
220                 else:
221                     character = character[0]
222                     if character['name'] != '': print(character['name'], corefMentions[index].corefClusterID,
                        corefMentions[index].mentionType, corefMentions[index].gender, corefMentions[index].number
                        , corefMentions[index].animacy)
223
224                 else: print(token.originalText, corefMentions[index].corefClusterID, corefMentions[index].
                        mentionType, corefMentions[index].gender, corefMentions[index].number, corefMentions[index]
                        ].animacy)
225
226                 else: print(token.originalText, corefMentions[index].corefClusterID, corefMentions[index].
                        mentionType, corefMentions[index].gender, corefMentions[index].number, corefMentions[index]
                        ].animacy)
227
228                 else: print(token.originalText)
229
230                 print("\n\n")

```

E.4. ner_and_sen_extraction_v2

```
1  #!/bin/bash/python3
2
3  import sys, time, pandas, numpy
4  from corenlp_wrapper import CoreClient2
5  from stanza.server import Document
6  from nltk.tag import pos_tag
7  from nltk.tokenize import sent_tokenize, word_tokenize
8  #from urllib.error import HTTPError
9
10 from NER_tagger_v3 import NERTagger
11 from fanfic_util import FanficGetter, FanficHTMLHandler, Fanfic
12
13 ### VARIABLES ###
14 CHARACTERS_TO_CSV = '/home/maria/Documents/Fanfic_ontology/fic_characters.csv'
15 SENTENCES_TO_CSV = '/home/maria/Documents/Fanfic_ontology/fic_sentences.csv'
16 CANON_DB = '/home/maria/Documents/Fanfic_ontology/canon_characters.csv'
17 ERRORLOG = '/home/maria/Documents/Fanfic_ontology/TFG_logs/ner_and_sen_extraction_v2_errorlog.
    txt'
18
19 ROMANCE_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/romance_fic_paths_shortened.txt'
20 FRIENDSHIP_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/
    friendship_fic_paths_shortened.txt'
21 ENEMY_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/enemy_fic_paths3.txt'
22
23 FEMALE_TAGS = ['She/Her Pronouns for ', 'Female ', 'Female!', 'Female-Presenting ']
24 MALE_TAGS = ['He/Him Pronouns for ', 'Male ', 'Male!', 'Male-Presenting ']
25 NEUTRAL_TAGS = ['They/Them Pronouns for ', 'Gender-Neutral Pronouns for', 'Gender-Neutral ', '
    Agender ', 'Genderfluid ', 'Androgynous ', 'Gender Non-Conforming ']
26
27 ### FUNCTIONS ###
28
29 def print_coref_mention(mention, sent):
30     token = sent.token[mention.headIndex]
31     coref_cluster = token.corefClusterID
32     #print(mention.mentionID, coref_cluster, mention.mentionType, token.originalText, mention.
        gender, mention.animacy, mention.number)
33
34 def print_ner_mention(mention):
35     print(mention.entityMentionIndex, mention.canonicalEntityMentionIndex, mention.ner, mention.
        gender, mention.entityMentionText)
36
37 def merge_character_mentions(fic_index, character_entities, character_mentions,
        tagger_characters):
38     # create characters from their NER IDs
39     ner_ids = [char['nerID'] for char in character_entities]
40     ner_ids = set(ner_ids) #remove duplicates
41
42     characters = []
43     for ner in ner_ids:
```

```

44 character = {}
45 character['ficID'] = fic_index
46 character['nerID'] = [ner]
47 character['Name'] = 'Jane Doe' #filler
48 character['Gender'] = 'X' #filler
49 character['Mentions'] = 0
50 character['clusterID'] = [-1] #filler
51 character['canonID'] = -1 #to be used later
52 characters.append(character)
53
54 # fill in character information using its clusters and mentions
55 for char in characters:
56     for mention in character_mentions:
57         if mention['nerID'] == char['nerID'][0]:
58             no_cluster = False
59             char['Mentions'] = char['Mentions']+1
60
61         if mention['MentionT'] == 'PROPER':
62             char['Name'] = mention['Name']
63             char['Gender'] = mention['Gender']
64
65     cluster = char['clusterID']
66     if mention['clusterID'] not in cluster: char['clusterID'].append(mention['clusterID'])
67
68     for char in characters:
69         if len(char['clusterID']) == 1:
70             ner_char = list(filter(lambda entity: entity['nerID'] == char['nerID'][0], character_entities)
71                                 )
72             char['Name'] = ner_char[0]['Name']
73             char['Gender'] = ner_char[0]['Gender']
74             char['Mentions'] = len(ner_char)
75
76     for char in characters: char['clusterID'].remove(-1) #remove filler
77
78     # merge repeated character entries into single entry
79     aux = characters.copy()
80     for char in characters:
81         aux.remove(char)
82     for c in aux:
83         cluster = c['clusterID']
84         if len(cluster) == 1 and cluster[0] in char['clusterID']:
85             if char['Name'] == c['Name'] and char['Gender'] == c['Gender']:
86                 char['Mentions'] = char['Mentions']+c['Mentions']
87                 char['nerID'].append(c['nerID'][0])
88
89     aux.remove(c)
90     characters.remove(c)
91
92     # merge in the NERTagger characters
93     for tagger_char, mentions in tagger_characters.items():

```

