UNIVERSITÀ DEGLI STUDI DI TORINO

Dipartimento di Informatica

Corso di Laurea in Informatica

Sistemi Cognitivi

Metodo di Rocchio, Metadati, Ontologia con Protègè

Docente:
Prof. Radicioni

Studente:
Giulia Monticone

Dicembre 2018, 18

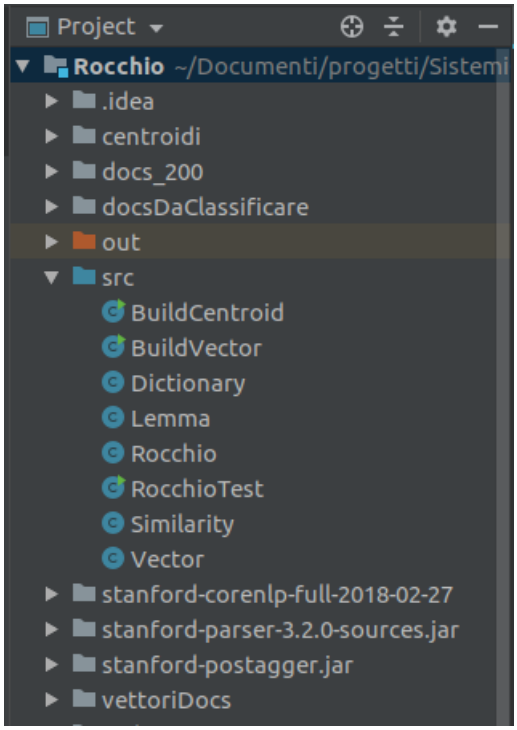
Indice

Capitolo 1

Metodo di Rocchio

1.1 Introduzione

Il codice di questa esercitazione è composto da tre main:



- **BuildVector**: qui vengono invocate le classi con i metodi che creano i vettori di ogni documento contenuto nella cartella `docs_200`, i vettori calcolati vengono salvati nella cartella `vettoriDocs`;
- **BuildCentroid**: qui vengono invocati metodi che creano i centroidi, per 10 classi, in base ai training set passati e vengono salvati all'interno della cartella `centroidi`;
- **RocchioTest**: qui vengono passati i documenti che si vogliono classificare, contenuti nella cartella `docsDaClassificare`.

1.2 Feature vector model

Partendo dal main `BuildVector`, vengono letti i documenti contenuti nella cartella `docs_200` e per ognuno di essi avvengono i seguenti passaggi:

- tokenizzazione: vengono individuati i token di ogni frase del documento in esame;
- eliminazione delle stopwords: vengono eliminati pronomi, articoli, preposizioni ed altro, prese dall'elenco di stopwords di riferimento `stopwords-it.txt`;
- lemmatizzazione: vengono calcolati i lemmi di ogni singola parola rimasta tramite l'utilizzo del documento `morph-it_048.txt`.

Questi passi vengono sviluppati nella classe **Lemma.java**.
Una volta finita la "pulizia" del singolo documento, vengono aggiunte nel dizionario le parole incontrate, come coppia $< w_i,n >$: parola i-esima con a fianco il numero di documenti in cui compare; questo passaggio viene fatto nella classe **Dictionary.java** e l'output viene salvato nel file `dictionary.txt`.
Dopo aver parsificato tutti i documenti e costruito il dizionario, vengono costruiti i **feature vector** per ogni documento:

$$\vec{d_j} = (w_{1,j}, w_{2,j}, ..., w_{m,j})$$

dove $w_{i,j}$ è il peso della parola i-esima del documento j-esimo calcolato tramite la seguente formula:

$$w_{i,j} = tf_{i,j} * idf_i$$

dove: $tf_{i,j}$ è la frequenza di $w_{i,j}$ nel documento j-esimo e idf_i è la `inverse document frequency` della parola i-esima calcolata tramite la formula $log(N/n_i)$ (N numero totale di documenti, n_i numero di documenti in cui la parola i-esima compare). Questi calcoli vengono eseguiti nella classe **Vector.java**.

