textract Documentation

Release 1.3.0

Dean Malmgren

June 23, 2015

Contents

1 Currently supporting **3**

2 Related projects **5**
2.1 Command line interface 5
2.2 Python package 6
2.3 Installation 8
2.4 Contributing 9
2.5 Change Log 11

3 Indices and tables **15**

As undesireable as it might be, more often than not there is extremely useful information embedded in Word documents, PowerPoint presentations, PDFs, etc—so-called "dark data"—that would be valuable for further textual analysis and visualization. While *several packages* exist for extracting content from each of these formats on their own, this package provides a single interface for extracting content from any type of file, without any irrelevant markup.

This package provides two primary facilities for doing this, the *command line interface*

```
textract path/to/file.extension
```

or the *python package*

```
# some python file
import textract
text = textract.process("path/to/file.extension")
```

CHAPTER 1

Currently supporting

textract supports a growing list of file types for text extraction. If you don't see your favorite file type here, Please recommend other file types by either mentioning them on the issue tracker or by *contributing a pull request*.

- `.csv` via python builtins
- `.doc` via antiword
- `.docx` via python-docx
- `.eml` via python builtins
- `.epub` via ebooklib
- `.gif` via tesseract-ocr
- `.jpg` and `.jpeg` via tesseract-ocr
- `.json` via python builtins
- `.html` and `.htm` via beautifulsoup4
- `.mp3` via SpeechRecognition and sox
- `.msg` via msg-extractor
- `.odt` via python builtins
- `.ogg` via SpeechRecognition and sox
- `.pdf` via pdftotext (default) or pdfminer
- `.png` via tesseract-ocr
- `.pptx` via python-pptx
- `.ps` via ps2text
- `.rtf` via unrtf
- `.tiff` via tesseract-ocr
- `.txt` via python builtins
- `.wav` via SpeechRecognition
- `.xlsx` via xlrd
- `.xls` via xlrd

CHAPTER 2

Related projects

Of course, textract isn't the first project with the aim to provide a simple interface for extracting text from any document. But this is, to the best of my knowledge, the only project that is written in python (a language commonly chosen by the natural language processing community) and is *method agnostic about how content is extracted*. I'm sure that there are other similar projects out there, but here is a small sample of similar projects:

- Apache Tika has very similar, if not identical, aims as textract and has impressive coverage of a wide range of file formats. It is written in java.
- textract (node.js) has similar aims as this textract package (including an identical name! great minds...). It is written in node.js.
- pandoc is intended to be a document conversion tool (a much more difficult task!), but it does have the ability to convert to plain text. It is written in Haskell.

Contents:

2.1 Command line interface

2.1.1 textract

Command line tool for extracting text from any document.

```
usage: textract [-h]
                [-e {aliases,ascii,base64_codec,big5,big5hkscs,bz2_codec,charmap,cp037,cp1
                [-m METHOD] [-o OUTPUT] [-O OPTION] [-v]
                filename
```

Positional arguments:

filename Filename to extract text.

Options:

-e=utf_8, --encoding=utf_8 Specify the encoding of the output.

Possible choices: aliases, ascii, base64_codec, big5, big5hkscs, bz2_codec, charmap, cp037, cp1006, cp1026, cp1140, cp1250, cp1251, cp1252, cp1253, cp1254, cp1255, cp1256, cp1257, cp1258, cp424, cp437, cp500, cp720, cp737, cp775, cp850, cp852, cp855, cp856, cp857, cp858, cp860, cp861, cp862, cp863, cp864, cp865, cp866, cp869, cp874, cp875, cp932, cp949, cp950, euc_jis_2004, euc_jisx0213, euc_jp, euc_kr, gb18030, gb2312, gbk, hex_codec, hp_roman8, hz,

idna, iso2022_jp, iso2022_jp_1, iso2022_jp_2, iso2022_jp_2004, iso2022_jp_3, iso2022_jp_ext, iso2022_kr, iso8859_1, iso8859_10, iso8859_11, iso8859_13, iso8859_14, iso8859_15, iso8859_16, iso8859_2, iso8859_3, iso8859_4, iso8859_5, iso8859_6, iso8859_7, iso8859_8, iso8859_9, johab, koi8_r, koi8_u, latin_1, mac_arabic, mac_centeuro, mac_croatian, mac_cyrillic, mac_farsi, mac_greek, mac_iceland, mac_latin2, mac_roman, mac_romanian, mac_turkish, mbcs, palmos, ptcp154, punycode, quopri_codec, raw_unicode_escape, rot_13, shift_jis, shift_jis_2004, shift_jisx0213, string_escape, tactis, tis_620, undefined, unicode_escape, unicode_internal, utf_16, utf_16_be, utf_16_le, utf_32, utf_32_be, utf_32_le, utf_7, utf_8, utf_8_sig, uu_codec, zlib_codec

