Memoria

Javier Antonio Román López

15 de abril de 2018

1. Resolición

- Se han realizado tres funciones:

- tripletFd: Recibe la dirección de un fichero y una dirección de la dbpedia de donde sacar las tripletas, escribe en el fichero todas las tripletas, no retorna nada.
- labels: Recibe la direccíon del fichero de tripletas y la propiedad sobre la que buscara, retorna una lista de labels, por ejemplo [[”Blade”, Runner”], [”Scott”, ...], ...].
- rank: Recibe una lista de labels y retorna la lista con las 10 palabras más similares al vector promedio de word2vec de todos los vectores de las palabras de las labels.

Importando el programa se podra realizar el ejercício los pasos serían: Generar un documento con tripletas usando ”tripletFd” respecto a una tripleta concreta, posteriormente obtener todos los labels con la función ”labels” pasandole como argumentos la dirección del fichero de tripletas y la propiedad, esto devolvera una lista de labels que procesada por la función ”rank” devolvera el resultado deseado.
Si se ejecuta el programa directamente este realizara una prueba con el documento ”test2.txt”, encontrado en la carpeta test, y mostrando por pantalla el listado de palabras.

2. Decisiones

- De las dos partes que podemos diferenciar, se opto en el caso de las consultas a la dbpedia por las librerías de ”rdflib” y ”SPARQLWrapper” ya que ambas constan de una buena documentación,un facil manejo de las mismas y facilidad para realizar las consultas a la dbpedia. Para word2vec se busco alguna librería en python que lo tuviese implementado, se encontraron dos ”gensim” y ”tensorflow” ambas con una extensa y buena documentación, en este caso se opto por ”gensim” antes que la otra por el codigo baste más reducido a la hora de implementar el algoritmo y su compatibilidad con numpy. Por ultimo el uso de numpy ha sido para reducir los tiempos de ejecución en algunos calculos, como el vector promedio, ya que para listas con gran cantidad de datos numpy es bastante más eficiente que las listas, por defecto, de python.

El separar el ejercicio en tres funciones se hizo pensando en el uso de las tres para otros ambitos y para facilitar tanto la comprensión como la realización del ejercicio.

3. Conclusión

- Los tiempos de ejecución vienen muy limitados por la lentitud de las consultas a la dbpedia lo que puede significar un gran problema a la hora de necesitar analizar un gran número de tripletas y el modelo puede no ser el más optimo dando lugar a posibles sobre-entrenamientos o resultados no tan acertados o esperados.

4. Mejoras

- Una revisión general del código replanteando como se hacen algunos calculos e intendando, si se puede, reducir su complejidad. O la forma de hacer el modelo de word2vec para mejorar los resultados.

- En el caso de word2vec se podría mirar otras librerias, por ejemplo TensorFlow, que implementen word2vec y si dichas implementaciones pueden ser más eficientes que la usada. También en la documentación de Gensim sobre word2vec se habla de que con el uso de librerias como Cython y/o Scipy se puede aumentar considerablemente el rendimiento.

- La paralelización del programa tambien podría ser una buena forma de optimizar el rendimiento, sobre todo para las consultas. Ya que la concurrencia en Python no es muy eficiente se podria intentar usar procesos en C o Erlang, bastante más eficientes, para dicho cometido.