```

94  for char in characters:
95  if tagger_char.lower() == char['Name'].lower(): #add characters mentions if the character is
    already named
96  if mentions > char['Mentions']:
97  char['Mentions'] += mentions-char['Mentions']
98  break
99
100
101  return characters
102
103  def link_characters_to_canon(characters, canon_db):
104
105  for character in characters:
106  for index, canon_character in canon_db.iterrows():
107  if type(canon_character['Other names']) == str:
108  other_names = [name.lower() for name in canon_character['Other names'].split(',')]
109  #print(other_names) #debug
110
111  else: other_names = ['']
112
113  if canon_character['Name'].lower() == character['Name'].lower():
114  #print(canon_character['Name'].lower(), character['Name'].lower()) #debug
115  character['canonID'] = index
116  break
117
118  elif character['Name'].lower() in [name.lower() for name in canon_character['Name'].split(' ')]
    ]:
119  character['canonID'] = index
120  break
121
122  elif character['Name'].lower() in other_names:
123  character['canonID'] = index
124  break
125
126  #if a character is not canon its canonID will not be changed from -1
127
128  return characters
129
130  def make_gender_tags(tags, character_name):
131  for tag in tags:
132  tag = tag+character_name
133
134  return tags
135
136  def decide_gender(characters, canon_db):
137  handler = FanficHTMLHandler()
138
139  for character in characters:
140  if character['Gender'] == 'UNKNOWN':
141  fic_link = '/home/maria/Documents/Fanfic_ontology/TFG_fics/html/gomensfanfic_'+str(character['
    ficID'])+'.html'

```

```

142 fic_tags = handler.get_tags(fic_link)
143 character_name = character['Name'].capitalize()
144
145 f_gender = make_gender_tags(FEMALE_TAGS, character_name)
146 m_gender = make_gender_tags(MALE_TAGS, character_name)
147 n_gender = make_gender_tags(NEUTRAL_TAGS, character_name)
148
149 male = female = neutral = False
150
151 if any(tag in fic_tags for tag in f_gender): female = True
152 elif any(tag in fic_tags for tag in m_gender): male = True
153 elif any(tag in fic_tags for tag in n_gender): neutral = True
154
155 if female and male: character['Gender'] = 'NEUTRAL'
156 elif female: character['Gender'] = 'FEMALE'
157 elif male: character['Gender'] = 'MALE'
158 elif neutral: character['Gender'] = 'NEUTRAL'
159 else: #if there are no gender tags applicable to the character we assume the canon gender
160 if character['canonID'] > -1: #if the character is not canon the gender will remain unknown
161 canon_gender = canon_db.iloc[character['canonID']]['Gender']
162 character['Gender'] = canon_gender
163
164 return characters
165
166
167 def get_longest_lists(coref_chains): #returns the two longest chains in the coreference graph
168 longest = []
169 longest.append(max(list(coref_chains), key=len))
170
171 coref_chains.remove(longest[0])
172
173 longest.append(max(list(coref_chains), key=len))
174
175 return longest
176
177 def merge_sentences(fic_index, fic_dataset, character_sentences):
178 sen_ids = []
179 merged_sentences = []
180
181 for sentence in character_sentences:
182 sent = {}
183 if sentence['senID'] in sen_ids: #If a dict for this sentence already exists
184 sent = list(filter(lambda s: s['senID'] == sentence['senID'], merged_sentences))
185 #print(len(sent)) #debug
186 sent = sent[0]
187
188 if sentence['nerIDs'] > -1 and sentence['nerIDs'] not in sent['nerIDs']: sent['nerIDs'].append
    (sentence['nerIDs'])
189 if sentence['Clusters'] > -1 and sentence['Clusters'] not in sent['Clusters']: sent['Clusters'
    ].append(sentence['Clusters'])
190

```



