Arcas Documentation

Release 1.0.0

Nikoleta Glynatsi

Nov 12, 2019

Contents

Arcas is Python library which allow users to collect academic articles' metadata.

A large number of scholarly databases and collections offer some form of API access. An API is an online tool to access data straight from the databases. Arcas is a tool designed to help communicate/ping various of these APIs.

Arcas offers access to metadata of articles from the following journals and pre-prints:

- IEEE
- PLOS
- Nature
- Springer
- arXiv

Note some journals might require you to register and generate an application *key*. Currently, the following journals require you to register your application:

- IEEE
- Spinger

Guidelines for adding your key to the right place can be found under the *How to: Register Application and use api_key*.

CHAPTER 1

Table of Contents

1.1 Installing Arcas

From PyPi:

```
$ pip install arcas
```

From GitHub:

```
$ git clone https://github.com/Nikoleta-v3/Arcas.git
$ cd Arcas
$ pip install -r requirements.txt
$ python setup.py install
```

Arcas is supported by Python 3.5.

1.2 Tutorials

Arcas' tutorials cover the basic usage of the library.

These include retrieving medata of a single article from a single API, retrieving the same article from various APIs and finally retrieving a large number of medata from different APIs.

Contents:

1.2.1 Tutorial I: Retrieving a single article

In this tutorial the aim is to retrieve a single article for the journal arXiv, where the word 'Game' is contained in the title or the abstract.

Initially, let us import Arcas:

```
>>> import arcas
```

The APIs, are implemented as classes. Here we make an API instance of the API arXiv:

```
>>> api = arcas.Arxiv()
```

We will now create the query, to which arXiv listens to. `records` is the number of records we are requesting for:

```
>>> parameters = api.parameters_fix(title='Game', abstract='Game', records=1)
>>> url = api.create_url_search(parameters)
```

The query will be used to ping the API and afterwards we parse the xml file that has been retrieved:

```
>>> request = api.make_request(url)
>>> root = api.get_root(request)
>>> raw_article = api.parse(root)
>>> article = api.to_dataframe(raw_article[0])
```

Note that we are using the library pandas to store the results. The data frame contains metadata on an article as they are recorded in the journal arXiv. Here we can type the following to see the columns of the data frame:

```
>>> article.columns
Index(['url', 'key', 'unique_key', 'title', 'author', 'abstract', 'doi',
   'date', 'journal', 'provenance', 'primary_category', 'category',
   'score', 'open_access'],
  dtype='object')
```

and we can ask for the title:

```
>>> article.title.unique()
    array([ 'A New Approach to Solve a Class of Continuous-Time Nonlinear
    Quadratic Zero-Sum Game Using ADP'], dtype=object)
```

Note that you might be getting a different title that me. That is fine it's just that new articles have been added to the API's database.

The structure of the results is discussed in depth in *result set*.

1.2.2 Tutorial II: Retrieve an article from various APIs

In this tutorial we are aiming to make a similar query, to that in *tutorial I*, from different APIs.

To achieve that we will use a `for` loop, to loop over a list of given APIs classes. For each instance then repeat the following procedure:

```
>>>  for p in [arcas.Ieee, arcas.Plos, arcas.Arxiv, arcas.Springer, arcas.Nature]:

...      api = p()
...      parameters = api.parameters_fix(title='Game', abstract='Game', records=1)
...      url = api.create_url_search(parameters)
...      request = api.make_request(url)
...      root = api.get_root(request)
...      raw_article = api.parse(root)

...      for art in raw_article:
...          article = api.to_dataframe(art)
...          api.export(article, 'results_{}.json'.format(api.__class__.__name__))
```

The `export` function, is a function that writes the results to a json file. Here the results of each API are stored to a different file named after which API they come from.

Note that you need to require a `key` before being able to use `arcas.Ieee` and `arcas.Springer`.

1.2.3 Tutorial III: Retrieving a large number of articles

Now that we have learned to ping several APIs for a single article, we will repeat the procedure for a large number of articles. In this example the number of articles we would like to retrieve is 20 from each API.