-m=, --method= Specify a method of extraction for formats that support it

-o=-, --output=- Output raw text in this file

-O, --option Add arbitrary options to various parsers of the form KEYWORD=VALUE. A full list of available KEYWORD options is available at http://bit.ly/textract-options

-v, --version show program's version number and exit

Note: To make the command line interface as usable as possible, autocompletion of available options with textract is enabled by @kislyuk's amazing argcomplete package. Follow instructions to enable global autocomplete and you should be all set. As an example, this is also configured in the virtual machine provisioning for this project.

2.2 Python package

This package is organized to make it as easy as possible to add new extensions and support the continued growth and coverage of textract. For almost all applications, you will just have to do something like this:

```
import textract
text = textract.process('path/to/file.extension')
```

to obtain text from a document. You can also pass keyword arguments to `textract.process`, for example, to use a particular method for parsing a pdf like this:

```
import textract
text = textract.process('path/to/a.pdf', method='pdfminer')
```

or to specify a particular output encoding (input encodings are inferred using chardet):

```
import textract
text = textract.process('path/to/file.extension', encoding='ascii')
```

2.2.1 Additional options

Some parsers also enable additional options which can be passed in as keyword arguments to the `textract.process` function. Here is a quick table of available options that are available to the different types of parsers:

parser	option	description
gif	language	Specify the language for OCR-ing text with tesseract
jpg	language	Specify the language for OCR-ing text with tesseract
png	language	Specify the language for OCR-ing text with tesseract
pdf	language	For use when `method='tesseract'`, specify the language
tiff	language	Specify the language for OCR-ing text with tesseract

As an example of using these additional options, you can extract text from a Norwegian PDF using Tesseract OCR like this:

```
text = textract.process(
    'path/to/norwegian.pdf',
    method='tesseract',
    language='nor',
)
```

2.2.2 A look under the hood

When `textract.process('path/to/file.extension')` is called, `textract.process` looks for a module called `textract.parsers.extension_parser` that also contains a `Parser`.

`textract.parsers.`**`process`**(*filename*, *encoding='utf_8'*, ***kwargs*)

This is the core function used for extracting text. It routes the `filename` to the appropriate parser and returns the extracted text as a byte-string encoded with `encoding`.

Importantly, the `textract.parsers.extension_parser.Parser` class must inherit from `textract.parsers.utils.BaseParser`.

class `textract.parsers.utils.`**`BaseParser`**

Bases: `object`

The *`BaseParser`* abstracts out some common functionality that is used across all document Parsers. In particular, it has the responsibility of handling all unicode and byte-encoding.

`decode`(*text*)

Decode `text` using the chardet package.

`encode`(*text*, *encoding*)

Encode the `text` in `encoding` byte-encoding. This ignores code points that can't be encoded in byte-strings.

`extract`(*filename*, ***kwargs*)

This method must be overwritten by child classes to extract raw text from a filename. This method can return either a byte-encoded string or unicode.

`process`(*filename*, *encoding*, ***kwargs*)

Process `filename` and encode byte-string with `encoding`. This method is called by *`textract.parsers.process()`* and wraps the *`BaseParser.extract()`* method in a delicious unicode sandwich.

Many of the parsers rely on command line utilities to do some of the parsing. For convenience, the `textract.parsers.utils.ShellParser` class includes some convenience methods for streamlining access to the command line.

class `textract.parsers.utils.`**`ShellParser`**

Bases: *`textract.parsers.utils.BaseParser`*

The *`ShellParser`* extends the *`BaseParser`* to make it easy to run external programs from the command line with Fabric-like behavior.

run(*command*)
Run `command` and return the subsequent `stdout` and `stderr` as a tuple. If the command is not successful, this raises a `textract.exceptions.ShellError`.

temp_filename()
Return a unique tempfile name.

2.2.3 A few specific examples

There are quite a few parsers included with `textract`. Rather than elaborating all of them, here are a few that demonstrate how parsers work.

class `textract.parsers.epub_parser.`**Parser**
Bases: *textract.parsers.utils.BaseParser*

Extract text from epub using python epub library

extract(*filename*, ***kwargs*)

class `textract.parsers.doc_parser.`**Parser**
Bases: *textract.parsers.utils.ShellParser*

Extract text from doc files using antiword.

extract(*filename*, ***kwargs*)

2.3 Installation

One of the main goals of textract is to make it as easy as possible to start using textract (meaning that installation should be as quick and painless as possible). This package is built on top of several python packages and other source libraries. Assuming you are using `pip` or `easy_install` to install textract, the python packages are all installed by default with textract. The source libraries are a separate matter though and largely depend on your operating system.

