Code Documentation

Questions Answering

Relationship Identification and Linking

File: **Entity Recognition in QA.ipynb**

get_label(URI):

The function **"get_Label"** returns the label of the provided URI.

Parameters:

- **URI**: URI of ontology

I makes a SPARQL query to dbpedia as follows,

```
SELECT ?label WHERE { <URI> rdfs:label ?label FILTER (lang(?label) = 'en') }
```

to fetch labels for ontology URIs.

Example:
If the provided URI is http://dbpedia.org/ontology/birthDate, it returns the string **birth date**.

---

get_properties(URI)

The function **"get_properties"** returns the properties of the provided URI.

Parameters:

- **URI**: URI of any resource

I makes a SPARQL query to dbpedia as follows,

```
select distinct ?prop where { <URI> ?prop ?ent }
```

to fetch properties of the resource URI provided.

It filters the out properties of the types *“property”* and *“ontology”* and returns a list in the format of *[property, type, URI, label]*. It retrieves the label by calling the *get_labels* function.

Example:
If the provided URI is http://dbpedia.org/resource/John_F._Kennedy, it returns the list of all properties in this resource.

---

get_closest_word(word, URI, json_unit, with_synonyms=False)

The function **"get_closest_word"** returns the property of the URI which is closest to the provided word.

Parameters:

- **word**: the word or phrase which needs to be matched with the properties
- **URI**: URI of resource, properties of which needs to be matched with the *word*.
- **with_synonyms**: when set to true it looks for word matching by using synonyms from the dictionary. Only works if the number of words in the string passed is 1.
- **json_unit**: It is the section of the json structure that is created with all the details of answer extraction. Details of fetching closest words are populated and added to the json structure in this section.

It first retrieves properties from the URI using *get_properties* function.
If the *with_synonyms* is set to *True,* it fetches for synonyms of the word provided from Oxford Dictionary API and calls a server to fetch the average of all synonyms with the properties. Then retrieves the property with the highest average and returns it.
If the *with_synonyms* is set to *False,* it fetches for values phrase matches between the passed phrase and the property label from the server. Then it retrieves the property with the highest value and returns it.

Example:
If the provided word is **wife** for http://dbpedia.org/resource/John_F._Kennedy, it returns the property **spouse**.

---

fetch_compound(word, model)

The function **"fetch_compound"** returns compound words for words from the given model after dependency parsing.

Parameters:

- **word**: the base word for which compound word needs to be fetched.
- **model**: model retrieved by dependency parsing.

It simply looks up for the word in the model provided and looks up for words associated with it by the tag *'compound'* and returns an array of the words.

Example:
If the provided word is **date**, whereas we are actually looking for **birth date** in the question, it will return so by looking up the model

---

call_sparql(keyword, URI, pType)

The function **"call_sparql"** returns values of properties provided as keywords in the given URI.

Parameters:

- **keyword**: the specific property to be fetched in the resource URI.
- **URI**: URI of a resource.
- **pType:** Since we use only 2 types of properties, *"property"* and *"ontology"* these are the allowed values for *pType*. These types are specified in the SPARQL query.

It makes a SPARQL query as follows,

```
SELECT ?keyword WHERE { <URI> pType:keyword ?keyword }
```

and returns the relevant results retrieved.

Example:
If we are looking for **spouse** of http://dbpedia.org/resource/John_F._Kennedy it will return http://dbpedia.org/resource/Jacqueline_Kennedy_Onassis

---

getResources(text)

The function **"getResources"** returns URI for a marked entity in a sentence.

Parameters:

- **text**: It has a sentence(question) in the format shown in the example below.

It makes a call to an API provided by http://agdistis.aksw.org to retrieve the DBpedia resource link for the annotated words and returns it.

Example:
If the provided text is **Who are the family members of [Peter Griffin]?**, it returns http://dbpedia.org/resource/Peter_Griffin

---

answer(question)

The function **"answer"** returns an answer for a question using the functions above.

