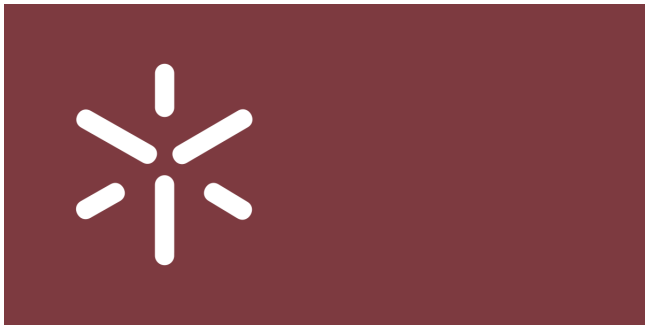

Universidade do Minho

Mestrado em Engenharia Informática

Scripting no Processamento de Linguagem Natural

OwlReady2

Autores:

Frederico Pinto	A73639
Rui Vieira	A74658

22 de Abril de 2019

Conteúdo

1 Introdução

Com o objetivo de conhecermos outras ferramentas úteis para o uso na nossa vida profissional, foi nos proposto pelos docentes da unidade curricular de *Scripting no Processamento de Linguagem Natural* este trabalho prático, que se baseia na ferramenta *OwlReady2*.

Neste documento, iremos apresentar o *OwlReady2* e quais as suas funcionalidades, no final iremos exibir uma implementação da ferramenta no contexto de *NLP* por nós criada.

2 OwlReady2

2.1 Objetivo

O *OwlReady2* é um *package* para o tratamento de ontologias em *Python*. É uma ferramenta bastante poderosa que permite carregar ou criar ontologias *OWL 2.0* como objetos *Python*, modifica-los, guarda-los e até mesmo correr um *Reasoner* que permite inferir informação sobre a ontologia.

Para além disso, este *package* apresenta um acesso à ontologia de uma maneira muito clara e com bastantes benefícios quer a nível de *performance* como também relativamente ao consumo de memória, em comparação com outros *packages* desenvolvidos para outras linguagens de programação.

2.2 Funcionalidades

2.2.1 Gestão da Ontologia

```
from owlready2 import *

# Criação de uma ontologia a partir um novo IRI:
ontology = get_ontology("http://test.org/onto.owl")

# Carregar uma ontologia a partir de uma já existente
# O caminho pode ser um URL ou uma diretoria
# Formatos aceites: NTriples, RDF ou OWL
ontology = get_ontology("/path/owl").load()

# Guardar a ontologia num ficheiro
# Exporta no formato NTriples ou RDF
ontology.save(file = "ontology.rdf", format = "rdfxml")

# Questionar a ontologia sobre todos os individuos que estão
    relacionados pela relação "has_topping"
onto.search(has_topping = "*")
```

Listing 1: Exemplos de gestão da ontologia.

Para além disso, o *OwlReady2* permite interrogar a ontologia sobre outros parâmetros, como as relações e classes existentes, ou até mesmo efetuar uma procura por *IRI*. Contudo, este tipo de procura não se compara a *queries Sparql* pois está cingida a algumas questões e apenas mostra dados explícitos. Apesar disso, o *OwlReady2* apresenta a possibilidade de efetuar *queries Sparql*, mas para isso utiliza o *RDFlib*.

2.2.2 Gestão de Classes

Para criar uma classe no *OwlReady2* temos que criar uma classe em *Python*. Essa nova classe na ontologia pode ser subclasse de uma classe já existente ou então é subclasse da classe *owlready2.Thing* que é a classe pai de toda a ontologia.

```
# Carregar ontologia
ontology = get_ontology("http://test.org/onto.owl")

with ontology:
    # Criação da classe Drug que é subclasse de Thing
    class Drug(Thing):
        pass
    # Criação da classe DrugAssociation que é subclasse de Drug
    class DrugAssociation(Drug):
        pass

    print(DrugAssociation.is_a)
    # Result = [ontology.Drug]

    print(Drug.subclasses())
    # Result = [ontology.DrugAssociation]
```

Listing 2: Exemplos de gestão de classes na ontologia.

2.2.3 Gestão de Indivíduos

A criação de indivíduos utilizando o *OwlReady2* tem por base a criação de objetos de certas classes previamente definidas como mostrado na secção anterior.

Tomemos como classes existentes na nossa ontologia, as criadas no secção anterior, *Drug* e *DrugAssociation*.

```
my_drug = Drug("my_drug")

print(ontology.my_drug)
#Result: ontology.my_drug

print(my_drug.name)
# Result: my_drug

print(my_drug.iri)
# Result: http://test.org/onto.owl#my_drug

# Ver os individuos que pertencem a uma classe
print(Drug.instances())
# Result: [ontology.my_drug]
```

```
#Destruir uma entidade, sendo um individuo, classe ou
    propriedade
destroy_entity(whatever)
```

Listing 3: Gestão de individuos na ontologia.