2.3.1 Ubuntu / Debian

There are two steps required to run this package on Ubuntu/Debian. First you must install some system packages using the apt-get package manager before installing textract from pypi.

```
apt-get install python-dev libxml2-dev libxslt1-dev antiword unrtf poppler-utils pstotext tesseract-o
flac ffmpeg lame libmad0 libsox-fmt-mp3 sox
pip install textract
```

Note: It may also be necessary to install `zlib1g-dev` on Docker instances of Ubuntu. See issue #19 for details

2.3.2 OSX

These steps rely on you having homebrew installed as well as the cask plugin (`brew install caskroom/cask/brew-cask`). The basic idea is to first install XQuartz before installing a bunch of system packages before installing textract from pypi.

```
brew cask install xquartz
brew install poppler antiword unrtf tesseract
pip install textract
```

Note: pstotext is not currently a part of homebrew so `.ps` extraction must be enabled by manually installing from source.

Note: Depending on how you have python configured on your system with homebrew, you may also need to install the python development header files for textract to properly install.

2.3.3 Don't see your operating system installation instructions here?

My apologies! Installing system packages is a bit of a drag and its hard to anticipate all of the different environments that need to be accomodated (wouldn't it be awesome if there were a system-agnostic package manager or, better yet, if python could install these system dependencies for you?!?!). If you're operating system doesn't have documenation about how to install the textract dependencies, please *contribute a pull request* with:

1. A new section in here with the appropriate details about how to install things. In particular, please give instructions for how to install the following libraries before running `pip install textract`:
   - libxml2 2.6.21 or later is required by the `.docx` parser which uses lxml via python-docx.
   - libxslt 1.1.15 or later is required by the `.docx` parser which users lxml via python-docx.
   - python header files are required for building lxml.
   - antiword is required by the `.doc` parser.
   - pdftotext is *optionally* required by the `.pdf` parser (there is a pure python fallback that works if pdftotext isn't installed).
   - pstotext is required by the `.ps` parser.
   - tesseract-ocr is required by the `.jpg`, `.png` and `.gif` parser.
   - sox is required by the `.mp3` and `.ogg` parser. You need to install ffmpeg, lame, libmad0 and libsox-fmt-mp3, before building sox, for these filetypes to work.
2. Add a requirements file to the requirements directory of the project with the lower-cased name of your operating system (e.g. `requirements/windows`) so we can try to keep these things up to date in the future.

2.4 Contributing

The overarching goal of this project is to make it as easy as possible to extract raw text from any document for the purposes of most natural language processing tasks. In practice, this means that this project should preferentially provide tools that correctly produce output that has words in the correct order but that whitespace between words, formatting, etc is totally irrelevant. As the various parsers mature, I fully expect the output to become more readable to support additional use cases, like extracting text to appear in web pages.

Importantly, this project is committed to being as agnostic about how the content is extracted as it is about the means in which the text is analyzed downstream. This means that `textract` should support multiple modes of extracting text from any document and provide reasonably good defaults (defaulting to tools that tend to produce the correct word sequence).

Another important aspect of this project is that we want to have extremely good documentation. If you notice a type-o, error, confusing statement etc, please fix it!

2.4.1 Quick start

1. Fork and clone the project:

```
git clone https://github.com/YOUR-USERNAME/textract.git
```

2. Contribute! There are several open issues that provide good places to dig in. Check out the contribution guidelines and send pull requests; your help is greatly appreciated!

Depending on your development preferences, there are lots of ways to get started developing with textract:

Developing in a native Ubuntu environment

3. Install all the necessary system packages:

```
./provision/travis-mock.sh
./provision/debian.sh

# optionally run some of the steps in these scripts, but you
# may want to be selective about what you do as they alter global
# environment states
./provision/python.sh
./provision/development.sh
```

4. On the virtual machine, make sure everything is working by running the suite of functional tests:

```
nosetests
```

These functional tests are designed to be run on an Ubuntu 12.04 LTS server, just like the virtual machine and the server that runs the travis-ci test suite. There are some other tests that have been added along the way in the Travis configuration. For your convenience, you can run all of these tests with:

```
./tests/run.py
```

Current build status:

Developing with Vagrant virtual machine

3. Install Vagrant and Virtualbox and launch the development virtual machine:

```
vagrant plugin install iniparse
vagrant up && vagrant provision
```

On `vagrant ssh`ing to the virtual machine, note that the `PYTHONPATH` and `PATH` environment variables have been altered in this virtual machine so that any changes you make to textract in development are automatically incorporated into the command.