```

191 else: #Create new sentence dict
192     sen_ids.append(sentence['senID'])
193
194     sent['ficID'] = fic_index
195     sent['ficDataset'] = fic_dataset
196     sent['senID'] = sentence['senID']
197     sent['Sentiment'] = sentence['Sentiment']
198     sent['Verbs'] = sentence['Verbs']
199
200     if sentence['nerIDs'] > -1: sent['nerIDs'] = [sentence['nerIDs']]
201     else: sent['nerIDs'] = []
202
203     if sentence['Clusters'] > -1: sent['Clusters'] = [sentence['Clusters']]
204     else: sent['Clusters'] = []
205
206     merged_sentences.append(sent)
207
208     return merged_sentences
209
210 def extract_data_from_annotations(annotation):
211     sentences= annotation.sentence
212     all_ner_mentions = annotation.mentions #NERMention[]
213     all_coref_mentions = annotation.mentionsForCoref #Mention[]
214     chains = annotation.corefChain #CorefChain[], made up of CorefMention[]
215
216
217     #coref_chains = []
218     #for i in range(0, len(chains)): #debug
219     #     coref_chains.append(chains[i].mention)
220     #coref_chains = get_longest_lists(coref_chains)
221
222     # lists to store the return values in
223     character_entities = []
224     character_mentions = []
225     character_sentences = []
226     ## Extract characters from NER and coreference mentions ##
227     if len(all_ner_mentions) > 0:
228         for ner in all_ner_mentions:
229             if ner.ner == 'PERSON':
230                 sentence = {}
231                 sen = sentences[ner.sentenceIndex]
232
233                 string_sen = ''
234                 for token in sen.token:
235                     string_sen += ' '+token.originalText
236
237                 sentence['senID'] = sen.sentenceIndex
238                 sentence['Sentiment'] = sen.sentiment
239                 #sentence['Verbs'] = get_lemmatized_verbs(string_sen)
240                 sentence['Verbs'] = string_sen
241                 sentence['nerIDs'] = ner.canonicalEntityMentionIndex

```

```

242 sentence['Clusters'] = -1 #filler
243 character_sentences.append(sentence)
244
245 character = {}
246 #token = tokens[ner.tokenStartInSentenceInclusive]
247 #coref_in_token = token.
248
249 character['senID'] = sen.sentenceIndex
250 character['nerID'] = ner.canonicalEntityMentionIndex
251 character['nerMentionIndex'] = ner.entityMentionIndex
252 character['Name'] = ner.entityMentionText
253 character['Gender'] = ner.gender
254 character['MentionT'] = 'PERSON'
255 character_entities.append(character)
256
257 #end FOR ner
258 #end IF len
259 if len(all_coref_mentions) > 0:
260     for mention in all_coref_mentions:
261         if mention.mentionType == 'NOMINAL':
262             print_coref_mention(mention, sentences[mention.sentNum])
263
264         elif mention.mentionType in ['PROPER', 'PRONOMINAL']:
265             sentence = {}
266             sen = sentences[mention.sentNum]
267
268             string_sen = ''
269             for token in sen.token:
270                 string_sen += ' ' + token.originalText
271
272             sentence['senID'] = sen.sentenceIndex
273             sentence['Sentiment'] = sen.sentiment
274             #sentence['Verbs'] = get_lemmatized_verbs(string_sen)
275             sentence['Verbs'] = string_sen
276             sentence['nerIDs'] = -1 #filler
277             sentence['Clusters'] = mention.corefClusterID
278
279             character_sentences.append(sentence)
280
281             if mention.sentNum != sen.sentenceIndex: print('false')
282
283             character = {}
284
285             token = sen.token[mention.headIndex]
286             ner_in_token = all_ner_mentions[token.entityMentionIndex]
287             character['senID'] = sen.sentenceIndex
288             character['nerID'] = ner_in_token.canonicalEntityMentionIndex
289             character['clusterID'] = mention.corefClusterID
290             character['Name'] = mention.headString
291             character['Gender'] = mention.gender
292             character['MentionT'] = mention.mentionType

```