1.3 Applicazione del metodo di Rocchio

I documenti che sono stati manipolati fin'ora sono stati divisi nelle seguenti classi: `ambiente, cinema, cucina, economia_finanza, motori, politica, salute, scie_tecnologia, spettacoli e sport`; per ognuna di esse è stato calcolato il **centroide**.
Per prima cosa, nel main `BuildCentroid` sono stati definiti i seguenti insiemi

$$POS_i = \{d_j \in T_r | \Phi(d_j, c_i) = True\}$$
$$NEG_i = \{d_j \in T_r | \Phi(d_j, c_i) = False\}$$

dove: T_r è il training set di documenti, $\Phi(d_j, c_i)$ è la funzione che restituisce True se il documento j-esimo appartiene alla classe i-esima, False altrimenti; inoltre $POS_i \cup NEG_i = T_r$.

Si può ora passare al calcolo del centroide C_i per la classe c_i come

$$\vec{C_i} = \{f_{1,i}, f_{2,i}, ..., f_{m,i}\}$$

dove $f_{k,i}$ è la k-esima feature calcolata nel seguente modo

$$f_{k,i} = \beta * \sum_{d_j \in POS_i} \frac{w_{k,j}}{|POS_i|} - \delta * \sum_{d_j \in NEG_i} \frac{w_{k,j}}{|NEG_i|}$$

Il tutto viene svolto nella classe **Rocchio.java**.

1.4 Classificazione di nuovi documenti

Per classificare dei documenti, dal main `RocchioTest` si procede nel seguente modo:

- si procede con il calcolo dei lemmi e la costruzione del feature vector;
- si leggono i centroidi calcolati precedentemente e per ognuno di essi si calcola la *similarità* tra il feature vector del documento e il centroide c_i;

La similarità viene calcolata utilizzando il `cosine similarity`:

$$sim_{cos}(d_j, c_i) = \frac{\sum_{k=1}^{N} w_{k,d} * w_{k,c}}{\sqrt{\sum_{k=1}^{N} w_{k,d}^2} * \sqrt{\sum_{k=1}^{N} w_{k,c}^2}}$$

Il coseno fra due documenti identici sarà 1, 0 se risultano essere ortogonali.

I documenti che si vogliono classificare si trovano nella cartella `docsDaClassificare` mentre l'output della classificazione viene salvato nel file *classification.txt*.