Often, we are looking for hundreds of articles. Rather than asking the API for all the results at once, the APIs offer a paging mechanism through `start` and `records`. That way we can receive chunks of the result set at a time. `start` defines the index of the first returned article and `records` the number of articles returned by the query.

```
>>> for p in [arcas.Ieee, arcas.Plos, arcas.Arxiv, arcas.Springer, arcas.Nature]:
...     for start in range(2):
...
...         api = p()
...         parameters = api.parameters_fix(title='Game', abstract='Game',
...                                         records=10, start=(start * 10))
...         url = api.create_url_search(parameters)
...         request = api.make_request(url)
...         root = api.get_root(request)
...         raw_article = api.parse(root)
...
...     for art in raw_article:
...         article = api.to_dataframe(art)
...         api.export(article, 'results_{}.json'.format(api.__class__.__name__))
```

In our example this might not seem as an important difference. But assume you were asking for a hundred of articles. Some APIs have a limited number of articles that be can returned, thus using this practice we avoid overloading the API.

Note that you need to require a `key` before being able to use `arcas.Ieee` and `arcas.Springer`.

1.3 Guides

Contents:

1.3.1 How to use Arcas from the command line

Arcas is a tool which can be used by the command line as well.

To get information on the arguments we can pass we type the following command in a command prompt:

```
$ arcas_scrape --h
Arcas. A library to facilitate scraping of APIs for scholarly resources.

Usage:
    arcas_scrape  [-h] [-p API] [-a AUTHOR] [-t TITLE] [-b ABSTRACT] [-y YEAR]
                  [-r RECORDS] [-s START] [-v VALIDATE] [-f FILENAME]
    arcas_scrape --version
```

(continues on next page)

(continued from previous page)

```
Options:
    -h --help               Show this
    --version               Show version.
    -p API                  The online API, from a given list, to parse [default:␣
↪arxiv]
    -a AUTHOR               Terms to search for in Author
    -t TITLE                Terms to search for in Title
    -b ABSTRACT             Terms to search for in the Abstract
    -y YEAR                 Terms to search for in Year
    -r RECORDS              Number of records to fetch [default: 1]
    -s START                Sequence number of first record to fetch [default: 1]
    -v VALIDATE             Checks if query returned with arguments asked [default:␣
↪False]
    -f FILENAME             Name of json file [default: results.json]
```

1.3.2 How to: Register Application and use *api_key*

Open APIs exist to allow users to access academic meta data easily. Some of those APIs may require a user to register their application in order to do so.

Currently, the following APIs implemented within the library will require you to register:

- IEEE Xplore; registration link: https://developer.ieee.org/member/register
- Springer Open Access API; registration link: https://dev.springernature.com/login

One you have registered as a user and have register your application, it will be given an application key.

In order to be able to use the APIs listed here via Arcas you will have to add this key to the *api_key.py* file under the API's respective folder.

Firstly, you will have to clone the repository from GitHub using the following command:

```
$ git clone https://github.com/Nikoleta-v3/Arcas.git
```

Once you have a copy of the repository you can see that there is a folder for each API located at *src/arcas*. We can see this by typing the following commands:

```
$ cd Arcas/src/arcas
$ ls
arXiv  IEEE  __init__.py  nature  PLOS  __pycache__  Springer  tools.py  version.py
```

Both *IEEE* and the *Springer* folders has an *api_key.py* file which is where we need to add your application key.

For example lets consider *IEEE*. Using the following command we list the files within the folder:

```
$ ls IEEE
api_key.py  __init__.py  main.py
```

We can also see what's in the *api_key.py* file:

```
$ cat IEEE/api_key.py
api_key = 'Your key here'
```

All we need to do is replace our key with the `'Your key here'` and save.

Once this is done all you have to do is go ahead and install the library. We need to navigate to the top of the repository:

```
$ cd ...
```

and then just use the following command to install the package:

```
$ python setup.py install
```

We will need to add your Springer key in `src/arcas/Springer/api_key.py` as well so we can use both APIs.

Once we have done this we should be ready to use all the APIs available to us via Arcas.

1.3.3 How to: Collect articles' based on *title*

Academic articles are published with a given title by their authors. Some times we found ourselves in search of articles relevant to our field and we do not know where to start. The most common approach is to search articles where a word describing our topic of interest is included in the article's title.

For example a mathematician might be interested in looking for articles' that the world eigenvalues appears on the title.