```

293 character_mentions.append(character)
294
295
296 #end FOR mention
297 #end IF len
298
299 return character_entities, character_mentions, character_sentences
300
301 def character_and_sentence_extraction(fics):
302     # Declarations
303     client = CoreClient2()
304     NERtagger = NERTagger()
305     #chains = []
306
307     print('\n##### Starting CoreNLP server and processing fanfics. . .#####\n')
308     start= time.time()
309
310
311     annotated_fics, error = client.parse(fics)
312     if error:
313         print('Error: not all fanfics could be processed by the server')
314         recovered_fics = []
315         for fic in annotated_fics:
316             if fic.annotations is not None: recovered_fics.append(fic)
317
318         print('Program is continuing with ', len(recovered_fics), ' of the original ', len(fics), ' fics')
319         annotated_fics = recovered_fics
320
321
322
323     end = time.time()
324     print("Client closed. " + str((end-start)/60) + " mins elapsed")
325
326     print("Processing annotation data...")
327     start = time.time()
328
329
330
331     num_fics = len(annotated_fics)-1
332     for i, fic in enumerate(annotated_fics):
333         # Declarations
334         character_entities = []
335         character_mentions = []
336         character_sentences = []
337
338         # First we process the character entities with NERTagger
339         print('Processing fic ' + str(i) + ' of ' + str(num_fics) + ' (Fic #' + str(fic.index) + ')')
340
341         # Preprocess and tag characters with NERTagger
342         processed_fic = preprocess_fic(fic)
343         tagger_characters = NERtagger.parse(processed_fic)

```

```

344 #print(tagger_characters) #debug
345
346 canonized_characters = []
347 for annotation in fic.annotations:
348     entities, mentions, sentences = extract_data_from_annotations(annotation)
349
350     character_entities = character_entities + entities
351     character_mentions = character_mentions + mentions
352     character_sentences = character_sentences + sentences
353
354     # Merge all character mentions into unique characters
355     unique_characters = merge_character_mentions(fic.index, character_entities, character_mentions
        , tagger_characters)
356
357     # Link characters to their canon version, if it has one
358     canonized_characters = link_characters_to_canon(unique_characters, canon_db)
359     #print(canonized_characters[:10])
360
361     # Decide genders for all characters
362     canonized_characters = decide_gender(canonized_characters, canon_db)
363
364     # Merge all sentences into unique sentences with the characters they mention
365     merged_sentences = merge_sentences(fic.index, fic.dataset, character_sentences)
366     #print(merged_sentences[:10]) #debug
367
368     fic.set_characters(canonized_characters)
369     fic.set_sentences(merged_sentences)
370
371
372 end = time.time()
373 print(". . .annotation data processed. " + str((end-start)/60) + " mins elapsed")
374
375
376 return annotated_fics
377
378 def preprocess_fic(fic):
379     tagged_fic = []
380
381     for chapter in fic.chapters:
382         tokenized_chapter = [word_tokenize(sent) for sent in sent_tokenize(chapter)]
383         tagged_chapter = [pos_tag(word) for word in tokenized_chapter]
384
385         tagged_fic.extend(tagged_chapter)
386
387     return tagged_fic
388
389 def get_fanfics(start, end, dataset):
390     fGetter = FanficGetter()
391     fics = []
392     if dataset == 'r':
393         fGetter.set_fic_listing_path(ROMANCE_LISTING_PATH)

```

```

394 fics = fGetter.get_fanfics_in_range(start, end) #get ROMANCE fanfics
395
396 elif dataset == 'f':
397     fGetter.set_fic_listing_path(FRIENDSHIP_LISTING_PATH)
398     fics = fGetter.get_fanfics_in_range(start, end) #get FRIENDSHIP fanfics
399
400 elif dataset == 'e':
401     fGetter.set_fic_listing_path(ENEMY_LISTING_PATH)
402     fics = fGetter.get_fanfics_in_range(start, end) #get ENEMY fanfics
403
404 else: print('Dataset '+dataset+' does not exist')
405
406 if dataset != 'e': #ENEMY dataset can contain more than one chapter per fanfic
407     for fic in fics:
408         if len(fic.chapters) != 1:
409             f = open(ERRORLOG, 'a')
410             f.write('Error on fanfic #'+str(fic.index)+' from dataset '+dataset+' : num chapters = '+str(
411                 len(fic.chapters))+'\n')
412             f.close()
413
414         raise Exception('ERROR: a multi-chapter fanfic was found (fic index #'+str(fic.index)+'')')
415
416     return fics
417
418 def count_fics_already_processed():
419     processed_sentences = pandas.read_csv(SENTENCES_TO_CSV)
420
421     r_sentences = processed_sentences[processed_sentences['ficDataset'] == 'ROMANCE']
422     f_sentences = processed_sentences[processed_sentences['ficDataset'] == 'FRIENDSHIP']
423     e_sentences = processed_sentences[processed_sentences['ficDataset'] == 'ENEMY']
424
425     num_r = len(set(r_sentences['ficID']))
426     num_f = len(set(f_sentences['ficID']))
427     num_e = len(set(e_sentences['ficID']))
428
429     return {'r': num_r, 'f': num_f, 'e': num_e}
430
431 ### MAIN ###
432 # Loading canon DB...
433 canon_db = pandas.read_csv(CANON_DB)
434
435 if len(sys.argv) == 4:
436     start_index = int(sys.argv[1])
437     end_index = int(sys.argv[2])
438     dataset = sys.argv[3]
439
440     count_processed = count_fics_already_processed()
441
442     if start_index < count_processed[dataset]: print('Careful: you seem to be re-processing some
443         fanfics')

```

