SearchBetter Documentation

Release 0.3.0

Neel Mehta

Jul 15, 2017

Contents

SearchBetter lets you make powerful, fast, and drop-in search engines for any dataset, no matter how small or how large. It also offers built-in query rewriting, which uses NLP to help your search engines show users more, and more relevant, search results for any search term.

With query rewriting, your search engine can return results for additional topics that are semantically related to the user's original saerch. For instance, a search for *machine learning* would normally only return results for pages that contain the words "machine learning". But with query rewriting, you would get results not only for *machine learning* but also, say, *artificial intelligence* and *neural networks*.

SearchBetter lets you power up your search engines with minimal effort. It's especially useful if you have a small dataset to search on, or if you don't have the time or data to make complex custom query rewriting algorithms.

CHAPTER 1

Getting started

Using SearchBetter in your Python project is easy:

```
pip install searchbetter
```

Refer to the SearchBetter GitHub repository for more instructions and the interactive demo for usage examples.

CHAPTER 2

Contents of this documentation

Rewriter

The Rewriter module contains classes that help you rewrite single words and phrases to a set of semantically related words and phrases. If you search for all of these words and phrases on a search engine, you can get more, and more useful, results than if you just searched for the original word or phrase.

For instance, *acceleration* could be rewritten to *{ acceleration, velocity, mass, force, physics }*.

Visit the interactive demo to learn how to use these classes.

Usage

First, `pip install searchbetter`.

Then, in your Python code:

```
from searchbetter import rewriter
```

Documentation

class `rewriter.`**`ControlRewriter`**

A rewriter that's basically a no-op. Just returns the term you give it. This is mostly useful for testing purposes.

`rewrite`(*term*)

Rewrites a term to a list containing just itself. This is the degenerate case of query rewriting - the original term isn't actually rewritten at all.

`rewrite(x) == [x]` for all x.

Parameters **`term`** (*`str`*) – a string to rewrite

Returns a list containing just `term`

Return type list(str)

class rewriter.**Rewriter**

Abstract class around a query rewriter, which takes a given term and rewrites it to a set of semantically related terms. This, hopefully, helps search engines return more, and more useful, results.

rewrite(*term*)

Rewrites a term to a list of new terms to search with. Abstract base!

Parameters **term** (*str*) – a string to rewrite

Returns a list of semantically related strings, including `term`

Return type list(str)

class rewriter.**WikipediaRewriter**

A class to rewrite queries using Wikipedia's Category API.

rewrite(*term*)

Given a base term, returns a list of related terms based on the Wikipedia category API.

For example, visit your favorite Wikipedia page and look for the list of Categories at the very bottom of the page.

Parameters **term** (*str*) – a string to rewrite

Returns a list of semantically related strings, including `term`

Return type list(str)

class rewriter.**Word2VecRewriter**(*model_path*, *create=False*, *corpus=None*, *bigrams=True*)

A class to rewrite queries using Word2Vec, an NLP package that finds semantically related words and phrases to inputted words and phrases. Word2Vec must be trained on a user-provided dataset before it is used.

__init__(*model_path*, *create=False*, *corpus=None*, *bigrams=True*)

Initializes the rewriter, given a particular Word2Vec corpus. A good example corpus is the Wikipedia Text8Corpus. You only need the corpus if you are recreating the model from scratch.

If `create == True`, this generates a new Word2Vec model (which takes a really long time to build.) If `False`, this loads an existing model we already saved.

Parameters

- **model_path** (*str*) – where to store the model files. This file needn't exist, but its parent folder should.
- **create** (*bool*) – True to create a new Word2Vec model, False to use the one stored at `model_path`.
- **corpus** (*Iterable*) – only needed if `create=True`. Defines a corpus for Word2Vec to learn from.
- **bigrams** (*bool*) – only needed if `create=True`. If True, takes some more time to build a model that supports bigrams (e.g. *new_york*). Otherwise, it'll only support one-word searches. `bigram=True` makes this slower but more complete.

decode_term(*encoded*)

Converts an encoded search term into something more human readable, like *hadrians_wall* to *hadrians wall*.

Parameters **term** (*str*) – a cleaned term from *Word2VecRewriter:encode_term*.

Returns a more human-readable version of the inputted term.

Return type str

encode_term(*term*)
Converts a search term like *Hadrian's Wall* to *hadrians_wall*, which plays better with Word2Vec. Primarily for internal use.

Parameters **term** (*str*) – a search term you'd normally feed into *Word2VecRewriter.rewrite*.

Returns a cleaned up version of the term, which works better in *rewrite*.

Return type str

rewrite(*term*)
Rewrites a term to a list of new terms to search with. These words are the *k* most similar words and phrases to the inputted term, as judged by Word2Vec. Here, `k==10`.

Parameters **term** (*str*) – a string to rewrite

Returns a list of semantically related strings, including `term`

Return type list(str)

Search

The classes in this module

Visit the interactive demo to learn how to use these classes.

Usage

First, `pip install searchbetter`.

