WebResourceManager

Table of contents

- Introduction
- Documentation
- Usage

Introduction

The Web Crawling Framework is capable of producing a large volume of data, but it is important that the information is managed in a way that it is useful. A common theme for companies with a lot of data is that they cannot make use of it. Even though our data is represented many ways (HTML, text, lists of words, vectors, etc.), it is important that it is always traceable back to the source as well as accessible and thus queryable. It needs to be traceable back to the source so we can justify the conclusions we make with tangible evidence rather than simply pointing to the output of a system. Aggregating this information in a queryable format makes it usable for query formulation and building company profiles. The complexity of the architecture and variety of data types (chiefly HTML, PDF, text, URLs) render most traditional data storage solutions incapable of helping us. The variety of our data, integration needs, and the desire to have random access to source documents mean that we had to make our own data storage solution, but our solution still needs to fulfill the traditional role of a data management solution: a uniform as possible entries, queryability, and efficiency.

Web Resource Manager was designed to meet these needs for a database of web resources. Each data entry is given a UUID (Universally Unique Identifier) and the instance saves a mapping from URL to UUID. Both the text and source for each web resource is saved using this UUID. The source document (HTML or PDF) is saved in "source/.(pdf/html)" while the text information, UUID, generating query, and URL is saved in a JSON file at "docs/.json." Using this uniform data storage system and a simple API. **Web Resource Manager** makes storing and querying the contents and source files of web resources trivial.

Documentation

`__init__(rel_path=None)`

Constructor

Parameters

- rel_path (string) : the relative path to the parent directory of "data" holding the WebResourceManager data

Returns

- None

`__iter__()`

An iterator function with the ability to be accessed by multiple instances at once in a safe way.

Parameters

- instances (int) : the number of instances using the iterator (default = 1)
- iam (int) : the current instance's assignment [0-*instances*) (default = 0)

Returns

- dict : A dictionary which is the profile if found, else None (Yields)

`__getitem__(key)` and get(key)

Loads and returns the web resource profile specified by key

Parameters

- key (string) : universally unique identifier (UUID) for a web resource being tracked by the Web Resource Manager instance

Returns

- dict : A dictionary which is the profile if found, else None

`__len__()`

Returns the number of web resources being tracked by the instance

Returns

- int : number of web resources in the instance

`__repr__()` and `__str__()`

Returns a sorted and indented dictionary representation of URLs to UUIDs of the web resources

contained.

Returns

- string : a sorted and indented dictionary representation of URLs to UUIDs of the web resources map

absorb_file_manager(other_wrm)

This object starts tracking all of the files in the other_file_manager by adding the files to its dictionary. other_file_manager and its files are not altered in any way.

Parameters

- other_wrm (string) : another WebResourceManager object

Returns

- None

get_corpus(process_all=False)

Converts the text to a format more suitable for NLP processing

The function decodes using the "UTF-8" codec, ignoring errors, uses regular expressions to remove email addresses and all non-alpha-numeric characters except ' and -, then lemmatizes and stems using NLTK's WordNetLemmatizer and PorterStemmer respectively.

Parameters

- process_all (bool) : indicates whether you would like to use the 'corpus' fields of previously processed web resources (False) or reprocess all of the resources (True)

Returns

- None

get_TaggedDocuments()

A generator that yields the **corpus** fields of the contained documents as gensim's TaggedDocument objects with the **id** and **query** fields as tags.

get_texts()

Simple generator function so you can iterator over the text fields of documents. Can be used as a shortcut for iterating over the documents and then extracting the text field. See Usage for examples.

get_uuid(url)

- **url** is the URL of the web resource

Returns the UUID of the web resource with the specified URL.

load(file_name=None)

- **file_name** is the name of the file you wish to load a WebResourceManager from.

Reads in the WebResourceManager object from a file. The default is "data/webresourcemanager.json" and if rel_path was specified it will be "rel_path/data/webresourcemanager.json"

read_in_from_directory(directory)

- **directory** is the name of the directory you would like to read files in from.

NOTE: In order to avoid weird interactions, this function ignores **rel_path**. You put the path from the current working directory in as **directory**.

For each non-directory file in the directory, it attempts to load the file and add it to its dictionary. If this fails, it prints an error message that reads:

> "File {} was not in the proper format to be tracked with WebResourceManager".format(file)

read_in_from_iterator(iterator_of_docs)

- **iterator_of_docs** is an iterator of documents which are just dictionaries with a **text**, **query**, **url**, and either **html** or **pdf** attributes.

For each document, the method generates a UUID, adds the URL to UUID mapping to the dictionary, writes the appropriate source file (< UUID >.html or < UUID >.pdf), addes a timestamp (seconds since epoch), and writes a content JSON with the **id**, **query**, **text**, **time**, and **url** fields.

rank_by_relevance()

For each file in the WebResourceManager, it passes it's text attribute to **get_relevance_score** and assigns the return value to the document's **relevance_score** attribute.

string_to_uuid(string)

- **string** is a string, in practice it is generally the URL of the web resource

Returns the UUID for the web resource.

save(file_name=None)

- **file_name** is the name of the file you wish to save the WebResourceManager to.