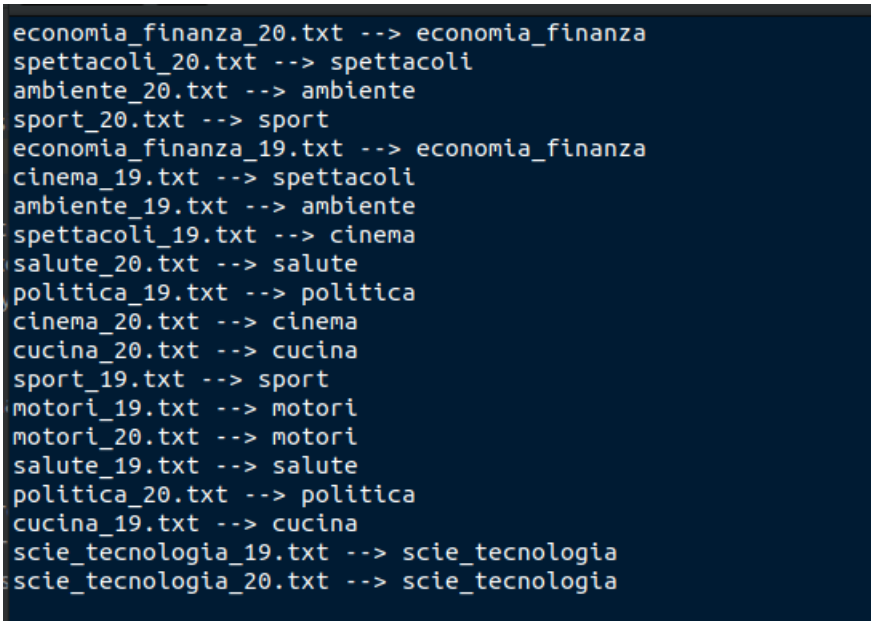


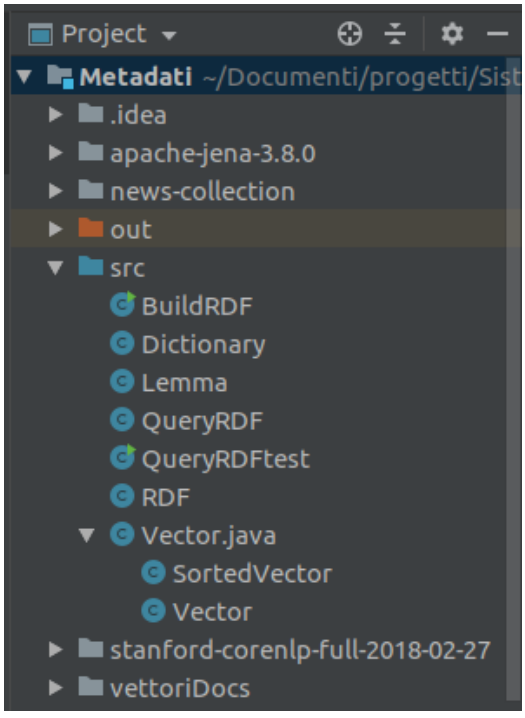
Figura 1.1: Output della classificazione

Capitolo 2

Creazione di metadati

2.1 Introduzione

Il codice di questa esercitazione è composto da due main:



- **BuildRDF**: qui vengono invocate le classi con i metodi che leggono gli articoli contenuti nella cartella `news-collection`, vengono costruiti i vettori dei termini contenuti in essi e salvati nella cartella `vettoriDocs`, infine viene contruito il modello RDF di questi articoli;
- **QueryRDFtest**: qui viene invocato il metodo nella classe QueryRDF.java che interroga il modello RDF costruito in precedenza.

2.2 Annotazione dei dati in formato RDF

Partendo dal main `BuildRDF`, vengono letti i documenti contenuti nella cartella `news-collection` e per ognuno di essi vengono eseguiti i seguenti passaggi:

- vengono estratte le informazioni dagli articoli, quali: title, subject, description, date, creator e publisher;
- per costruire il "subject" dell'articolo vengono presi i primi 3 termini del *feature vector* rappresentante l'articolo; esso viene costruito tramite:
  - tokenizzazione;
  - eliminazione delle stopwords;
  - lemmatizzazione tramite *Morphology*.

  Il tutto viene sviluppato nella classe **Lemma.java** per la "pulizia" del testo e in **Vector.java** per costruire il feature vector.
  Per prendere i 3 termini più frequenti è stato ordinato il vettore (tramite il SortedVector);
- le informazioni così estratte vengono passate alla classe **RDF.java** per creare il modello RDF utilizzando i metatag Dublin Core.

2.2.1 Creazione del modello RDF

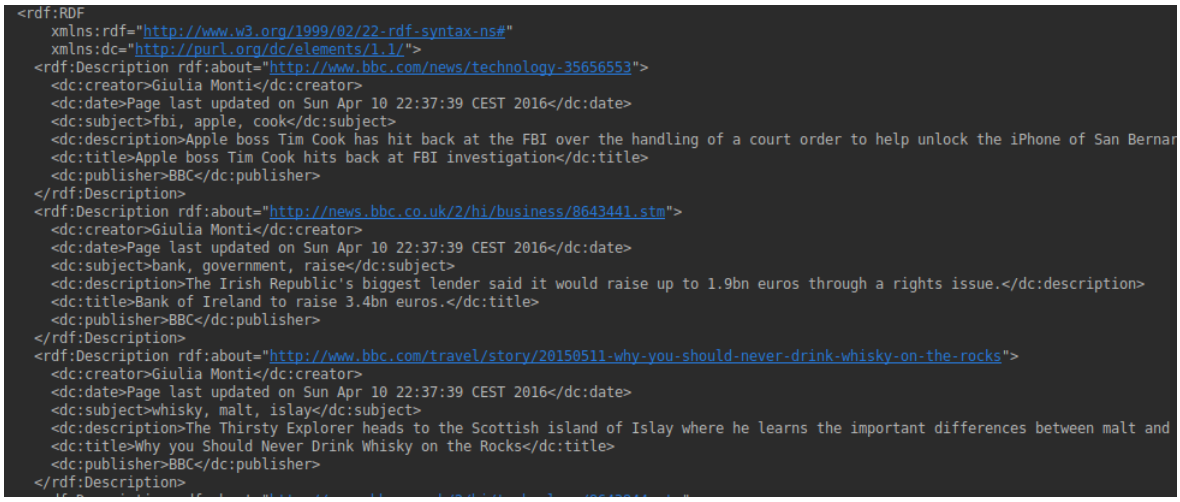
È stata utilizzata la libreria di JENA per modellare RDF ed eseguire in seguito le interrogazioni tramite SPARQL.
La seguente figura mostra la creazione del modello di uno degli articoli:

```
model.createResource(infoRDF.get(i).get(0))
     .addProperty(DC.publisher,infoRDF.get(i).get(6))
     .addProperty(DC.title,infoRDF.get(i).get(1))
     .addProperty(DC.description,infoRDF.get(i).get(2))
     .addProperty(DC.subject,infoRDF.get(i).get(3))
     .addProperty(DC.date,infoRDF.get(i).get(4))
     .addProperty(DC.creator,infoRDF.get(i).get(5));
```

