SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

SISTEMAS BASADOS EN CONOCIMIENTOS

Autor:

David Alejandro Burneo Valencia

David Josue Jiménez Jiménez

Karla Lizbeth Ochoa Ludeña

Docente:

Ing. Janneth Chicaiza

Tema:

PROYB2-E1 – TRANSFORMACIÓN, ENLAZADO Y ALMACENAMIENTO DE DATOS.

Abr/2021 – Ago/2021

Documentación

Definición de URIs.

Para la definición de URIs, tomamos en cuenta los siguientes parámetros:

RECURSO	CRITERIO CONSIDERADO
Artículo	Se tomó en cuenta el identificador único que proporciona Semantic Scholar.
Referencias	En este caso se utilizó el DOI, eliminando todos los caracteres especiales que puedan dar problema.
Nombres	Se utilizó el nombre del autor quitando los espacios en blanco.

Transformación y almacenamiento de datos RDF.

Tabla resumen de datos recolectados:

Por cada clase del modelo ontológico se generaron el siguiente número de instancias

Modelo Ontológico	Número de instancias
foaf:Person	5345
vivo:Article	12182
skos:Concept	17
c4o:GlobalCitationCount	1496
bibo:Journal	433

Pre-procesamiento de datos:

Para la generación del archivo RDF no fue necesario mayor limpieza de datos, dado que los datos consumidos de Semantic Scholar no requieren de esto, a diferencia de otras bases de datos. Sin embargo, se realizó un proceso de limpieza para la generación de URIS, en el cual se procedió a reemplazar los siguientes caracteres:

“ ”	“_”
“Ñ”	“N”
“ñ”	“n”
“/”	“”
“(”	“”
“)”	“”
“%”	“”
“.”	“”

Transformación de datos:

1. **Librería.** Para la generación de rdf se ha implementado la librería para Java denominada Jena
2. **Definición de Clases y Propiedades.** Como primer paso se han definido las clases y propiedades que se van a utilizar según el modelo consolidado en clases. Cada una de ellas se definió con un nombre que sea fácil de identificar a cual clase o propiedad está haciendo referencia

3. **Recorrido y creación de objetos.** Se implementa un método *for* en el cual se realizan varias iteraciones acorde al número de artículos registrados en la base de datos. Por cada iteración se realiza lo siguiente:
   - **3.1. Crear el objeto.** Por cada iteración realizada se hace un llamado a la base de datos para obtener un articulo a la vez y se crea el objeto con todos los datos que se encuentran registrados (Articulo, autores, referencias, campos de estudio, etc.)
   - **3.2. Agregar clases.** Se agregan las clases en función de lo consultado a la base de datos
   - **3.3. Varios datos.** Para las relaciones que son según la base de datos de uno a muchos (como lo pueden ser varios autores para un artículo) se implementa un método for anidado, en el cual se agregan los valores en referencia a un articulo
   - **3.4. Control de existencia.** Antes de agregar los datos se realiza un método condicional *if* en el cual se verifica la existencia del valor a agregar. Si éste existe da paso a la agregación correspondiente. Caso contrario continua a la siguiente propiedad.
4. **Salida.** Luego de haber terminado las iteraciones se procede a generar un nuevo archivo con la extensión "rdf" en el cual se escribirán los datos procesados por Jena.

Almacenamiento:

El repositorio para almacenar los datos es Graph Db, el motivo es que es una base de datos de gráficos, también conocida como base de datos semántica, es una aplicación de software diseñada para almacenar, consultar y modificar gráficos de red. Un gráfico de red es una construcción visual que consiste en nodos y bordes. Cada nodo representa una entidad (como una persona) y cada borde representa una conexión o relación entre dos nodos.

Enlazado post-transformación:

Este código es para crear las tripletas consumiendo la data de la base de datos:

```
def create_abstract(Articles):
    mentions = []
    categories = []
    triples = []
    for x in Articles:
        print(x.get_id())
        if x.get_abstract() != None:
            dbCategories = semantic_annotation.getAnnotationsAbstract(x.get_paperId(), x.get_abstract())
            for a in dbCategories:
                mentions.append(a[0])
                categories.append(a[1])
            mentions = list(set(mentions))
            categories = list(set(categories))
            # print(mentions)
            # print(categories)
            idArticle = x.get_paperId()

            triples.append(create_triples(idArticle, mentions, categories))

    result = []
    for item in triples:
        for item2 in item:
            if item2 not in result:
                result.append(item2)

    with open('tripletas.rdf', 'w') as temp_file:
        for item in result:
                temp_file.write("%s\n" % item)
```

```
def create_triples(idArticle, mentions, categories):
    triples_mention = []
    mention_type = []
    triples_subject = []
    triples = []

    for a in mentions:
        triples_mention = idArticle + " schema:mentions " + a + " ."
        mention_type = "<" + a + ">" + " rdfs:label " + "'" + getLabel(a) + "' ."
        triples.append(triples_mention)
        triples.append(mention_type)

    for b in categories:
        triples_subject = idArticle + " dct:subject " + b + " ."
        triples.append(triples_subject)

    print(triples)
    return (triples)
```

El siguiente código es para sacar las URLs, de tagme, en la línea 28 es lo que verifica si la URL existe o no. y si existe la retorna para agregar la URI.

```
import requests

token = '76df1f8b-d548-4db5-8aba-af5e8f468e94-843339462' # colocar aqui el token
url_endpoint = 'https://tagme.d4science.org/tagme/tag?lang=en&include_abstract=true&include_categories=true&gcube-token='
headers = {'user-agent': 'Mozilla/5.0', 'accept': 'application/json', 'content-type': 'application/json'}
dbr = 'http://dbpedia.org/resource/'
dbc = 'http://dbpedia.org/resource/Category:'

def getAnnotationsAbstract (paper_id, text):
    url = url_endpoint + token + '&text=' + text

    resp = requests.get(url, headers=headers).json()
    resp.keys()
    annotations = resp['annotations']

    dbCategories = []

    for i in range(len(annotations)):
        ann = annotations[i]['spot']
        if annotations[i]['rho'] > 0.3 and annotations[i]['link_probability'] > 0.3:
            ann = ann.capitalize()
            ann = dbr + ann.replace(' ', '_')
            # Guardar como recursos de la Dbpedia:
            for c in annotations[i]['dbpedia_categories']:
                # dbCategories.append([paper_id, ann, annotations[i]['rho'],
                #                      annotations[i]['link_probability'], dbc + c.replace(' ', '_')])
                status = requests.get(ann)
                if status.status_code == 200:
                    dbCategories.append([ann, dbc + c.replace(' ', '_')])
                break
```

El enlace del GITGUB estará disponible en:

https://github.com/daburneo1/project_sbc