BatteryDataExtractor

Release v1.0

Shu Huang

Apr 05, 2023

CONTENTS:

BatteryDataExtractor is a battery-aware text-mining software embedded with BERT models for automatically extracting chemical information from scientific literature.

Features:

- Open-source battery-specific literature-mining toolkit
- BERT-based token-classification models: abbreviation detection, part-of-speech tagging, chemical-named-entity recognition
- Double-turn question-answering model for the data extraction of materials and properties
- State-of-the-art performance on downstream evaluation data sets
- Updated NLP plugins: new web scrapers, document readers, and tokenizers
- New options: database auto-saving, original text-saving, and device-selection

Citing:

BatteryDataExtractor: battery-aware text-mining software embedded with BERT models

CHAPTER

ONE

GETTING STARTED

1.1 Installation

You can use `pip` to install BatteryDataExtractor:

```
pip install batterydataextractor
```

1.2 Reading a Document

Most commonly, you want to pass an entire document file to BatteryDataExtractor. BatteryDataExtractor comes with three built-in Document readers that can read HTML or XML files. These readers are responsible for detecting different elements of a document and recompiling them into a single consistent document structure:

```
>>> from batterydataextractor import Document
>>> f = open('paper.html', 'rb')
>>> doc = Document.from_file(f)
```

Each reader will be tried in turn until one is successfully able to read the file. If you know exactly which readers you want to use, it is possible to specify a list as an optional parameter:

```
>>> f = open('rsc_article.html', 'rb')
>>> doc = Document.from_file(f, readers=[RscHtmlReader()])
```

Note: Always open files in binary mode by using the 'rb' parameter.

Alternatively, you can load a document into BatteryDataExtractor by passing it some text:

```
>>> doc = Document('The capacity of graphite is 372 mAh/g.')
```

At present, the available readers are:

- RscHtmlReader - For RSC HTML articles
- ElsevierXmlReader - For Elsevier XML papers
- SpringerXmlReader - For Springer XML with JATS format
- PlainTextReader - Generic plain text

1.3 Document Elements

Once read, documents are represented by a single linear stream of *element* objects. This stream is now independent of the initial document type or the source:

```
>>> doc.elements
[Metadata('title, authors, journal, date, abstract...'),
Heading('Abstract'),
Paragraph('The first paragraph of text...'),
...]
```

Element types include Title, Heading, Paragraph, Citation, Table, Figure, Caption and Footnote. You can retrieve a specific element by its index within the document:

```
>>> para = doc.elements[14]
>>> para
Paragraph(id=None, references=[], text='The mechanism of lithium intercalation in the so-
↪called ‘soft’ anodes, i.e. graphite or graphitable carbons, is well known. It develops␣
↪through well-identified, reversible stages, corresponding to progressive intercalation␣
↪within discrete graphene layers, to reach the formation of LiC6 with a maximum␣
↪theoretical capacity of 372 ± 2.4 mAh g1.')
```

You can also get the individual sentences of a paragraph:

```
>>> para.sentences
[Sentence('The mechanism of lithium intercalation in the so-called ‘soft’ anodes, i.e.␣
↪graphite or graphitable carbons, is well known.', 0, 123),
 Sentence('It develops through well-identified, reversible stages, corresponding to␣
↪progressive intercalation within discrete graphene layers, to reach the formation of␣
↪LiC6 with a maximum theoretical capacity of 372 ± 2.4 mAh g1.', 124, 344)]
```

Or, the individual tokens:

```
>>> para.tokens
[[Token('The', 0, 3),
Token('mechanism', 4, 13),
Token('of', 14, 16),
Token('lithium', 17, 24),
Token('intercalation', 25, 38),
Token('in', 39, 41),
Token('the', 42, 45),
Token('so', 46, 48),
Token('-', 48, 49),
...
]]
```

1.4 NLP toolkits

BatteryDataExtractor can produce a list of individual chemical entity mentions (CEMs) of the document:

```
>>> doc.cems
[Span('lithium', 17, 24),
 Span('graphite', 76, 84),
 Span('carbons', 100, 107),
 Span('graphene', 239, 247),
 Span('LiC6', 282, 286)]
```

Each mention is returned as a `Span`, which contains the mention text, as well as the start and end character offsets within the containing document element.