Figura 2.1: Esempio di costruzione del modello con le RDF API formite da JENA

Dalla Figura 2.1 si possono vedere i metatag di Dublin Core utilizzati.

Vengono messi in un unico RDF tutti gli articoli che vengono letti e parsificati ottenendo il seguente output:



```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://www.bbc.com/news/technology-35656553">
    <dc:creator>Giulia Monti</dc:creator>
    <dc:date>Page last updated on Sun Apr 10 22:37:39 CEST 2016</dc:date>
    <dc:subject>fbi, apple, cook</dc:subject>
    <dc:description>Apple boss Tim Cook has hit back at the FBI over the handling of a court order to help unlock the iPhone of San Bernar
    <dc:title>Apple boss Tim Cook hits back at FBI investigation</dc:title>
    <dc:publisher>BBC</dc:publisher>
  </rdf:Description>
  <rdf:Description rdf:about="http://news.bbc.co.uk/2/hi/business/8643441.stm">
    <dc:creator>Giulia Monti</dc:creator>
    <dc:date>Page last updated on Sun Apr 10 22:37:39 CEST 2016</dc:date>
    <dc:subject>bank, government, raise</dc:subject>
    <dc:description>The Irish Republic's biggest lender said it would raise up to 1.9bn euros through a rights issue.</dc:description>
    <dc:title>Bank of Ireland to raise 3.4bn euros.</dc:title>
    <dc:publisher>BBC</dc:publisher>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.bbc.com/travel/story/20150511-why-you-should-never-drink-whisky-on-the-rocks">
    <dc:creator>Giulia Monti</dc:creator>
    <dc:date>Page last updated on Sun Apr 10 22:37:39 CEST 2016</dc:date>
    <dc:subject>whisky, malt, islay</dc:subject>
    <dc:description>The Thirsty Explorer heads to the Scottish island of Islay where he learns the important differences between malt and
    <dc:title>Why you Should Never Drink Whisky on the Rocks</dc:title>
    <dc:publisher>BBC</dc:publisher>
  </rdf:Description>
```


Figura 2.2: Parte di output del modello RDF creato

L'intero RDF viene salvato nel file di nome *newsCollections.rdf*.

2.3 Interrogazione di triple store

Il file generato precedentemente costituisce un `triple store`.
All'interno del main `QueryRDFtest` viene invocata la classe **QueryRDF.java** nella quale vengono create le query ed eseguite, utilizzando SPARQL tramite JENA.

```
//create new query
String queryStr= "PREFIX dc: <http://purl.org/dc/elements/1.1/>" +
        "SELECT ?title " +
        "WHERE {?predicate dc:title ?title . ?predicate dc:creator \"Anakin Skywalker\".}";
Query query = QueryFactory.create(queryStr);
//execute query and results
QueryExecution exec = QueryExecutionFactory.create(query,this.model);
ResultSet results = exec.execSelect();
//output
System.out.println("\nTitoli dei documenti che hanno come autore 'Anakin Skywalker'");
ResultSetFormatter.out(System.out, results, query);
exec.close();
```

Figura 2.3: Esempio di creazione ed esecuzione di una query

Di seguito l'output della query eseguita:

```
Titoli dei documenti che hanno come autore 'Anakin Skywalker'
-----------------------------------------------------
| title                                             |
=====================================================
| "Disney announces Monsters Inc sequel"            |
| "Why you Should Never Drink Whisky on the Rocks" |
-----------------------------------------------------
```