Parameters:

- **question**: The question one needs to ask.

It is the main function that utilizes all the functions above to fetch an array results for a provided question. It first applies dependency parsing on the question. Then applies dependency parsing on the questions. From the parsing tree it creates a stack based on which the results must be fetched. Then it pops each element of the stack and fetches its relevant details like an URI or a text using the functions above.
As in the example below. The question provided is **"What was the religion of the wife of JFK?"** From this it creates a stack *[JFK, wife, religion]*. Then we look for the URI corresponding to JFK and fetch it properties. Then we match *wife* with its corresponding properties, we find the URI for the *spouse* of JFK, Jacqueline Kennedy. Then we look for properties of available in the resources URI and match it with the religion of Jacqueline Kennedy and find the URI for Catholic Church and return it.

Example:
If we are looking for an answer for **"What was the religion of the wife of JFK?"**, it should return the URI http://dbpedia.org/resource/Catholic_Church

---

File: **OxfordDictionary.py**

getSynonyms(word_id)

The function **"getSynonyms"** returns set of synonyms for a word.

Parameters:

- **word_id**: word id of the word of whose synonyms are desired.

This function makes an api call to https://od-api.oxforddictionaries.com to retrieve synonyms of a certain word.

Example:
If the provided text is **Who are the family members of [Peter Griffin]?**, it returns http://dbpedia.org/resource/Peter_Griffin

---

Server Code: **app.py**

It uses the model provided by https://github.com/mmihaltz/word2vec-GoogleNews-vectors for word2vec matching.

@app.route("/<path:synonyms>/<path:properties>",methods=["GET"])

simple(synonyms, properties)

The function **"simple"** responds does a w2v matching between a set of url encoded strings of a set of synonyms and a set of properties, joined by '+' and return a list of values of corresponding matches.

Parameters:

- **synonyms**: url encoded synonyms joined by '+'.
- **properties**: url encoded properties joined by '+'.

If one wants to find the similarity between a set of synonyms and a set of words, all synonyms and properties must be concatenated into one string joined by "+", then url encoded and make a request to the server at *url_of_server/url_encoded_concatenated_string_of_synonyms/url_encoded_concatenated_string_of_properties.*

Example:
Let's say our synonyms are **[wife, partner]** and our properties are **[car, house, cat, spouse, socks]**. The synonyms are concatenated to a single string **wife+partner** and properties into **car+house+cat+spouse+socks**. Then they are url encoded into **wife%2Bpartner** and **car%2Bhouse%2Bcat%2Bspouse%2Bsocks,** respectively. A request is made at ***url_of_server*/wife%2Bpartner/car%2Bhouse%2Bcat%2Bspouse%2Bsocks.** It returns a stringified list [['spouse', 0.7435570728358986], ['house', 0.5647266430982613], ['car', 0.6114013465662559], ['socks', 0.6723798365759176], ['cat', 0.0]].

@app.route("word2vecsimilarity/<path:phrases>/<path:labels>",methods=["GET"])

phrase_label_similarity_cd(phrases, labels)

The function **"phrase_label_similarity_cd"** responds does a w2v matching between a set of url encoded strings of a set of phrases and a set of labels, joined by '+' and return a list of values of corresponding matches.

Parameters:

- **phrases**: url encoded phrases joined by '+'.
- **labels**: url encoded labels joined by '+'.

If one wants to find the similarity between a set of phrases and a set of words, all phrases and labels must be concatenated into one string joined by "+", then url encoded and make a request to the server at *url_of_server/url_encoded_concatenated_string_of_phrases/url_encoded_concatenated_string_of_labels.* It returns a list of values based on cosine similarity of the passed phrases and labels.

Example:

Similar to the last example.

---

phrase_similarity(_phrases_1, _phrases_2)

The function **"phrase_similarity"** returns cosine distance of two phrases.

Parameters:

- **_phrases_1**: first phrase
- **_phrases_2**: second phrase