4. See *step 4* in the Ubuntu development environment. Current build status:

Developing with Docker container

3. Go to the Docker documentation and follow the instructions under "If you'd like to try the latest version of Docker" to install Docker.

4. Just run `tests/run_docker_tests.sh` to run the full test suite. Current build status:

2.5 Change Log

This project uses semantic versioning to track version numbers, where backwards incompatible changes (highlighted in **bold**) bump the major version of the package.

2.5.1 latest changes in development for next release

2.5.2 1.3.0

- support for `.rtf` files (#84)
- support for `.msg` files (#87 and #17 by @anthonygarvan)

2.5.3 1.2.0

- support for `.tiff` files (**'#81'_**)
- added support for other languages for tesseract (#76 by @anderser)
- added `--option/-O` flag to pass arbitrary arguments for things like languages into textract
- several bug fixes, including:
  - fix bug with doing OCR on multi-page pdfs and removing temporary directory (#82 by @pudo)
  - correctly accounting for whitespace in `.odt` documents (#79 by @evfredericksen)
  - standardizing testing environment to be compatible with different versions of third-party command line tools (#78)

2.5.4 1.1.0

- support for `.wav`, `.mp3`, and `.ogg` files (#56 and #62 by @arvindch)
- support for `.csv` files (#64)
- support for scanned `.pdf` files with tesseract (#66 by @pudo)
- support for `.htm` files (**'#69'_**)
- several bug fixes, including:
  - `.odt` parser now correctly extracts text in order (#61 by @levivm)
  - fixed Docker development environment compatability with the Vagrant VM environment (#73 by @ShawnMilo)
- several internal improvements, including:
  - improvements in the python documentation (#70)
  - improved html output with reduced whitespace around inline elements in output text (#58 by @eiotec)

2.5.5 1.0.0

- **standardized encoding of output with ‘‘-e/–encoding‘‘ option** (#39)
- support for `.xls` and `.xlsx` files (#42 and #55 by @levivm)
- support for `.epub` files (#40 by @kokxx)
- several bug fixes, including:
  - removing tesseract version info from output of image parsers (#48)
  - problems with spaces in filenames (#53)
  - concurrancy problems with tesseract (#44 by @ShawnMilo, #41 by @christomitov)
- several internal improvements, including:
  - switching to using class-based parsers to abstract away the common functionality between different parser classes (#39)
  - switching to using a python-based test suite and added standardized text tests to make sure output is consistent across file types (#49)
  - including support for Docker-based testing (#46 by @ShawnMilo)

2.5.6 0.5.1

- several bug fixes, including:
  - documentation fixes
  - shell commands hanging on large files (#33)

2.5.7 0.5.0

- support for `.json` files (#13 by @anthonygarvan)
- support for `.odt` files (#29 by @christomitov)
- support for `.ps` files (#25)
- support for `.gif`, `.jpg`, `.jpeg`, and `.png` files (#30 by @christomitov)
- several bug fixes, including:
  - improved fallback handling in `.pdf` parser if the `pdftotext` command line utility isn't installed (#26)
  - improved documentation for installation instructions on non-Ubuntu operating systems (#21, #26)
- several internal improvements, including:
  - cleaned up implementation of extension parsers to avoid magic

2.5.8 0.4.0

- support for `.html` files (#7)
- support for `.eml` files (#4)
- automated the documentation for the python package using sphinx-apidoc in docs/Makefile (#9)

2.5.9 0.3.0

- support for `.txt` files, haha (#8)
- fixed installation bug with not properly including requirements files in the manifest

2.5.10 0.2.0

- support for `.doc` files (#2)
- support for `.pdf` files (#3)
- several bug fixes, including:
  - fixing tab complete bug no file paths (#6)
  - fixing tests to make sure the work properly on travis-ci

2.5.11 0.1.0

- Initial release, support for `.docx` and `.pptx`

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

B

BaseParser (class in textract.parsers.utils), 7

D

decode() (textract.parsers.utils.BaseParser method), 7

E

encode() (textract.parsers.utils.BaseParser method), 7
extract() (textract.parsers.doc_parser.Parser method), 8
extract() (textract.parsers.epub_parser.Parser method), 8
extract() (textract.parsers.utils.BaseParser method), 7

P

Parser (class in textract.parsers.doc_parser), 8
Parser (class in textract.parsers.epub_parser), 8
process() (in module textract.parsers), 7
process() (textract.parsers.utils.BaseParser method), 7

R

run() (textract.parsers.utils.ShellParser method), 7

S

ShellParser (class in textract.parsers.utils), 7

T

temp_filename() (textract.parsers.utils.ShellParser method), 8