Figura 2.4: Output della query in Figura 2.3

Capitolo 3

Sviluppo di un'ontologia con Protégé

3.1 Introduzione

È stata creata un'ontologia sul dominio del film:

- un singolo film è composto da: titolo, anno di uscita e numero di oscar vinti; questi vengono raggruppati sotto la classe **Film**;
- i film vengono divisi in `SingleFilm` (ad esempio "The imitation game") e `FilmSeries` se fanno parte di una serie (ad esempio "Il ritorno dei Jedi" fa parte della serie di "Star Wars");
- i film vengono classificati anche in base ai numeri di oscar vinti, il titolo e l'anno di uscita;
- i film appartegono a delle categorie;
- le **Categorie** sono di tipo diverso: azione, avventura, commedia, drammatico, fantasy, fantascientifico, storico, thriller;
- agli **Attori** sono assegnati i film in cui hanno recitato;
- i film sono stati girati in vari **Stati**, per esempio: Tunisia, Argentina, Italia, ecc...;
- gli stati appartengono a dei **Continenti** (America, Asia, Europa, Africa, Oceania)

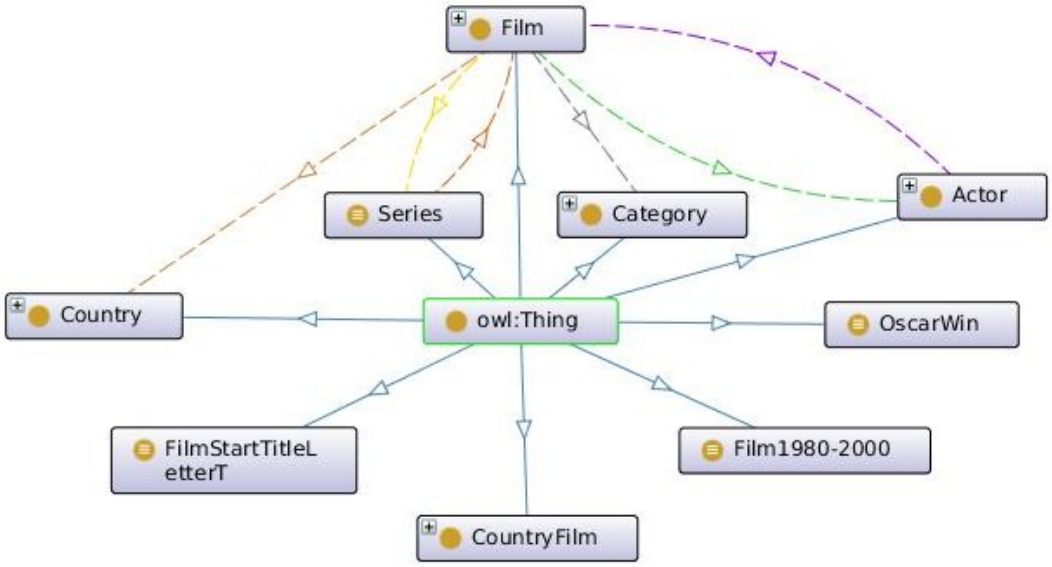


Figura 3.1: Classi dell'ontologia dei Film

3.2 Classificazione dell'ontologia

I film vengono classificati in base:

- alle loro *data property*: titolo, anno di uscita e oscar vinti;

Vengono inoltre classificati i film in base alle loro *object property*:

- in quale/i Stato/i è stato girato;
- quale/i attori hanno fatto parte del cast;
- a quale/i categorie appartiene.

Di seguito vengono riportato alcuni grafi creati da Protègè tramite il plugin OntoGraph:

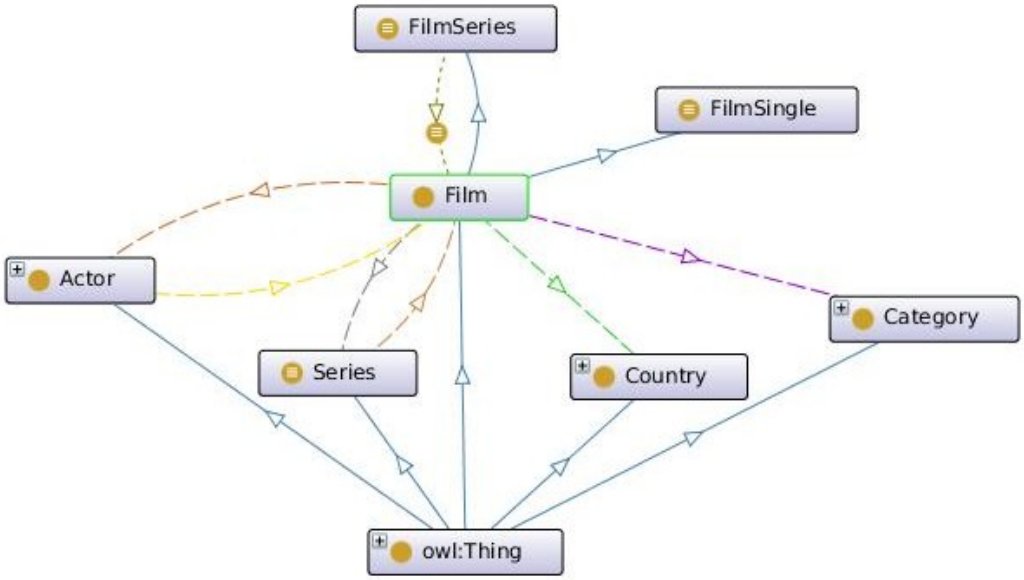


Figura 3.2: Classe Film

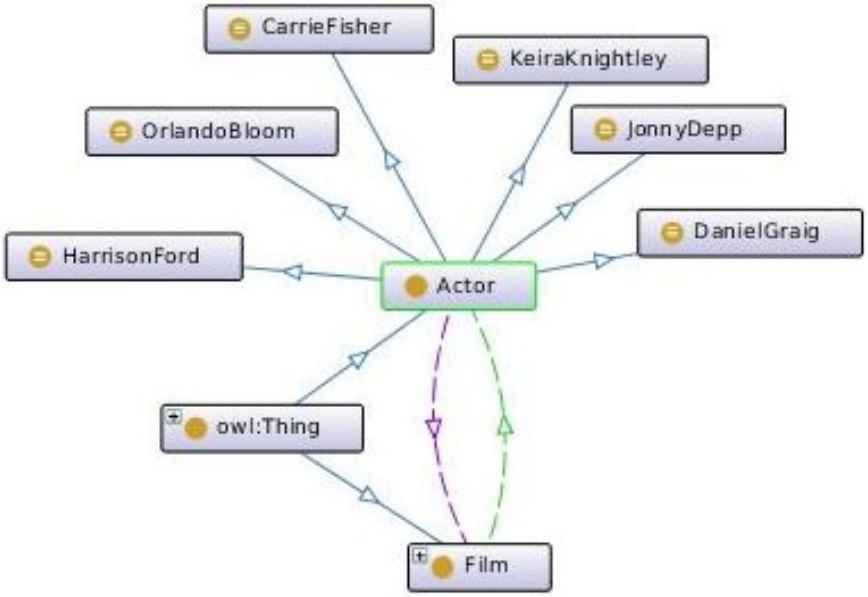


Figura 3.3: Classe Actor

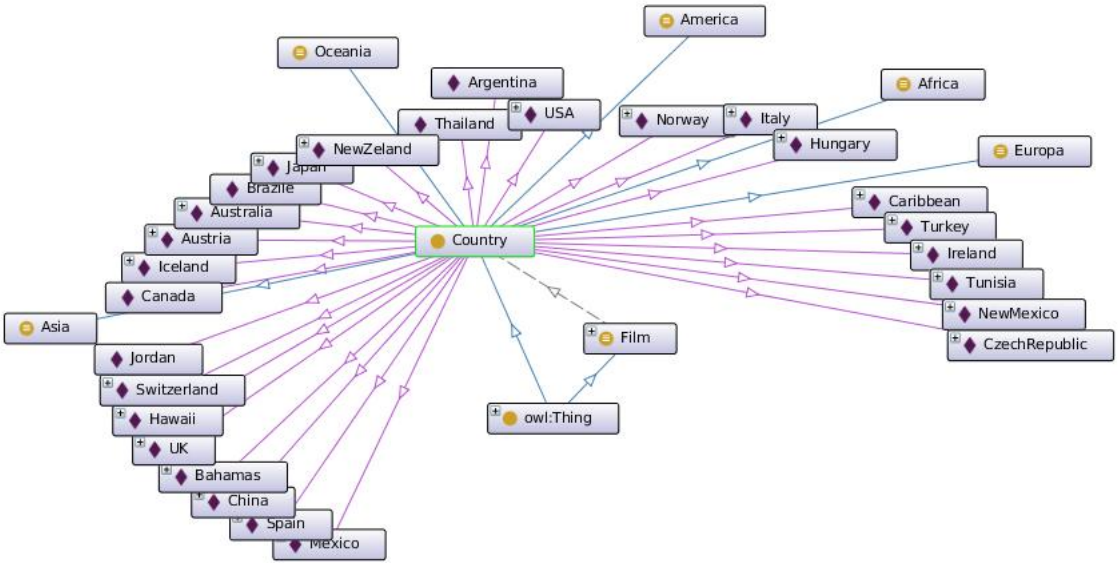


Figura 3.4: Classe Country

3.2.1 Classificazione con HermiT

Il reasoner *HermiT* può determinare se l'ontologia è inconsistente o meno e supporta diversi servizi di ragionamento specializzati come la classificazione di classi e proprietà.
Di seguito vengono riportate alcune delle classificazioni fatte con HermiT sull'ontologia in esame:

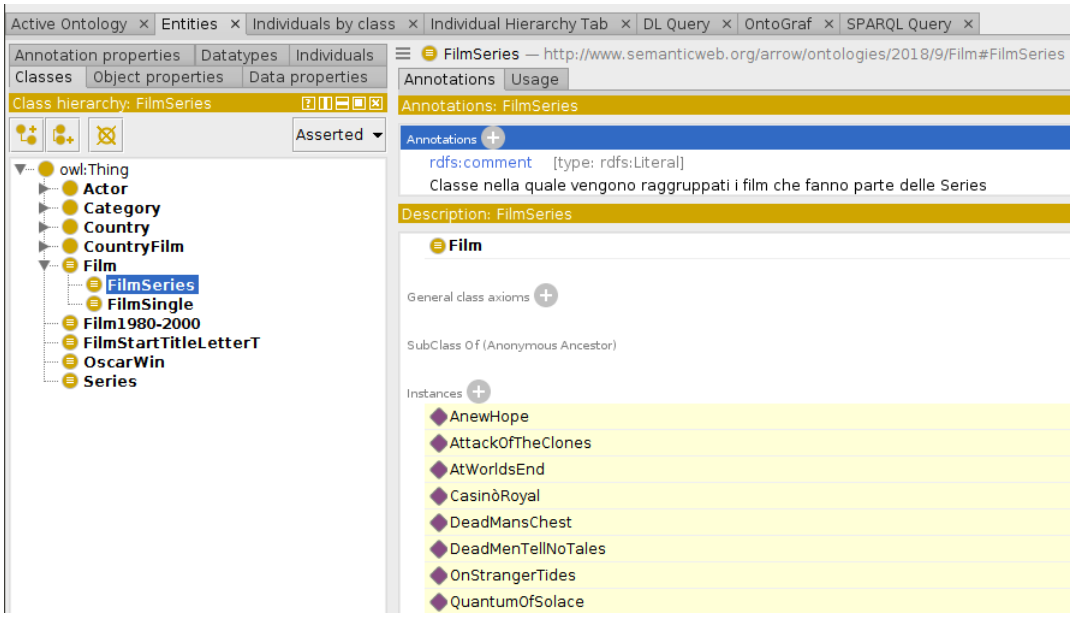


Figura 3.5: Classificazione dei film delle serie

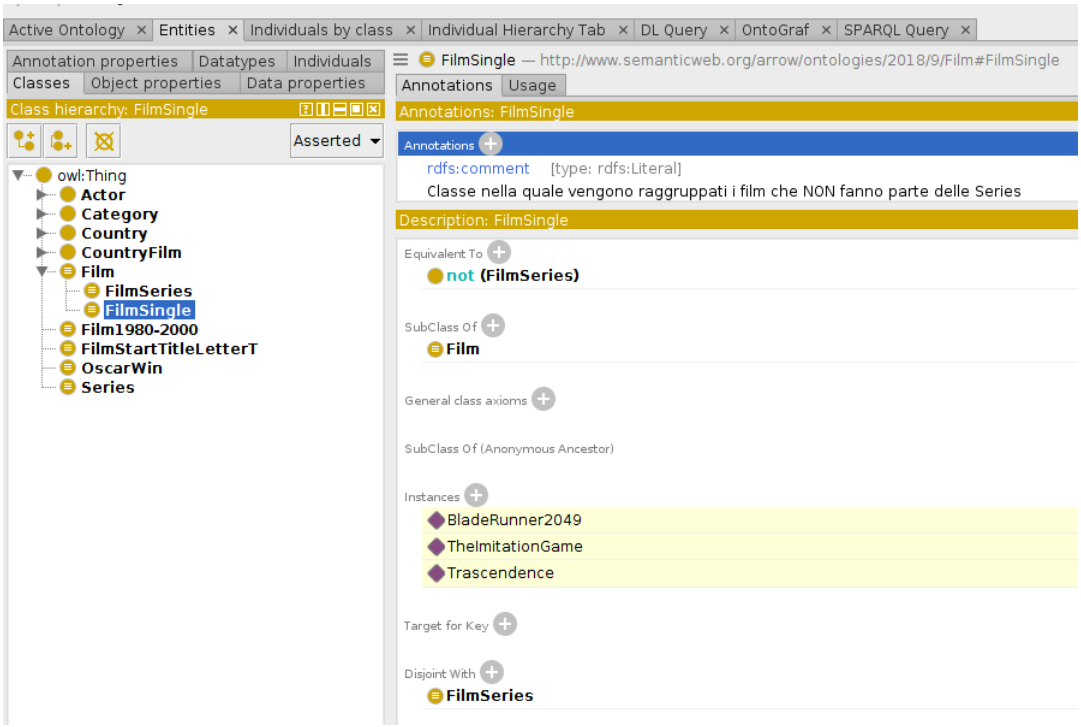


Figura 3.6: Classificazione dei film che non fanno parte di serie

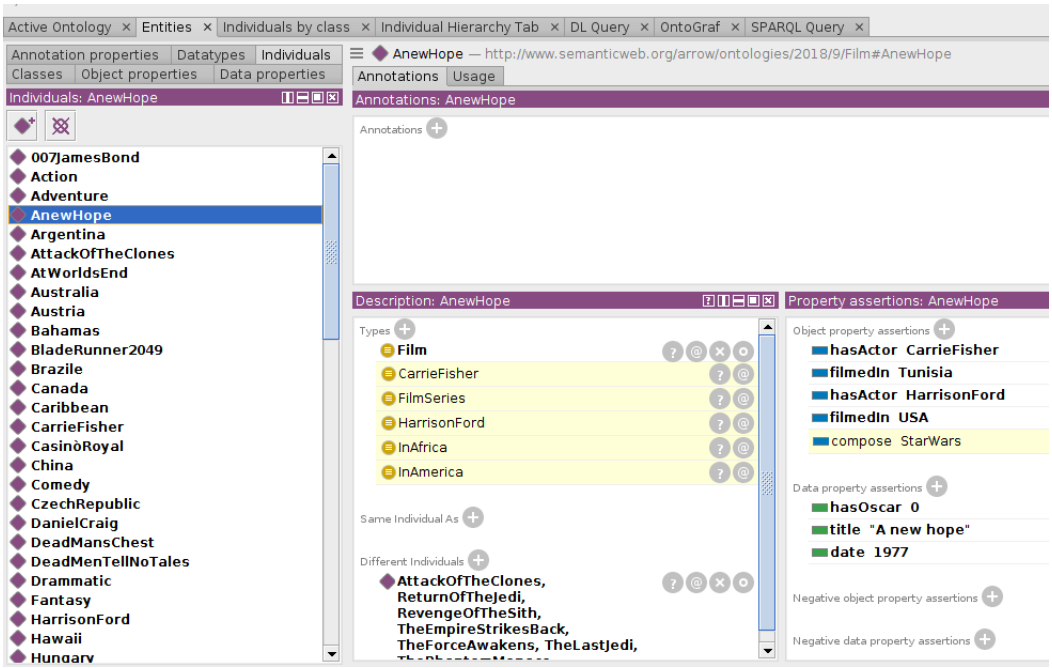


Figura 3.7: Classificazione dell'individuo "A new hope"