É possível também criar indivíduos sem fornecer nome, neste caso o *Owlready2* atribui-lhe um. Sendo sempre diferente, pois este nome é o identificador do indivíduo.

2.2.4 Gestão de Propriedades

Após mostrarmos a gestão de classes e a gestão de indivíduos, vamos agora apresentar a gestão de propriedades que vão relacionar classes com classes *(Propriedades de Objectos)*, ou classes com atributos *(Propriedades de Dados)*.

2.2.4.1 Propriedades de Objetos

```
from owlready2 import *

# Carregar a ontologia
ontology = get_ontology("http://test.org/onto.owl")

with ontology:
    # Classe Drug
    class Drug(Thing):
        pass
    # Classe Ingredient
    class Ingredient(Thing):
        pass
    # Definição da propriedade
    class is_ingredient_of(ObjectProperty):
        # Tem como dominio um objecto da classe Ingredient
        domain     = [Ingredient]
        # Tem como contradominio um objeto da classe Drug
        range      = [Drug]
    class has_for_ingredient(ObjectProperty):
        # Tem como dominio um objecto da classe Drug
        domain     = [Drug]
        # Tem como contradominio um objeto da classe Ingredient
        range      = [Ingredient]
        # Definir a relação inversa
        inverse_property = is_ingredient_of
```

Listing 4: Exemplo de criação de uma propriedade de objetos.

Após a criação da classe que representa a relação podemos agora representar os indivíduos de certas classes e as suas relações.

```
# Criação de um individuo da classe Drug
my_drug = Drug("my_drug")

# Criação de um individuo da classe Ingredient
codeine = Ingredient("codeine")

# Criação de um individuo da classe Ingredient
acetaminophen = Ingredient("acetaminophen")

# Unir os individuos pela propriedade criada
my_drug.has_for_ingredient.append(acetaminophen)
my_drug.has_for_ingredient.append(codeine)

print(my_drug.has_for_ingredient)
# Result: [onto.acetaminophen, onto.codeine]
```

Listing 5: Criação de indivíduos e as suas propriedades de objetos.

2.2.4.2 Propriedades de Dados

Iremos mostrar agora como definir uma propriedade de dados no *OwlReady2*, é de salientar que apenas existem estes tipos de dados:

- Int
- Float
- Bool
- Str (string)
- owlready2.normstr (normalized string, a single-line string)
- owlready2.locstr (localized string, a string with a language associated)
- datetime.date
- datetime.time
- datetime.datetime

Nesta demonstração, assumimos a continuação do código mostrado na subsubsecção anterior.

```
with ontology:
    # Adicionar uma propriedade de dados
    class has_for_cost(DataProperty):
        # A que classe se aplica
        domain    = [Drug]
        # Qual é o tipo de dados
        range     = [float]

# Adicionar ao individuo da classe Drug a propriedade
my_drug.has_for_cost = 2.1

print(my_drug.has_for_cost)
# Result: 2.1
```

Listing 6: Exemplo de criação de uma propriedade de dados.

Para além do que falamos, o *OwlReady2* apresenta muitas outras funcionalidades sobre as propriedades tais como definir sub-propriedades e relações indiretas.

2.2.5 Restrições de Propriedades

Outra funcionalidade que o *OwlReady2* fornece é a capacidade de adicionar restrições a propriedades existentes. Exemplos dessa restrições são mostradas abaixo.

```
from owlready2 import *

ontology = get_ontology("http://test.org/onto.owl")

# Definição da ontologia
with onto:
    class Drug(Thing):
        pass
    class ActivePrinciple(Thing):
        pass
    # Outra maneira de definir de uma propriedade de objetos
    class has_for_active_principle(Drug >> ActivePrinciple):
        pass

    # Restrição
    class NonPlaceboDrug(Drug):
        equivalent_to = [Drug & has_for_active_principle.some(
    ActivePrinciple)]

    # Restrição de Cardinalidade
    class DrugAssociation(Drug):
```

```
21         equivalent_to = [Drug & has_for_active_principle.min(2,
   ActivePrinciple)]
```

Listing 7: Exemplo de criação restrições sobre propriedades.

3 Implementação

Chegamos agora à fase de apresentação da aplicação por nós criada que está inserida no contexto *NLP* e utiliza o *OwlReady2*.

Para mostrar as capacidade das ferramentas que utilizamos decidimos fazer um *script* que recebe um ficheiro de texto, contendo esse texto informações sobre uma ou mais empresas. Dessas informações queremos retirar os produtos que as empresas vendem e os respetivos preços bem como a localização das suas lojas.