```

443     else:
444
445     print('Fetching fic texts from database '+dataset+'...')
446     start = time.time()
447
448     fic_list = get_fanfics(start_index, end_index, dataset)
449
450
451     #fic_texts = [fic.chapters for fic in fic_list] #debug
452     #print(len(fic_texts[0]), type(fic_texts[0][0])) #debug
453
454     end = time.time()
455     print("...fics fetched. Elapsed time: ", (end-start)/60, " mins")
456
457     processed_fics = character_and_sentence_extraction(fic_list)
458
459     all_characters = []
460     all_sentences = []
461     for fic in processed_fics:
462         all_characters = all_characters + fic.characters
463         all_sentences = all_sentences + fic.sentences
464
465     print('Saving data to csv file...')
466     start = time.time()
467
468     # Create dataframe from dicts and save to csv
469     c_df = pandas.DataFrame.from_dict(all_characters)
470     s_df = pandas.DataFrame.from_dict(all_sentences)
471
472     c_df.to_csv(CHARACTERS_TO_CSV, mode='a', index=False, header=False)
473     s_df.to_csv(SENTENCES_TO_CSV, mode = 'a', index=False, header=False)
474
475     end = time.time()
476     print("...saved. Elapsed time: ", (end-start)/60, " mins")
477
478     elif len(sys.argv) == 3:
479         start_index = int(sys.argv[1])
480         end_index = int(sys.argv[2])
481
482         count_processed = count_fics_already_processed()
483
484         if start_index < count_processed['r']: print('Careful: you seem to be re-processing some
485             fanfics')
486     else:
487
488     print('Fetching fic texts...')
489     start = time.time()
490
491     fic_list = get_fanfics(start_index, end_index, 'r')
492

```

```

493 #fic_texts = [fic.chapters for fic in fic_list] #debug
494 #print(len(fic_texts[0]), type(fic_texts[0][0])) #debug
495
496 end = time.time()
497 print "...fics fetched. Elapsed time: ", (end-start)/60, " mins")
498
499 processed_fics = character_and_sentence_extraction(fic_list)
500
501 all_characters = []
502 all_sentences = []
503 for fic in processed_fics:
504     all_characters = all_characters + fic.characters
505     all_sentences = all_sentences + fic.sentences
506
507 print('Saving data to csv file...')
508 start = time.time()
509
510 # Create dataframe from dicts and save to csv
511 c_df = pandas.DataFrame.from_dict(all_characters)
512 s_df = pandas.DataFrame.from_dict(all_sentences)
513
514 c_df.to_csv(CHARACTERS_TO_CSV, mode='a', index=False, header=False)
515 s_df.to_csv(SENTENCES_TO_CSV, mode = 'a', index=False, header=False)
516
517 end = time.time()
518 print "...saved. Elapsed time: ", (end-start)/60, " mins")
519
520 elif len(sys.argv) == 2:
521     if sys.argv[1] == 'c':
522         count = count_fics_already_processed()
523         print('Romance fics processed: ', count['r'], '\nFriendship fics processed: ', count['f'], '\n'
              'Enemy fics processed: ', count['e'])
524     else: print('Incorrect use of command line')
525
526 elif len(sys.argv) == 1:
527     fGetter = FanficGetter()
528     fGetter.set_fic_listing_path(ROMANCE_LISTING_PATH)
529
530 print('Fetching fic texts...')
531 start = time.time()
532
533 fic_list = fGetter.get_fanfics_in_range(0, 5)
534 #fic_texts = [fic.chapters for fic in fic_list]
535 #print(len(fic_texts[0]), type(fic_texts[0][0])) #debug
536
537 end = time.time()
538 print "...fics fetched. Elapsed time: ", (end-start)/60, " mins")
539
540 processed_fics = character_and_sentence_extraction(fic_list)
541
542 all_characters = []

```