Writes the WebResourceManager object to the **file_name** if specified. Otherwise, it writes to "data/webresourcemanager.json" or "rel_path/data/webresourcemanager.json" if rel_path was specified.

train_classifier()

Trains the **classifier** on the WebResourceManager's documents

Basic Usage

This section acts as a tutorial for how to use WebResourceManager without reading through every function in the Documentation section.

What is a WebResourceManager?

A WebResourceManager is basically a dictionary that converts URLs to UUIDs (to label and lookup files) with some fun functionalities. For example, because of the file labeling and data storage scheme, given a UUID or URL of a web resource, we can instantly pull up the source file (HTML or PDF) or the document file (attributes such as the text and URL).

How do I Create a WebResourceManager?

You create a WebResourceManager object just like any other object, and the constructor doesn't do any heavy lifting.

```
wrm = WebResourceManager()
```

The only thing interesing in the constructor is the **rel_path** keyword argument which allows you set the relative path from your current working environment to the directory where your data is housed. For example, let's say your directory layout looks like this:

project\

```
|----data\
|     |---docs\
|     |---source\
|----scripts\
     |----my_file.py
```

To use WebResourceManager from **my_file.py** (for example with an import statement), you want to set the **rel_path** variable to ".." because WebResourceManager always expects to have access to a "data" directory as well as "data/docs" and "data/source" subdirectories. However, you could also just put a "data" fold in "scripts" with both "docs" and "source" subdirectories as well.

How do I Populate a WebResourceManager?

To populate a WebResourceManager, you can use **load()** if you already have written a WebResourceManager object to a JSON. The **load()** function looks to "data/webresourcemanager.json" by default, but if you specify **rel_path** will look to "rel_path/data/webresourcemanager.json", and you can also just specify your own file_name which ignores **rel_path** and the "webresourcemanager.json" convention.

If you haven't written out a WebResourceManager or want to add more documents, you have two options: **read_in_from_iterator()** and **read_in_from_directory()**.

read_in_from_iterator() takes an iterator of documents. Documents are simply dictionaries with a **text**, **url** and either **html** or **pdf** attribute. The function generates a UUID using **string_to_uuid(string)**, puts the ID and URL pair in the dictionary, adds a time stamp, writes the source file, and writes the document file.

read_in_from_directory() requires documents that have already been written by a WebResourceManager and when you specify the a directory (ignoring **rel_path**), it attempts to read in all of the document files in the directory, printing error messages for files it cannot parse.

```
tmp = WebResourceManager() \
tmp.load(os.path.join("data/webresourcemanager", file_name)
```

If you have another WebResourceManager, you can also "absorb" those files by adding them to your dictionary. This is different from copying because you retain any files you were tracking previously. It is also important to note that the physical files and the other WebResourceManager are not altered during this function call, you just gain their files.

```
wrm = WebResourceManager() \
tmp = WebResourceManager() \
wrm.absorb_file_manager(tmp)
```

Accessing Files in WebResourceManager

Once you have a WebResourceManager object and with files, accessing the files is very easy!

To get the UUID of a web resource if you have its URL just use **get_uuid()**

```
wrm.get_uuid("http://money.cnn.com/quote/forecast/forecast.html?symb=BASFY") \
--> bd841c9a-ea85-54e4-9efa-1d627abbfa36
```

To access a file once you have a UUID, you can use **get()** of just use square brackets [ ] to access the document file.

```
wrm["bd841c9a-ea85-54e4-9efa-1d627abbfa36"]
```

To access a specific attribute of the file, because document files are just dictionaries, you can just append [attribute] to the end of that call. For example, if you wanted the query that generated the document, you would use:

```
wrm["bd841c9a-ea85-54e4-9efa-1d627abbfa36"]['query'] \
--> test
```

Iterating over files is a very common task and WebResourceManager has made that easy as well! You have a few options. First, you could iterate over WebResourceManager's **url_to_uuid** dictionary by calling **values()** on it to get a list of UUIDs in the file.

```
for elem in wrm.url_to_uuid.values(): \
   print(elem)
```

This is not recommend and there is built in support for iterating! The `__iter__()` function is a generator which returns an iterator over the files. This means that when you make a "for x in WebResourceManager" statement, it will give you the files you want. The following example prints the **id** attribute of the documents in the WebResourceManager:

```
for elem in wrm: \
   print(elem['id'])
```

We are of couse working with text documents however, so to save you the trouble of iterating and explicitly saying ['text'], there is the **get_texts()** generator which returns an iterator over the **text** attributes of the tracked files.

```
for elem in wrm.get_texts(): \
   print(elem)
```