Then, in your Python code:

```
from searchbetter import search
```

Documentation

class search.**EdXSearchEngine**(*dataset_path*, *index_path*, *create=False*)
edX

__init__(*dataset_path*, *index_path*, *create=False*)
Creates a new search engine that searches over edX courses.

Parameters

- **{string}** (*index_path*) – the path to the edX course listings file.
- **{string}** – the path to a folder where you'd like to store the search engine index. The given folder doesn't have to exist, but its *parent* folder does.
- **{bool}** (*create*) – If True, recreates an index from scratch. If False, loads the existing index

count_words()
Returns the number of words in the underlying Udacity dataset.

create_index()
Creates a new index to search the dataset. You only need to call this once; once the index is created, you can just load it again instead of creating it afresh all the time.

Returns the index object.

class search.**GenericSearchEngine**

An abstract class for any search engine, whether that's an external API you've already built or a Whoosh-based search engine you can make from scratch via searchbetter.

This class encapsulates some useful functionality like query rewriting that can benefit any search engine, even one not made using SearchBetter tools.

Extending this class is easy - you just need to provide a search function and a few other details, and we'll build in functionality from there.

process_raw_results(*raw_results*)

After rewriting, we'll pass the full list of results in here for you to clean up. This could include sorting, removing duplicates, etc. (What you can do, and how you do it, really depends on what kind of objects your search engine returns.)

search(*term*)

Runs a plain-English search and returns results. :param term {String}: a query like you'd type into Google. :return: a list of dicts, each of which encodes a search result.

set_rewriter(*rewriter*)

Sets a new query rewriter (from this_package.rewriter) as the default rewriter for this search engine.

single_search(*term*)

Runs the search engine on a single term (no rewriting or anything), returning a list of objects.

Subclasses must implement!

Parameters **term** (*str*) – a word or phrase to search for

Returns a list of objects that were found. Can be anything: dicts, strings, custom objects, whatever.

Return type list(object)

class search.**HarvardXSearchEngine**(*dataset_path*, *index_path*, *create=False*)

HX

__init__(*dataset_path*, *index_path*, *create=False*)

Creates a new HarvardX search engine. Searches over the HarvardX/DART database of all courses and course materials used in HarvardX. This includes videos, quizzes, etc.

TODO: consider renaming to DART, probz

Parameters

- **{string}** (*index_path*) – the path to the HarvardX course catalog CSV file.
- **{string}** – the path to a folder where you'd like to store the search engine index. The given folder doesn't have to exist, but its *parent* folder does.
- **{bool}** (*create*) – If True, recreates an index from scratch. If False, loads the existing index

create_index()

Creates a new index to search the dataset. You only need to call this once; once the index is created, you can just load it again instead of creating it afresh all the time.

Returns the index object.

class search.**PrebuiltSearchEngine**(*search_fields*, *index_path*)

A search engine designed for when you're just given a model file and can use that directly without having to build anything.

class search.**UdacitySearchEngine**(*dataset_path*, *index_path*, *create=False*)
Udacity

__init__(*dataset_path*, *index_path*, *create=False*)
Creates a new Udacity search engine.

Parameters

- **{string}** (*index_path*) – the path to the Udacity API JSON file.
- **{string}** – the path to a folder where you'd like to store the search engine index. The given folder doesn't have to exist, but its *parent* folder does.
- **{bool}** (*create*) – If True, recreates an index from scratch. If False, loads the existing index

count_words()
Returns the number of words in the underlying Udacity dataset.

create_index()
Creates a new index to search the Udacity dataset. You only need to call this once; once the index is created, you can just load it again instead of creating it afresh all the time.

class search.**WhooshResult**(*dict_data*, *score*)
Encodes a search result from a Whoosh-based search engine. Basically a wrapper around a result dict and its relevance score (higher is better).

get_dict()
Get the underlying dict data

class search.**WhooshSearchEngine**(*create*, *search_fields*, *index_path*)
An abstract class for custom, Whoosh-based search engines.

A batteries-included search engine that can operate on any given dataset. Uses the Whoosh library to index and run searches on the dataset. Has built-in support for query rewriting.

__init__(*create*, *search_fields*, *index_path*)
Creates a new search engine.

Parameters

- **{bool}** (*create*) – If True, recreates an index from scratch. If False, loads the existing index
- **{str[]}** (*search_fields*) – An array names of fields in the index that our search engine will search against.
- **{str}** (*index_path*) – A relative path to a folder where the whoosh index should be stored.

create_index()
Creates and returns a brand-new index. This will call get_empty_index() behind the scenes. Subclasses must implement!

get_empty_index(*path*, *schema*)
Makes an empty index file, making the directory where it needs to be stored if necessary. Returns the index.

This is called within create_index(). TODO this breakdown is still confusing

get_num_documents()
Returns the number of documents in this search engine's corpus. That is, this is the size of the search engine.

load_index()
Used when the index is already created. This just loads it and returns it for you.

single_search(*term*)
Helper function for search() that just returns search results for a single, non-rewritten search term. Returns a list of results, each of which is a Result object. The makeup of the results objects varies from search engine to search engine.

OVERRIDDEN from GenericSearchEngine.

search.**pack_byte**()
S.pack(v1, v2, ...) -> string

Return a string containing values v1, v2, ... packed according to this Struct's format. See struct.__doc__ for more on format strings.

search.**unpack_byte**()
S.unpack(str) -> (v1, v2, ...)

Return tuple containing values unpacked according to this Struct's format. Requires len(str) == self.size. See struct.__doc__ for more on format strings.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

s

Index

Symbols

C

D

E

G

H

L

P

R

S

U

W