Initially we need to chose a publisher, for this example we assume that we are interested in the articles published by Nature:

```
>>> import arcas
>>> api =  arcas.Nature()
```

Now all that is needed to specify in the parameters that we want *title='eigenvalues'*:

```
>>> parameters = api.parameters_fix(title='eigenvalues')
>>> url = api.create_url_search(parameters)
```

The query will be used to ping the API and afterwards we parse the response that has been retrieved:

```
>>> request = api.make_request(url)
>>> root = api.get_root(request)
>>> raw_article = api.parse(root)
>>> article = api.to_dataframe(raw_article[0])
```

We can perform an insanity check and reassure that `'eigenvalues'` is indeed within the title:

```
>>> 'eigenvalues' in article['title'].unique()[0]
True
```

Note that Arcas can be used from the command line as well. If we wanted to reproduced the same example the command would be:

```
$ arcas_scrape -p nature -t "eigenvalues"
```

1.3.4 How to: Collect articles' based on *abstract*

Often we might search articles based on words that can be found withing the abstract of the article. For example one might interested in an article's metadata for which the word eigenvalues is within the abstract.

For this example we are going to be using the API of Nature:

```
>>> import arcas
>>> api =  arcas.Nature()
```

Now all that is needed to specify in the parameters that we want *abstract='eigenvalues'*:

```
>>> parameters = api.parameters_fix(title='eigenvalues')
>>> url = api.create_url_search(parameters)
```

The query will be used to ping the API and afterwards we parse the response that has been retrieved:

```
>>> request = api.make_request(url)
>>> root = api.get_root(request)
>>> raw_article = api.parse(root)
>>> article = api.to_dataframe(raw_article[0])
```

Note that Arcas can be used from the command line as well. To reproduce the query in the command line would would type the following:

```
$ arcas_scrape -p nature -a "eigenvalues"
```

1.3.5 How to: Collect articles' based on *year*

Publication date of an article is another search field available with Arcas. Consider an example whereas we are interested in articles that have been published on a specific year.

Let us assume that we are interested in the first article that will is returned by Plos that has been publish in 1993:

```
>>> import arcas
>>> api =  arcas.Plos()
```

Now all that is needed to specify in the parameters that we want *year=1993*:

```
>>> parameters = api.parameters_fix(year=1993)
>>> url = api.create_url_search(parameters)
```

The url can be used to retrieve the response which is then passed to a data frame:

```
>>> request = api.make_request(url)
>>> root = api.get_root(request)
>>> raw_article = api.parse(root)
>>> article = api.to_dataframe(raw_article[0])
```

The same example can be used to collect the article using the command line:

```
$ arcas_scrape -p plos -y 1993
```

1.3.6 How to: Collect articles' based on *journal*

Articles can also be retrieved using the full journal name/publication title.

Thus sometime we might not be specifying only the publisher but the exact journal as well. This can be done using the argument *journal*.

```
>>> import arcas
>>> api =  arcas.Nature()
```

Assume that we would like to fetch an article from Nature's Blood Cancer Journal. The query message will be the following:

```
>>> parameters = api.parameters_fix(journal='Blood Cancer Journal')
>>> url = api.create_url_search(parameters)
'http://www.nature.com/opensearch/request?&query=prism.publicationName=Blood Cancer␣
↪Journal'
```

1.3.7 How to: Collect articles' based on *category*

Subject terms are often given to articles either by the authors or the journals themselves. Arcas allow the user to search articles that satisfies a given subject term using the `category` argument.

For example the query for a game theoretic article in arXiv would be the following:

```
>>> import arcas
>>> api =  arcas.Nature()
>>> parameters = api.parameters_fix(category='Game Theory')
>>> url = api.create_url_search(parameters)
'http://www.nature.com/opensearch/request?&query=dc.subject adj Game Theory'
```

1.4 Reference

Contents:

1.4.1 Search Parameters

The table below outlines the parameters that can be passed to the query interface:

Parameter	Description
`author`	Searches both first name and last name.
`title`	Locate documents containing a word or phrase in the "article title" element.
`abstract`	Locate documents containing a word or phrase in the "abstract" element.
`year`	The value for publication year.
`category`	Allows users to search the by keywords given to an article.
`journal`	Locate documents containing a word or phrase in the "full journal/publication title" element.
`records`	The number of records to fetch.
`start`	Sequence number of first record to fetch.