You can also output the abbreviations found in the document:

```
>>> doc.abbreviation_definitions
[[('Abbr: ', 'THF')], [('LF: ', 'Tetrahydrofuran')]]
```

CHAPTER

TWO

BERT-BASED AUTOMATED MODEL

2.1 Double-turn Question-answering Model

BatteryDataExtractor can create a BERT-based automated data extraction model for materials property by just providing the list of names of properties based on the double-turn question-answering strategy:

```
>>> from batterydataextractor.doc import Document
>>> doc = Document("The theoretical capacity of graphite is 372 mAh/g... In the case of␣
↪LiFePO4 chemistry, the absolute maximum voltage is 4.2V per cell.")
>>> doc.add_models_by_names(["capacity", "voltage"])
>>> records = doc.records
>>> for r in records:
>>>     print(r.serialize())
{'PropertyData': {'value': [372.0], 'units': 'mAh / g', 'raw_value': '372 mAh / g',
↪'specifier': 'capacity', 'material': 'graphite', 'confidence_score': 0.6248}}
{'PropertyData': {'value': [4.2], 'units': 'V', 'raw_value': '4.2 V', 'specifier':
↪'voltage', 'material': 'LiFePO4', 'confidence_score': 0.6432}}
```

Users can also extract non-battery-related properties and provide the confidence-score threshold. The original text can be saved by setting `original_text` as True:

```
>>> text = 'The melting point (mp) of Aspirin (C9H8O4): 146-147 °C.'
>>> doc = Document(text)
>>> property_names = ["mp"]
>>> doc.add_models_by_names(property_names, confidence_threshold=0.01, original_
↪text=True)
>>> for record in doc.records:
>>>     print(record.serialize())
{'PropertyData': {'value': [146.0, 147.0], 'units': '° C', 'raw_value': '146-147 ° C',
↪'specifier': 'mp', 'material': 'Aspirin', 'confidence_score': 0.3717, 'original_text':
↪'The melting point ( mp ) of Aspirin ( C9H8O4 ) : 146-147 ° C .'}}
```

2.2 General Question-answering Model

Similarly, the model of data extraction of general information can be created by providing the name of the general infomation:

```
>>> from batterydataextractor.doc.text import Paragraph
>>> text = '1H NMR spectra were recorded on a Varian MR-400 MHz instrument.'
>>> doc = Paragraph(text)
>>> doc.add_general_models(["apparatus"], confidence_threshold=0.1, original_text=True)
>>> for record in doc.records:
>>>     print(record.serialize())
{'GeneralInfo': {'answer': 'Varian MR - 400 MHz instrument', 'specifier': 'apparatus',
↪'confidence_score': 0.5065, 'original_text': '1H NMR spectra were recorded on a Varian␣
↪MR - 400 MHz instrument .'}}
```

Or users can ask self-defined questions by setting `self_defined` as True:

```
>>> from batterydataextractor.doc.text import Paragraph
>>> text = 'For current LIBs based on OLE system, the employed cathodes could be mainly␣
↪divided into two categories: LCO is still very popular in the consumer electronics␣
↪market and Ni-rich compounds have already taken a place in the electric vehicles where␣
↪the Tesla LiNi0.8Co0.15Al0.05O2 (NCA) cathode is a good example.'
>>> doc = Paragraph(text)
>>> doc.add_general_models(["Which cathode is commonly used in electric vehicles?"],␣
↪confidence_threshold=0.1, self_defined=True)
>>> for record in doc.records:
>>>   print(record.serialize())
{'GeneralInfo': {'answer': 'Ni - rich compounds', 'specifier': 'Which cathode is␣
↪commonly used in electric vehicles?', 'confidence_score': 0.1489}}
```

CHAPTER

THREE

NATURAL LANGUAGE PROCESSING

BatteryDataExtractor also includes state-of-the-art Natural Language Processing (NLP) facilities, as described here.

3.1 Tokenization

Sentence Tokenization

Use the `sentences` property on a text-based document element to perform sentence segmentation. The sentence tokenizer is based on `en_core_sci_sm` package that is trained specifically on scientific text by SciPy:

```
>>> from batterydataextractor.doc import Paragraph
>>> para = Paragraph('The mechanism of lithium intercalation in the so-called ‘soft’␣
↪anodes, i.e. graphite or graphitable carbons, is well known. It develops through well-
↪identified, reversible stages, corresponding to progressive intercalation within␣
↪discrete graphene layers, to reach the formation of LiC6 with a maximum theoretical␣
↪capacity of 372 ± 2.4 mAh g1.')
>>> para.sentences
[Sentence('The mechanism of lithium intercalation in the so-called ‘soft’ anodes, i.e.␣
↪graphite or graphitable carbons, is well known.', 0, 123),
 Sentence('It develops through well-identified, reversible stages, corresponding to␣
↪progressive intercalation within discrete graphene layers, to reach the formation of␣
↪LiC6 with a maximum theoretical capacity of 372 ± 2.4 mAh g1.', 124, 344)]
```

Each sentence object is a document element in itself, and additionally contains the start and end character offsets within its parent element.