```

543 all_sentences = []
544 for fic in processed_fics:
545     all_characters.extend(fic.characters)
546     all_sentences.extend(fic.sentences)
547
548     #print(len(all_characters), len(all_sentences)) #debug
549     #print(type(all_characters), type(all_sentences)) #debug
550
551     #print(all_characters[:5]) #debug
552     #print(all_sentences[:5]) #debug
553
554     # Create dataframe from dicts and save to csv
555     c_df = pandas.DataFrame.from_dict(all_characters)
556     s_df = pandas.DataFrame.from_dict(all_sentences)
557
558     #c_df.to_csv(CHARACTERS_TO_CSV, mode='a', index=False, header=True)
559     #s_df.to_csv(SENTENCES_TO_CSV, mode = 'a', index=False, header=True)
560
561 else:
562     print('Incorrect use of command line')

```


F. ANEXO: PROGRAMA FINAL

```
1  #!/usr/bin/bash/python3
2
3  from nltk.tag import pos_tag
4  from nltk.tokenize import word_tokenize, sent_tokenize
5  from fanfic_util import *
6  from NER_tagger import NERTagger
7  from corenlp_util import CoreNLPPDataProcessor, CoreWrapper
8
9  import sys, time, random, pandas
10
11  ### VARIABLES ###
12  FIC_LISTING_PATH = '/home/maria/Documents/Fanfic_ontology/html_fic_paths.txt'
13  FIC_NAME_PATH = '/home/maria/Documents/Fanfic_ontology/TFG_fics/html/'
14
15  MAX_FIC_CHARACTERS = 50000
16
17  ### FUNCTIONS ###
18
19  def tag_with_NERTagger(fic):
20      nerTagger = NERTagger()
21      tagged_fic = []
22
23      #tokenize fanfic
24      for chapter in fic.chapters:
25          tokens = [word_tokenize(sent) for sent in sent_tokenize(chapter)]
26          tagged_chapter = [pos_tag(word) for word in tokens]
27
28          tagged_fic.extend(tagged_chapter)
29
30      #tag NERs with NERTagger
31      ner_characters = nerTagger.parse(tagged_fic)
32
33      return ner_characters
34
35  def call_corenlp(fic):
36      client = CoreWrapper()
37
38      print('\n## Starting CoreNLP server and processing fanfic. . .\n')
39      start= time.time()
40
41
42      annotated_fics, error = client.parse(fic)
43      if error:
44          print('Error: not all fanfics could be processed by the server')
45      recovered_fics = []
46      for fic in annotated_fics:
47          if fic.annotations is not None: recovered_fics.append(fic)
48
49      print('Program is continuing with ', len(recovered_fics), ' of the original ', len(fics), ' fics')
```

```

50 annotated_fics = recovered_fics
51
52
53
54 end = time.time()
55 print("Client closed. " + str((end-start)/60) + " mins elapsed")
56
57 return annotated_fics
58
59 def calculate_sentiment_percent(fic_sentiment):
60 total = fic_sentiment['Num sentences']
61 fic_sentiment.pop('Num sentences')
62
63 sentiment_percents = []
64 for _, count in fic_sentiment.items():
65 sentiment_percents.append(count*100/total)
66
67 return sentiment_percents
68
69
70 ### MAIN ###
71 getter = FanficGetter()
72 handler = FanficHTMLHandler()
73
74 if len(sys.argv) == 1:
75 print('Fetching fanfic...')
76
77 fic_not_found = True
78 while fic_not_found:
79 fic_id = random.randint(0,20190)
80 fic_path = FIC_NAME_PATH+'gomensfanfic_'+str(fic_id)+'.html'
81 #print(fic_id, fic_path) #debug
82
83 fic = getter.get_fanfics_in_range(fic_id, fic_id+1)
84 fic = fic[0]
85
86 if len(fic.get_string_chapters()) < MAX_FIC_CHARACTERS: fic_not_found = False
87
88 print('Fic #' + str(fic.index) + ' fetched.')
89
90 print('Tagging characters with NERTagger...')
91 start = time.time()
92 nertagger_characters = tag_with_NERTagger(fic)
93 #for char in nertagger_characters: print(char) #debug
94 end = time.time()
95 print('...done in ' + str((end-start)/60) + ' minutes.')
96
97 annotated_fic = call_corenlp(fic)
98 #print(len(annotated_fic)) #debug
99 annotated_fic = annotated_fic[0]
100

```