If a search argument is not available for a given API a message will be displayed.

1.4.2 Results set

Each response of the API returns a list of metadata for a given article. This list differs for each API. Arcas is designed to return a similar set of metadata for any given API. Thus the json results of Arcas has the following list of metadata:

- **`key`**
  - A generated key containing an authors name and publication year (e.g. Glynatsi2017)
- **`unique_key`**

  - A unique key generated using the hashlib python library. The hashable string is created by: [author name, title, year,abstract]

- **title**
  - Title of article
- **author**
  - A single entity of an author from the list of authors of the respective article
- **abstract**
  - The abstract of the article
- **date**
  - Date of publication
- **doi**
  - Article's doi
- **url**
  - Article's url
- **journal**
  - Journal of publication
- **pages**
  - Pages of publication
- **key_word**
  - A single entity of a keyword assigned to the article by the given journal
- **provenance**
  - Scholarly database for where the article was collected
- **category**
  - A list of subjects given to the article by the authors
- **score**
  - Score given to article by the given journal
- **open_access**
  - A boolean describing whether the article is open access or not

Note that if a specific result is not available by an API, not because is missing but because is not implemented, Arcas returns `'Not available'` for the value of that column.

1.4.3 List of available APIS

A list of the APIs you can ping with Arcas. Contents:

arXiv API

arXiv API is hosted at arXiv.org, is a document submission and retrieval system that is heavily used by the physics, mathematics and computer science communities.

arXiv is set as the default API for Arcas. For more information on interacting with the api visit the official site for the user's manual: https://arxiv.org/help/api/user-manual.

The `Arxiv` class supports the following arguments as search fields:

- `author`
- `title`
- `abstract`
- `category`
- `journal`
- `records`
- `start`

The most recent check of compatibility between Arcas and the arXiv API was done on the 27th of August 2018.

IEEE Xplore

Query the Institute of Electrical and Electronics Engineers content repository and retrieve results for manipulation and presentation on local web interfaces.

Information on the IEEE Xplore can be found on the official site: https://developer.ieee.org/docs.

IEEE Xplore API requires a user to register their application in order to use the API. Once the application has been registered an API key is generated a user can use Arcas to collect articles.

Guidelines on using your API key with Arcas can be found under *How to: Register Application and use api_key*.

The `Ieee` class supports the following arguments as search fields:

- `author`
- `title`
- `abstract`
- `category`
- `journal`
- `year`
- `records`
- `start`

The most recent check of compatibility between Arcas and the nature.com OpenSearch API was done on the 27th of August 2018.

nature.com OpenSearch API

The nature.com OpenSearch API provides an open, bibliographic search service for content hosted on nature.com, comprising around half a million news and research articles and citations

For more information on interacting with the nature.com OpenSearch API visit the official site: https://www.nature.com/opensearch/.

The `Nature` class supports the following arguments as search fields:

- `author`
- `title`
- `abstract`
- `category`
- `journal`
- `year`
- `records`
- `start`

The most recent check of compatibility between Arcas and the nature.com OpenSearch API was done on the 27th of August 2018.

Springer Open Access API

Springer Open Access API - Provides metadata and full-text content for more than 370,000 online documents from Springer open access xml, including BioMed Central and SpringerOpen journals.

Information on the Springer Open Access API can be found on the official site: https://dev.springer.com/restfuloperations. In order to use the Springer Open Access API a user must register an application and generate an application key which is used in the query message for access. Guidelines on using your API key with Arcas can be found under *How to: Register Application and use api_key*.

`Springer` class supports the following arguments as search fields:

- `author`
- `title`
- `journal`
- `category`
- `records`
- `start`

and the most recent check of compatibility between Arcas and the Springer Open AccessI API was done on the 27th of August 2018.

PLOS Search API

Query content from the seven open-access peer-reviewed journals from the Public Library of Science using any of the twenty-three terms in the PLOS Search.

For more information on PLOS Search API visit the official site: http://api.plos.org/ under Documentation.

`Plos` class supports the following arguments as search fields:

- `author`
- `title`
- `abstract`

- `category`
- `journal`
- `year`
- `records`
- `start`

The most recent check of compatibility between Arcas and the nature.com OpenSearch API was done on the 27th of August 2018.

CHAPTER 2

Indices and tables

- search