Word Tokenization

Use the `tokens` property to get the word tokens:

```
>>> para.tokens
[[Token('The', 0, 3),
Token('mechanism', 4, 13),
Token('of', 14, 16),
Token('lithium', 17, 24),
Token('intercalation', 25, 38),
Token('in', 39, 41),
Token('the', 42, 45),
Token('so', 46, 48),
```

(continues on next page)

(continued from previous page)

```
Token('-', 48, 49),
...
]]

>>> para.sentences[0].tokens
[Token('1,4-Dibromoanthracene', 0, 21),
 Token('was', 22, 25),
 Token('prepared', 26, 34),
 Token('from', 35, 39),
 Token('1,4-diaminoanthraquinone', 40, 64),
 Token('.', 64, 65)]
```

There are also `raw_sentences` and `raw_tokens` properties that return strings instead of `Sentence` and `Token` objects.

3.2 Set CPU/GPU Device

Each document and paragraph are assigned a default `device` value as -1 (CPU). You can set the `device` as a local GPU rank (e.g. 0, 1) to accelerate the NLP pipeline:

```
>>> para.device = 0
>>> s = Sentence("Li-ion battery")
>>> s.device = 1
>>> print(doc.device, para.device, s.device)
-1, 0, 1
```

3.3 Part-of-speech Tagging

BatteryDataExtractor contains a chemistry-aware Part-of-speech tagger that is based on the fine-tuned base BERT-cased model. Use the `pos_tagged_tokens` property on a document element to get the tagged tokens:

```
>>> s = Sentence('1H NMR spectra were recorded on a 300 MHz BRUKER DPX300 spectrometer.')
>>> s.pos_tagged_tokens
[('1H', 'CD'),
('NMR', 'NNP'),
('spectra', 'NN'),
('were', 'VBD'),
('recorded', 'VBN'),
('on', 'IN'),
('a', 'DT'),
('300', 'CD'),
('MHz', '.'),
('BRUKER', 'NNP'),
('DPX300', 'NNP'),
('spectrometer', 'NN'),
('.', '.')]
```

Using Taggers Directly

All taggers have a `tag` method that takes a list of `RichToken` instances and returns a list of (token, tag) tuples. For more information on how to use these taggers directly, see the documentation for `BaseTagger`.

3.4 Lexicon

As BatteryDataExtractor processes documents, it adds each unique word that it encounters to the `Lexicon` as a `Lexeme`. Each `Lexeme` stores various word features, so they don't have to be re-calculated for every occurrence of that word.

You can access the Lexeme for a token using the `lex` property:

```
>>> s = Sentence('Sulphur and Oxygen.')
>>> s.tokens[0]
Token('Sulphur', 0, 7)
>>> s.tokens[0].lex.normalized
'sulfur'
>>> s.tokens[0].lex.is_hyphenated
False
```

3.5 Abbreviation Detection

Abbreviation detection is based on the fine-tuned BatteryOnlyBERT-cased model:

```
>>> p = Paragraph(u'Dye-sensitized solar cells (DSSCs) with ZnTPP = Zinc␣
↪tetraphenylporphyrin.')
>>> p.abbreviation_definitions
[[('Abbr: ', 'DSSCs'), ('Abbr: ', 'ZnTPP')],
 [('LF: ', 'Dye - sensitized solar cells'),
  ('LF: ', 'Zinc tetraphenylporphyrin')]]
```

3.6 Chemical Named Entity Recognition (CNER)

Chemical Named Entity Recognition (CNER) is based on the fine-tuned BatteryOnlyBERT-uncased model:

```
>>> doc.cems
  [Span('lithium', 17, 24),
   Span('graphite', 76, 84),
   Span('carbons', 100, 107),
   Span('graphene', 239, 247),
   Span('LiC6', 282, 286)]
```

Each mention is returned as a Span, which contains the mention text, as well as the start and end character offsets within the containing document element.