```

101 print('Processing CoreNLP data...')
102 coreProcessor = CoreNLPPDataProcessor(annotated_fic)
103 coreProcessor.extract_fic_characters()
104 core_characters = coreProcessor.fic.characters
105 #print(type(core_characters)) #debug
106
107 fic_sentiment = coreProcessor.extract_fic_sentiment()
108 #print(fic_sentiment) #debug
109
110 print('...data processed.')
111 fic_title = handler.get_title(fic_path)
112
113 print('-- DATA FOR FANFIC #' + str(fic.index) + ' --\n')
114 print(' Title: ' + fic_title)
115 print(' NERTagger characters:   {:<8} {:<30} {:<10}'.format('Canon ID', 'Name', 'Mentions'))
116 for character in nertagger_characters:
117     try: print('           {:<8} {:<30} {:<10}'.format(character['Canon ID'], character['
           Name'], character['Mentions']))
118     except TypeError: print(character) #debug
119
120 print('\n')
121
122 print(' CoreNLP characters:   {:<8} {:<20} {:<10} {:<10} {:<10} {:<10} {:<20}'.format('Canon
           ID', 'Name', 'MALE', 'FEMALE', 'NEUTRAL', 'UNKNOWN', 'Other names'))
123 for character in core_characters:
124     try: print('           {:<8} {:<20} {:<10} {:<10} {:<10} {:<10} {:<20}'.format(
           character['Canon ID'], character['Name'], character['Male mentions'], character['Female
           mentions'], character['Neutral mentions'], character['Unknown mentions'], character['Other
           names']))
125     except TypeError: print(character) #debug
126
127 print('\n')
128
129 #percentages = calculate_sentiment_percentages(fic_sentiment)
130 print(' Sentiment:           {:<10} {:<15} {:<15} {:<15} {:<15} {:<15} {:<15}'.format('Sentences', '
           Very positive', 'Positive', 'Neutral', 'Negative', 'Very negative', 'Weighted avg'))
131 print('           {:<10} {:<15} {:<15} {:<15} {:<15} {:<15} {:<15}'.format(fic_sentiment['Num
           sentences'], fic_sentiment['Very positive'], fic_sentiment['Positive'], fic_sentiment['
           Neutral'], fic_sentiment['Negative'], fic_sentiment['Very negative'], fic_sentiment['
           Weighted average']))
132 print('\n\n')
133
134
135 elif len(sys.argv) == 2:
136     try:
137         fic_id = int(sys.argv[1])
138         fic_path = FIC_NAME_PATH + 'gomensfanfic_' + sys.argv[1] + '.html'
139         print(fic_id, fic_path) #debug
140
141         if fic_id < 0 or fic_id > 20190: raise ValueError
142

```

```

143 except ValueError:
144     print('Fic index must be a natural number between 0 and 20190')
145     sys.exit()
146
147     print('Fetching fanfic #' + str(fic_id) + '...')
148     fic = getter.get_fanfic_in_path(fic_path)
149
150
151     print('Tagging characters with NERTagger...')
152     start = time.time()
153     nertagger_characters = tag_with_NERTagger(fic)
154     #for char in nertagger_characters: print(char) #debug
155     end = time.time()
156     print('...done in ' + str((end-start)/60) + ' minutes.')
157
158     annotated_fic = call_corenlp(fic)
159     #print(len(annotated_fic)) #debug
160     annotated_fic = annotated_fic[0]
161
162     print('Processing CoreNLP data...')
163     coreProcessor = CoreNLIDataProcessor(annotated_fic)
164     coreProcessor.extract_fic_characters()
165     core_characters = coreProcessor.fic.characters
166     #print(type(core_characters)) #debug
167
168     fic_sentiment = coreProcessor.extract_fic_sentiment()
169     #print(fic_sentiment) #debug
170
171     print('...data processed.')
172     fic_title = handler.get_title(fic_path)
173     fic_character_tags = handler.get_characters(fic_path)
174     character_tags = ''
175     for tag in fic_character_tags: character_tags += tag + ' '
176
177     print('-- DATA FOR FANFIC #' + str(fic_id) + ' --\n')
178     print(' Title: ' + fic_title)
179     print(' Character tags: ' + character_tags)
180     print(' NERTagger characters:   {:<8} {:<30} {:<10}'.format('Canon ID', 'Name', 'Mentions'))
181     for character in nertagger_characters:
182         try: print('               {:<8} {:<30} {:<10}'.format(character['Canon ID'], character['
            Name'], character['Mentions']))
183     except TypeError: print(character) #debug
184
185     print('\n')
186
187     print(' CoreNLP characters:      {:<8} {:<20} {:<10} {:<10} {:<10} {:<10} {:<20}'.format('Canon
        ID', 'Name', 'MALE', 'FEMALE', 'NEUTRAL', 'UNKNOWN', 'Other names'))
188     for character in core_characters:
189         try: print('               {:<8} {:<20} {:<10} {:<10} {:<10} {:<10} {:<20}'.format(
            character['Canon ID'], character['Name'], character['Male mentions'], character['Female
            mentions'], character['Neutral mentions'], character['Unknown mentions'], character['Other

```