Após traçados os objetivos, começamos por definir a estrutura da nossa ontologia.

```
with ontology:
    class Company(Thing):
        pass
    class Product(Thing):
        pass
    class Location(Thing):
        pass
    class is_product_of(ObjectProperty):
        domain = [Product]
        range  = [Company]

    class is_location_of(ObjectProperty):
        domain = [Location]
        range  = [Company]

    class has_product(ObjectProperty):
        domain = [Company]
        range  = [Product]
        inverse_property = is_product_of

    class has_location(ObjectProperty):
        domain = [Company]
        range  = [Location]
        inverse_property = is_location_of

    class has_cost(DataProperty, FunctionalProperty):
        domain = [Product]
        range  = [float]
```

Listing 8: Estrutura da Ontologia.

Após essa definição, avançamos para a parte de processamento e análise do texto.

Para isso, utilizou-se o *NLTK* (*Natural Language Tool Kit*), isso permitiu-nos descobrir o contexto do texto usando certas ferramentas disponibilizadas.

Numa primeira fase, dividiu-se o texto por frases. Iniciamos posteriormente uma análise por frase em que se dividiram as mesmas por palavras e descobriu-se as *tags* de discurso. Após a obtenção dessas *tags* aplicamos duas técnicas para descobrir o que necessitávamos. Construiu-se *chunks* para conseguirmos descobrir os produtos e os preços, mas para descobrir as empresas e as localizações utilizamos outra funcionalidade do *NLTK* que permite reconhecer entidades, procurando neste caso por *ORGANIZATION* para as empresas e por *GPE* para encontrar as localizações.

Após todo esse reconhecimento e tratamento dos dados encontrados, utilizou-se as ferramentas disponibilizadas pelo *OwlReady2* para criar os indivíduos e estabelecer as relações entre eles.

Por fim, importamos o ficheiro do tipo *rdf* gerado pelo *OwlReady2* para o *GraphDB* e apresentamos de seguida a representação em grafo da ontologia criada a partir do texto analisado.

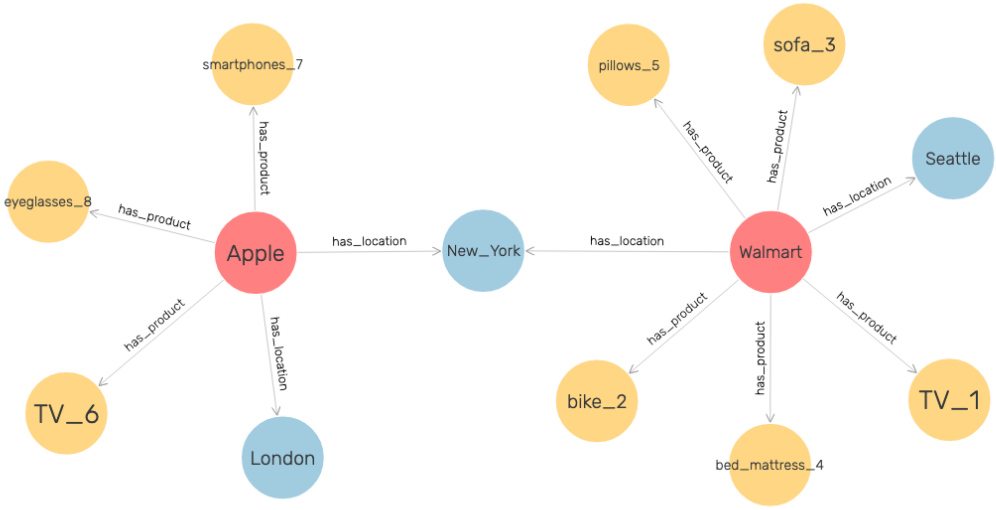

Figura 1: Representação da informação da ontologia no *GraphDB*

4 Conclusão

A concretização deste trabalho prático foi de grande importância para o nosso conhecimento, pois permitiu-nos conhecer o *OwlReady2* e qual as vantagens que este nos pode trazer, como verificamos implementando-o no contexto *NLP*. Para além disso, ao implementar a aplicação utilizando o *NLTK*, melhoramos os nossos conhecimentos sobre essa ferramenta e sobretudo mostrou-nos que é um *tool kit* de uma dimensão enorme e com uma grande capacidade para resolver muitos problemas.

Para finalizar, podemos dizer que para tratar ontologias em *Python*, o *OwlReady2* é um dos melhores *packages* pois é extremamente completo, capaz de efetuar todas as operações sobre ontologias de uma maneira bastante rápida e económica a nível de memória.