```

names']]))
190 except TypeError: print(character) #debug
191
192 print('\n')
193
194 #percentages = calculate_sentiment_percentages(fic_sentiment)
195 print(' Sentiment:      {:<10} {:<15} {:<15} {:<15} {:<15} {:<15} {:<15} {:<15}'.format('Sentences', '
    Very positive', 'Positive', 'Neutral', 'Negative', 'Very negative', 'Weighted avg'))
196 print('      {:<10} {:<15} {:<15} {:<15} {:<15} {:<15} {:<15} {:<15}'.format(fic_sentiment['Num
    sentences'], fic_sentiment['Very positive'], fic_sentiment['Positive'], fic_sentiment['
    Neutral'], fic_sentiment['Negative'], fic_sentiment['Very negative'], fic_sentiment['
    Weighted average']))
197 print('\n\n')
198
199 else: print('Incorrect usage of program.py. Correct use: program.py, program.py <fic_index>')

```

G. REFERENCIAS

Referencias

- Feature extraction: Tf-idf term weighting. https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction. (última visita: 26/11/2020).
- Barthes, R. (1968). La mort de l’auteur. *Manteia*, (5).
- Bird, S. (2012). Natural language processing with python. https://www.nltk.org/book_1ed/ch07.html. (última visita: 16/1/2021).
- Craven, M. and Kumlien, J. (1999). Constructing biological knowledge bases by extracting information from text sources. *SMB-99 Proceedings*.
- Eisenstein, J. (2018). *Natural Language Processing*. MIT Press.
- Ellis, L. (2018). Death of the author. https://www.youtube.com/watch?v=MGN9x4-Y_7A. (última visita: 2/12/2020).
- Finkel, J. R. et al. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*.
- Ivanov, G.-B. (2016). Complete guide to build your own named entity recognizer with python. <https://nlpforhackers.io/named-entity-extraction/>. (última visita: 28/4/2020).
- Jose, R. (2017). Information extraction. <https://github.com/rohitjose/InformationExtraction>. (última visita: 24/9/2020).
- Lafferty, J. et al. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of the 18th International Conference on Machine Learning 2001 (ICML 2001)*, pages 282–289.
- Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10.
- Lin, D. and Wu, X. (2009). Phrase clustering for discriminative learning. *Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP*, pages 151–213.
- Manica, M. and Auer, C. (2019). An information extraction and knowledge graph platform for accelerating biochemical discoveries. *Workshop on Applied Data Science for Healthcare at KDD*.
- Nothman, J. et al. (2013). Learning multilingual named entity recognition from wikipedia. *Artificial Intelligence*, 194:151–175.
- Peng, N. et al. (2017). Cross-sentence n-ary relation extraction with graph lstms. arXiv: 1708.03743 (última visita: 9/2/2021).
- Stuart, A. (2017). Dean winchester and commander shepard walk into a bar: Why fanon matters. *Uncanny Magazine*.
- Swift, J. (1998). Copyright 101: A brief introduction to copyright for fan fiction writers. *Woosh Magazine, Birthplace of the International Association of Xena Studies*.
- Wick, M. et al. (2009). An entity based model for coreference resolution. *University of Massachusetts*.
- Zelenko, D. et al. (2003). Kernel methods for relation extraction. *Journal of Machine Learning Research*, (3):1083–1106.